UNIVERSITY OF CALIFORNIA SAN DIEGO

A Framework for Generalized Steady State Neural Fluid Simulations

A Thesis submitted in partial satisfaction of the requirements
for the degree Master of Science

in

Computer Science

by

Steve Guerin

Committee in charge:

Professor Hao Su, Chair
Professor Gary Cottrell
Professor David Kriegman

2023

The Thesis of Steve Guerin is approved, and it is acceptable in quality and form for publication on microfilm and electronically.

University of California San Diego

2023

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT OF THE THESIS

A Framework for Generalized Steady State Neural Fluid Simulations

by

Steve Guerin

Master of Science in Computer Science

University of California San Diego, 2023

Professor Hao Su, Chair

Steady State fluid simulations are a critical piece of the mechanical engineering design loop but serve as a bottleneck due to the engineering overhead and the high computational load. In recent years, there has been an increase in the research activity in the field of neural fluid simulations, however, the current works have limited scope with respect to the fluid domains and flow regimes, restricting their generalizability and their potential industrial impact. This thesis seeks to introduce a foundational framework for generic 3D steady state neural fluid simulations, with the goal of simplifying and expediting the fluid simulation

process for engineering applications. To that end, this project introduces 3 key contributions: A python package for automated generation and post-processing of fluid simulations with the Ansys simulation suite, a benchmark dataset of over 3000 fluid channel geometries and roughly 3400 quality fluid simulations that meet engineering standards for accuracy, and a series of foundational experiments exploring steady state neural fluid simulations for internal flows - the first of its kind as far as the author is aware. Experimental results demonstrate that geometric deep learning models have the capacity to be used as a proxy for traditional fluid simulations, but also indicate that further research is required to continue to develop the dataset and architectures for this deep learning application.

Chapter 1

Introduction

This section will cover the industrial and theoretical background as well as the motivation for pursuing this project. Additionally, it covers the current state of the art in the space of neural fluid simulations, as well as the goals and core contributions of this project.

1.1 Background

1.1.1 Fluid Theory for Manifolds

The fundamental equation of fluid mechanics is the Navier-Stokes equation:

$$\frac{\delta(\rho u)}{\delta t} + \nabla \cdot (\rho\, u \otimes u) = -\nabla P + \nabla \cdot \tau + \rho g$$

Simply put, the equation relates the conservation of mass to the conservation of momentum in an infinitesimally small continuum. This equation is often simplified and rearranged to analyze larger systems. A common simplification is the Bernoulli Equation, which describes the steady-state behavior of an ideal incompressible fluid flowing through a pipe:

$$P_1 + \frac{1}{2}\rho v_1^2 + \rho g h_1 = P_2 + \frac{1}{2}\rho v_2^2 + \rho g h_2$$

This equation assumes that there are no friction losses throughout the pipe, and describes an energy balance between inlet and outlet points (Figure 1.1). The ***P*** term describes the energy contribution from Pressure, ***½pV²*** describes the contribution from Kinetic Energy, and ***pgh*** is the contribution from Potential Energy. It should be noted that the term ‘pipe’ will be used to

describe any internal flow channel moving forward, regardless of the consistency of the cross-sectional area or the number of inlet or outlet paths. 'Internal flow' here is defined as fluid flow that is completely surrounded by solid surfaces, and the fluid domain itself is described by the bounds of these solid surfaces.

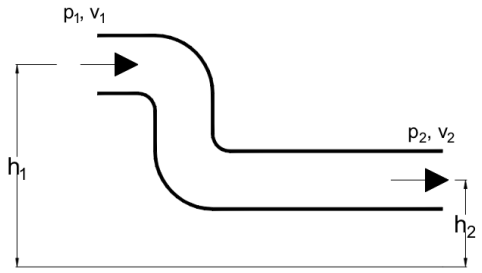


Figure 1.1 : Pipe highlighting key variables in Bernoulli Equation at inlet and outlet

In this form, the fluid is assumed to be incompressible and therefore the fluid density is constant throughout the entire domain. As such, the velocity of the fluid will be determined by the continuity equation, which states that the product of the pipe cross-sectional area and the bulk fluid velocity are constant at all locations along the pipe:

$$A_1 v_1 = A_2 v_2$$

Simply put, any changes to the bulk fluid velocity along the length of the pipe are directly related to changes in the pipe radius.

For practical applications involving pipe flows, an additional modification is made to the Bernoulli Equation. A term ***F*** is added to the left side of the equation to account for the effects of friction, which cannot be ignored. The rewritten equation takes the form:

$$P_1 + \frac{1}{2}\rho v_1^2 + \rho g h_1 - F = P_2 + \frac{1}{2}\rho v_2^2 + \rho g h_2$$

To understand the effects of friction in a pipe, envision a level pipe with constant cross-sectional area. It can be assumed that the height and the velocity will not change, which allows for the removal of the potential (***pgh)*** and kinetic (***½pV²***) energy terms from both sides of the equation. Rearranging the pressure terms yields the following equation:

$$P_2 - P_1 = F$$

This highlights a key phenomenon of pipe flow - the energy loss due to friction manifests itself as pressure loss along the length of the pipe.

The friction losses occur due to the fluid-solid interactions along the walls of the pipe. The frictional forces can be visualized when examining the axial fluid velocity profile in a pipe, where the velocity is 0 at the wall and reaches a maximum at the centerline of the pipe (Figure 1.2). The rate of energy loss due to friction can be calculated from two values: the roughness of the internal walls of the pipe, and a non-dimensional term called the Reynolds number.

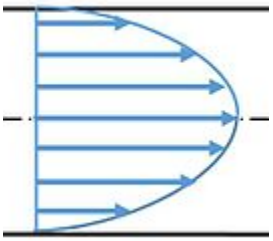


Figure 1.2: Axial fluid velocity profile within a pipe

The Reynolds number [32] is defined as:

$$Re = \frac{\rho VD}{v}$$

Where ρ, V, and ν represent the fluid density, velocity, and viscosity, respectively, and D represents the pipe diameter. Simply put, the Reynolds number represents the ratio of inertial forces to viscous damping forces in a given fluid. It is most commonly known for expressing whether a fluid flow is in the laminar or turbulent regime (Figure 1.3).

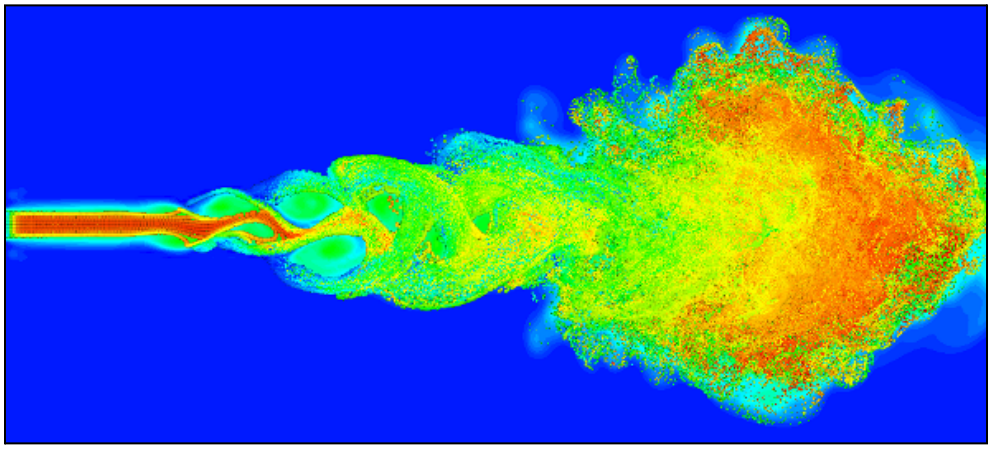


Figure 1.3: Fluid flow plume transitioning from laminar to turbulent flow regime [78]

For straight pipes, the energy losses due to friction can be determined via the Darcy-Weisbach Equation [33]:

$$h_f = f_D \frac{L}{D} \frac{V^2}{2g}$$

Where the loss due to friction h_f is equal to the friction factor f_D multiplied by the length of the pipe L, the square of the velocity V, divided by the pipe hydraulic diameter D and 2 times the acceleration due to gravity g. The loss due to friction here is defined as head loss h_f [33], which describes pressure loss in terms of potential energy. The two are related via the equation:

$$\Delta P = \rho g h_f$$

where $\boldsymbol{h_f}$ is the head loss, with units of length, $\boldsymbol{g}$ is the acceleration due to gravity, and ρ is density. Modifying the Darcy-Weisbach by substituting the head loss term yields the following equation:

$$\frac{\Delta P}{L} = f_D \frac{\rho V^2}{2D}$$

In this form, it can be seen that the pressure loss per unit length is proportional to the friction factor, the fluid density, and the square of the velocity, and inversely proportional to the pipe hydraulic diameter. As mentioned previously, the friction factor $\boldsymbol{f_D}$ has been determined experimentally to be a function of the fluid Reynolds number and relative wall roughness. This data is visualized in the Moody Diagram [34].

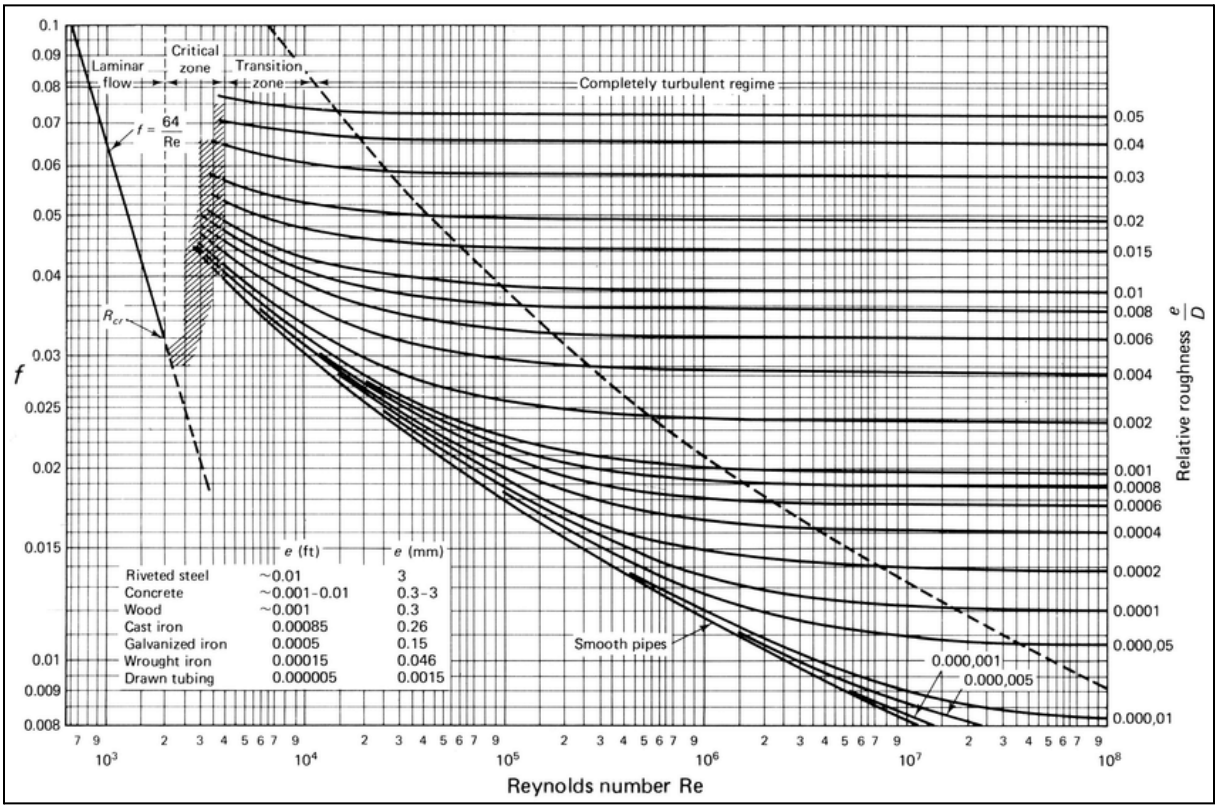


Figure 1.4: Moody Diagram [53]

Non-dimensional Analysis

Non-dimensional analysis is a mathematical method that re-parameterizes physical equations by grouping and substituting parameters in the equation in a manner that yields a dimensionless equation [33]. In the process, the equation is rescaled and simplified without loss of information.

A non-dimensionalization approach commonly used in fluid mechanics utilizes the Buckingham Pi Theorem, which states that if there is a physical equation involving $\boldsymbol{n}$ variables, and there are $\boldsymbol{k}$ fundamental dimensions involved in the equation, then that equation can be rewritten as a non-dimensional equation with $\boldsymbol{p = n - k}$ terms - where the new terms are comprised of the original terms. When looking at incompressible fluid flow without heat-transfer, there are 3 fundamental dimensions: mass, length, and time - therefore any equation can be reduced to a non-dimensional form with 3 fewer parameters, provided all three fundamental dimensions were present in the original equation [35]. The resulting non-dimensional terms also provide insight into how the relative scale of the original dimensional parameters affect the behavior of the system. The most famous non-dimensional parameter in fluid mechanics is the previously mentioned Reynolds number.

In the context of fluid mechanics, non-dimensional analysis has been used extensively in situations where the analytical solutions are not feasible and therefore experimentation is required. In the non-dimensional form, the number of free parameters is greatly reduced, requiring orders of magnitude fewer experiments to be conducted in order to properly characterize the system. Additionally, the non-dimensional parameters provide generalized scaling laws that allow for prediction of system behavior at scales that would be either infeasible or cost-prohibitive to prototype [36]. The Moody Diagram (Figure 1.4) demonstrates the utility

of non-dimensionalization, showing that the friction factor f_D can be determined as a function of the relative wall roughness ϵ and Reynolds number Re.

Need for Simulation Tools

The equations and experimental relationships detailed in Section 1.1.1 can be expected to provide accurate results when the system under analysis is consistent with assumptions made in deriving the equations. However, in many situations the system deviates from the assumptions, making it infeasible to use analytical methods. In such scenarios, numerical methods like finite element analysis provide a path to achieve accurate predictions of system behavior.

1.1.2 Finite Element Analysis and Computational Fluid Dynamics

Finite Element Analysis, originally developed as a computation tool for Structural Analysis [37], is widely used for modeling various physical phenomena including heat transfer and fluid mechanics (often referred to as computational fluid dynamics or CFD). This numerical method operates by dividing a physical domain into small, discrete elements, which are connected together to form a volumetric mesh. Each element of the mesh has a set of equations associated with it, and the equations are aggregated together to form a larger system of equations which is solved using numerical methods such as the Conjugate Gradient Iterative Solver [38].

There are four core stages of finite element simulations: physical domain definition, domain discretization, physics model definition, and numerical solver execution. The following sections will describe each of the stages through the lens of computational fluid dynamics.

Physical Domain Definition

The physical domain under analysis must first be defined in a digital format. Generally a CAD (computer aided design) model is developed using a parametric solid modeling software such as SolidWorks [39] or CATIA [40] and the resulting geometry is stored in a Boundary Representation Format [41]. Mesh based model definitions such as .STL [42] format are also viable inputs. The CAD file is then passed into a Meshing application in order to discretize the domain.

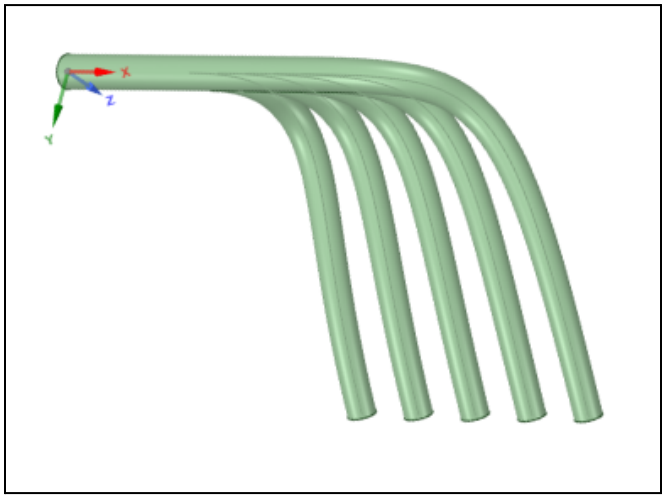


Figure 1.5: Example CAD file design in BREP format

Domain Discretization

The physical medium is divided into a series of discrete, interconnected polyhedral elements in a process that is commonly referred to as mesh generation. The resulting volumetric mesh will serve as an input to the simulation solver. There are numerous algorithms for discretizing the geometric domain into structured or unstructured meshes, including Grid-Based, Medial-Axis, Quad-Tree, and Advancing-Front methods [43]. Structured meshes are simpler and more computationally efficient to generate given their regular cell organization [44]. This

efficiency comes at the cost of flexibility, therefore unstructured meshes are often used to discretize complex geometries.

