FOSSEE Semester-Long Internship Report

On

EXTENDING THE CFDof ADD-ON IN FreeCAD: DYNAMIC TURBULENCE MODEL CONFIGURATION FOR OpenFOAM SIMULATIONS

Submitted by

ANUGYA CHAUBEY

Bachelor of Technology (B.Tech.)
VIT Bhopal University

Principal Investigator

Prof. Parees Palkar

FOSSEE
Indian Institute of Technology Bombay

July 2026

Acknowledgement

I would like to express my sincere gratitude to the **FOSSEE (Free/Libre and Open Source Software for Education) team at the Indian Institute of Technology (IIT) Bombay** for providing me with the opportunity to participate in the **Semester-Long Internship Program**. Working as a member of the **OpenFOAM GUI** project has been an intellectually enriching and professionally rewarding experience. The internship provided an excellent platform to contribute to the development of an engineering software tool while gaining valuable exposure to open-source software development, computational fluid dynamics (CFD), and collaborative software engineering practices.

I am especially grateful to my mentor for their constant guidance, technical insights, and constructive feedback throughout the internship. Regular progress reviews, along with timely reviews of pull requests and code implementations, greatly enhanced both the quality of this project and my learning experience.

I would also like to express my sincere appreciation to **Prof. Parees Palkar** and **Prof. Evan Fernandes** for their invaluable support, encouragement, and vision in leading the FOSSEE initiative. Their efforts in promoting open-source engineering software have created exceptional learning opportunities for students and have inspired countless contributors to participate in meaningful technical projects.

My heartfelt thanks also go to all the members of the **OpenFOAM GUI** development team. Their collaboration, technical discussions, code reviews, and willingness to share knowledge fostered a highly supportive and productive development environment. Working alongside such a dedicated team helped me understand the importance of teamwork, version control, software maintainability, and systematic engineering design in large-scale open-source projects.

I would also like to express my gratitude to my home institution for providing the academic environment and foundational knowledge that enabled me to undertake this internship successfully.

This internship has significantly strengthened my technical skills in **Python, FreeCAD, OpenFOAM, and the CfdOF workbench**, while instilling in me the values of collaboration, perseverance, systematic debugging, and professional responsibility.

INDEX

List of Figures

List of Tables

List of Equations

Abstract

FreeCAD's **CfdOF** workbench provides an accessible graphical front-end for setting up and running OpenFOAM Computational Fluid Dynamics (CFD) simulations. However, the stock CfdOF interface exposes only a limited subset of OpenFOAM's turbulence modelling capabilities: users are restricted to choosing a broad category (Laminar, RANS, or LES) and must manually edit raw OpenFOAM dictionary files – such as *turbulenceProperties* and *fvSchemes* – to select a specific model or tune model coefficients. This project report presents the design, implementation, and debugging of a **dynamic, data-driven turbulence model configuration system** for the CfdOF add-on.

The work introduces a central turbulence model registry (*CfdTurbulenceModels.py*) that stores every supported RANS, LES, and DES model together with its default coefficients and OpenFOAM keyword mapping. This registry drives a redesigned Qt task panel that dynamically populates a model-selection dropdown and generates coefficient-editing spin boxes on the fly, while new FreeCAD document properties (`TurbulenceModel`, `TurbulenceModelCoeffs`, `PrintCoeffs`) persist the user's selections. The case-writer backend (*CfdCaseWriterFoam.py*) was extended to merge user overrides with registry defaults and generate a syntactically correct *constant/turbulenceProperties* file automatically.

During integration testing, a critical pipeline defect caused every turbulence coefficient to be written as `0.0`, silently corrupting generated cases. A systematic root-cause analysis traced this to a chain of five interacting bugs spanning the UI, the model registry, and the case writer, all of which were isolated and fixed. Finally, an automation script (*runCFD.sh*) was developed to reconcile CfdOF-generated cases with OpenFOAM v2306's newer *momentumTransport* dictionary format, supply the missing pressure-reference entries required by `pimpleFoam`, and execute the meshing-to-solver pipeline end-to-end inside a WSL2 Ubuntu environment, achieving an overall 20–50$\times$ reduction in simulation turnaround time through binary mesh I/O, parallel execution, and reduced write frequency. The resulting system allows a user to select a turbulence model and tune its coefficients entirely from within FreeCAD, with no manual editing of OpenFOAM dictionaries required.

1 Introduction

1.1 Background of the Topic

The **FOSSEE (Free/Libre and Open Source Software for Education) project** is a national initiative promoted by the **Indian Institute of Technology (IIT) Bombay**, funded by the National Mission on Education through Information and Communication Technology (NMEICT), Ministry of Education, Government of India. FOSSEE encourages the widespread adoption of Free/Libre and Open Source Software (FLOSS) within academic institutions, research laboratories, and engineering colleges across India, reducing dependence on expensive proprietary engineering tools. Among the many domains supported by FOSSEE is a dedicated **OpenFOAM GUI** project, which aims to make **FreeCAD**, a free and open-source parametric 3D CAD modeller, a fully capable graphical front-end for **OpenFOAM** CFD simulations through its **CfdOF** workbench.

FreeCAD's CfdOF add-on already provides several useful capabilities out of the box: geometry import from native FreeCAD models, automated mesh generation using OpenFOAM's meshing utilities, a basic solver setup panel (supporting both steady and transient solvers), boundary-condition assignment through the graphical interface, and the ability to run OpenFOAM simulations directly from within FreeCAD. These features allow a user to build a computational domain, generate a mesh, and execute a basic CFD simulation without touching a terminal.

1.2 Motivation of the Study

Despite these strengths, several advanced OpenFOAM capabilities remain inaccessible through the CfdOF user interface. In particular, turbulence modelling – one of the most frequently tuned aspects of any practical CFD case – is limited to a coarse category selection (Laminar / RANS / LES), with no way to choose a specific model such as `kOmegaSST` or `kEpsilon`, and no way to override model coefficients from the GUI. Enabling any of this functionality currently requires the user to manually locate and edit the *constant/turbulenceProperties* dictionary using a text editor, which reintroduces exactly the syntactic fragility and command-line dependency that the CfdOF add-on was created to eliminate.

The motivation for this project is therefore to close this gap: to design and implement a **dynamic turbulence model configuration system** inside CfdOF that lets a user pick a specific turbulence model and tune its coefficients entirely from the FreeCAD interface, with the corresponding OpenFOAM dictionary files generated automatically and correctly every time.

1.3 Problem Statement

The existing CfdOF interface exhibits the following concrete limitations, identified at the outset

of this project:

- **Coarse-grained turbulence category selection only:** Users may choose Laminar, RANS, or LES, but cannot select a specific model (e.g. `kOmegaSST`, `kEpsilon`, `Smagorinsky`) from the UI.
- **No coefficient tuning in the UI:** Model coefficients (e.g. α_{k1}, β_1 for k–ω SST) cannot be adjusted without manually editing OpenFOAM dictionaries.
- **Static, template-based backend:** The case-writer previously used a fixed template to emit *turbulenceProperties*, so user input was never reflected in the generated output files.
- **No support for advanced OpenFOAM features:** Capabilities such as dynamic mesh configuration (*dynamicMeshDict*) and multiphase solver setup are entirely absent from the UI and must be hand-written.
- **Manual configuration is error-prone for beginners:** OpenFOAM's dictionary syntax is strict; a single missing brace or semicolon causes cryptic run-time failures, which is precisely the friction the CfdOF add-on is meant to remove.

