Creating Computational Fluid Dynamics Simulations with OpenFOAM for Machine Learning Tasks

Anish Kambhampati
Department of Computer Science
Purdue University
West Lafayette, IN 47907
akambha@purdue.edu

Abstract

The rising interest in applying machine learning (ML) techniques to computational fluid dynamics (CFD) problems has created a need for large, high-quality datasets. However, the quantity and quality of existing datasets is inadequate and generating such datasets remains challenging due to the complexity and computational expense of CFD simulations. In this work, we present an end-to-end pipeline for creating CFD datasets suitable for ML training. Our method leverages OpenFOAM and supercomputing clusters to run hundreds of customized simulations across varied flow conditions, ParaView to automate post-processing and extract timestep data at mesh points, and a custom Python script to transform the data into TensorFlow records. We detail each stage of the pipeline, from mesh selection and case setup to trajectory automation and TensorFlow record generation. Two examples are provided – one subsonic and one supersonic – to demonstrate the pipeline's flexibility. Our framework enables researchers to efficiently generate diverse CFD datasets, facilitating the training of ML models on realistic, high-fidelity fluid dynamics data.

1 Introduction

Computational fluid dynamics (CFD) is a field of study focused on the numerical analysis and solution of problems involving fluid flows. By solving the governing equations of fluid motion, CFD enables researchers and engineers to simulate and predict the behavior of liquids and gases in complex environments [Anderson Jr. et al., 2009]. These simulations are invaluable across a wide range of industries, including aerospace, automotive, energy, and biomedical engineering, allowing for the analysis of systems that would be too expensive, dangerous, or impractical to study experimentally. Modern CFD approaches discretize the fluid domain into a mesh and iteratively solve the governing equations across this mesh to approximate fluid behavior over time. Advances in computational power and numerical methods have continually expanded the scope of CFD, enabling higher-fidelity simulations of turbulent flows, heat transfer, and multiphase phenomena [Ferziger et al., 2020, chap. 2].

While CFD simulations have become much more powerful, their computationally intensive nature along with more demanding requirements means that the most complex simulations still take a significant time to run. One idea that aims to reduce the time required is to leverage machine learning (ML) models (especially deep learning models) to approximate these simulations [Kochkov et al., 2021, Thuerey et al., 2020]. Of course, training these models requires a vast amount of data, and existing works have attempted to create some such datasets. As an example, DrivAerNet++ and WindsorML provide datasets pertaining to automotive aerodynamics [Elrefaie et al., 2024, Ashton et al., 2024]. However, these are very specific datasets and do not provide an easy way for others to

Preprint. Under review.

generate their own datasets. An end-to-end pipeline for creating CFD datasets for machine learning would make it much easier for researchers to train novel models.

In this work, we attempt to solve that issue. We present a complete pipeline for generating large-scale CFD datasets suitable for ML model training. We show how to import meshes into OpenFOAM, initialize the necessary boundary conditions, run hundreds of simulations with seeded values, post-process the results with ParaView, and finally generate TensorFlow records which can be used for downstream tasks. This process enables rapid generation of diverse CFD datasets across a range of flow conditions, enabling the creation of high-quality training data for CFD-based machine learning models.

2 Background

2.1 OpenFOAM

Over the past several decades, several multi-purpose CFD solvers have been developed. Among the most widely used is OpenFOAM, an open-source, C++-based, CFD solver with support for mesh handling, turbulence, heat transfer, parallelization, code extensibility, and more [Jasak, 2009]. In particular, OpenFOAM's open-source nature has made it a particularly compelling option due to the ability to extend the software [Jacobsen et al., 2012]. Even without adding to the codebase, the ability to edit text configuration files to change nearly everything about a simulation gives the software strong flexibility. It does, however, come with a rather steep learning curve.