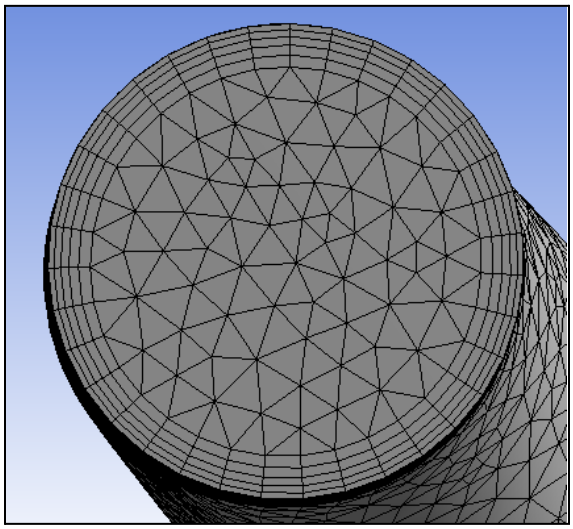


Figure 1.6: Volumetric mesh of a pipe with inflation layer near pipe walls

There are several configuration decisions that need to be made depending on the physics being modeled and the expected simulation behavior. A user must make selections on the element types (tetrahedral, hexahedral, polyhedral, pyramid, or wedge - see Figure 2.4), meshing method, average element size, and target quality [45]. Setting these parameters properly is essential to achieve a proper balance between simulation accuracy and simulation run-time. A mesh that is too coarse or is of poor quality will lead to inaccurate simulation results, while a mesh that is unnecessarily fine will result in wasted time, both during the mesh-generation and the simulation solver execution. Finding the right set of parameters is often achieved through an iterative loop of mesh-generation and simulation until a satisfactory mesh that balances resolution and accuracy has been achieved.

An important mesh quality metric for fluid simulations is the cell skew - Skewness is defined as the difference between the shape of the cell and the shape of an equilateral cell of

equivalent volume [45] (Figure 1.7). Highly skewed cells can introduce errors that prevent solution convergence (See Configuration and Execution), as information propagation across cell edges assumes that cell centroids are normal to edge joining the adjacent cells [46].

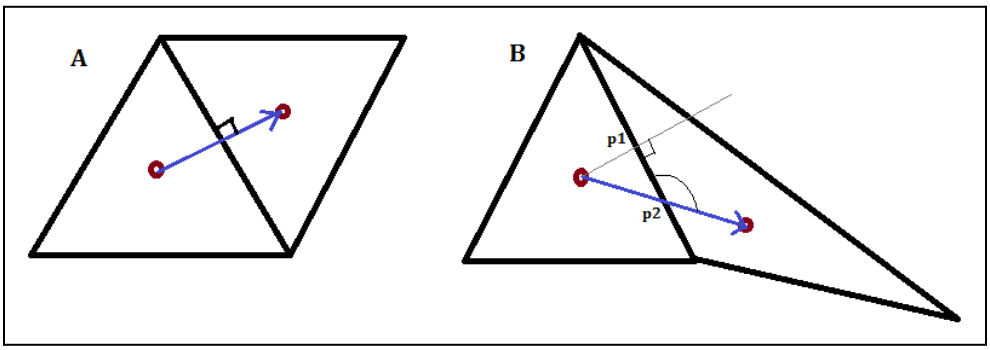


Figure 1.7: Comparison of ideal mesh cell (A) vs skewed cell (B) [54]

An additional mesh consideration for fluid simulations is the boundary layer - a region at the fluid-solid interface. This area requires additional focus due to the high velocity gradients found near the solid boundary [47]. A special type of mesh pattern called an inflation layer (Figure 1.8) is generally used in this region. Inflation layers consist of prismatic cells aligned with the gradient direction, and are often smaller than the cells in the bulk fluid domain.

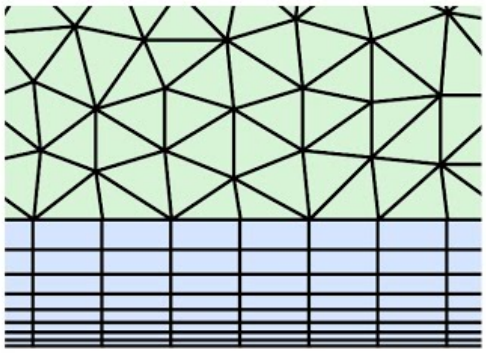


Figure 1.8: CFD mesh inflation layer

Setting the proper size for the inflation layers is critical for achieving simulation convergence. An adequate inflation layer thickness can be determined by calculating the

Dimensionless Wall Distance, also known as ***Y+*** [48], which describes the velocity behavior of the fluid at a given distance from the solid surface (Section 2.2.2). Sizing the inflation layer such that the first cell has a thickness ranging from 30 to 300 in the dimensionless ***Y+*** space allows for use of the wall functions [49], which will speed up the simulation without sacrificing accuracy. Determining the proper layer size prior to executing the simulation requires an understanding of the velocity profile along the length of the pipe. Using the continuity equation (Section 1.1.1), an educated guess can be made about the expected minimum and maximum velocities experienced along the pipe.

Physics Model Definition

A simulation package will offer multiple physics models for a given phenomena, where each model is specialized for a specific regime of behavior. It is up to the user to decide the correct model for a given simulation. This decision can largely be informed by intuition about the expected behavior of the system, but can also be accomplished through literature review. For example, Ansys Fluent offers three standard physics models for incompressible fluid flow based on the expected Reynolds number regime. The offerings include laminar, $\kappa\omega$-sst (shear stress transport) for turbulent flow, or transition-sst for flow that experiences both laminar and turbulent regimes along the length of the fluid domain. Incorrect model selection can prevent the simulation from converging to a solution or yield results that are not consistent with the physical phenomena [50].

Simulation Configuration and Execution

After selecting an adequate physics model, there are many parameters that can be configured for the model and the numerical solver. These parameters impact the stability and the

performance of the simulation. The standard turbulence model for incompressible fluid flow in Ansys Fluent provides 15+ parameters and constants that can be modified [22]. Generally the default values are sufficient, however, they may need to be tuned given the specific dynamics being modeled.

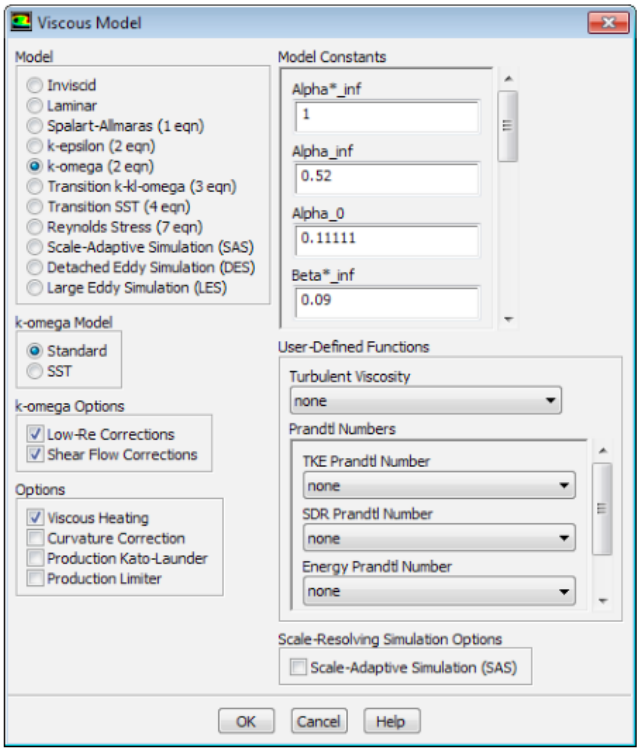


Figure 1.9: Ansys Fluent model setup dialog

Once the solver is configured, the user must provide fluid properties and boundary conditions to the solver. The fluid properties will vary depending on the specific type of simulation that is being conducted, however for steady-state incompressible flow, the fluid density and viscosity are the key parameters. Additionally, for internal channel flows, the relative wall roughness must be provided. Boundary conditions - the physical values at the edges of the domain being modeled - provide necessary constraints to the numerical solver. A common

boundary condition configuration for steady-state simulations is to define a velocity at the inlet of the fluid domain and a pressure value at the outlet of the domain.

Finally, the convergence criteria and the number of solver iterations are set. After each iteration of the solver, the conservation of mass and momentum are evaluated for the entire domain. Any violations of conservation equations, referred to as residuals, are tracked by the simulation software. The convergence criteria provides the numerical solver a target residual to achieve in order to consider the simulation complete. For most applications, a residual value of 10^{-4} is considered sufficient, although values down to 10^{-6} can be set [51]. Multiple terms are tracked for convergence, and the specific terms depend on the physics model, however velocity and continuity values are tracked in most steady state incompressible flow models.

The iterative solver will run indefinitely if it is unable to achieve the target residual values, therefore a maximum number of iterations must be set in order to prevent the simulation from running indefinitely.

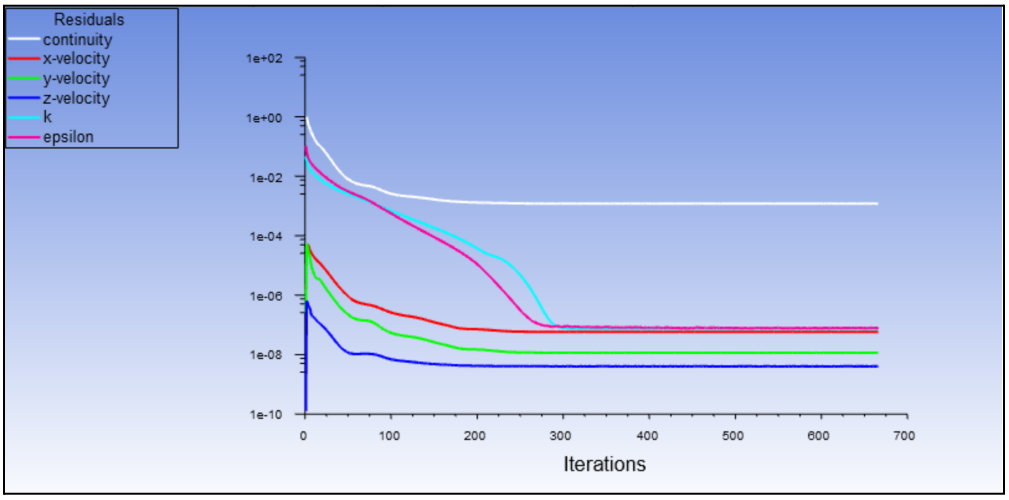


Figure 1.10: Solver residuals chart

Once the simulation is complete, result visualization software allows the user to inspect the results and understand the behavior of the physics being studied. For steady state incompressible fluid simulations, common visualizations include the resulting velocity and pressure fields within the domain.

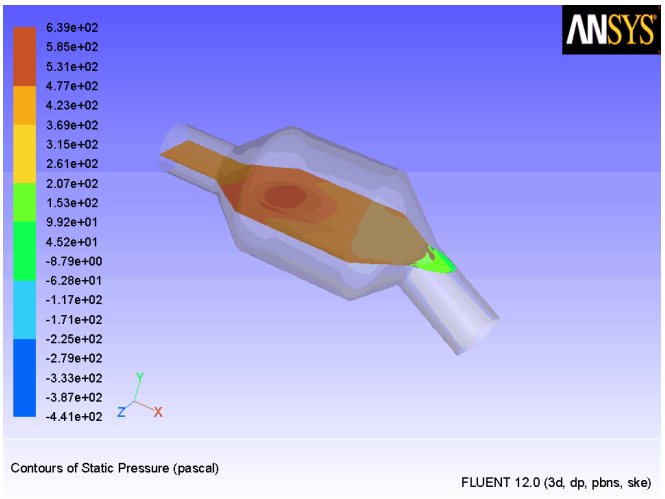


Figure 1.11: Pressure field visualization of a slice of the fluid domain [55]

1.1.3 Simulation as Part of the Design Process

For a mechanical system, there are two operating regimes that must be considered: the transient phase, which consists of the system startup and wind-down, and the steady-state. While the system must perform at a high level during the transient phase, the steady-state regime makes up the bulk of the operating time for a given system. Therefore, the bulk of the design efforts should be focused on optimizing the steady-state performance. For applications such as automotive, aerospace, and energy, incremental optimizations yield massive societal impact. Modern engineering design loops are conducted in a digital space with CAD models and finite element simulations, iteratively improving the design based on the simulation feedback until a

desired level of performance is achieved. Once a satisfactory result has been obtained, a physical version of the model is built and tested to validate the digital results.

Simulation-centric design loops have proven to be more effective than traditional design-build-test loops in terms of cost and performance. However, there are still significant areas for improvement in the simulation pipeline.

1.2 Project Motivation

During my time working in the Aerospace industry, I used the three primary CFD softwares - Ansys Fluent (the most prevalent), Siemens Star-CCM+, and OpenFOAM, which is open source. While working with these softwares I encountered four common pain points: the learning curve, solver run-time, varying degrees of automation, and costs associated with use.

Learning Curve

Industrial grade CFD code operates on the philosophy that an expert user needs full control of the parameters at each stage of the simulation pipeline. It is non-trivial to determine the correct setup and parameters for a simulation, and incorrect selections lead to poor simulation performance. Troubleshooting is also non-trivial and can require significant effort to correct simulation performance. This lack of abstraction in the simulation setup creates a barrier to entry for prospective CFD users and requires a significant time commitment to learn the fundamentals that are required to generate physically accurate simulations.

Solver Run-time

Finite element simulations work by breaking the physical medium into a series of discrete elements, and large or complex systems require a greater number of elements in order to achieve

an acceptable level of accuracy. Given that the computation run-time scales linearly with respect to the number of elements [26], engineers are forced to balance the trade-offs between simulation fidelity and run-time. While CFD solvers allow for parallelization, there is often a licensing or infrastructure cost associated with parallelization [27], which introduces another trade-off between the explicit financial cost and the implicit time cost.

Cost of Use

Both Ansys and Siemens CFD are proprietary simulation softwares with licensing models that scale based on access to compute infrastructure, requiring users to pay to remove software imposed throttles. There are multiple license options, however the standard 1-seat local license can cost $25,000+ for Ansys Fluent [25], and $30,000+ for Star-CCM+ [28]. Ansys does provide a student license that ships with significant limits. GPU usage is prevented while the CPU utilization is limited to 16 cores, and the finite element mesh is capped at 512,000 elements for fluid simulations [29]. Star-CCM+ provides a full-featured academic license, however there are costs incurred by the university to provide the software to students [30].

OpenFOAM is completely free to use but does not ship with a GUI, and has an equally if not more complex simulation setup process. Users of this software are forced to trade the financial cost for significantly more time spent traversing the learning curve.

Varying Levels of Automation

OpenFOAM by nature of being a CLI tool is completely automated and runs via C++ and JSON configuration files, and Siemens Star CCM+ provides a Java API that allows for scripting at various stages of the simulation pipeline. Ansys, by contrast, does not have a unified API to interact with their program. This is due to the fact that the Ansys product has been built largely

through acquisition over the years, and therefore the modules have distinct API's built on different programming languages. These modules live inside a setup and orchestration GUI program called WorkBench. WorkBench itself has a Python API, but it is meant primarily to run pre-configured simulation templates and does not provide the granular API's necessary to build an automated simulation pipeline.

1.2.1 A More Efficient Approach

At one of my previous employers, the Additive Rocket Corporation, we leveraged metal additive manufacturing [31] in order to create geometries that are not feasible to manufacture with traditional methods. Due to the cost of the novel manufacturing method, we had a large incentive to optimize and validate the design digitally before fabricating and testing. In order to accomplish this, we conducted a significant number of steady state fluid simulations, iterating and improving the designs based on the feedback from the simulation. The complexity of the 3D-printable designs led to larger and more time consuming simulations, which became a significant bottleneck in the design process. In order to address this bottleneck, we found that for certain designs, we could parameterize the geometry and fluid properties, reducing the simulation inputs to a vector. After generating a few hundred simulations for a rocket motor injector, we attempted to train an Artificial Neural Network to regress a scalar value of pressure drop through the injector. The trained model performed at a level that made it a viable proxy for the CFD simulation, with a throughput thousands of times greater.

From this experience I realized the potential impact of a generalized neural fluid simulation model. By abstracting away the complexity of the setup, simulation becomes much

more accessible. Additionally, the speedup from neural simulations has the ability to alter the economics and time-scales of system design by orders of magnitude.