Consequently, the core objective of this project was to **extend the CfdOF add-on to expose additional OpenFOAM simulation settings directly inside FreeCAD**, with a primary focus on turbulence model selection, coefficient configuration, and automatic, correct generation of the corresponding OpenFOAM configuration files.

1.4 Literature Review / Existing System Overview

In the pre-project state of the workbench, the CfdOF Physics panel offered only a coarse Laminar / RANS / LES category selection, with no specific model dropdown; any finer configuration required a user to manually open and edit the generated OpenFOAM dictionary files before running a simulation. This workflow – FreeCAD model, basic CfdOF UI, limited simulation options, manual editing of OpenFOAM files, and finally running the simulation – is precisely the sequence this project set out to shorten.

OpenFOAM (Open-source Field Operation And Manipulation) is a widely used open-source C++ library for Computational Fluid Dynamics, offering a large suite of finite-volume solvers and turbulence models. However, its architecture is fundamentally text-file driven: cases are configured through a hierarchy of C++-style dictionaries (*controlDict*, *fvSchemes*, *fvSolution*, *turbulenceProperties*, *transportProperties*, and others), which are strict in syntax and offer little fault tolerance. The CfdOF workbench for FreeCAD (maintained at `https://github.com/jaheyns/CfdOF`) was created specifically to abstract this text-file complexity behind a graphical interface, following a Geometry → Mesh → Physics → Case → Solver pipeline. This project builds directly on top of that existing architecture, extending its *Physics* stage

to support fine-grained turbulence configuration rather than replacing any part of the existing pipeline.

Summary

This chapter introduced the FOSSEE OpenFOAM GUI initiative, the existing capabilities and limitations of the CfdOF add-on, and the motivation for extending its turbulence modelling support. The next chapter summarizes the CFD and turbulence-modelling theory relevant to the models integrated into the UI, before the report proceeds to describe the system architecture and the resulting implementation.

2 CFD and Turbulence Modelling Background

2.1 Governing Equations

OpenFOAM numerically solves the incompressible Navier–Stokes equations using the Finite Volume Method (FVM). The mass continuity equation for an incompressible fluid ($\rho =$ constant) is:

$$\nabla \cdot \mathbf{u} = 0 \tag{2.1}$$

and the momentum (Navier–Stokes) equation is:

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{g} \tag{2.2}$$

where $\mathbf{u}$ is the velocity vector, p is pressure, $\nu = \mu/\rho$ is the kinematic viscosity, and $\mathbf{g}$ is the body-force vector. These two equations form the mathematical foundation of every solver exposed through CfdOF, including `icoFoam`, `simpleFoam`, and `pimpleFoam`.

2.2 Reynolds-Averaged Navier–Stokes (RANS) Framework

High Reynolds-number engineering flows are turbulent and multi-scale; resolving every scale directly (Direct Numerical Simulation) is computationally prohibitive for practical geometries. The RANS approach instead decomposes flow variables into a mean and a fluctuating component and time-averages the Navier–Stokes equations. This introduces an unclosed term, the Reynolds stress tensor, which – via the Boussinesq eddy-viscosity hypothesis – is modelled in terms of a turbulent (eddy) viscosity ν_t. Two-equation RANS models solve additional transport equations to determine ν_t at every point in the domain; the two models most relevant to this project are summarized below, since they are the models most commonly selected by CfdOF users.

2.3 Standard k–ϵ Model

The standard k–ϵ model closes the RANS system by solving transport equations for the turbulent kinetic energy k and its dissipation rate ϵ:

$$\frac{\partial k}{\partial t} + \nabla \cdot (\mathbf{u}k) = \nabla \cdot \left[\left(\nu + \frac{\nu_t}{\sigma_k} \right) \nabla k \right] + P_k - \epsilon \tag{2.3}$$

with the eddy viscosity given by $\nu_t = C_\mu k^2/\epsilon$. The model coefficients (C_μ, C_1, C_2) are precisely the values exposed for editing in the new `TurbulenceModelCoeffs` property described in Section 4.2.

2.4 k–ω SST Model

The k–ω Shear Stress Transport (SST) model, developed by Menter, blends the k–ω formulation near walls with the k–ϵ formulation in the free stream, giving robust predictions across a wide range of adverse pressure-gradient flows. Its transport equations for k and the specific dissipation rate ω involve coefficients such as α_{k1} and β_1:

$$\frac{\partial \omega}{\partial t} + \nabla \cdot (\mathbf{u}\omega) = \frac{\gamma}{\nu_t} P_k - \beta \omega^2 + \nabla \cdot [(\nu + \sigma_\omega \nu_t)\nabla \omega] + \mathrm{CD}_{k\omega} \tag{2.4}$$

`kOmegaSST` is the default RANS model shown throughout this report's generated-output examples, as it was the model most frequently used to validate the new configuration system.

2.5 LES and DES Categories

In addition to RANS models, the turbulence registry developed in this project also enumerates Large Eddy Simulation (LES) models – `Smagorinsky`, `WALE`, and `dynamicKEqn` – which directly resolve large turbulent eddies while modelling only the smallest sub-grid scales, and a single Detached Eddy Simulation (DES) hybrid model, `kOmegaSSTDES`, which switches between RANS and LES treatment depending on local mesh resolution.

Summary

This chapter summarized the governing equations solved by OpenFOAM and the RANS/LES/DES turbulence models relevant to the configuration system built during this project. The following chapter describes how these models were organized into a structured software architecture inside the CfdOF workbench.

3 System Architecture

3.1 CfdOF Repository Structure

Before any code was written, the initial phase of the project was dedicated to understanding the CfdOF repository layout and the FreeCAD document-object / task-panel pattern it follows. Table 3.1 summarizes the key files identified as relevant to the turbulence-modelling extension, and their role in the overall Model → View → Controller-style architecture used by FreeCAD workbenches.

Table 3.1: CfdOF Repository Structure Relevant to This Project

Directory / File	Purpose	Relevance to Project
CfdOF/Solve/CfdPhysicsSelection.py	FreeCAD document object storing all physics settings (solver type, turbulence category, etc.)	Primary file extended to store turbulence model and coefficient data
CfdOF/Solve/CfdCaseWriterFoam.py	Reads model object properties and writes the full OpenFOAM case directory	Extended to write dynamic turbulence and scheme files
Gui/TaskPanelPhysics.ui	Qt Designer XML layout for the Physics task panel	Modified to add the turbulence model dropdown, coefficient controls, and print-coefficients checkbox
CfdOF/Solve/TaskPanelCfdPhysicsSelection.py	Python controller class wiring UI signals to the model object	Extended with signal/slot connections for the new UI elements
CfdOF/Solve/CfdTurbulenceModels.py	Central data dictionary of all supported turbulence models (new file)	Created during this project as the single source of truth for models and default coefficients
Data/defaults/	Template OpenFOAM case files used as a base for generated cases	Referenced for default *fvSchemes* and *turbulenceProperties* content

3.2 Design Principle: Central Model Registry

The core architectural decision made during this project was to store all turbulence-model metadata in a single, central Python dictionary rather than scattering hard-coded model names and coefficient values across the UI and the case writer. This registry, *CfdTurbulenceModels.py*,

groups models by category and stores each model's default coefficients and its OpenFOAM keyword, as summarized in Table 3.2.