OpenFOAM stores all the necessary files to run a simulation in a case directory. A typical case directory contains three main folders: system, constant, and one or more time directories (such as 0 for initial conditions). The system directory holds files that control the simulation setup, including mesh settings (blockMeshDict), solver parameters (fvSchemes, fvSolution), and runtime settings (controlDict) Medina et al. [2015]. The constant directory contains physical properties, turbulence models, and mesh data, including the polyMesh subdirectory that stores the mesh information. Time directories like 0 (and later ones such as 0.5, 1, etc.) hold field data for variables like velocity (U), pressure (p), and temperature (T) at different simulation times. This organized structure allows OpenFOAM to be highly modular and extendable but the complexity involved in knowing what to change creates the steep learning curve.

2.2 ParaView

ParaView is an open-source multi-platform data analysis and visualization tool widely used in scientific computing. It supports a variety of data formats, including native OpenFOAM outputs, and provides a rich set of tools for post-processing, such as slicing, probing, and plotting field variables over time [Ahrens et al., 2005]. Importantly, ParaView offers a Python scripting interface (pvpython) which is vital in automating post-processing tasks necessary to create large-scale simulations and datasets.

3 Our Pipeline

3.1 Setting Up the Case

Before we start running multiple simulations, it is important to make sure that we have one set of parameters working properly.

3.1.1 Mesh Generation

Creating a mesh is the most important part of the entire process of creating training dataset as it defines the environment in which the simulation will run. The specifics of how to create a mesh are outside the scope of this paper. It is important, however, to ultimately create a mesh file that can be easily imported into OpenFOAM. The software natively supports meshes from a variety of software, including Fluent, STAR-CD, and Gambit [OpenFOAM Foundation]. There have also been efforts from third parties to extend support to other formats like CGNS [Scheufler, 2023]. Once a mesh has

been selected and imported into OpenFOAM, it is customary to run the `checkMesh` command. This instructs OpenFOAM to examine the mesh and look for problematic spots [Medina et al., 2015].

Before importing the mesh into OpenFOAM, the case directory structure must be created. While it is possible to create each file manually, it is far easier (and more common) to copy a tutorial and modify the files as needed. Choosing the right case to copy requires knowing which solver to use, of which OpenFOAM has plenty. While guiding the process of solver selection is beyond the scope of this paper, some factors to consider are: whether the simulation is steady-state or transient, whether the simulation should have compressible or incompressible flow, and the speed at which the fluid will flow. (Certain solvers, for example, are better suited to supersonic and/or hypersonic simulations.) The pipeline we describe will work regardless of which one is chosen. Whatever solver is chosen, it is important to make sure that the `writeFormat` key in system/controlDict is set to `ascii` for this stage. Otherwise, the TensorFlow creation script will be unable to read in the faces and points properly.

3.1.2 Running OpenFOAM

After a tutorial is copied and the mesh imported, the boundary conditions (BCs) will need to be adjusted to match those in the imported mesh. This pertains most strongly to files in the 0/ directory but it may be necessary to change some of the constant/polyMesh files if the import was not perfect. Once the case files have been adjusted, it should be possible to run the simulation. Afterwards, creating an empty `para.foam` file in the case directory and opening it in ParaView will show the results of the simulation.

Once ParaView has been opened, we change the visible mesh regions to view the patch regions instead of the internalMesh. We then activate its SpreadSheetView to view the CSV data at a particular timestep, as shown in Figure 1. This lets us download the BC CSV at a particular timestep. While this may seem strange, there is a reason for doing this – it allows us to construct a mapping between every point and a BC. OpenFOAM does not compute values at nodes, instead computing them at cell centers. While this may be fine for some researchers, forcing this restriction would make it harder to compare OpenFOAM results against those from other solvers. Additionally, ParaView's automated timestep CSV generation does not store BC information, so this step is required to perform the necessary association. As we only do this once per case, it does not hinder the automation of running hundreds of cases.

(a) Changing ParaView mesh regions. (b) Selecting the proper ParaView View.