Since my experience with the Additive Rocket Corporation in 2018, there has been a significant amount of research work exploring neural networks as a proxy for physical simulations, demonstrating feasibility as a surrogate for finite-element simulation methods. However, the current work on steady state neural fluid simulations has been narrowly scoped - either for specific fluid properties or specific geometries [1, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15]. This is due to the fact that it is extremely difficult to set up and generate large amounts of quality fluid simulation data, and it is difficult to map that data to a format that is ingestible for deep learning. Additionally, the projects have used different CFD software packages to generate the simulation data, which makes it difficult for subsequent work to be conducted. Ansys should be the de facto tool for research work in this domain due to the free academic license, its prevalence in industry, and significant amount of supporting documentation. However, Ansys use has been limited in published works.

In order to realize the goal of a geometry and fluid agnostic simulation model, there needs to be a common, accessible framework for generating and post-processing massive amounts of simulation data. To the best of my knowledge, such a tool for Ansys does not exist, therefore I set out to create the framework that would serve as a foundation for others to develop, build and collaborate in this space.

1.3 Related Work

The first use of a neural network as a method for predicting a steady state fluid field occurred in 2016 when Guo et al [1] at Autodesk demonstrated that convolutional neural

networks can be used for 2D and 3D open-channel flow problems. In that same year, Thompson et al [2] demonstrated that voxel neural networks could be used as a surrogate for a computationally expensive task in an Eulerian Fluid Solver - which is used in animation applications [3] (Figure 1.12).

Figure 1.12: Eulerian fluid simulation animation [2]

After publication of those two seminal papers, there has been a gradually increasing volume of work using deep learning for fluid simulation, including transient and steady state simulations for engineering applications [4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 15], as well as for animation [2, 12, 14].

Fluid Simulations Properties

Due to the considerable amount of setup and refinement necessary to execute fluid simulations, many of the papers make use of open channel in a fixed rectangular fluid domain, where the fluid flows around one or more objects [1, 2, 4, 5, 7, 8, 9, 10, 11, 12, 13, 14]. Doing so allows for simpler automation and also places the resulting data in a format that is more easily consumed by deep learning models, primarily voxel or convolutional neural networks [1, 2, 4, 7, 9, 10, 11, 13]. Additionally, Kashefi et al. [15] set up a circular domain, while Pajaziti et al [6]

use a template aorta shape with surface deformations, in both cases allowing for the use of a consistent meshing scheme.

Many of the papers fix or restrict the variation of the fluid properties, allowing the model to learn from only a small range of Reynolds numbers, therefore restricting the range of fluid behavior that is understood by the model. With respect to the engineering-centric works, [2, 4, 6, 9, 15] fix all fluid properties, while [8, 10, 12] vary a few parameters. Theurey et al. [13] utilizes a full range of Reynolds numbers (0.5 - 5M), however the simulation is restricted to airflow over an airfoil.

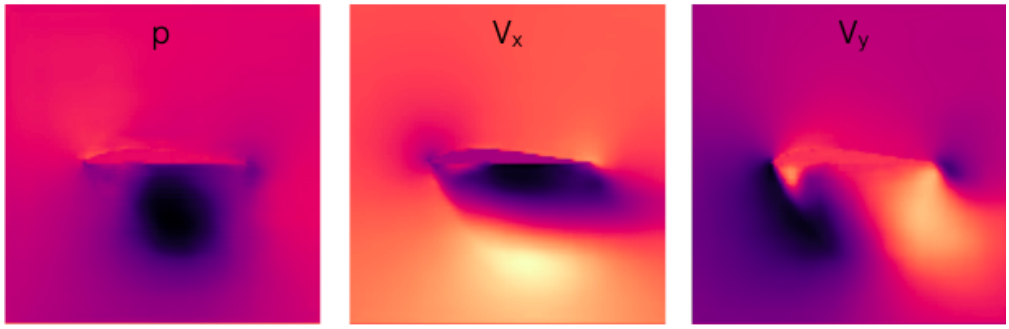


Figure 1.13: Pressure and velocity fields around an airfoil [13]

For the open channel simulations, many of the works focus on fluid flow around airfoils [5, 7, 8, 12, 13] or geometric primitive shapes [1, 4, 5, 8, 10, 11, 12, 15], and validate on more complex objects [1, 5]. The animation-focused projects leverage more complex objects [2, 14] that demonstrate the models' ability to generate realistic looking fluid flows.

In order to generate the simulation training data, [1, 2, 4, 7, 8, 9, 12, 13, 14, 15] use open source CFD software, including OpenLB [16], OpenFOAM [17], Sailfish [18], MantaFlow [20], SU2 [19], and FLeX [21]. Proprietary Softwares, including Ansys [22], Matlab FEATool [23], and COMSOL [24] are also utilized by [5, 6, 10, 12]. The resultant dataset sizes range anywhere

from 640 simulations [2] up to 500,000 [1], while a majority fall between 3,000 and 30,000 simulations [5, 6, 7, 8, 9, 12, 13, 14, 15].

Architectures and Problem Formulation

A bulk of the projects leverage convolutional or voxel neural networks in a UNet architecture [4, 7, 9, 10, 11, 12, 13]. Point cloud models are also used by [14, 15], where [14] developed a continuous point convolution to regress fluid particle location updates in a transient Lagrangian fluid model [3], while [15] utilized a PointNet architecture to predict the steady state flow field.

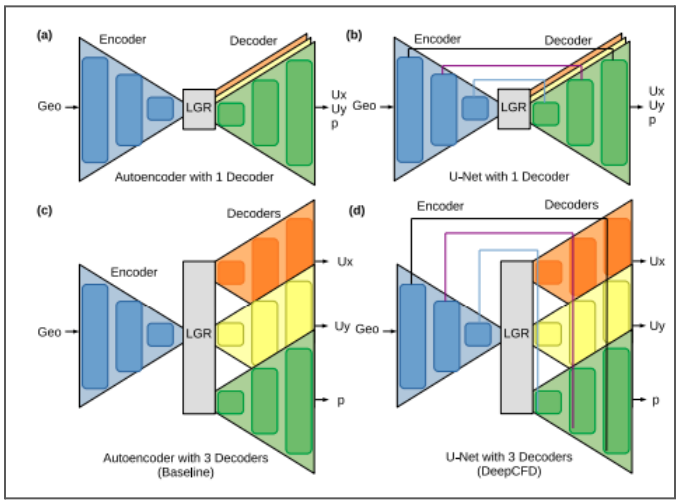


Figure 1.14: UNet with multiple decoder branches [4]

Graph neural networks have been utilized in both steady state and transient simulations [5, 8, 12], taking advantage of the natural graph form of a volumetric mesh. The lone project that attempted to model an internal fluid flow simulated flow through an aorta [6]. In this project the authors use a single fixed mesh for all simulations, and are able to generate pseudo-unique geometries perturbing the node locations (Figure 1.15). Doing so allowed them to project the mesh features onto a vector and then utilize an MLP to regress the target values.

Figure 1.15: Template aorta (left) and deformed aorta (right) meshes

Broadly, the task of a neural fluid simulation model is mapping the fluid properties, boundary conditions, and geometric representation of the fluid domain to the target velocity and pressure field values. For the models that leverage a CNN architecture, the fluid domain is defined on a rectangular grid where each pixel represents a fluid or solid region. These regions are either represented by a binary encoding [2, 7, 9, 10, 13] or SDF [1, 4]. The added distance context from the SDF representation proved to be effective for improving model performance. For the point cloud models, the points serve as the natural representation of the fluid domain. In the paper examining a Lagriangian model [14], there are two classes of points. One class represents individual fluid particles, while the other represents a discretized sample of the solid surfaces present in the scene (Figure 1.16). In [15] the points are extracted from the nodes of the simulation mesh and represent the fluid domain while implicitly representing the geometry through negative space. The projects that leverage a GNN architecture represent the mesh natively as the graph input.

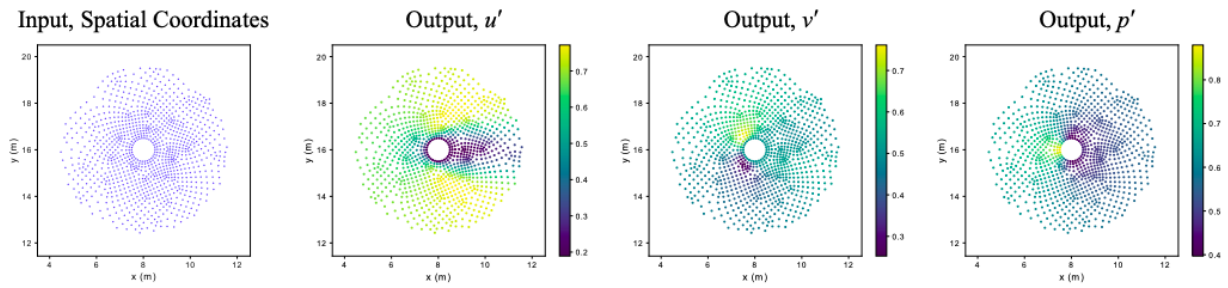


Figure 1.16: PointNet model inputs and regression predictions [15]

In addition to the geometry/fluid domain representation, the model must be given context about the boundary conditions and fluid properties, however this requirement is lifted in the projects where the boundary conditions and fluid properties remain constant throughout all samples [1, 4, 5, 6, 7, 9, 15]. For steady state simulations, input velocity is provided as a global context to all points [8, 12, 10, 13], in addition to other project specific properties, such as airfoil angle of attack [8]. Transient simulations require different inputs, depending on the application. For the projects that use a model as a proxy for the computationally expensive steps of a numerical solver, they provide specific inputs such as the velocity divergence field [2] or the terms of the Navier Stokes Equation [11].

Generally, the goal of a fluid simulation is to understand the resultant velocity and pressure fields. Therefore the task is to regress the velocity and pressure values at each node, whether that node is represented natively as part of a graph, or downsampled into a pixel or voxel [1, 4, 6, 7, 8, 9, 10, 11, 12, 13, 15] (Figure 1.16). In addition to regressing field values, global values such as drag force can also be calculated [5].

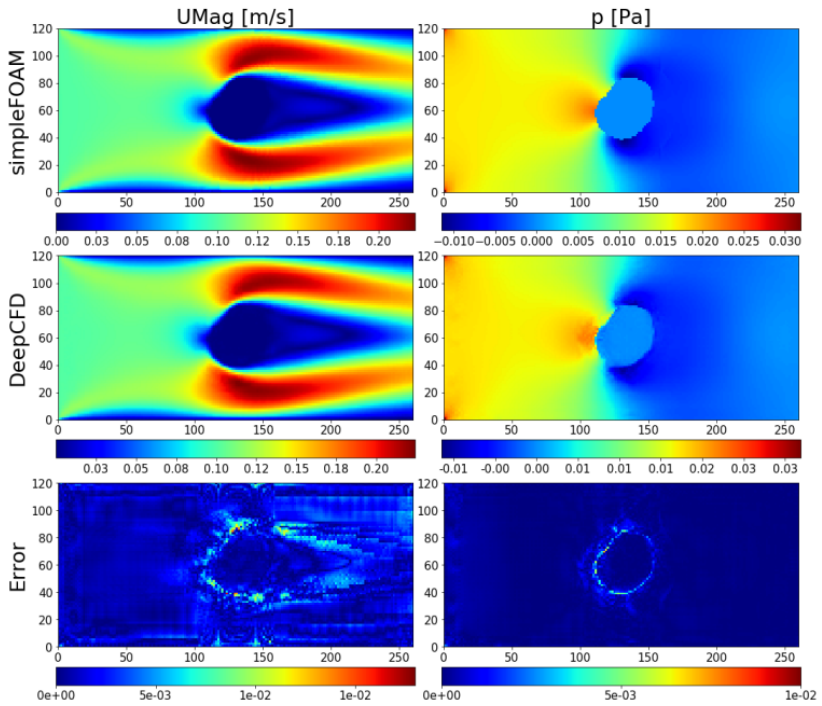


Figure 1.17: Comparison of ground truth vs model output for DeepCFD [4]

Models for transient simulations attempt to regress the velocity and pressure fields at a timestep *t* in the future, either directly [11, 12] or by calculating corrections that are part of a solver loop [2, 14].

Training and Evaluation Metrics

Given the task of regressing field values, L1 and L2 loss is widely used. Several papers use an equal weighted loss between Velocity (optionally V*x,* V*y,* V*z*) and Pressure [1, 8, 9, 10, 12, 13, 15], while others weight the velocity and pressure components of the loss based on the relative contribution to the total loss [4, 5, 7]. Transient simulation models take a similar approach, incorporating loss over multiple time steps [2, 14] to incentivize the models towards accuracy over longer time horizons.

Normalization strategies vary. Target values are scaled from a linear range of [-1,1] in [8] while [11, 12] mean normalize the data, with [12] using unit variance as the scaler. Non-dimensionalization (Section 1.1.1) strategies are also leveraged in [13, 15], where the target values are normalized first by their characteristic properties [36], with [15] subsequently normalizing by the range of values contained within each sample onto a [0,1] scale. Such a strategy takes advantage of the nature of open channel flows, where the field values will be unchanged except for regions near the solid objects present in the fluid domain.

Evaluating the effectiveness of the models is non-trivial. While L1 and L2 metrics provide a measure of the scale of the regression error, the error does not take into account the overall scale of the model. For example, regression error for a velocity value near a solid boundary, which is often near 0, may have low L1 regression loss while having an extremely large percent error. Conversely, for extremely small ground truth velocity values, a large percent error does not indicate that the model is performing poorly, especially if the absolute scale of the error is small relative to the bulk values present in the field. As such, approaches for evaluating model performance have varied among authors. A direct L1, L2, or RMSE loss is used by [4, 9, 8, 10, 12], while [1, 5, 13] calculate a relative loss by dividing out by the ground truth value. For a full list of training and evaluation metrics, see the appendix table A4.

Performance

Given the diverse nature and specific goals of the papers, it is not feasible to provide a direct comparison of steady-state simulation performance between papers. However it can be stated that several authors [1, 4, 6, 8, 10, 13] are able to achieve performance that indicates neural networks are a viable surrogate for narrow scenarios presented in the papers:

Table 1.1: Training and evaluation model performance of related works

Reference #	Evaluation Metric	Value
1	Av. Relative Error (magnitude)	1.98% (2D), 2.69% (3D)
4	MSE (Vx, Vy, P)	2.0303
5	1 - L2 Norm of Percent Error (Accuracy) for Pressure Profile and Drag Coefficient	C_P: 0.8203 C_D: 0.8601 (2D) C_D: .703 (3D)
6	MAE normalized by range of ground truth values in sample	10.19 % (Pressure) 4.47 % (Velocity)
8	RMSE (Vx, Vy, Pressure)	5.81×10^{-2}
9	Magnitude of Velocity Error	Not Published
10	MSE (Vx, Vy, Pressure)	0.345
13	Mean Relative Error (Vx, Vy, Pressure)	2.6%

It is also worth mentioning that [2, 12, 14] are able to achieve animation results that, albeit subjectively, appear physically realistic.

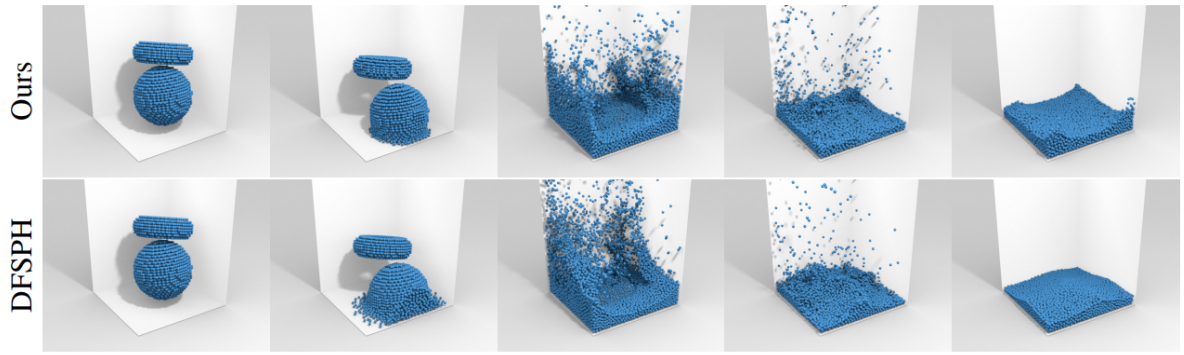


Figure 1.18: Comparison of ground truth vs model output the Lagrangian animation model [14]

1.4 Key Contributions

The goal of this project was to build a framework that will provide a foundation for others to pursue research on generalized neural fluid simulations. To that end, there are several components that the framework must possess in order to provide any utility for the research community. These components are: programmatic geometry generation in a format that can be consumed by the simulation software, automated simulation generation and execution, and a tool to map the resulting simulation data into a deep learning friendly format. The simulation portion of the project was accomplished through use of the Ansys simulation suite.