Table 3.2: Central Turbulence Model Registry (*CfdTurbulenceModels.py*)

Category	**Models**	**Example Coefficients**
RANS	kOmegaSST, kEpsilon, realizableKE	Cmu, C1, C2, alphaK1, beta1
LES	Smagorinsky, WALE, dynamicKEqn	Ck, Ce
DES	kOmegaSSTDES	Hybrid RANS/LES coefficients
Laminar	None	None

Four helper functions were built on top of this registry and used consistently by both the UI layer and the backend case writer, ensuring that model names, keywords, and coefficients are defined in exactly one place:

- `getModelsForCategory(category)` – returns the list of models available for a RANS/LES/DES/Laminar category, used to populate the dropdown.
- `getModelCoefficients(model)` – returns the default coefficient dictionary for a given model, used to populate the coefficient spin boxes and to merge with user overrides.
- `getOfKeyword(model)` – returns the exact OpenFOAM keyword string (e.g. `RASModel`) expected in *turbulenceProperties*.
- `getSimulationType(category)` – maps a UI category (RANS/LES/Laminar) to the OpenFOAM `simulationType` entry.

Summary

This chapter described the CfdOF files touched by this project and the central-registry design pattern adopted to avoid hard-coding turbulence data in multiple places. The next chapter walks through the implementation of this architecture in detail, including the UI redesign, the backend property changes, the case-writer integration, and the debugging of a critical pipeline defect.

4 Design and Implementation

4.1 Environment Setup and OpenFOAM Familiarization

The initial phase of implementation focused on building the domain knowledge required to modify OpenFOAM-generating code safely. Activities included exploring FreeCAD's interface and the Part / Part Design workbenches, creating simple parametric solids (cube, cylinder, and a hollow pipe built with a Boolean cut), and studying the canonical five-stage CFD workflow: geometry creation, mesh generation, boundary-condition definition, solver execution, and post-processing.

Ubuntu was installed via the Windows Subsystem for Linux (WSL2), and OpenFOAM v2306 was installed and verified by running the standard `cavity` lid-driven cavity tutorial to completion. This established a working, independent OpenFOAM environment against which the FreeCAD-generated cases could later be validated. The OpenFOAM case directory structure and its key dictionaries – *fvSchemes* (numerical discretization schemes), *fvSolution* (solver settings), *transportProperties* (fluid properties), and *turbulenceProperties* (turbulence model configuration) – were studied in detail, and the divergence scheme in *fvSchemes* was manually changed from `Gauss linear` to `Gauss upwind` to compare accuracy against numerical stability.

This manual, hands-on debugging phase surfaced several practical OpenFOAM errors – missing file headers, incorrect scheme syntax, missing turbulence-related files, and incorrect transport property definitions – all of which had to be understood and fixed by hand before any automation could be trusted to reproduce the same files programmatically. This step was considered essential: since OpenFOAM is a strictly file-driven system, UI-based automation implemented without a precise understanding of syntax and required parameters would simply produce incorrect output faster.

4.2 Dynamic Turbulence Model Selection UI

With the domain knowledge in place, the next stage of implementation replaced the static category-only turbulence setup with a dynamic, reactive UI. Three files were modified: *CfdPhysicsSelection.py* (backend data model), *TaskPanelPhysics.ui* (Qt Designer layout), and *TaskPanelCfdPhysicsSelection.py* (controller logic).

At the backend level, three new FreeCAD document properties were added to the physics object:

- `TurbulenceModel` (String) – stores the specific selected model name (e.g. `"kOmegaSST"`).
- `TurbulenceModelCoeffs` (Map) – stores any coefficient values the user has explic-

itly overridden.

- `PrintCoeffs` (Bool) – controls the OpenFOAM `printCoeffs` flag written to *turbulenceProperties*.

On the UI side, the old static `turbulenceComboBox` and `turbulenceModelFrame` widgets were removed and replaced with a `QComboBox` for model selection, a `QCheckBox` for the print-coefficients flag, and a `QGroupBox` container intended to later host dynamic coefficient controls. The controller logic populates the dropdown according to the selected radio button (RANS / LES / DES), using the model lists defined in Table 3.2, so that the dropdown reacts instantly to the user's category selection:

```
cb_turbulence_model.currentIndexChanged.connect(
    self.onModelChanged
)


# accept()
self.obj.TurbulenceModel = self.form.cb_turbulence_model.currentText()
self.obj.PrintCoeffs = self.form.check_print_coeffs.isChecked()
```

Implementing this stage required resolving several practical Qt/FreeCAD integration issues, including malformed *.ui* XML after widget removal, `AttributeError` exceptions caused by stale references to the removed widgets, a dropdown that failed to refresh on category change, and FreeCAD property caching that required a full application restart to observe. Table 4.1 summarizes the resulting improvement over the original system.

Table 4.1: Turbulence Configuration: Old System vs. New System

Feature	**Old**	**New**
Model Selection	Category only	Specific models
UI Response	Static	Dynamic
Persistence	No	Yes
Coefficients	Not available	Planned in next implementation stage

Figure 4.1 shows the resulting turbulence model selection panel as it appears inside the FreeCAD Physics task panel, with the category radio buttons, the dynamically populated model dropdown, and the print-coefficients checkbox described above.