Figure 1: ParaView setup process.

3.2 Automating Simulations

Having successfully run a single case and created a point-BC mapping CSV, the next step is to automate the running of multiple cases. To do so, we make use of sed. Sed is a text-based stream

editor present in many UNIX systems [Dougherty and Robbins, 1997]. In our case, we use sed to place dummy text where we want to set BC values generated dynamically at runtime. Listing 1 shows an example of this for inlet velocity.

Listing 1: OpenFOAM boundary condition for inlet velocity.

```
inlet
{
    type            fixedValue;
    value           uniform (INITIAL_VELOCITY 0 0);
}
```

Once all dynamically-generated values have been manually replaced with dummy text, we can move to the actual data generation. Our script for running all the individual cases is designed for usage with SLURM, a resource manager widely used in supercomputing clusters. Algorithm 1 shows it does at a high level.

Algorithm 1: OpenFOAM Case Generation

Input: Start index s, End index e
Output: Compressed simulation archives for each case
Load modules: GCC, OpenMPI, OpenFOAM, etc.;
Fix random seed for reproducibility;
Create arrays for each dynamically generated value;
for *each index* i *from* s *to* e **do**
    Create a new case directory by copying the base case;
    Replace placeholders in boundary files with those in arrays using `sed`;
    Convert mesh to OpenFOAM format;
    Run `checkMesh` to verify mesh quality;
    Solve the flow using the chosen solver;
    Compress simulation outputs into a `.tar.gz` archive;
    Delete the temporary case directory to save disk space;

We note that OpenFOAM comes with OpenMPI support. This makes running simulations much faster. Table 1 gives an example from a sample case we ran. While the speedup may look attractive, there is a downside – whenever using OpenMPI, running `reconstructPar` afterwards is necessary to generate the final timestep directories. While the time this takes is negligable if the number of timesteps is small, it becomes prohibitively costly if the number of timesteps is large. Whether using multiple cores will save time depends on the number of timesteps being computing and the number of cores being used. The optimal core count will vary by case.

Table 1: Effect of increasing core count on solver total time.

Cores	Total Time (mm:ss)	Speedup
1	06:39	1.00×
2	03:51	1.73×
4	02:23	2.79×
8	01:50	3.62×

One other note is that when compressing timestep directories, we include the directory at time 0. This is not the default with OpenFOAM but is necessary as the values at timestep 0 vary based on case. Additionally, we need an accurate timestep 0 to open case in ParaView, which we need to create the timestep point CSV files. Finally, we recommend running around 20% more cases than required as it is likely that some cases will break in either this phase or the next.

3.3 Post-Processing

3.3.1 Generating Timestep CSVs

As stated earlier in the paper, OpenFOAM does not store timestep data at nodes, instead storing data at cell centers. To get the timestep data at the nodes, we use ParaView scripting which performs its own interpolation to get the proper data. The ParaView script, which uses `pvpython`, is shown in Listing 2. The most important line is that which defines the point arrays as it specifies what data gets written to the CSV files.

Listing 2: ParaView script for timestep CSV generation.

```
from paraview.simple import *
parafoam = OpenFOAMReader(registrationName='para.foam',
    FileName='para.foam')
SaveData('../$snapshot_dir/timestep.csv', proxy=parafoam,
    WriteTimeSteps=1,
    WriteTimeStepsSeparately=1,
    Filenamesuffix='_%05d',
    PointDataArrays=['T', 'U', 'p', 'rho'],
    CellDataArrays=['T', 'U', 'p', 'rho'],
    FieldDataArrays=['CasePath'],
    Precision=6,
    AddTime=1)
```

This script is just a smaller part of the larger SLURM script that reads in all the raw data tar files generated earlier and creates CSV tar files. The associated SLURM script is shown in Algorithm 2.