Geometry is generated using SpaceClaim [56], using four classes of pipe geometry (Figure 2.2). This initial set of geometry classes was chosen to demonstrate the potential for programmatic generation of irregular fluid domains. Across the 4 geometry classes, over 3060 unique geometry files were created.

The Simulation Pipeline consists of five components - geometry selection, fluid property selection, geometry meshing, fluid model selection, and simulation execution. After the simulation runs have been completed, there are several steps of post-processing to prepare the raw simulation data for model ingestion. A quality control routine parses the simulation logs and removes simulations that failed to complete. Another routine then processes the simulation data, parsing the solution files and creates the requisite model-consumable files for the training dataset. In total over 3400 simulations were generated.

Lastly, an exploration of geometric deep learning architectures (voxels and point clouds) for internal flow simulations was conducted. As far as I am aware, this is the first set of experiments for geometry and fluid-property agnostic models in steady state application. This exploration was conducted to examine the impact of various hyperparameters detailed in the

current state of art works in order to understand how they might map over to internal flow domains.

Chapter 2

Simulation Generation Framework

This chapter will cover in detail the core contribution of this project. The following sections include: geometry generation, simulation pipeline, quantitative and quality characterization of the resulting dataset, and the data post processing pipeline.

2.1 Geometry Generation

A major challenge for creating large fluid simulation datasets is the ability to generate realistic and variable geometries that will allow the model to generalize. Open flow simulations are less constrained by the geometry selection due to the regular structure of the simulation domain (rectangular, cylindrical, circular) - however many of the works utilize simple geometries that are not encountered in reality (triangles, squares, etc…). It is my understanding that these simplified geometries are used because they are easy to generate and reduce the complexity of meshing downstream in the simulation pipeline. Internal channel flows pose a greater challenge due to the irregular geometry of the fluid domain and the unique input and output regions for each file. The simulation pipeline must have knowledge about which faces of the geometry represent the input and output in order to specify the boundary conditions for the simulation. This requirement also prevents the use of many widely available digital geometry representations, which are often encoded in a surface mesh format. Instead the geometry is

required to be in a Boundary Representation (BREP) format. In a BREP format, the geometry is represented by the surface of the geometry, splitting the design into a series of discrete faces that are connected by edges [41]. By representing the geometry through discrete faces, the fluid domain can more easily be segmented into the input, output, wall and bulk fluid domain.

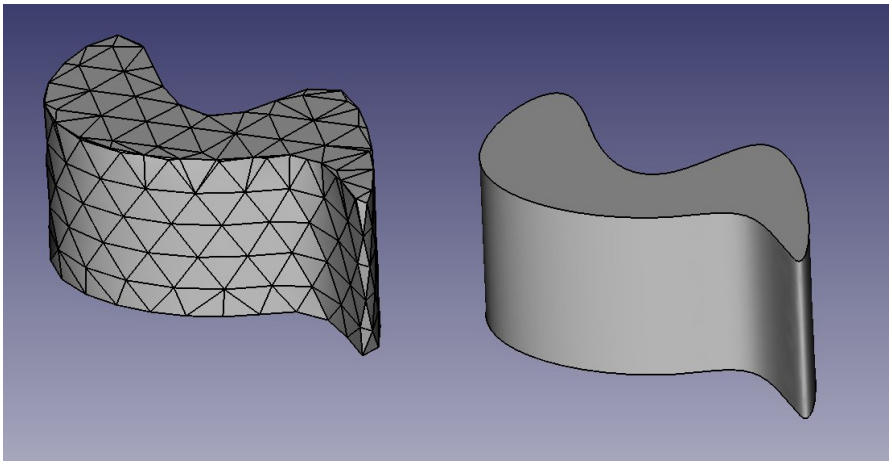


Figure 2.1: Representation of geometry using mesh (left) and BREP (right) format [67]

A solid model is often designed using a GUI, a process that does not lend itself to bulk geometry generation. While there are programmatic solid modeling softwares such as OpenSCAD [70], and OpenCascade [71], these are not commonly used in industry and are non-trivial to use. Additionally, the softwares do not easily provide a way to label the faces of the geometry, which makes it difficult to integrate into an automated pipeline. This ability to label the faces is necessary as it will be utilized in both the meshing (Section 2.2.3) and the simulation execution (Section 2.2.5) stages of the pipeline.

Fortunately, Ansys provides a CAD modeling software package, called SpaceClaim, as part of its software suite. This software exposes a python API that allows programmatic generation geometry and simpler programmatic face labeling. As a part of the broader Ansys ecosystem, the SpaceClaim native files are easily consumed by the downstream stages of the

simulation pipeline. For this reason, SpaceClaim was chosen as the geometry generation tool. Using this CAD software, four classes of pipe geometries were created: single-inlet-single-outlet pipes (referred to as simple pipes), y-junction pipes, single-inlet-multiple-outlet pipes (referred to as one-to-many pipes), and classic manifolds (referred to as simple manifold). Each class of geometry is generated with random permutations that allow each CAD model to have a distinct shape. While the specific implementation details for each geometry are beyond the scope of this paper, characteristics of the classes will be provided. Additionally it should be noted that all generated geometries have a circular cross section with a diameter between 0.1 and .2 meters, and maintain a length/diameter ratio between 5 and 20.

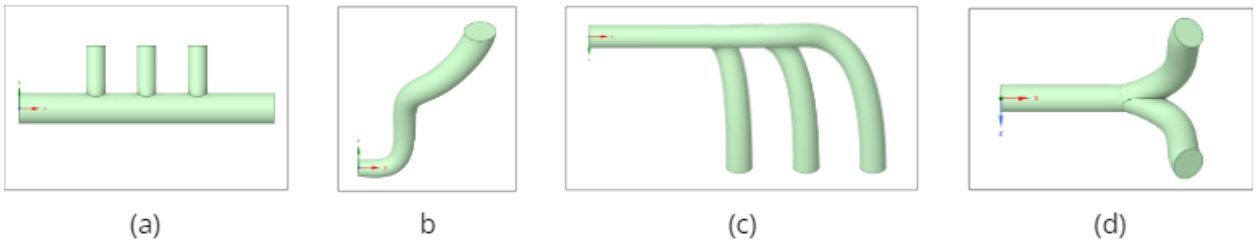


Figure 2.2: Classes of generated geometry: simple manifold (a), simple pipe (b), one-to-many (c), and y-junction (d).

The simple pipe follows a random trajectory with five randomly placed control points, and contains variable sized inlets and outlets. Similarly, the split paths of the y-junction pipe follow a random trajectory away from a variable length stem. The split path diameters and the stem diameters are also set randomly. The one-to-many pipe utilizes a variable length stem portion that serves as the branching point for a variable number of outlets (which range between 2 and 5). The outlet points are located at a random orientation and distance relative to the stem portion of the pipe, and are linearly placed with increasing distance in the X+ direction. Finally, the simple manifold has a variable length and a fixed diameter throughout the main pipe. The

outlets are evenly spaced along the length of the manifold and have a diameter that is sized as a variable ratio of the main pipe diameter, as well as a variable length.

In total, 3060 geometries were generated with over 500 files for each class. Additionally, for the y-junction and one-to-many classes, each geometry created 2 distinct files with reversed encoding of the inlets and outlets so that the simulation dataset contained both expansion and contraction along the fluid domain.

2.2 Simulation Pipeline

The simulation pipeline contains the following steps: geometry selection, fluid property selection, meshing, fluid model selection, simulation execution, and solution post processing. The first 6 steps are managed by an orchestration script that runs the pipeline in a loop until the desired number of simulations have been generated. For each simulation, the script uses the python subprocess command to open the Ansys Workbench in console mode and pass along the pipeline script which is then executed by the Workbench scripting console. When the simulations have been completed, the post processing routine is executed against the batch of newly generated simulations.

2.2.1 Geometry Selection

A geometry class is specified for each batch of simulations, and each simulation randomly selects one of the geometries generated for the chosen class. Once a geometry is loaded, the routine calculates the area on the inlet and outlet faces. If the area ratio between inlet and outlet is greater and 5:1 or less than 1:5, the geometry is thrown out and a new geometry is selected. This area ratio constraint was implemented because it was found that greater ratios led to convergence issues, likely due to the fact that the flow behavior changes drastically along the

length of the domain and requires a more sophisticated meshing approach that varies the inflation layer sizing along the length of the domain.

2.2.2 Fluid Property Selection

Once the geometry has been selected, the fluid properties are then chosen. The fluid density, viscosity, Reynolds number, wall roughness, and outlet pressure are sampled from their respective distributions (Section 2.3, Appendix A1). The Reynolds number, wall roughness, and fluid viscosity distributions were generated using log-linear scales, while fluid density and outlet pressure followed linear distributions.

The Reynolds number samples ranged from the laminar value of 100 to the highly turbulent value of 10^6. The cutoff of 10^6 was selected because that is the range where the pressure losses taper off in the Moody Diagram (Figure 1.4). The fluid density ranges from 600 kg/m^3 (ethane) to 3120 (bromine) kg/m^3, and while not exhaustive, these ranges sufficiently captured the range of common fluids [57]. The viscosity varies between values of 0.0003 to 1.4 Pa-s, which are the viscosities of acetone and corn syrup, respectively [59]. Wall Roughness ranges are also derived from the Moody Diagram, with values between 10^{-6} to 0.05. Outlet pressure varies between atmospheric pressure (14.7 psi) up to 200 psi, a common value of consumer facing compressors [58]. It should also be noted that the fluid theory dictates that outlet pressure has no impact on the pressure loss encountered throughout the length of the pipe. Nevertheless it is included as one of the experiments (Section 3.2) evaluates the deep learning model's ability to calculate the pressure values along the length of the pipe in the absolute scale.

After the fluid properties are randomly selected, the Reynolds number, density, viscosity and geometry inlet diameter are used to calculate the inlet velocity via the Reynolds number

equation. After this calculation, the continuity equation (Section 1.1.1) is used to calculate the expected velocity at the outlet. If the outlet velocity exceeds 100 m/s, it is capped at 100 m/s and then the inlet velocity and Reynolds number are updated accordingly.

The next step is to set the target size for the first boundary layer which is accomplished by setting a target Y+ value (Section 1.1.2), and using the following equations to calculate the initial layer height:

$$Y+ = \frac{y\mu_\tau}{\nu}$$

where y is the layer height, μ_τ is the friction velocity, and ν is the kinematic viscosity. The friction velocity is defined as:

$$\mu_t = \sqrt{\frac{\tau_\omega}{\rho}}$$

where τ_ω is the wall shear stress and ρ is the density. The wall shear stress can be calculated using the shear stress equation [76]:

$$\tau_\omega = \frac{1}{2} C_f \rho V_{av}^2$$

where C_f is the skin friction coefficient, and V_{av}^2 is the mean velocity. The skin friction coefficient is related to the Darcy friction factor f_D by the equation $f_D = 4C_f$. The friction factor can be determined using the Swamee-Jain equation - an empirical fit of the Moody Diagram (Figure 1.4) [77] :

$$f_d = \frac{0.25}{\left[log\left(\frac{\epsilon}{3.7d} + \frac{5.74}{Re^{0.9}}\right)\right]^2}$$

where ϵ is the wall roughness, Re is the Reynolds number and d is the pipe diameter. The inlet diameter, Reynolds number, wall roughness, inlet velocity, density, and viscosity are all known parameters. This allows for solving f_d, and subsequently τ_ω and μ_τ. With a set $Y+$ and a known μ_τ and ν, the first boundary layer size can be calculated.

Initially a target Y+ value of 100 is set, which is in the standard region for use of wall functions (Section 1.1.2). Using the maximum value of expected V_{in} or V_{out} in the equations, the initial layer size is calculated. If the layer size is outside of the range of the 0.01 - 0.1 of the Inlet Diameter, the parameters are adjusted until the inflation layer size is within the adequate range. This constraint was established to balance the mesh quality and total node count within the mesh. In the event that the initial calculation of the inflation layer was less than 1% of the diameter, the target Y+ value is increased to 200. If the resulting layer height is still too small, the inlet velocity is iteratively reduced by 10% until the layer height is within the allowable range. Similarly, if the layer height is greater than 10% of the diameter, the Y+ target is set to 30 and reevaluated, at which point if the resulting height is still too large, the Y+ target is changed to a value of 3, at which point the near-wall function is used [72]. If the layer height still exceeds the maximum value, the inlet velocity is iteratively increased by 10% until the boundary layer height is less than 10% of the diameter.

Once the boundary layer size is determined, the turbulent inlet intensity is calculated [49]. This metric describes the development of the fluid flow as it enters the fluid domain. The fluid solver requires this term for turbulent flow in order to accurately calculate the pressure loss through the domain. For all simulations, the intensity value is set such that the flow is considered fully developed at the inlet of the fluid domain.

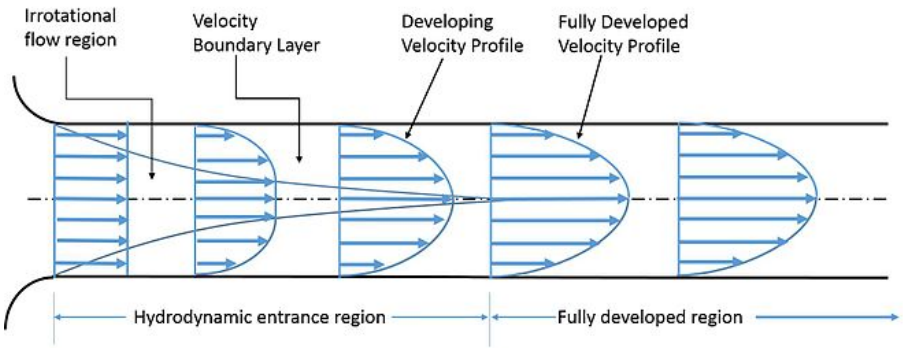


Figure 2.3: Turbulent flow development profile [69]

2.2.3 Meshing

Once the fluid properties have been established, the meshing program - Ansys Mechanical [73], is opened via the Ansys WorkBench API and the geometry is loaded into the program. Ansys Mechanical provides a scripting API through the Mechanical GUI, but does not provide access to this API through WorkBench. Fortunately, a workaround does exist. The WorkBench *SendCommand* function can be used along with the raw string of the python code. Ansys Mechanical will execute this code as if it was being run through the GUI.

Once the geometry is loaded, parameters for the meshing engine are established. These parameters include the cell type, inflation layer size, the target cell size, and the target mesh quality, amongst others. For this project, prismatic mesh cells were utilized for the inflation layer and tetrahedral cells were used for the rest of the fluid domain. It should be noted that more complex volumetric cell types are often used in modern CFD simulations, however simpler elements were chosen to reduce the complexity of the post processing routine (Section 2.2.6).

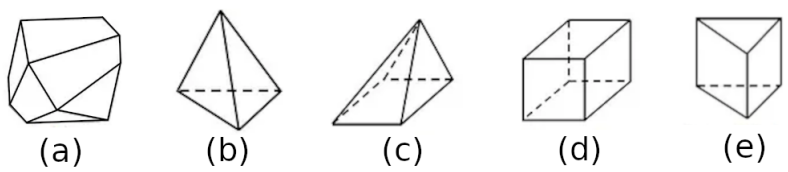


Figure 2.4: Volumetric mesh cell types: polyhedral (a), tetrahedral (b), pyramidal (c), hexahedral (d), prismatic/wedge (e) [68]

When the meshing run is completed, the cell count is checked to ensure that the total is less than 500,000, which is the upper limit that is allowed in the academic license for Ansys Fluent. If the resulting mesh is over the limit, the target cell size is increased by 10% and the meshing algorithm is rerun on loop until the resulting cell count is under 500,000. The distribution of cell counts, and mesh quality metrics can be found in Section 2.3.1.

2.2.4 Fluid Model Selection

Similar to Mechanical, Ansys Fluent does not expose an API through WorkBench. Within the Fluent GUI, there is a CLI that they refer to as the Textual User Interface that allows a user to setup and execute a simulation through command line inputs. Fortunately, the CLI can also be reached via the WorkBench *SendCommand* function.

Once the mesh is loaded into Fluent, the fluid properties (density and viscosity) are updated and then the fluid model is selected based on the Reynolds number value that was generated during the parameter selection. For Reynolds number values less than 2000, the laminar model is chosen, and for values greater than 3000, the $\kappa\omega$-sst model is chosen. If the Reynolds number is between 2000 and 3000, the transition-sst model is selected. For all fluid models, the default parameters are utilized.

After the fluid model is selected, the inlet velocity and outlet pressure boundary conditions are specified. For simulations in the turbulent regime, the turbulent inlet intensity is

also specified at the inlet boundary. Finally the wall roughness is set on the body of the fluid domain.