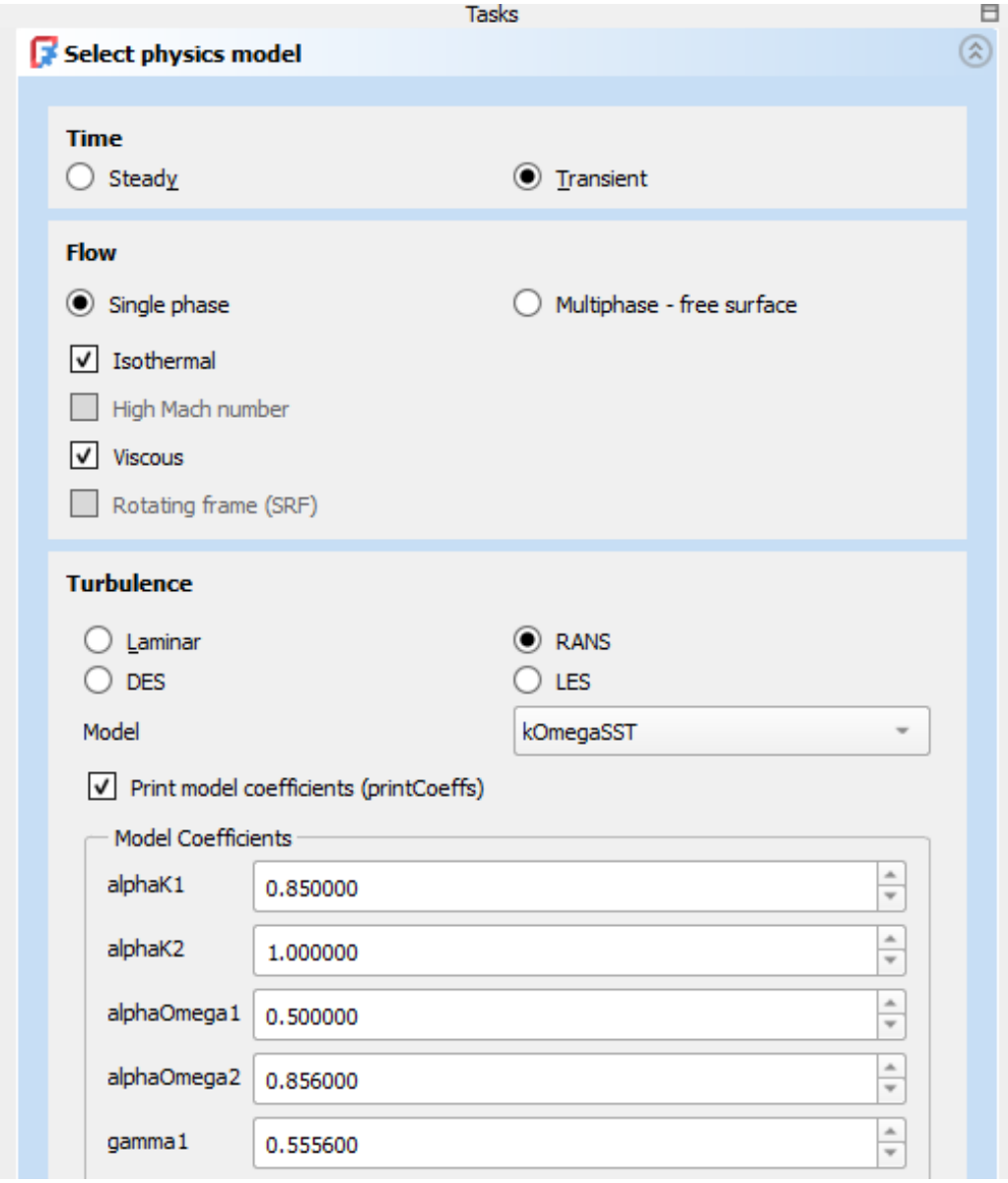

Figure 4.1: Dynamic turbulence model selection panel inside the FreeCAD CfdOF Physics task panel.

The same registry-driven, dynamically populated dropdown pattern established for turbulence model selection was also applied to the numerical-scheme configuration controls, allowing the *fvSchemes* and *fvSolution* settings relevant to the active solver and turbulence category to be selected directly from the task panel rather than edited by hand. Figure 4.2 shows this dynamic scheme/solution selection interface.

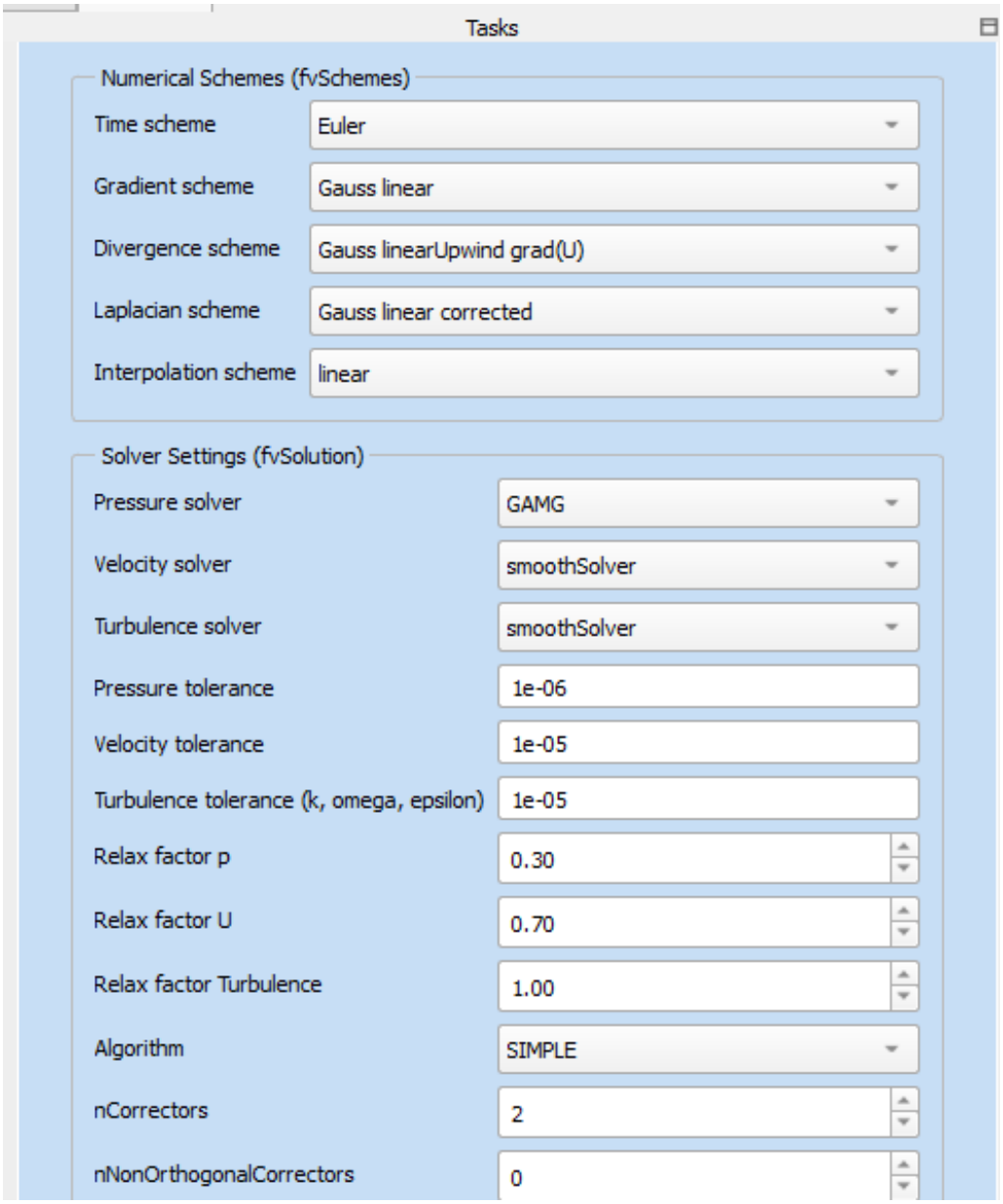

Figure 4.2: Dynamic *fvSchemes* and *fvSolution* file selection interface inside the FreeCAD CfdOF task panel.

4.3 Coefficient Editing and OpenFOAM File Generation

The following implementation stage completed the loop from UI selection through to a written OpenFOAM file. The *CfdTurbulenceModels.py* registry (Table 3.2) and its four helper functions were finalized, and the coefficient `QGroupBox` was made fully dynamic: whenever the selected model changes, the panel is rebuilt by reading the model's default coefficients from the registry and overlaying any values the user had previously saved:

```
coeffs = getModelCoefficients(model)
saved  = obj.TurbulenceModelCoeffs
```

```
for name, default in coeffs.items():
    value = saved.get(name, default)
    # spin box created and pre-filled with `value`
```

The case-writer backend (*CfdCaseWriterFoam.py*) was then extended to read the selected model and coefficients from the physics object and generate *constant/turbulenceProperties* directly, replacing the previous fixed template. The generated fields include the OpenFOAM `simulationType`, the specific model keyword (e.g. `RASModel`), the coefficient sub-dictionary, and the `printCoeffs` flag, producing output such as:

```
simulationType RAS;
RASModel       kOmegaSST;

kOmegaSSTCoeffs
{
    alphaK1  0.85;
    beta1    0.075;
}
```

The full pipeline was validated end-to-end against the WSL2 OpenFOAM v2306 installation configured earlier in the project, following the sequence User Input → Category → Model → Coefficients → Backend → File → OpenFOAM. At this stage, the system was assessed as substantially complete, with one critical defect remaining: every coefficient in the generated file was being written as `0.0` rather than its correct default or overridden value. Investigating and resolving this defect became the primary focus of the following implementation stage.