Algorithm 2: OpenFOAM Post-Processing and CSV Dataset Generation

Input: Start index s, End index e
Output: Compressed CSV archive for each simulation case
Load modules: GCC, OpenMPI, OpenFOAM, ParaView;
Set random seed to that used in Algorithm 1;
Create arrays for each dynamically generated value;
for *each index* i *from* s *to* e **do**
    Create a new case directory by copying the base case;
    Copy and extract the corresponding simulation results tar file;
    Run the ParaView script to export timestep data into CSV files;
    Compress all generated CSV files into a single tar.gz archive;
    Remove temporary folders and the case directory;

3.3.2 Creating the TensorFlow Record

The final step in our pipeline is to use the generated timestep CSV files to create a TensorFlow record which can be used for training ML models. What features make it into the record will vary by use case. As a general rule, we recommend including at least timestep interval, points, faces, velocity, temperature, and pressure. This stage does not have a SLURM script associated at the whole process involves a single Python file. Algorithm 3 gives a summary of the process.

Once this step is done, we have the TensorFlow record and the pipeline is complete. This file can then be used for all sorts of ML training tasks. The full pipeline is summarized in Figure 2.

4 Examples

4.1 Subsonic Channel Mesh

Our first example involves simulating subsonic, laminar, compressible airflow through an unstructured 2D mesh (called channel), shown in Figure 3. Flow comes in from the left and exits on the right. We use the rhoPimpleFoam solver to solve for the flow in this domain.

Algorithm 3: TensorFlow Record Creation from OpenFOAM Simulation Data

Input: OpenFOAM mesh files, reference CSV, trajectory snapshot archives
Output: TFRecord file containing mesh and dynamic field data
Load OpenFOAM mesh points and triangular faces;
Load reference CSV mapping points to boundary conditions (BCs);
Filter points to retain only those lying on the $z = 0$ plane;
Assign a node type (wall, inlet, outlet, etc.) to each mesh point based on the reference CSV;
Initialize static fields: node types, mesh connectivity (cells), and point coordinates;
Define metadata and save to `meta.json`;
for *each trajectory index* **do**
    if *trajectory index is not excluded* **then**
        Extract the compressed snapshot archive;
        Initialize arrays for velocity, pressure, temperature, and density;
        for *each timestep CSV* **do**
            Read and filter point data;
            Match CSV points to mesh points using nearest-neighbor search;
            Populate arrays with velocity, pressure, temperature, and density values;
        Serialize the fields and write to the TFRecord;
        Delete temporary extracted snapshot folder;
Finalize and close the TFRecord file;

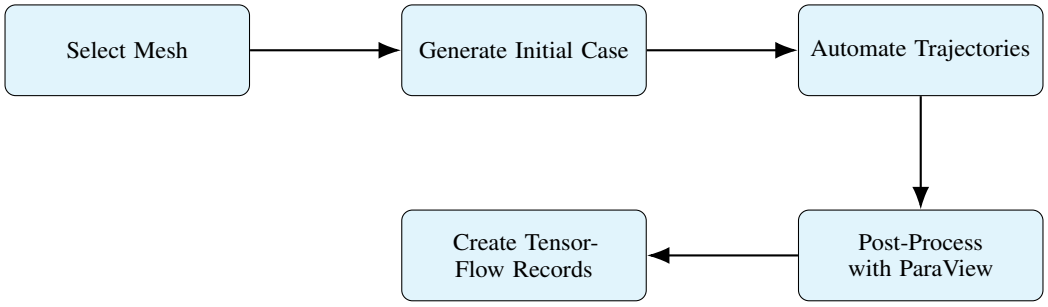


Figure 2: A summary of the end-to-end pipeline.