2.2.5 Simulation Execution

Before starting the simulation, the convergence criterion is set at 0.0001 or 0.0002 for all residual parameters, and the solver is provided an upper limit of 2000 iterations. The simulation is then allowed to run until completion, at which point the residual values are logged and the node-valued velocity and pressure results are written to a file.

2.2.6 Post Processing

After a batch of simulations has been completed, the post processing routine first begins by scrubbing the simulation output folders and quarantining any simulations that failed to complete during the batch generation. Given the high rate of simulations converging to the desired value (Figure 2.7), residual values were not evaluated when making the decision to quarantine output data.

With a set of clean simulation results, the next step is to process the solution data and mesh files to map the data into a deep learning friendly format. Ansys is able to export the volumetric mesh in a GMSH format [60], which provides a flattened list of points, cells, and faces that make up the mesh. The python package meshio [61] has the ability to process Ansys-generated meshes in the GMSH format, however at the time that this portion project was completed, the library was unable to process all the cell types present in the meshes used in this project. Therefore I created a custom mesh file processor that extends the functionality of

meshio. The output file is a mesh adjacency graph that encodes node location, fluid zone type (inlet, outlet, body, wall), and the node connectivity.

The shape and curvature of the fluid domain has a direct impact on the direction of the fluid flow and is useful information to provide a deep learning model. This information is not provided by the GMSH or solution file, and must be extracted from the calculated adjacency graph. Some mesh faces on the inlet and outlet are composed of four nodes (due to the prismatic cells of the inflation layer). These prismatic faces are split into triangular faces, and then the surface normals are calculated and added to the solution data.

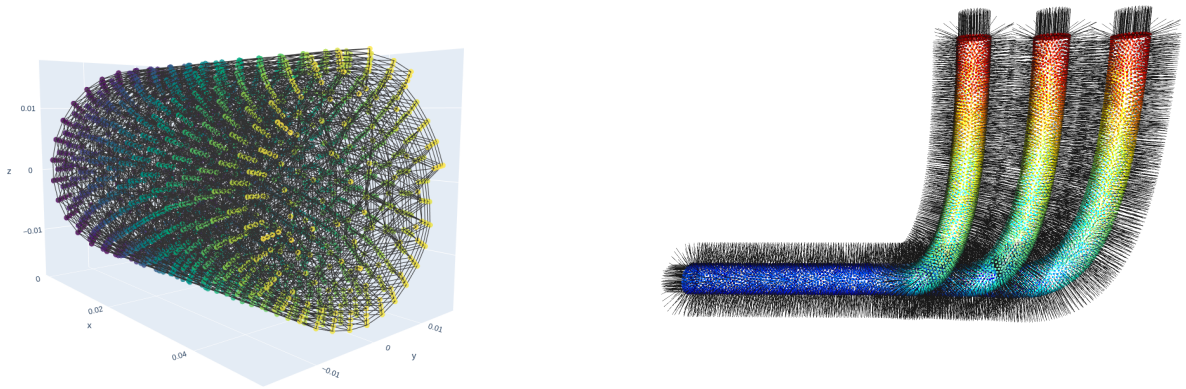


Figure 2.5: Extracted volumetric mesh (left) and calculated surface normals (right)

The exported solution file is stored in csv format and lists the node locations, velocity components (x,y, and z), as well as the pressure values. The node ordering in the solution file is different from the ordering in the mesh file, and therefore, a node-matching routine is executed to identify similar nodes based on the node location. Once that the matching is completed, the values from the solution file are reordered to match the sequence from the mesh file and the solution values are then stored in the polygonal PLY format [62]. Additionally, the fluid properties from the simulation are extracted from the log files and stored in a pickle file format [63] for use as model inputs.

2.3 Dataset

In total, 3061 training samples and 356 validation samples were generated, for a total dataset size of 3417 with a rough 90/10 split between training and validation samples. The relative proportions of the geometry classes within the data can be seen in table 2.1.

Each sample folder of the dataset contains the following data:

- the mesh graph adjacency information stored in a pickle file
- the simulation outputs (x, y, z location, Vx, Vy, Vz, Pressure, Node Zone) and the surface normals information stored in a .ply file
- the fluid simulation properties stored in a pickle file format

along with the original output data from the simulation - results, mesh file, and log files. The dataset is structured in this way to allow future researchers to plug and play while also providing flexibility for different preprocessing routines and hyperparameters combinations. The dataset contains a greater proportion of the one-to-many and simple-manifold geometry classes with the intent of providing a greater sample of the more complex fluid flows.

Table 2.1: Distribution of geometry classes within the dataset

Geometry Class	Training Samples	Validation Samples
Simple Pipe	576	80
Y-Junction	762	86
One-to-Many	860	105
Simple Manifold	863	85

2.3.1 Dataset Metrics

A stated goal of this project was to develop an initial dataset of validated, high quality fluid simulations with a wide range of properties. And in large part, that goal was achieved. 87.8% of simulations reached the target residual convergence for the continuity term - either set to 0.0001 or 0.0002 depending on the geometry class. The volumetric meshes achieved an average quality of 0.729, and with minimum values above 0.2, which is a sufficient quality level [74]. On average, each simulation required 8.02 minutes to converge to a solution, using an average of 59663 nodes in the simulation.

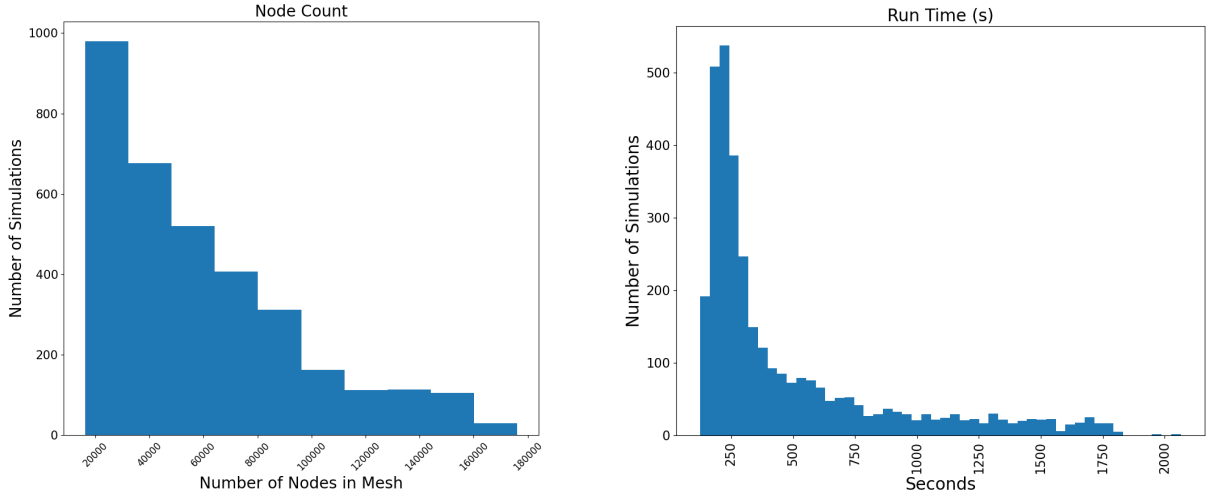


Figure 2.6: Distribution of (a) simulation run-time and (b) node counts in the volumetric meshes

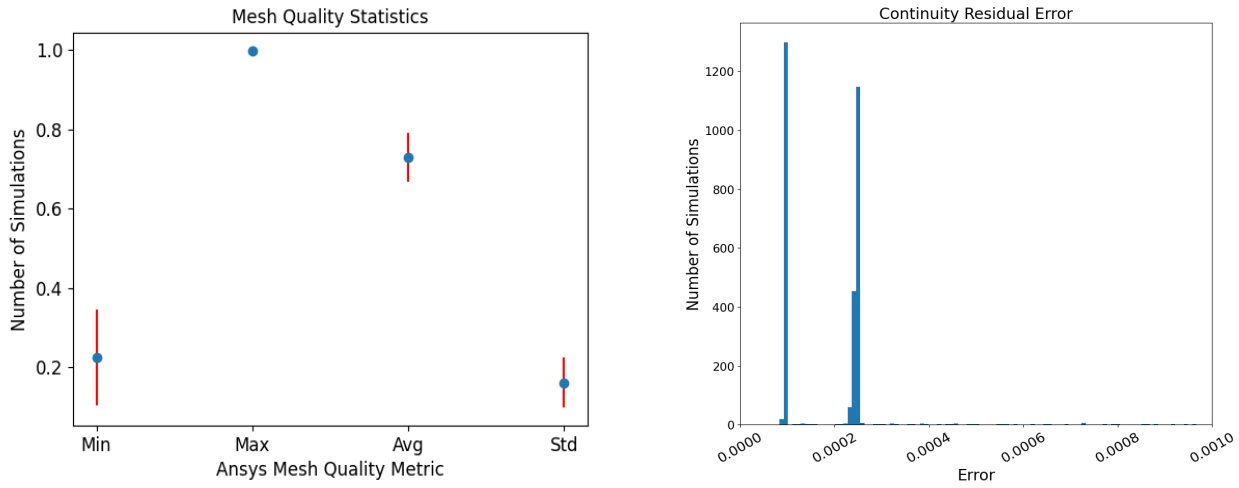


Figure 2.7: Residual value for the continuity metric, and the dataset distribution of the mesh quality values

Additionally, the distribution of inlet Reynolds number and wall roughness utilized in the simulations followed the target log-linear distribution specified in the parameter distribution routine (Section 2.2.2). It is important to note that even with the adjustments that can be potentially made to the velocity, and therefore the Reynolds number, during the parameter selection and meshing routines, the actual distribution does not deviate significantly from the intended distribution.

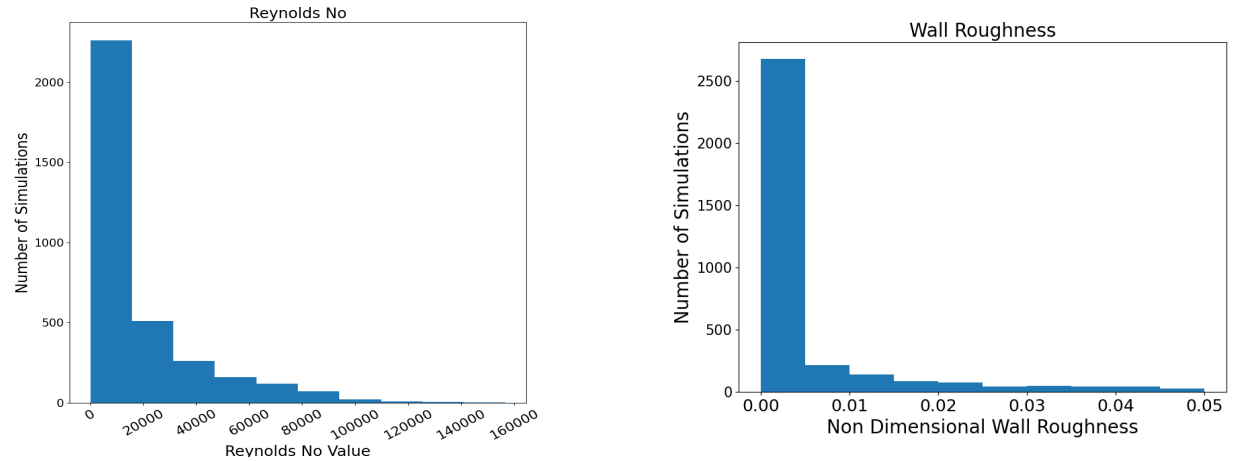


Figure 2.8: Distribution of inlet Reynolds number and Wall Roughness utilized in simulation dataset

For the distributions of the other fluid parameters utilized in the simulations please refer to the Appendix tables A1, A2, and A3.

2.4 Data Pipeline

The data pipeline was built for maximum flexibility to allow for easier experimentation. The pipeline code includes a generic data utilities package for manipulating the simulation data as well as PyTorch Dataset modules for preparing the input based on the type of geometry representation, whether it be sparse voxel, point cloud, or graph representation.

2.4.1 Architecture Specific Network Input

Separate modules are provided to prepare the sample data for its respective architecture. The modules are built in a way that allows a user to call a single function with several configuration parameters as an input and it returns a training sample in the correct shape to be processed by the neural network.

There are several common steps that are shared between the modules before the data is transformed into the final shape. These steps include geometry rotation, surface extraction, normalization, non-dimensionalization, input and target split, and global context allocation.

Rotation

In order to prevent the model from overfitting for fluid flow in a specific orientation, the geometries and their respective velocity and surface-normal vectors are rotated randomly about the origin so that the network is seeing new samples during each pass through the training set despite the fixed number of geometries and simulations.

Surface Extraction

Although not explored in the experiments for this project, another relevant problem formulation is to restrict the geometry input to just the surface of the geometry. Many times, engineers are only concerned with the values at the inlet and outlet of the fluid domain, and therefore the regression task can be constrained to the inlet and outlet of the domain. The data loading routine provides a feature flag for restricting the domain to the surface zones of inlet, outlet, and fluid wall. Additionally, a flag is exposed to restrict the regression loss to just the inlet and outlet nodes, masking nodes belonging to all other zones.

Normalization

The fluid parameters and regression values are normalized based on the scalers derived from the training split of the dataset. The velocity values are normalized to a [-1,1] range due to the directional nature of the vector, while the pressure values are scaled from [0,1] because pressure is a scalar value. Similarly, the scalar values of density, viscosity, Reynolds number, wall roughness are normalized to a [0,1] range. The node locations are also scaled and translated to a [0,1] bounding box, while also generating a [0,1] scale factor that describes the size of the fluid domain relative to others in the dataset.

Non-dimensionalization

As stated in Section 1.1.1, non-dimensionalization reduces the number of free parameters in the underlying problem and maps the underlying equation to a unitless space. In this context, non-dimensionalization is achieved by making the velocity, pressure, and location terms dimensionless. All velocity values are divided by the inlet velocity value to create $V^* = V/V_{in}$,

pressure is mapped to a unitless change in pressure $P^* = (P - P_{out}) / P_{out}$, and location is divided by the scale of the fluid domain $X^* = X/L$. By mapping the inputs and outputs to a unitless space, theoretically the regression task for the model becomes easier.

Input and Target Split

The processed solution data is stored in a single file and must be split into the input and target arrays. The network input requires the geometry encoding, fluid parameters, and the boundary conditions present in the problem. As such, the input vector consists of fluid properties, zone encoding, node locations, as well as the inlet velocity and outlet pressure values. The target vector contains the velocity vector (Vx, Vy, Vz) components and the pressure scaler P at each node location.

Global Context Values

The parameters of the fluid simulation are provided as global context to all of the nodes as part of the input feature vector. The number and type of parameters will vary depending on the hyperparameter configuration. One configuration worth noting is the decision to provide the boundary conditions (V_{in} , P_{out}) to all nodes or to restrict the boundary condition context to the inlet and outlet nodes - a configuration that mirrors the setup of traditional finite element simulations.

Architecture Specific Formatting

After the shared preprocessing steps, the samples are mapped to their intended formats for consumption by the model. The sparse voxel implementation utilizes the Minkowski Engine

[64] package and aggregates the input and target arrays into a sparse voxel data structure. The point cloud implementation takes a different approach to subsampling, taking a random subset of the solution based on a hyperparameter configuration for the quantity of sample points. Finally, the graph implementation utilizes the mesh adjacency information and includes the node connectivity as an additional input feature to the network.

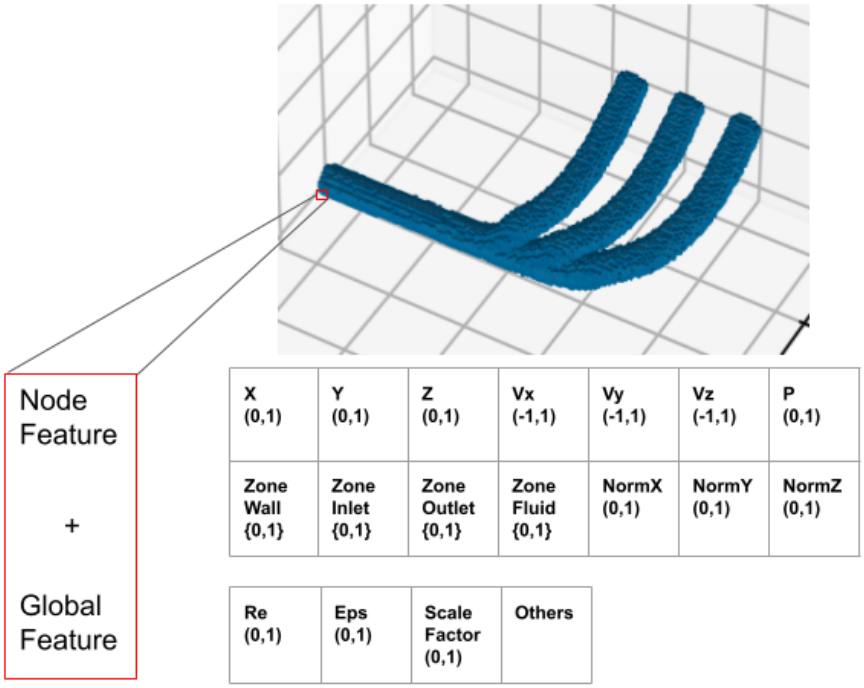


Figure 2.9: Node level and global context features that serve as the network input

CHAPTER 3

Experiments and Results

Experiments were conducted with the goal of establishing baseline performance for internal channel flows, utilizing architectures found in the neural fluid simulation literature. Additionally, hyperparameters and normalization schemes were explored to gain insight into performance optimization. Voxel and Point Cloud UNet models were developed and passed through multiple rounds of hyperparameter tuning to establish a set of parameters that were then used for full training runs. Details of the architectures, experiments, and model performance are provided in the following sections.