4.4 Root-Cause Diagnosis of the Zero-Coefficient Bug

Rather than patching the symptom directly, this stage began with a systematic root-cause analysis, which revealed that the zero-coefficient defect was **not a single bug but a chain of five interacting failures** spanning the UI layer, the model registry, and the case writer, each of which silently amplified the next. Table 4.2 lists each defect and its corresponding fix.

Table 4.2: Bugs Identified and Resolved

#	File	Bug	Fix
1	*CfdCaseWriterFoam.py*	Wrong argument passed into the coefficient-lookup call	Passed the selected model name explicitly
2	*CfdCaseWriterFoam.py*	FreeCAD's `PropertyMap` type is not a native Python `dict`, so `.items()` silently returned nothing useful	Explicitly converted with `dict(obj.TurbulenceModelCoeffs)`
3	UI Task Panel	Coefficient spin boxes were never told to display the loaded value	Added an explicit `spinBox.setValue(value)` call
4	*CfdTurbulenceModels.py*	The registry's default coefficient dictionary was returned by reference, so overriding one field mutated the shared defaults for all models	Returned a `.copy()` of the coefficient dictionary
5	*CfdCaseWriterFoam.py*	An unnecessary `float()` conversion crashed on already-numeric `PropertyMap` values	Removed the redundant conversion

Tracing the failure end-to-end showed exactly how these five defects compounded: the UI displayed `0.0` because the spin box was never explicitly set (#3); this incorrect `0.0` was then saved back as a user "override" (#1); the `PropertyMap` object failed to convert cleanly into a plain dictionary during merging (#2); as a result the registry's defaults were being ignored entirely during the merge (#4); and the wrong `simulationType` keyword was ultimately selected by the case writer as a downstream consequence – so the entire generated pipeline produced zeros, even though each individual component appeared to work correctly in isolation.

Two representative fixes are shown below. The first corrects the missing UI update:

```
# BEFORE (broken): value read but never displayed
value = saved.get(coeff, defaultVal)

# AFTER (fixed): value read, converted, and shown in the UI
value = float(saved.get(coeff, defaultVal))
spinBox.setValue(value)
```

The second establishes the correct default/override merge order, ensuring registry defaults are always preserved unless explicitly overridden by the user:

```
defaults  = getModelCoefficients(model)
overrides = dict(obj.TurbulenceModelCoeffs)

merged = defaults.copy()
merged.update(overrides)
```

A temporary debug print statement was added to the case writer to independently verify the model, the registry defaults, the user overrides, and the final merged dictionary at run time, confirming (for the `kOmegaSST` model with no user overrides) that `Merged` correctly matched `Defaults: {alphaK1: 0.85}`. With all five defects resolved, the generated *turbulenceProperties* file once again contained correct, non-zero coefficient values, and the key lesson – that FreeCAD's `PropertyMap` is not interchangeable with a native Python dictionary, that UI widgets must always be explicitly updated with `setValue()`, and that shared mutable defaults must always be `.copy()`-ed before being merged – was documented for the rest of the team.

4.5 OpenFOAM v2306 Compatibility and Solver Automation

With turbulence-coefficient generation verified as correct, the final implementation stage addressed a separate class of problem: structural incompatibilities between CfdOF-generated cases and the newer OpenFOAM v2306 solver conventions, which caused `pimpleFoam` to fail even on an otherwise correctly configured case. Three specific defects were diagnosed and fixed:

- **Missing pressure reference (`pRefCell`):** `pimpleFoam` still validates the legacy `SIMPLE` control block even in fully transient PIMPLE mode, and fails fatally (`"Please supply pRefCell or pRefPoint"`) if it is absent. The fix adds a minimal `SIMPLE { pRefCell 0; pRefValue 0; }` block without altering the active `PIMPLE` solver settings.
- **Missing *momentumTransport* dictionary:** OpenFOAM v2306 solvers expect turbulence configuration under the newer *constant/momentumTransport* filename, while CfdOF still generates the legacy *turbulenceProperties* name only. The automation script copies the generated *turbulenceProperties* content into a correctly named *momentumTransport* file (renaming the internal `object` field accordingly) without modifying any physics values – purely a structural fix.
- **Invalid *controlDict* include:** A stale `#include "turbulenceLib"` directive left over from an older template caused a fatal parse error; it is stripped automatically with `sed -i '/turbulenceLib/d' system/controlDict`.

An explicit design constraint was maintained throughout this stage: **the automation script performs only structural fixes and never modifies user-specified physics or turbulence**

values – and, critically, the legacy `SIMPLE` block is added purely to satisfy `pimpleFoam`'s reference-pressure check and must never be used to replace the active `PIMPLE` solver configuration, since doing so would silently change the solution algorithm the user selected.

A single automation script, *runCFD.sh*, was written to apply all of the above fixes automatically and then drive the complete pipeline: mesh generation, copying the mesh into the correct decomposed-case structure, applying the structural fixes above, converting the mesh to binary format, running the solver (optionally in parallel via MPI), and reconstructing the parallel results into a single serial case, following the sequence Mesh → Copy Mesh → Fix Files → Convert Format → Run Solver → Reconstruct.

Alongside correctness fixes, several performance optimizations were incorporated into the automation script, together yielding an overall speed-up of **20–50**× relative to the naive, unoptimized pipeline (Table 4.3).

Table 4.3: Solver Performance Optimizations

Optimization	**Speed-up**
Binary mesh format	3–5×
Parallel execution (4 cores, MPI)	3–4×
Reduced mesh size (target 100k–300k cells)	5–10×
Reduced write frequency	1.5×
Total (combined)	**20–50**×

Summary

This chapter traced the implementation from environment setup and manual OpenFOAM familiarization, through the dynamic turbulence-model selection UI and automatic *turbulenceProperties* generation, to the diagnosis and resolution of a five-part chained defect that had been silently zeroing all coefficients, and finally to OpenFOAM v2306 solver-compatibility fixes together with a fully automated, performance-optimized *runCFD.sh* pipeline. The following chapter presents the verification results and generated output produced by the completed system.

5 Results and Discussion

5.1 Verified Generated Output

After the defect-resolution work described in Section 4.4, the complete UI → Backend → Case Writer → OpenFOAM pipeline was re-verified. Selecting `kOmegaSST` with no coefficient overrides consistently produced:

```
simulationType RAS;
RASModel       kOmegaSST;

kOmegaSSTCoeffs
{
    alphaK1  0.85;
    beta1    0.075;
}
```

matching the registry's documented default values exactly, and any coefficient explicitly changed by the user in the UI was confirmed to correctly override the corresponding default in the generated file, while all other coefficients remained at their registry defaults – the intended behaviour of the merge logic described in Section 4.3.