We have boundary conditions for the inlet, outlet, top, bottom, body, z0, and z1. The final two are not too important – they are only needed as OpenFOAM requires a 3D mesh even if the flow is only being simulated through a 2D domain. We set BCs for pressure, temperature, and velocity. They are as follows:

- Pressure at the inlet and internal field is fixed to $6 \times 10^4\,\mathrm{Pa}$. Pressure at the outlet, top, and bottom is set as wave transmissible. This prevents resonance from occurring within the mesh. Pressure at the body (the diamond-shaped object near the left side), z0, and z1 is set to zero gradient.
- Temperature at the inlet, top, bottom, and body is fixed as $278\,\mathrm{K}$. Temperature at the outlet, z0, and z1 is set to zero gradient.

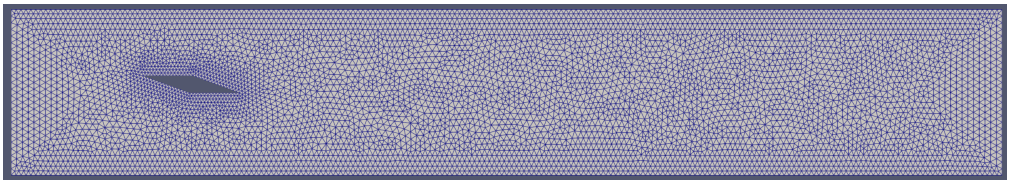


Figure 3: Unstructured channel mesh, shown in ParaView.

- Velocity at the inlet and internal field (the starting velocity across the whole mesh) is fixed at $10\,\mathrm{m/s}$ in the positive x direction. The outlet is set to a special BC (pressureInletOutletVelocity) which is essentially a zero gradient BC in the direction of the flow with some caveats for backflow. The top, bottom, and body are initialized to a no slip condition, and z0 and z1 are initialized to a slip condition.

Setting these, along with a timestep of 0.001 seconds, let us create our first simulation. The velocity at the final timestep of 1 second is shown in Figure 4. After confirming that the simulation works, we grab a reference csv as described earlier.

Figure 4: Velocity magnitude at $t = 1$ second, channel simulation.

The next step in the pipeline is to automate the running multiple cases with varied parameters. We chose to vary velocity from 10-150 $\mathrm{m/s}$ and temperature from 220-300 K. As we wanted 200 runs (also called trajectories), we ran 230 to deal with any runs that may have failed. With the high maximum velocity, we had to scale our timestep interval down to 5×10^{-5} seconds. We ran simulations for 1.2 seconds and wrote to disk every 1×10^{-4} seconds, for a total of 12,000 records per trajectory. After running, each compressed trajectories uses around 18GB of space. We then use the ParaView script to convert each run to a compressed archive of timestep CSVs. The final step was to then convert these archives into a TensorFlow record. Of the 230 simulations we ran, 24 of them ran into some issue, typically with OpenFOAM crashing or the ParaView script unable to read a particular timestep. This was fine, however, as we had over 200 usable trajectories. Our TensorFlow record stores the mesh points, cells, x and y velocity at each timestep, pressure at each timestep, temperature at each timestep, density at each timestep, and the write interval 1×10^{-4}. Once complete, our record had a size of approximately 280GB. It is successfully being used to train some of our ML models.

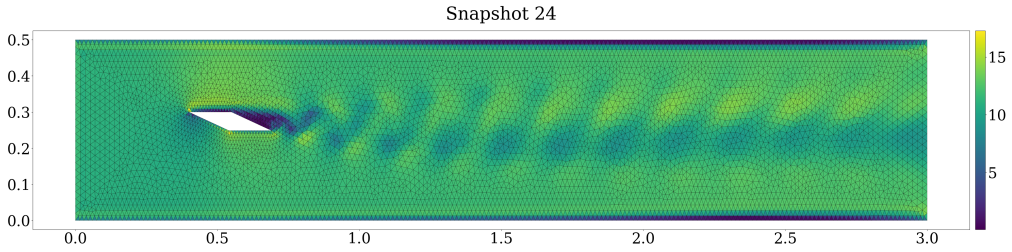
As a sanity check, we also wrote a script that reads in the values provided by OpenFOAM and creates visualizations using matplotlib. We wrote scripts that perform the visualization based on both cell data (which reads ASCII OpenFOAM timestep data directly) and point data (which reads the ParaView-generated timestep data). These scripts (in particular the point data one) can also be used to visualize the output of ML models. The processes used in the points script were used heavily in creating the TensorFlow records. Figure 5 shows the resulting views at the final snapshot of the initial simulation, the same velocity magnitude data shown in Figure 4. The similarity between these two figures indicates that our TensorFlow record accurately represents the OpenFOAM data.