3.1 Model Architecture

All experiments utilized a UNet Architecture due to its ability to capture local detail while also encoding global context, which is necessary to regress node/voxel information throughout the fluid domain. Additionally, the experiments examined a variant of the UNet that utilizes multiple decoding branches - one for each regression target (V_x , V_y , V_z , P) - an approach that was utilized by [1, 4, 10]. For all experiments, the architecture downsampling layers continued until reaching a single point/voxel with a feature vector that stores the global context. At each level of the network, 1 or more residual residual blocks [75] were used before passing the outputs through a down/upsampling layer. The output layer consists of a series of voxel/point-wise linear layers before a final linear layer regresses the target values. For the

single-decoder branch network the output is of shape n x 4, and for the multi-decoder branch network the output is of shape n x 1 for each branch.

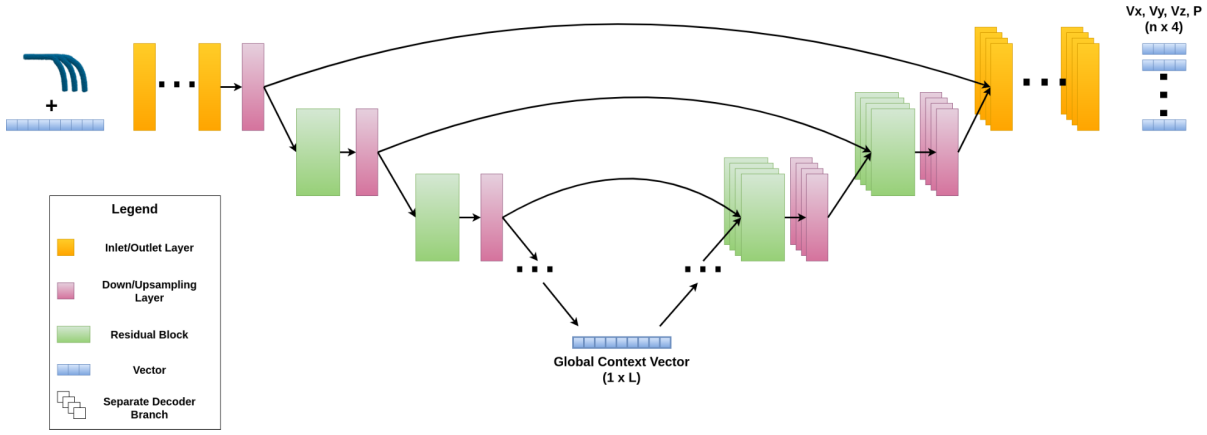


Figure 3.1: UNet architecture diagram with separate decoder branch variant

Sparse Voxel Implementation Details

The sparse voxel model utilizes the Minkowski Engine [64] to aggregate all of the points for a given sample into a discretized set of points and features. For all experiments, a 256^3 voxel domain size is used. For all feedforward convolutional layers, a kernel of size 3x3x3 is used with a stride of 1, while downsampling layers use a stride of 2. Given the effective downsampling by a factor 2 per layer, there are a total of 9 levels in the sparse voxel implementation.

Point Cloud Implementation Details

The point cloud implementation utilizes the ConvPoint continuous convolution [65]. The ConvPoint function operates in continuous space by learning a kernel that consists of a set of points and weights $(\boldsymbol{c}_i, \boldsymbol{w}_i) \in K$, as well as a distance weighting function (Figure 3.2). The operators receives the set of input points and features $X = \{(\boldsymbol{p}_j, \boldsymbol{x}_j)\}$, calculates the relative

distance between the kernel point c_i and input point p_j, and applies the distance weighting function to calculate a scalar which is then applied to the output of $w_i \cdot x_j$. The number of kernel points c and the number of input points p are also hyperparameters that are provided to the operator.

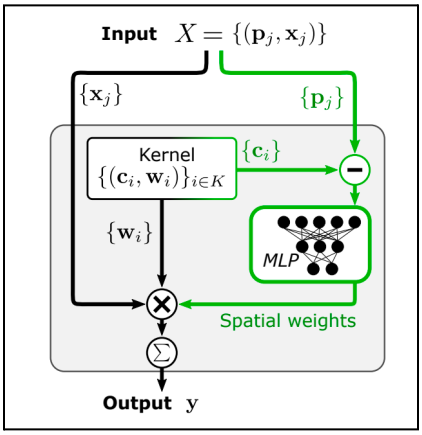


Figure 3.2: ConvPoint Convolution Operator [65]

In order to calculate the neighborhood for the convolution, the number of output points q must first be specified. There are various schemes for selecting q, however the default implementation is a pseudo-random selection of the input points using a scoring mechanism. Each time a point is selected for the output, a value of 100 is added to the point, and the point will not be chosen until all other points have an equal or higher score. This helps ensure that output points are not selected multiple times. Alternatively, one can specify the specific output points for the operator to utilize. This approach is useful for a UNet, because it allows for use of the same points at a given level in both the downsampling and upsampling portion of the network, providing a more consistent route for delivering layer-specific context through the skip connections. Once q is specified, the nearest neighbors are calculated using a k-d tree and the neighbor points are passed into the operator.

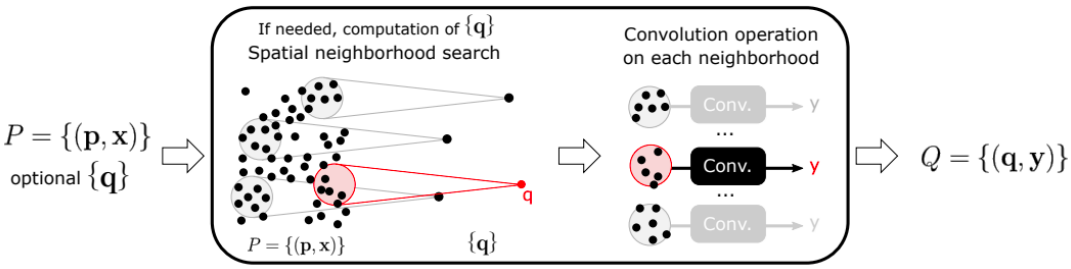


Figure 3.3: Neighborhood computation for ConvPoint operator

For the Point Cloud experiments, a random subset of 4096 points are selected from the input sample. This number was selected ultimately due to computational constraints. Given the number of nodes present per sample in the simulation dataset (Appendix Table A2), on average this value represents 6.67% of the total points, with minimum and maximum percentages of 2.3% and 25%, respectively. Within the UNet architecture, the points are downsampled at each new level of the network on the following schedule: 4096, 2048, 1024, 512, 256, 128, 64, 16, 1.

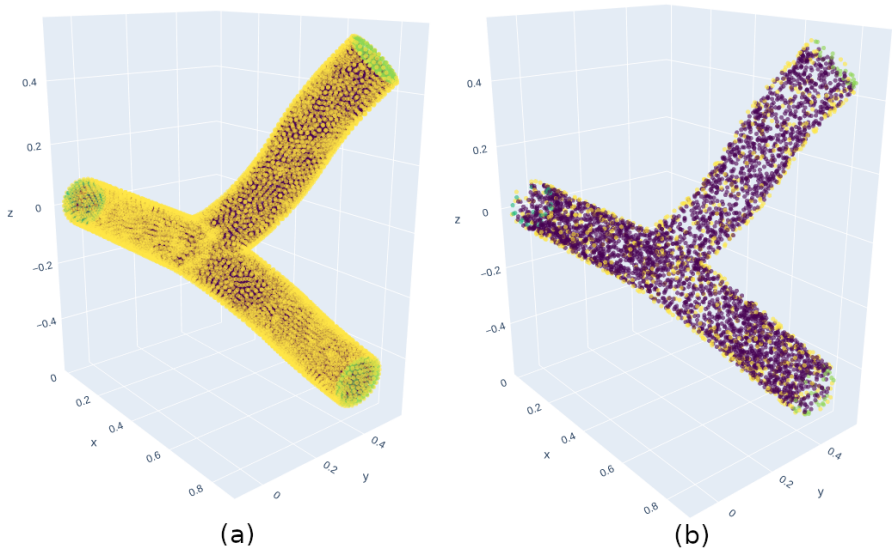


Figure 3.4: Point Cloud before (a) and after (b) subsampling to 4096 points. Zones are color coded as follows: yellow - wall, purple - fluid body, green - inlet and outlet.

By default, the ConvPoint function normalizes the point locations in the receptive field to the unit ball, however since the network input is already normalized to a unit scale, this step is skipped within the operator. At each downsampling convolution, the network randomly selects points to be used at the next layer, while the upsampling layers utilize the same points from the downsampling layers for the reasons mentioned above. To simplify the experiments, the same value is used to set the neighborhood size and the number of points in the ConvPoint convolution kernel. This value is used throughout the entire network, and is set as a hyperparameter.

3.2 Experiments

The approach for the experiments consisted of executing a hyperparameter search for the voxel and point cloud models, with a separate routine for the standard UNet and the UNet with multiple decoding branches. Once a fixed set of hyperparameters were established, training runs were executed, evaluating the impact of different normalization and global context schemes. Finally, the performance of the models were evaluated quantitatively and qualitatively.

3.2.1 Hyperparameter Tuning

Hyperparameter tuning was executed utilizing the Optuna optimization framework [66], using random search over the hyperparameter space. The optimization loop consisted of multiple runs, given the large number of hyperparameters. Each tuning run consisted of 100 iterations, running for 1 epoch, with early stopping if a given model was performing significantly worse than others in the set. The first tuning run was used to identify the parameters that exhibited the largest impact on model performance. The following run was used to fine-tune the important parameters and to establish optimal values for the key parameters. Finally, a third run was

executed with the key parameters set to fixed values. The best performing set of hyperparameters for the final search was then used in the full training runs.

Hyperparameters

The voxel and point cloud models explored the same hyperparameters, with the exception of the ConvPoint operator parameters. For a full list of the hyperparameter ranges for each of the four main search routines, refer to the appendix tables A5 and A6. It should be noted that the normalization scheme and fluid parameter context scheme were held fixed during the hyperparameter search, utilizing the standard normalization ([-1/0,1] range for absolute scale of values) and providing the velocity and pressure boundary conditions only to the inlet and outlet nodes of the geometry. The impact of these two hyperparameters were then examined during the full training runs.

The normalization hyperparameter has 2 variants: standard and non-dimensionalized. The standard normalization scheme, as described above, operates on the absolute scale of all fluid parameters, scales, and regression values. The non-dimensionalized approach (Section 2.4.1) maps the regression values to their non-dimensional terms $\boldsymbol{V^* = V/V_{in}}$ and $\boldsymbol{P^* = (P - P_{out}) / P_{out}}$, and utilizes a scalar that normalizes these non-dimensional terms to [-1/0,1] range. By mapping the problem to the non-dimensional space, theoretically the number of fluid parameters that need to be provided as context are also reduced. As shown in the Moody Diagram (Figure 1.4), the pressure loss through the pipe is a function of the Reynolds number and the Wall Roughness, therefore those are the only two fluid parameters that should need to be provided as context if using non-dimensional terms. Additionally, since the regression values $\boldsymbol{P^*}$ and $\boldsymbol{V^*}$ are normalized

by the inlet velocity and outlet pressure, they do not need to be provided as input features to the network.

The fluid parameter context scheme has different variants depending on the type of normalization that is utilized. For the standard normalization scheme, the fluid parameters - Reynolds number, density, viscosity, wall roughness, inlet diameter, and scale factor are provided as features regardless of context variant. The difference lies in the where the boundary conditions of the inlet velocity and the outlet pressure are shared. One approach shares the boundary conditions with just the inlet and outlet nodes, which parallels the setup of traditional simulations. The other approach provides the boundary condition information to all sample nodes, which theoretically reduces the burden on the network to propagate the boundary condition context to the rest of the nodes.

When using the non-dimensionalized terms, the fluid parameter context variation is simplified. Theoretically, only the Reynolds number, wall roughness, and scale factor are required, and those 3 terms represent the first variation. The second approach includes all of the fluid parameters: Reynolds number, density, viscosity, wall roughness, inlet diameter, and scale factor - to see if there is some benefit to provide this information explicitly to the network.

Training Routine

After establishing an optimal set of hyperparameters, the models were trained for a maximum of 1000 epochs with a 90/10 train/validation split. The learning rates were reduced by a factor of 2 when a plateau was reached on the validation loss, with a patience of 20 epochs before reduction. The learning rate was reduced a maximum of 5 times before stabilizing at a value 1/32 of the original rate. An early stopping mechanism was also put in place, allowing for

60 epochs of stagnating performance on the validation loss before ending the training run. At the end of each epoch, a model checkpoint was stored if it had achieved a better level of performance on the validation set, preventing degradation if training continued into the overfitting regime.

3.3 Results and Discussion

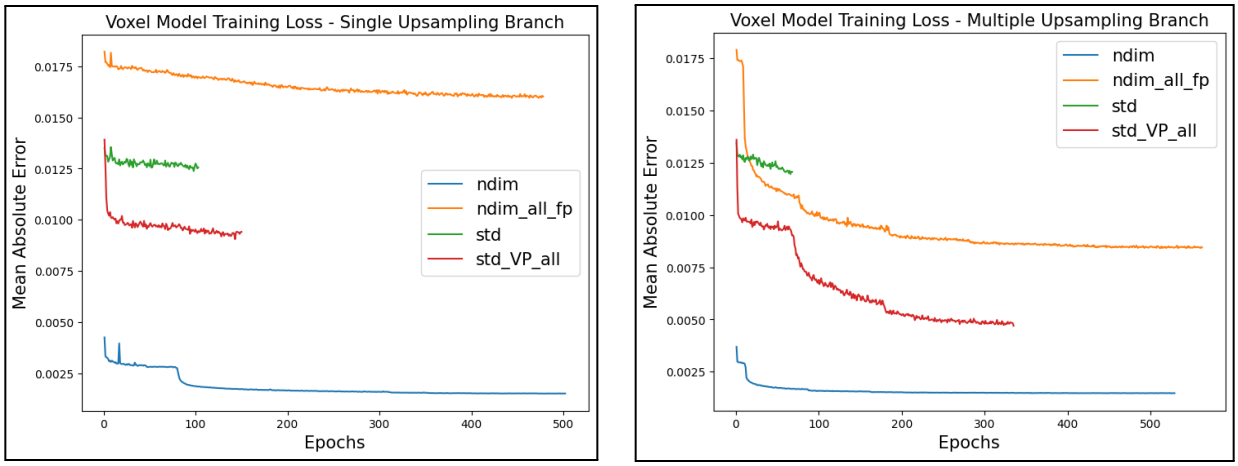
Models were trained on custom-built PC's using AMD Ryzen Processors with 32 GB CPU ram and 24 GB of GPU Ram, utilizing an NVidia RTX 3090 or 4090 series GPU. The available GPU Ram was a critical bottleneck during the training process, as the GPU frequently ran out of memory during the hyperparameter tuning, preventing evaluation of larger model configurations.

Throughout the hyperparameter tuning process, it was determined that Mean Absolute Error training loss metric yielded the best validation results for both the point cloud and voxel model architectures. For validation, Weighted Mean Average Percent Error was utilized. This metric was selected for its ability to account for the relative scale of values across the fluid domain, specifically the node/voxel values near the fluid walls where the fluid velocity can be near zero. Additionally, for the velocity regression targets, weighting the loss by the ground truth value can be interpreted as accounting for a given node's contribution to the overall mass flow through the domain. This behavior provides a better sense of the model's ability to honor the conservation equations (Section 1.1.1) in addition to providing a better measure of the model accuracy on the bulk fluid properties.