5.2 Solver Execution Verification

Using the *runCFD.sh* automation script on the OpenFOAM v2306 / WSL2 Ubuntu configuration described in Table 5.1, cases generated by the extended CfdOF add-on were meshed with `cfMesh`, fixed for solver compatibility, and executed with `pimpleFoam` without any of the structural errors described in Section 4.5 recurring and without any manual intervention.

Table 5.1: Verified System Configuration

Parameter	**Value**
OpenFOAM Version	2306
Solver	pimpleFoam
Mesh Generator	cfMesh
Parallel Execution	4 cores (MPI)
Operating System	WSL2 Ubuntu
Automation Script	*runCFD.sh*

5.3 Acceptance Criteria and Testing Summary

Prior to closing out the coefficient bug, the following acceptance criteria were defined and subsequently verified against the fixed pipeline, as summarized in Table 5.2.

Table 5.2: Acceptance Criteria Verification Summary

Criterion	Result
Correct default coefficients written when no override is given	PASSED
User coefficient overrides correctly applied	PASSED
Correct `simulationType` (RAS / LES / laminar) written	PASSED
`printCoeffs` flag correctly reflected in output	PASSED
No run-time errors during case generation or solving	PASSED

Summary

This chapter presented the verified output of the completed turbulence-configuration pipeline, confirming that both the coefficient-generation defect (Section 4.4) and the OpenFOAM v2306 solver-compatibility issues (Section 4.5) were fully resolved, and that the acceptance criteria defined for the feature were all satisfied. The following two chapters describe a further extension built on top of this configuration system – an integrated turbulence parameter calculator and its associated automatic field-initialization pipeline – before the report concludes with a discussion of the broader significance of these results, the current limitations of the system, and directions for future work.

6 Turbulence Parameter Calculator

6.1 Objective

The turbulence-model configuration system described in the preceding chapters allows a user to select a specific RANS, LES, or DES model and tune its coefficients from within FreeCAD. It does not, however, address a second and equally common source of manual effort in CFD case setup: computing the numerical values used to *initialize* the turbulence field variables in the OpenFOAM *0/* directory. Prior to this extension, a user who had selected a turbulence model still had to compute the turbulent kinetic energy, dissipation rate, or specific dissipation rate by hand – using an external calculator or spreadsheet – and manually transcribe the results into the *0/k*, *0/epsilon*, and *0/omega* files before a simulation could be run. This chapter presents the turbulence parameter calculator, a GUI-driven component that automates this computation and links it directly into the case-generation pipeline, removing the manual transcription step entirely.

6.2 Turbulence Quantity Formulation

Turbulence field initialization in OpenFOAM is conventionally expressed in terms of a reference velocity U, a characteristic geometric length scale L, and a turbulence intensity I. The turbulent kinetic energy is obtained from the velocity fluctuation implied by the turbulence intensity:

$$k = \frac{3}{2}\left(UI\right)^2 \tag{6.1}$$

Given k and a characteristic length scale, the turbulent dissipation rate follows from a mixing-length closure using the model constant C_μ:

$$\epsilon = \frac{C_\mu^{3/4}\, k^{3/2}}{L} \tag{6.2}$$

and the specific dissipation rate, required by k–ω family models, follows directly from k and ϵ:

$$\omega = \frac{\epsilon}{C_\mu\, k} \tag{6.3}$$

These three relations form the computational core of the calculator: k is required to initialize every turbulent case regardless of model family, while ϵ and ω are required selectively depend-

ing on whether the active model belongs to the k–ϵ or k–ω family, as summarized in Table 6.2.

6.3 The CfdTurbulenceCalculator Object

The calculator is implemented as CfdTurbulenceCalculator, a FreeCAD document object that follows the same property-based pattern already used elsewhere in CfdOF – most notably by *CfdPhysicsSelection.py* for turbulence model storage. Representing the calculator as a document object, rather than a transient dialog computation, allows its inputs and computed outputs to persist with the FreeCAD document and to be inspected or edited later through the standard property editor, consistent with the rest of the workbench's Model-View-Controller architecture.

Table 6.1: CfdTurbulenceCalculator Object Properties

Property	Type	Role
ReferenceVelocity	Float (input)	Characteristic inlet or free-stream velocity U used in the k relation.
CharacteristicLength	Float (input)	Geometric length scale L (e.g. hydraulic diameter) used in the ϵ relation.
TurbulenceIntensity	Float (input)	Fractional turbulence intensity I at the domain inlet.
Cmu	Float (input)	Turbulence model constant C_μ, defaulted from the active model's registry entry.
TurbulentKineticEnergy	Float (computed)	Stores the resulting k value.
DissipationRate	Float (computed)	Stores the resulting ϵ value.
SpecificDissipationRate	Float (computed)	Stores the resulting ω value.

The four input properties are exposed for editing through the task panel described in Section 6.4, while the three computed properties are recalculated by the object's execute() method whenever an input changes, in keeping with FreeCAD's standard recompute mechanism. Because the computed values are ordinary document-object properties, downstream components – most importantly the case writer described in Section 6.5 – can read them directly through the FreeCAD document API without re-implementing the underlying turbulence relations or duplicating the calculator's logic. This separation keeps the turbulence formulae defined in exactly one place and allows the calculator to be reused independently of case generation, for example to report the initialization values to the user before a simulation is run.

6.4 GUI Integration

The calculator is exposed to the user through the same command-and-task-panel pattern used

throughout the FreeCAD workbench for other CfdOF operations, keeping its interaction model consistent with the rest of the Solve workflow.

- **`CommandCfdTurbulenceCalculator`:** A FreeCAD `Gui.Command` subclass that defines the calculator's toolbar icon, menu text, and activation logic. Activating the command creates (or re-selects) a `CfdTurbulenceCalculator` object in the active document and opens its task panel.
- **Registration in *InitGui.py*:** The command is registered with FreeCAD's command manager and added to the Solve workbench's toolbar and *Solve* menu, alongside the existing physics and meshing commands, so that the calculator appears as a natural extension of the existing solver-setup workflow rather than a separate tool.
- **Task panel:** A dedicated task panel presents input fields for reference velocity, characteristic length, turbulence intensity, and C_μ, together with read-only fields displaying the computed k, ϵ, and ω values. On `accept()`, the panel writes the validated input fields back onto the `CfdTurbulenceCalculator` object's properties and triggers a document recompute, which invokes the object's `execute()` method and refreshes the computed properties before the panel closes.

Figure 6.1 shows the resulting turbulence parameter calculator task panel, with the four editable input fields and the three read-only computed fields described above.