4.2 Supersonic Diamond Mesh

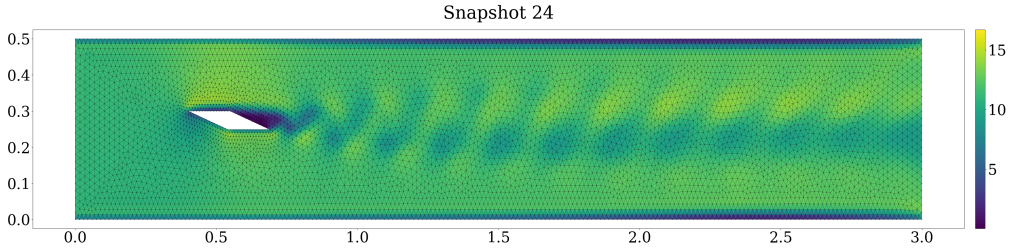
Our second example involves simulating supersonic, laminar, compressible airflow through an unstructured 2D mesh (called diamond), shown in Figure 6. We use the rhoCentralFoam solver to solve for the flow in this domain. Compared to rhoPimpleFoam, rhoCentralFoam is better at solving for supersonic flow.

We again have boundary conditions at the inlet, outlet, top, bottom, body, z0, and z1 for pressure, temperature, and velocity. These are as follows:

- Pressure at the inlet, top, and bottom is set to wave transmissible. Pressure at the other boundaries is set to zero gradient. The internal field is set to a uniform $6 \times 10^4\,\mathrm{Pa}$.
- Temperature at the inlet, internal field, and body is fixed at 300K. Temperature at the other boundaries is set to zero gradient.
- Velocity at the inlet and internal field is fixed at $660\,\mathrm{m/s}$ in the positive x direction. The top and bottom are set to slip, and the body is set to no slip. Finally, the outlet, z0, and z1 are set to zero .gradient.



(a) Cell-based visualization.



(b) Point-based visualization.

Figure 5: Matplotlib views for channel simulation, velocity magnitude.

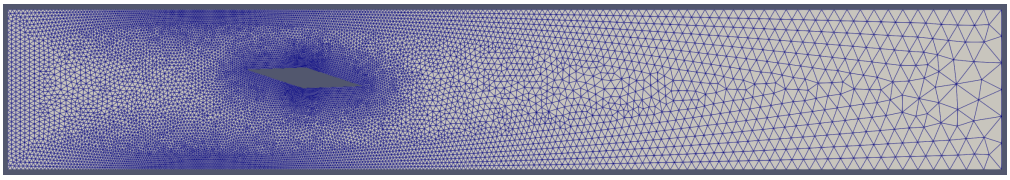


Figure 6: Unstructured diamond mesh, shown in ParaView.

Due to the supersonic nature of the flow, we set the timestep to 1×10^{-8} seconds. We also set a maximum CFL value of 0.1, which limits the maximum timestep increment for stability purposes [Mahgerefteh et al., 2009]. We write every 1×10^{-5} seconds and run for 1×10^{-3} seconds for a total of 100 snapshots of data. Figure 7 shows the resulting velocity magnitude at the final timestep, along with the points script view.