3.3.1 Results

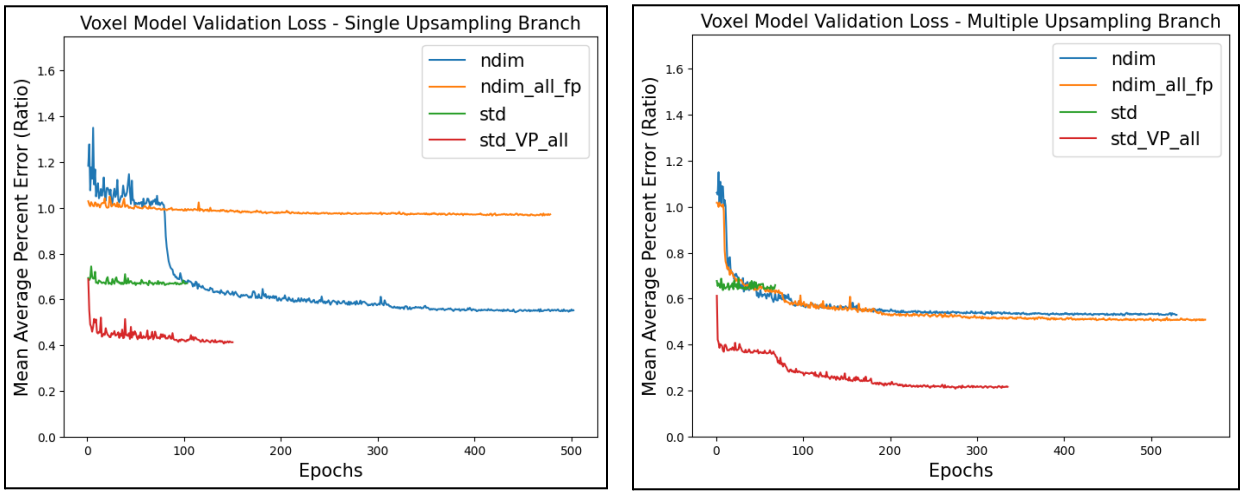
Voxel Model

The single and multiple-decoder branch architectures each used a distinct set of hyperparameters determined via the hyperparameter tuning process. For each architecture, training runs were conducted for each of the 4 permutations of normalization and global context as described in Section 3.2.1. In total, 8 training runs were conducted. The hyperparameters used for each network can be found in appendix table A7. The results of the training routine can be visualized below in figures 3.5 and 3.6 below, along with table 3.1. It should be noted that the validation loss plotted in the figures shows the weighted average percentage error in the normalized scale, which in the case of the non-dimensionalized values, is showing the relative error of $\boldsymbol{P^*}$ and $\boldsymbol{V^*}$ as opposed to the linearly normalized velocity and pressure values. The tabulated values in table 3.1 map the validation loss back to the dimensionalized space so that the error values can be compared directly. On average, dimensionalized validation error for the non-dimensionalized models is approximately 10% less than what is seen in the figures below.



(a) (b)

Figure 3.5: Training Loss Charts for single-decoder branch (a) and multi-decoder branch (b) networks. ndim, ndim_all_fp, std, and std_VP_all stand for non-dimensionalized, non-dimensionalized with all fluid properties, standard, and standard with velocity and pressure context for all nodes, respectively.



(a) (b)

Figure 3.6: Validation loss sharts for single-decoder branch (a) and multi-decoder branch (b) networks. ndim, ndim_all_fp, std, and std_VP_all stand for non-dimensionalized, non-dimensionalized with all fluid properties, standard, and standard with velocity and pressure context for all nodes, respectively.

The results indicate that the models using non-dimensionalized inputs did not generate more accurate validation results compared to the models that used standard min-max normalization. Despite the superior training loss demonstrated by one of the non-dimensionalized input schemes, it would appear that the non-dimensionalized scale maps to a different range for the error validation metric. Another interesting phenomena is that the models which used non-dimensionalized inputs along with the additional fluid properties (ndim_all_fp) performed worse than the models that did not utilize additional fluid properties as input features. It would be expected that a model could use these additional parameters as context to improve the results, or learn to ignore the additional parameters and achieve near-identical performance to the other non-dimensionalized models. However that was not the case as the models with basic non-dimensionalization outperformed in training for both single and multi-decoder branch models and in validation for the single branch model. The models that utilized non-dimensionalization achieved incrementally better performance than the models that used standard normalization (std) with the input features provided only on the fluid boundaries. This would indicate that there is little to be gained from the non-dimensionalized input scheme if using that same network architecture.

Table 3.1: Voxel model performance

Model Type	Train Loss (min)	Train Loss (last)	Val Loss (min)	Val Loss (last)
single ndim	0.00151	0.00152	0.54355	0.55220
single ndim w/ all_fp	0.01595	0.01602	0.96579	0.97150
single std	0.01238	0.01255	0.66171	0.67066
single std VP all	0.00906	0.00941	0.40689	0.41259
multi ndim	0.00146	0.00146	0.52653	0.52912
multi ndim all_fp	0.00842	0.00846	0.50257	0.50811
multi std	0.01196	0.01207	0.63692	0.65851
multi std VP all	0.00469	0.00465	**0.20760**	0.21597

Comparing the standard normalization schemes (std and std_VP_all), it can be clearly seen that providing the input feature context to all voxels dramatically improves performance, yielding results of 40% vs 66% error for the single-decoder branch model, and 20% vs 63% error for the multi-decoder branch. Additionally, utilizing the multi-decoder branch architecture improved results across the board when compared to its single-branch counterpart, achieving a top performance of 20% validation error for the variant using standard normalization with context provided to all voxels (std_VP_all). While the multi-decoder branch models lose the context of the other values in their respective decoder branches, it would appear that allowing each decoder branch to specialize on a specific target value provides the model needed capacity.

Figures 3.7 and 3.8 below provide a comparison of the predictions from the top performing model and the ground truth values from a sample simulation. Given the irregular fluid domain, cross sectional slices are taken at various lengths and angles along the pipe. To simplify the visualization, the velocity components are aggregated into a single velocity magnitude value.

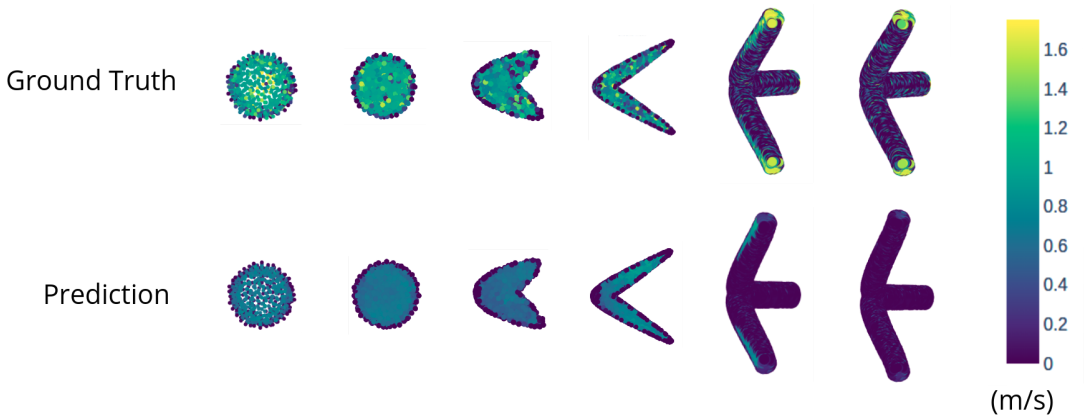


Figure 3.7: Voxel model velocity profile comparison

From the visualization it can be seen that the model tends to over smooth the velocity field for a given cross sectional area, lacking the axial variance seen in the ground truth values. The regressed pressure values are closer to the ground truth values for most of the fluid domain.

This is likely due to the fact that the pressure gradient mainly varies longitudinally through the domain and likely experiences a less severe change compared to the velocity values, which makes the regression task simpler for the model. To see the pressure values on a different scale which is capped to the range of the ground truth values, see Figure A2 in the appendix.

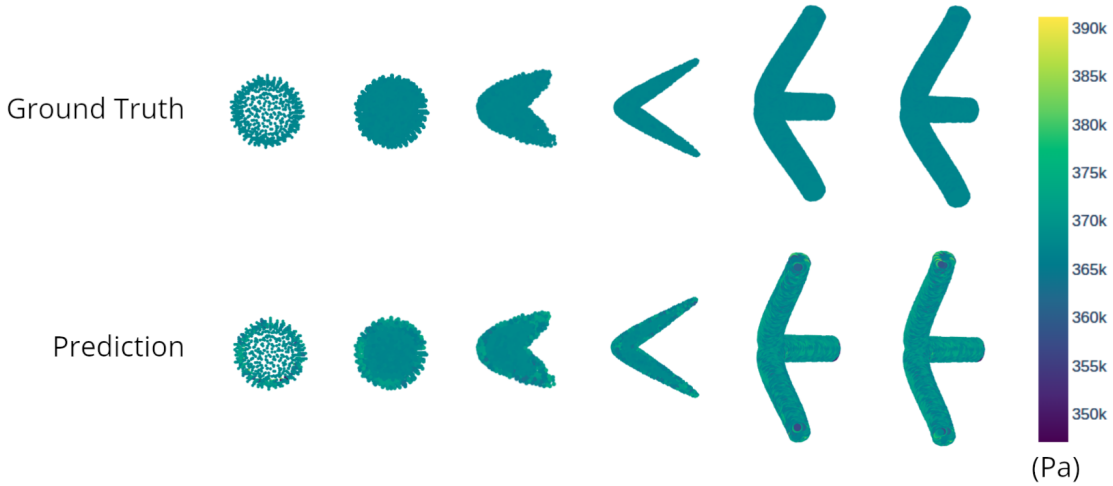


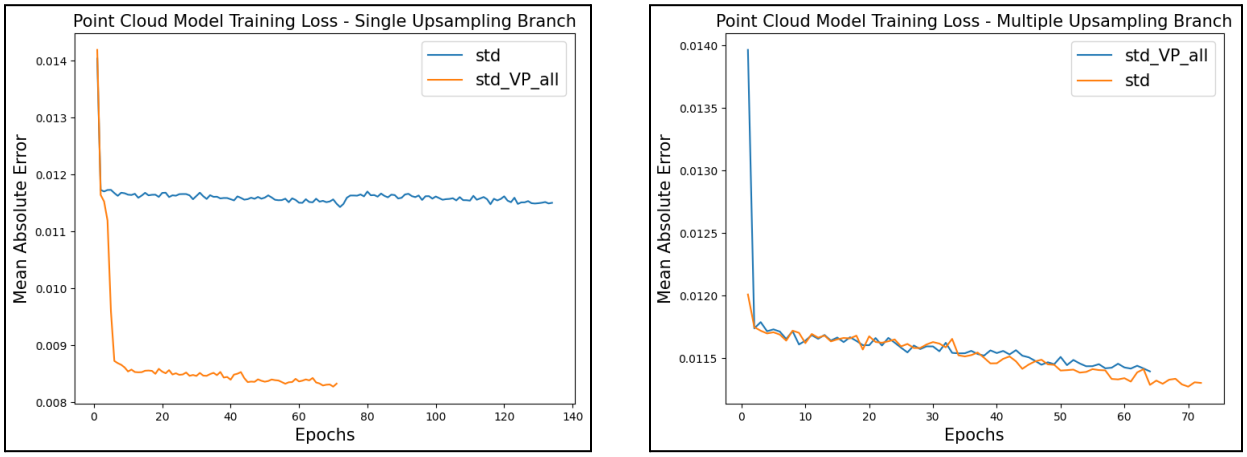
Figure 3.8: Voxel model pressure profile comparison

Point Cloud

The point cloud experiments followed a similar experimental structure as the voxel models, but were limited to the standard normalization schemes due to time constraints of the project. In total, 4 training runs were conducted across the single and multi-decoder branch architectures and the hyperparameters for these models can be found in Appendix table A8.

The training results visualized in figures 3.9 and 3.10, as well as table 3.2 show mixed results between model permutations. The multi-decoder branch architectures do not provide any performance gain when compared to the single-decoder architecture. The context permutation where the velocity and pressure are provided to all input nodes yielded a performance

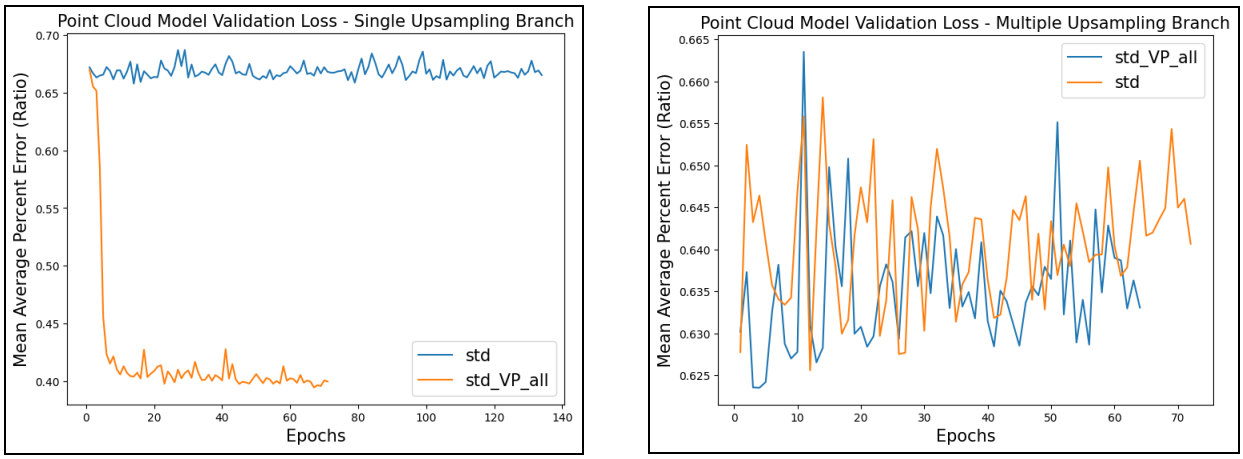
improvement only for the single-decoder architecture, and did not improve performance against the baseline for the multi-decoder architecture.



(a) (b)

Figure 3.9: Training loss charts for single-decoder branch (a) and multi-decoder branch (b) networks. std, and std_VP_all represent standard, and standard with velocity and pressure context for all nodes, respectively.

It appears that the models generally reached a saturated performance level early in the training runs and did not possess the capacity to continue to learn with multiple passes through the training set. The primary hypothesis for this behavior is that the number of sample points per geometry was not sufficient and did not provide the model with enough geometric context to regress the flow field. It is worth noting that the point cloud regression task is inherently more difficult than the voxel task because the point cloud model has incomplete information and must also regress the exact value of each point. The voxel model, by contrast, is provided with the entire downsampled geometry, and the per-voxel values are smoothed by averaging the point values located within a given voxel. Another potential reason for the limited model capacity is the overall size of the models were limited due to GPU constraints, as the hyperparameter tuning runs frequently ran up against GPU Ram limits, limiting search for larger models.



(a) (b)

Figure 3.10: Validation loss charts for single-decoder branch (a) and multi-decoder branch (b) networks. std, and std_VP_all represent standard, and standard with velocity and pressure context for all nodes, respectively.

Table 3.2: Point cloud model performance

Model Type	Train Loss (min)	Train Loss (last)	Val Loss (min)	Val Loss (last)
single std	0.01148	0.01150	0.65863	0.66517
single std VP all	0.00827	0.00832	**0.39464**	0.39985
multi std	0.01127	0.01130	0.62560	0.64066
multi std VP all	0.01139	0.01139	0.62352	0.63307

Visualizing the results of the best performing point cloud model, it is evident that the model is not able to adequately regress the velocity and pressure flow fields (Figures 3.11 and 3.12). The predicted velocity field indicates that the model defaulted to predicting 0 velocity for all nodes, lacking the capacity to regress values for non-wall nodes. The pressure field regression, which should be the simpler to predict due to its generally consistent value across the domain, also shows a great deviation, with the model adding variation that does not exist in the domain.