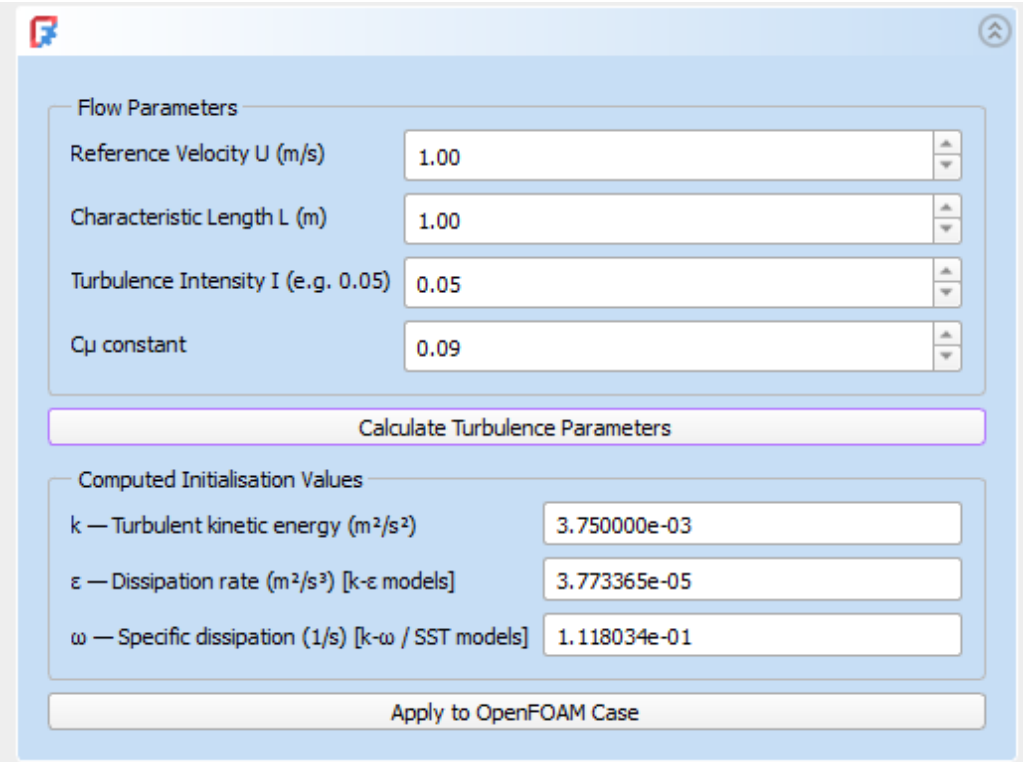

Figure 6.1: Turbulence parameter calculator task panel inside the FreeCAD CfdOF workbench.

This design keeps the task panel a thin presentation layer: it is responsible only for input validation and property assignment, while the turbulence-relation computation itself lives entirely inside the `CfdTurbulenceCalculator` object, so the same computation is available to any other part of the workbench that needs it – including the automatic case-writer integration described next – without going through the GUI at all.

6.5 Automatic Case Integration

To eliminate the manual editing of turbulence field files that motivated this feature, *CfdCaseWriterFoam.py* was extended with a helper routine that runs as part of the standard case-generation sequence, immediately after the existing *turbulenceProperties* generation step described in Section 4.3. The helper performs three tasks in sequence: it retrieves the computed k, ϵ, and ω values from the document's `CfdTurbulenceCalculator` object; it queries the active turbulence model from the physics object (via the same *CfdTurbulenceModels.py* registry used elsewhere in the case writer) to determine which field files are required; and it writes the `internalField` entry of each required file using the corresponding computed value, leaving all other entries governed by the existing OpenFOAM field-file templates untouched.

Table 6.2: Turbulence Field Files Populated by Selected Model Category

Model Category	*0/k*	*0/epsilon*	*0/omega*
k–ϵ family (`kEpsilon`, `realizableKE`)	✓	✓	–
k–ω family (`kOmegaSST`, `kOmegaSSTDES`)	✓	–	✓
Laminar	–	–	–

Because the field-selection logic is driven by the same model registry that already governs *turbulenceProperties* generation, the calculator's case-writer integration automatically stays consistent with the active model selection: changing the model in the Physics panel changes both the generated *turbulenceProperties* content and the set of field files initialized by the calculator, with no additional configuration required from the user.

Benefits

The turbulence parameter calculator delivers the following improvements to the CfdOF simulation-setup workflow:

- **Automated turbulence parameter computation** from standard reference-velocity, length-scale, and intensity inputs, removing the need for external calculators or spreadsheets.
- **GUI-driven initialization** that fits directly into the existing FreeCAD Solve workflow, requiring no OpenFOAM syntax knowledge from the user.

- **Automatic synchronization** between GUI-entered parameters and the generated OpenFOAM case, since the case writer reads computed values directly from the document object.
- **Elimination of manual editing** of the *0/k*, *0/epsilon*, and *0/omega* files.
- **Improved consistency and reproducibility** of turbulence field initialization across simulation cases.

7 Automatic Turbulence Field Initialization During Case Generation

7.1 Objective

While closely related to the calculator described in Section 6, the automatic field-initialization pipeline is documented here as a distinct architectural contribution, since it extends the OpenFOAM case-generation workflow itself rather than the turbulence-parameter computation that feeds it. Its objective is to guarantee that every case generated by CfdOF contains correctly initialized turbulence field variables by construction, so that no simulation can be started with unset or stale field values regardless of how the user arrived at the current model and parameter selection.

7.2 Case-Writer Pipeline Integration

The initialization logic is implemented as a dedicated helper function inside *CfdCaseWriterFoam.py*, invoked automatically as part of the case writer's normal execution sequence rather than as a separate, user-triggered step. The helper is responsible for four coordinated tasks: retrieving the current turbulence quantities from the calculator object; determining the active turbulence model from the physics object; selecting the corresponding OpenFOAM field dictionaries using the mapping in Table 6.2; and writing the computed values into the `internalField` entry of each selected file. Because this logic runs unconditionally during case generation, turbulence field initialization becomes an intrinsic property of every generated case rather than an optional post-processing step that a user could omit.

7.3 Component Interaction

This chapter's contribution is best understood as the integration layer connecting four previously independent components of the CfdOF architecture, illustrated by the data flow in Figure 7.1: the user's *physics selection* determines which turbulence model is active; the *turbulence calculator* computes the numerical quantities that model requires; the *case writer* queries both of these and resolves them into a concrete set of field files to write; and the resulting *generated OpenFOAM dictionaries* reflect both selections without either the physics object or the calculator object needing any awareness of OpenFOAM file syntax.

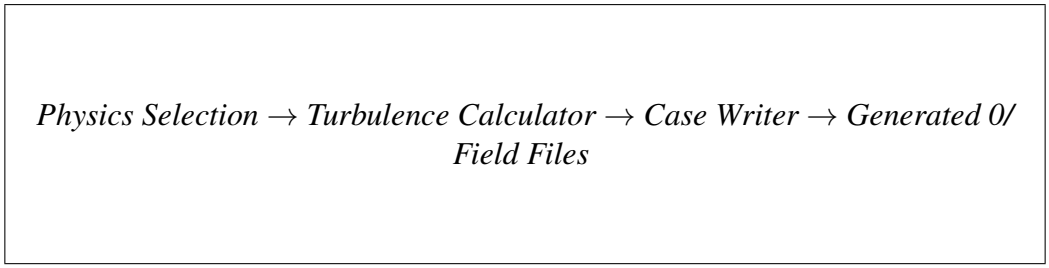

Figure 7.1: *Data flow through the automatic turbulence field-initialization pipeline. Each stage depends only on the interface exposed by the previous stage, not on its internal implementation.*

Restricting each component to a narrow, well-defined interface – the physics object exposes only the selected model name, the calculator exposes only computed scalar quantities, and the case writer alone is responsible for translating those quantities into OpenFOAM syntax – keeps the three components independently maintainable. The turbulence relations implemented by the calculator can be refined, and additional models can be added to the registry, without requiring any change to the case-writer's field-selection logic, since that logic depends only on the model category and not on how the underlying values were derived.