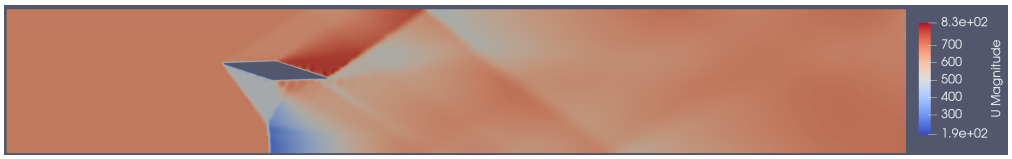
We did not execute the rest of the pipeline for this example. It is however, fairly simple to go through the rest of it and get the desired TensorFlow record needed for ML model trainings.

5 Future Directions

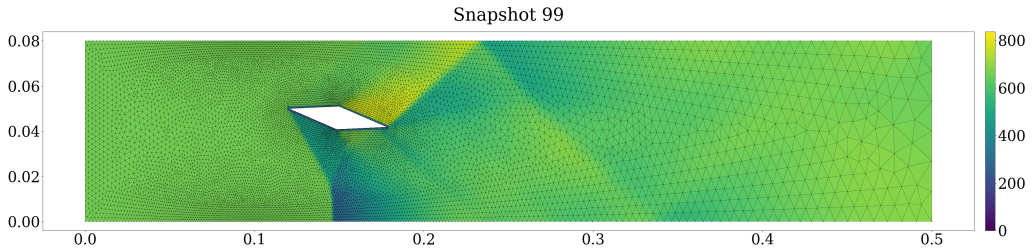
While the pipeline is robust and flexible, there are opportunities for improvement. One of the main areas to improve on is the setup of the initial case. While OpenFOAM is an excellent solver, its steep learning curve makes creating that first working case a difficult task. The BCs and solvers have to be set up perfectly and it is very difficult to pinpoint the source of an error. Tools that can help with setting up a working case given a particular mesh would be an invaluable resource, not only for our work but also for the CFD field as a whole.

Another area to explore would be how to speed up model training from the TensorFlow records. While not the primary focus of this paper, we did conduct some preliminary training tasks on the subsonic channel record. What we found was that training a model on the dataset would take weeks. This is an active area of research, and there is no doubt advancements here would enable the creation of larger and more accurate datasets.

One final direction future works could take would be to create this pipeline using other CFD solvers, such as Fluent and SU2. It would be invaluable to not only compare the results of these solvers



(a) ParaView visualization.



(b) Matplotlib, point-based visualization.

Figure 7: Velocity magnitude at $t = 1 \times 10^{-3}$ seconds, diamond mesh.

against each other, it would be even more valuable to see if machine learning models trained on the output of more than one would increase accuracy in predictions.

6 Conclusion

In this work, we explored a novel pipeline to generate CFD datasets for machine learning applications. We showed how to select a mesh, generate an initial working case, automate that case to run hundreds of trajectories, post-process with ParaView, and finally create the TensorFlow records needed for training. We give examples of our pipeline in action and show how it can be used to generate concrete datasets. Our hope is that the pipeline we created can be used by researchers in CFD and ML to generate their own datasets.

Acknowledgments and Disclosure of Funding

We would like to thank Matteo Ruggeri and Gabriel Buginga, PhD students at Purdue University, for their invaluable support. We also thank Dr. Bruno Ribeiro, an Associate Professor at Purdue University, and the Purdue MI.N.D.S lab for their mentorship and guidance.

References

J. Ahrens, Berk Geveci, and Charles Law. Paraview: An end-user tool for large data visualization. *Visualization Handbook*, page 18, 01 2005.

John D. Anderson Jr., Joris Degroote, Gérard Degrez, Erik Dick, Roger Grundmann, and Jan Vierendeels. *Computational Fluid Dynamics: An Introduction*. Springer Berlin, Heidelberg, 3 edition, 2009. ISBN 978-3-540-85055-7. doi: 10.1007/978-3-540-85056-4. URL `https://doi.org/10.1007/978-3-540-85056-4`.

Neil Ashton, Jordan B. Angel, Aditya S. Ghate, Gaetan K. W. Kenway, Man Long Wong, Cetin Kiris, Astrid Walle, Danielle C. Maddix, and Gary Page. Windsorml: High-fidelity computational fluid dynamics dataset for automotive aerodynamics. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 37823–37835. Curran Associates, Inc., 2024. URL `https://proceedings.neurips.cc/paper_files/paper/2024/file/42a59a5f35b1b3c3fd648397c88a7164-Paper-Datasets_and_Benchmarks_Track.pdf`.

Dale Dougherty and Arnold Robbins. *sed & awk: UNIX Power Tools*. " O'Reilly Media, Inc.", 1997.

Mohamed Elrefaie, Florin Morar, Angela Dai, and Faez Ahmed. Drivaernet++: A large-scale multimodal car dataset with computational fluid dynamics simulations and deep learning benchmarks. In A. Globerson, L. Mackey, D. Belgrave, A. Fan, U. Paquet, J. Tomczak, and C. Zhang, editors, *Advances in Neural Information Processing Systems*, volume 37, pages 499–536. Curran Associates, Inc., 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/013cf29a9e68e4411d0593040a8a1eb3-Paper-Datasets_and_Benchmarks_Track.pdf.

Joel H. Ferziger, Milovan Perić, and Robert L. Street. *Computational Methods for Fluid Dynamics*. Springer Cham, 4 edition, 2020. ISBN 978-3-319-99691-2. doi: 10.1007/978-3-319-99693-6. URL https://doi.org/10.1007/978-3-319-99693-6.

Niels G. Jacobsen, David R. Fuhrman, and Jørgen Fredsøe. A wave generation toolbox for the open-source cfd library: Openfoam®. *International Journal for Numerical Methods in Fluids*, 70(9):1073–1088, 2012. doi: https://doi.org/10.1002/fld.2726. URL https://onlinelibrary.wiley.com/doi/abs/10.1002/fld.2726.

Hrvoje Jasak. Openfoam: Open source cfd in research and industry. *International Journal of Naval Architecture and Ocean Engineering*, 1(2):89–94, 2009. ISSN 2092-6782. doi: https://doi.org/10.2478/IJNAOE-2013-0011. URL https://www.sciencedirect.com/science/article/pii/S2092678216303879.

Dmitrii Kochkov, Jamie A. Smith, Ayya Alieva, Qing Wang, Michael P. Brenner, and Stephan Hoyer. Machine learning–accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021. doi: 10.1073/pnas.2101784118. URL https://www.pnas.org/doi/abs/10.1073/pnas.2101784118.

Haroun Mahgerefteh, Yuri Rykov, and Garfield Denton. Courant, friedrichs and lewy (cfl) impact on numerical convergence of highly transient flows. *Chemical Engineering Science*, 64(23):4969–4975, 2009. ISSN 0009-2509. doi: https://doi.org/10.1016/j.ces.2009.08.002. URL https://www.sciencedirect.com/science/article/pii/S0009250909005351.

Humberto Medina, Abhinivesh Beechook, Jonathan Saul, Sophie Porter, Svetlana Aleksandrova, and Steve Benjamin. Open source computational fluid dynamics using openfoam. In *Royal Aeronautical Society, General Aviation Conference, London*, 2015.

OpenFOAM Foundation. Mesh conversion. URL https://www.openfoam.com/documentation/user-guide/4-mesh-generation-and-conversion/4.5-mesh-conversion. Accessed: 2025-04-26.

Henning Scheufler. cgnstofromfoam, 2023. URL https://github.com/HenningScheufler/cgnsToFromFoam/tree/of2306. Accessed: 2025-04-26.

Nils Thuerey, Konstantin Weißenow, Lukas Prantl, and Xiangyu Hu. Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows. *AIAA Journal*, 58(1):25–36, 2020. doi: 10.2514/1.J058291. URL https://doi.org/10.2514/1.J058291.