Benefits

The automatic field-initialization pipeline provides the following workflow improvements:

- **Automated OpenFOAM preprocessing**, removing turbulence field initialization as a manual step in the simulation setup workflow.
- **Guaranteed consistency** between the GUI configuration (model and parameters) and the generated case, since field files are always derived from the same document-object state.
- **Reduced manual intervention** and a correspondingly reduced opportunity for transcription error in field initialization.
- **Improved reproducibility** of simulation setup across users and cases.
- **A simplified end-to-end workflow** for users creating CFD simulations, extending the Geometry → Mesh → Physics → Case → Solver pipeline described in Section 1.4 with fully automated turbulence-field preprocessing.

8 Discussion and Conclusion

8.1 Summary of Findings

This project extended FreeCAD's CfdOF add-on with a dynamic, data-driven turbulence-model configuration system, replacing a static, category-only interface with one capable of selecting specific RANS, LES, or DES models and tuning their coefficients directly from the FreeCAD GUI. The principal engineering outputs of this work are:

- **Central Turbulence Model Registry:** A single source of truth (*CfdTurbulenceModels.py*) enumerating every supported model, its category, default coefficients, and OpenFOAM keyword, eliminating hard-coded model data from the UI and backend.
- **Dynamic, Reactive UI:** A redesigned Qt task panel that repopulates the model dropdown and coefficient input fields automatically based on the user's category and model selection, with values persisted as native FreeCAD document properties.
- **Correct Default/Override Merge Logic:** A verified merge pipeline that always preserves registry defaults unless explicitly overridden, addressing a five-part chained defect that had previously caused every generated coefficient to be silently zeroed.
- **Automatic OpenFOAM File Generation:** An extended case writer that generates a syntactically correct *constant/turbulenceProperties* file automatically, removing the need for any manual dictionary editing.
- **Solver Compatibility Automation:** A *runCFD.sh* script that reconciles CfdOF output with OpenFOAM v2306 conventions (pressure reference, *momentumTransport* naming, *controlDict* includes) and drives the full meshing-to-solver pipeline with a 20–50$\times$ performance improvement, without ever modifying user-specified physics.

8.2 Limitations

The current implementation exhibits the following constraints:

- **Coefficient coverage:** The registry currently documents the most commonly tuned coefficients for each model; less common coefficients are not yet exposed and would need to be added to *CfdTurbulenceModels.py* as they are identified.
- **Dynamic mesh and multiphase support:** As identified in the original project proposal, UI support for *dynamicMeshDict* configuration and multiphase solver setup (*transportProperties* for multiple phases) remains outside the scope completed during this project and is left for future work.
- **FreeCAD `PropertyMap` quirks:** As discovered during the root-cause debugging de-

scribed in Section 4.4, FreeCAD's `PropertyMap` type does not behave identically to a native Python dictionary, and any future extension of this feature must continue to explicitly convert and copy this type to avoid re-introducing similar defects.

- **Manual OpenFOAM v2306 compatibility layer:** The *runCFD.sh* script currently patches known incompatibilities as a post-processing step rather than CfdOF's case writer natively emitting the *momentumTransport* filename; a cleaner long-term solution would update the case writer itself once broader OpenFOAM version support is planned.

8.3 Future Work

Building on the architecture established in this project, natural next steps include:

1. **Dynamic Mesh Support:** Implementing a UI panel to enable and configure dynamic mesh simulations, automatically generating the *dynamicMeshDict* file for moving-boundary or mesh-deformation cases.
2. **Multiphase Flow Setup:** Adding UI options for selecting multiphase solvers and defining per-phase fluid properties, with automatic generation of *transportProperties* for multiphase cases.
3. **Numerical Scheme Configuration UI:** Extending the same registry-driven pattern used for turbulence models to *fvSchemes*, allowing users to select discretization schemes (gradient, divergence, Laplacian) and named preset profiles from the GUI.
4. **Native *momentumTransport* Support in the Case Writer:** Migrating the structural fix currently implemented in *runCFD.sh* directly into *CfdCaseWriterFoam.py*, so that OpenFOAM v2306-compatible cases are generated natively without a post-processing script.
5. **Automated Regression Testing:** Building a small suite of sample cases (one per turbulence category) that can be regenerated and diffed automatically on every code change, to catch pipeline regressions of the kind identified in Section 4.4 before they reach a user.

In conclusion, this project successfully extended the CfdOF add-on so that a user can select a specific turbulence model, tune its coefficients, and run a fully automated OpenFOAM simulation entirely from within FreeCAD – without manually editing a single OpenFOAM dictionary file. By replacing a static, category-only interface with a registry-driven, dynamically reactive one, and by rigorously tracing and resolving a chained, five-part pipeline defect rather than merely patching its symptom, this implementation directly supports the FOSSEE initiative's mandate to make accessible, robust, open-source engineering software available to academic institutions.

9 Appendix

9.1 Complete File Modification Log

Table 9.1: Complete List of Files Created or Modified During This Project

File	Change Type	Purpose
CfdOF/Solve/CfdTurbulenceModels.py	NEW	Central data dictionary for supported turbulence models, coefficients, and OpenFOAM keywords
CfdOF/Solve/CfdPhysicsSelection.py	MODIFY	Added `TurbulenceModel`, `TurbulenceModelCoeffs`, `PrintCoeffs` properties
Gui/TaskPanelPhysics.ui	MODIFY	Added turbulence-model dropdown, coefficient group box, print-coefficients checkbox; removed legacy widgets
CfdOF/Solve/TaskPanelCfdPhysicsSelection.py	MODIFY	Controller logic for dynamic model selection, coefficient loading, and signal/slot wiring
CfdOF/Solve/CfdCaseWriterFoam.py	MODIFY	Reads model and coefficients from the physics object; generates *turbulenceProperties* dynamically; merge-logic bug fixes
Data/defaults/ templates	MODIFY	Updated to use dynamic placeholders instead of static values
runCFD.sh	NEW	Automation script for OpenFOAM v2306 structural compatibility fixes and end-to-end solver execution

9.2 Sample Generated *turbulenceProperties* File

```
/*-------------------*- C++ -*-------------------*\
  Generated by: Extended CfdOF Turbulence Configuration System
\*-------------------------------------------------*/
FoamFile
{
    version     2.0;
    format      ascii;
    class       dictionary;
    object      turbulenceProperties;
}
// * * * * * * * * * * * * * * * * * * * * * * * * * * * * * //

simulationType RAS;

RAS
{
    RASModel        kOmegaSST;
    turbulence      on;
    printCoeffs     on;
```

```
    kOmegaSSTCoeffs
    {
        alphaK1     0.85;
        beta1       0.075;
    }
}
```

9.3 *runCFD.sh* Fix Snippets

```
# Fix 1: Supply a pressure reference for pimpleFoam
cat >> system/fvSolution << 'EOF'
SIMPLE
{
    pRefCell 0;
    pRefValue 0;
}
EOF

# Fix 2: Provide momentumTransport for OpenFOAM v2306
cp constant/turbulenceProperties constant/momentumTransport
sed -i 's/object turbulenceProperties;/object momentumTransport;/' \
    constant/momentumTransport

# Fix 3: Remove stale invalid include from controlDict
sed -i '/turbulenceLib/d' system/controlDict
```