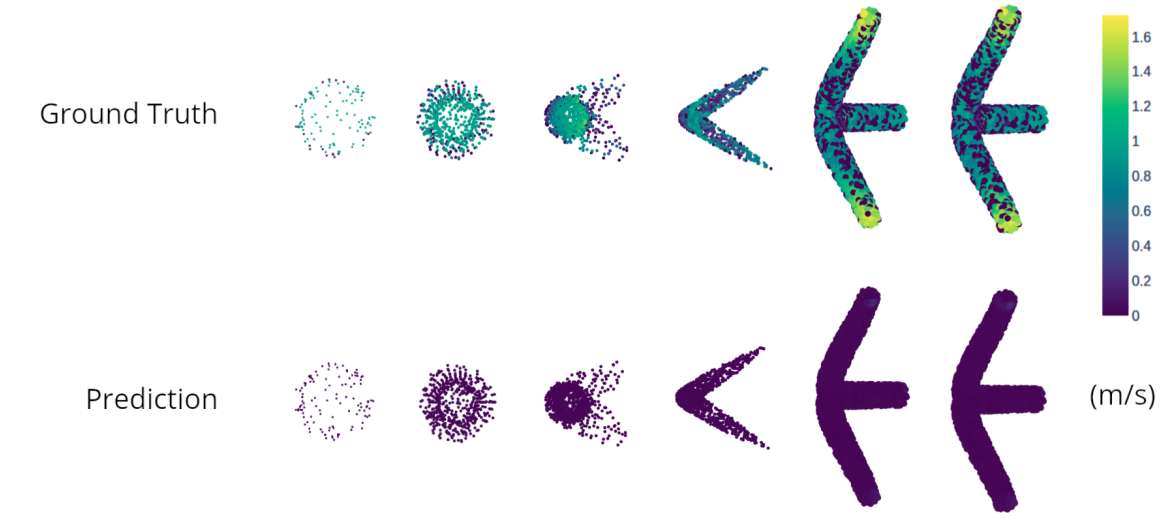


Figure 3.11: Point cloud model velocity profile comparison

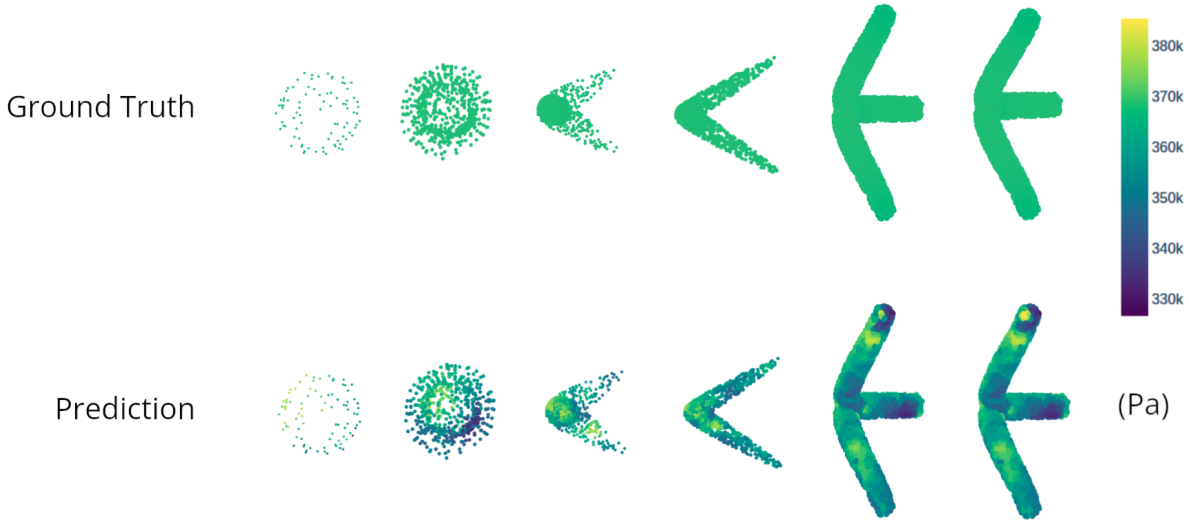


Figure 3.12: Point cloud model pressure profile comparison

3.3.2 Discussion

The experiments did not demonstrate any universal trends that improved performance across all permutations of the point cloud and voxel models. Providing the velocity and pressure values as context to all input nodes did improve performance across the implementations of the

voxel models, but did not improve the performance of the multi-branch point cloud model. That being said, providing the global context to all nodes, as opposed to just the domain boundaries, reduces the burden of propagating that information to the nodes, therefore it makes sense that the performance generally improved with this configuration.

Additionally, for the voxel model implementation, the multi-branch decoder implementations outperformed their respective single-branch implementations across the board, however this branch splitting did not provide any clear benefit to the point cloud models. Given the limited capacity that was demonstrated by the point cloud models, it is possible that the models were so under-provisioned that the hyperparameter configurations were not capable of improving performance and would require a significantly larger model to show any impact that was demonstrated in the voxel model experiments.

Non-dimensionalization has a theoretical argument for improving the model performance based on the fact that it reduces the number of free parameters and simplifies the underlying physics being modeled. However, the voxel experiments indicate that there is no benefit in mapping the parameters to the non-dimensional space if using an architecture that was tuned for the linearly normalized parameters. Perhaps the underlying functions being learned by model are different enough between the normalized and non-dimensionalized parameters that it requires tuning an architecture specifically for the non-dimensionalized parameters in order to see any gain in performance.

While the hyperparameter exploration may not have provided conclusive trends, the result visualization for the best voxel model indicates that it has sufficient capacity to regress something close to the ground truth flow field for the velocity and pressure components (Figure 3.7 and 3.8). And while the predicted flow field lacks precision, the results demonstrate that the

model was able to differentiate between fluid and solid boundaries, and also shows that the model internalized the continuity equation (Section 1.1.1), with the predicted flow fields increasing in speed as the area decreases. Such a model is not sufficient to drop in as a replacement for a finite element simulation, but indicates that geometric deep learning models do have the potential to understand the underlying physics steady state fluid simulations in irregular domains, and that further research should be conducted to more thoroughly explore the capabilities of these models.

CHAPTER 4

Project Evaluation and Next Steps

The 2 main goals of this project were to build a framework that would allow others to conduct research in the field of neural fluid simulations, and to evaluate if geometric deep learning models are capable of understanding the complexity of generic steady state fluid simulations. To a large degree, those objectives were achieved. The simulation pipeline code, post processing routines, and deep learning utility packages provide a solid foundation that can be reused and expanded upon by other research groups. Additionally, the experiments conducted demonstrate that existing geometric deep learning architectures have the ability to regress steady state fluid fields through irregular geometric domains.

A major contribution of this project was successfully using Ansys in a general automated pipeline. For the reasons mentioned in Section 1.2, Ansys has not been widely used in deep learning research, with research groups opting to use many other types of CFD software. In order

to develop a truly general neural fluid simulation model, massive amounts of data will need to be generated, and that will require collaboration and among research groups to build the necessary dataset. Currently, there is no standard simulation result format or quality threshold benchmark because groups are using different simulation tools with their own bespoke pipeline code. By proving that Ansys can be used in a general automated pipeline, my hope is that the academic community will converge on the use of Ansys and begin to establish the standards for quality and solution formatting. Once these have been established, the quality and volume of work in this space should increase drastically, given that much of the boilerplate code is being handled by the framework, freeing up researchers to focus on the deep learning aspects of the research.

4.1 Next Steps

Throughout this project, there were many questions and ideas that were not explored due to the scope constraints of this project. The experiments just scratched the surface of what can be explored, and future research can extend in many different directions. In this section I will discuss what I believe some of the future work might look like.

Simulation Framework

While architecture of the simulation pipeline package is structured in a modular and extensible way, the code is still very much research grade. A follow-up task after the completion of the thesis project will be to take a polishing pass on the code and then to look for existing projects in the neural simulation space where the code can be merged. While this code can exist as a standalone open-source project, professional and personal commitments will prevent my continued work on this project.

Simulation Pipeline Functionality

There are several areas where improvements and extensions can be made to the simulation pipeline, resulting in higher quality and more complex simulation data. An obvious improvement would be generating truly random geometries that while still properly encoding the domain boundaries. Accomplishing this through use of the SpaceClaim API will require use of template geometry shapes and an orchestration routine that combines the template shapes in a randomized way to form a fluid domain. The combination routine could be accomplished with a hand-crafted pseudo-random routine, or can make use of autoregressive geometry generation models [79] that use a codebook of shapes to generate a geometry.

The volumetric mesh cell types used in this project were chosen due to the simplicity of processing the output mesh file. In practice, many engineers use more complex mesh cell types such as hexahedral and polyhedral cells because they tend to lead to better simulation results. By extending the solution processing functionality to be able to handle these common volumetric cell types, the meshing routine can make use of these cell types and should improve the simulation quality. As an additional post-processing quality check, the total mass flux at the inlet and outlet of the fluid domain can be compared to evaluate how well the conservation of mass has been preserved by the simulation. Calculating this value provides another measure of the simulation quality.

The simulation pipeline is a general automated framework and can be easily modified to examine different physical phenomena such as structural mechanics or heat transfer. The framework can accommodate new simulation studies by adding code for the module in the Ansys suite that is responsible for the physics model and execution of the new phenomena. In the realm

of neural fluid simulations, natural next steps include examining transient simulations and more complex fluid-heat transfer models.

Deep Learning

Given the scope constraints of the project, there is much more that can be done on both the architecture and analysis fronts. The hyperparameter tuning routine was shared across permutations, and while that significantly reduced the time to achieve a fully trained model, it may have overly constrained the parameter search for the different permutations. It is worth examining if a hyperparameter search tuned specifically for non-dimensionalized inputs yields better results than those seen in the experiments. Another vector of research could be the loss metric utilized during the training process. The experiments in this project used equal weights between the loss terms - however [4, 5, 7] use different weights between the pressure and velocity terms so that component did not dominate. Additionally, one could look at integrating in a physics-informed loss term that penalizes the model for violating the conservation laws, either at the node level or at the domain level.

At an architecture-level, a natural progression is to look at a Graph Neural Network model which processes the input in the native mesh format. Due to the size and connectivity of the meshes from the simulations, processing the full input geometric will require multi-gpu compute infrastructure, although downsampling can be implemented. The point cloud model used in this project essentially accomplishes this downsampling task by dynamically finding N nearest neighbors. A downsampled mesh for a GNN model could take a similar approach, but could make the edges between the neighbors static and explicit. Another set of experiments could look at restricting the geometric input to just the surface of the fluid domain (Section 2.4).

While it is not known if this problem formulation is an easier regression task for a deep learning model, it would reduce the memory burden on the model and free up compute to focus on the higher value regions of the fluid domain.

On the analysis front, it is worth analyzing where the models succeed and fail in an attempt to understand why the model behaves the way it does, and hopefully inspire improvements in subsequent experiments. Metrics such as wall distance, longitudinal distance from the boundaries, curvature, and gradients, to name a few, can be compared against the regression error to see where the model struggles. This information can be used to craft loss functions that can address the phenomena that cause poor model performance, and hopefully improve overall model performance.

In short, there is a lot of work that can be done, and it is my hope that the work completed during this project can help others in the research community push the boundaries of this field.

Appendix

A1 Dataset Statistics

Table A1: Mesh quality statistics

	Minimum	**Maximum**	**Average**	**Standard Dev.**
Average Value	0.225	0.998	0.729	0.161
Standard Dev.	0.121	0.001	0.061	0.064

Table A2: Mesh node counts

Average	**Mediaan**	**Minimum**	**Maximum**	**Standard Dev.**
59663	49717	16275	176056	35649

Table A3: Simulation run time (s)

Average	**Median**	**Minimum**	**Maximum**	**Standard Dev.**
481.17	291.46	125.45	2064.67	402.16

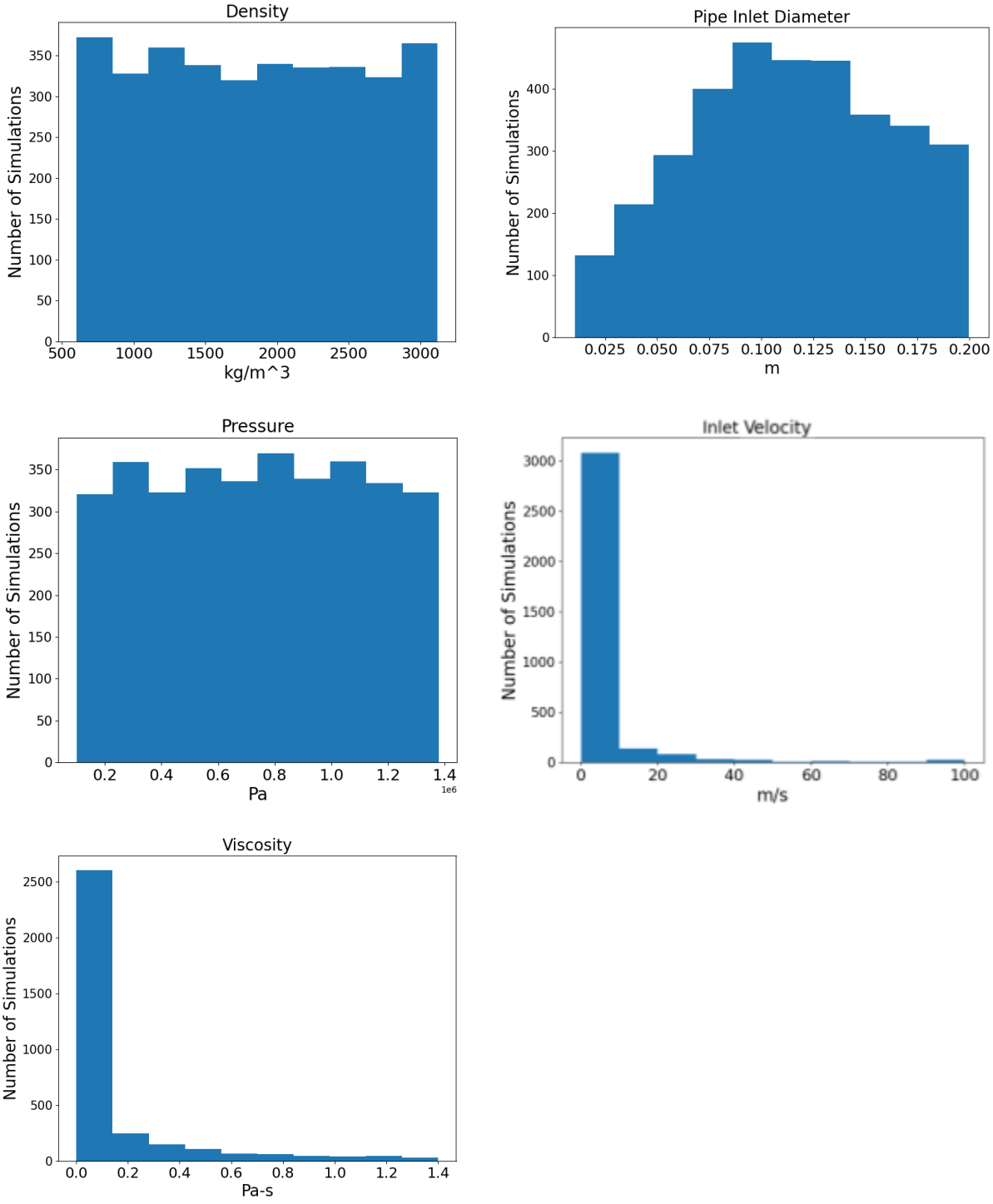


Figure A1: Distribution of fluid properties for simulation dataset

A2 Related Work

Table A4 : Training and evaluation metrics used in related works

Reference #	Training Loss Metric	Evaluation Metric
1	MSE	Av. Relative Error (magnitude)
2	Multi timestep MSE on divergence of v fields - weighted by distance to the boundary	Norm of max residual throughout the simulation
4	Term weighted, MSE for Velocity, MAE for Pressure	MSE
5	Term weighted MSE	1 - L2 Mean Percent Error (Acc)
6	MAE	MAE normalized by range of ground truth values in sample
7	MSE, pressure loss scaled by a factor of 10	Not mentioned
8	MSE	RMSE
9	MSE	Magnitude of Velo Error
10	MSE w/ Sum Reduction	MSE
11	Residual and divergence of Nav. Stokes eq at t+1	Parameter accuracy and divergence
12	MSE	RMSE
13	MAE	Mean Relative Error (normalized values)
14	2 time step position delta norm loss - with weighting based on nNeighbors in search radius	L2 Norm of delta to closest particle
15	MSE	Not mentioned

A3 Experiments

Table A5 : Voxel model hyperparameter ranges

Parameter Name	Description	Range
Batch Size		2 - 8
N Inlet and Outlet Layers	Number of convolutional blocks at the top level of the UNet	1 - 3
Inlet and Outlet Conv Channel	Channelwise depth of convolutional layer at inlet and outlet convolution blocks	16 - 256
N blocks	Number of residual convolution blocks at each encoding/decoder layer of the UNet	1-2 (multi-decoder branch) 1-4 (single-decoder branch)
Encoder Channels (Single-Decoder Branch)	Array specifying the number of convolutional channels for the encoder layers at each level of the Unet	One of the following: [32, 32, 64, 64, 128, 128, 256, 256] [64, 64, 128, 128, 256, 256, 512, 512] [64, 64, 128, 128, 256, 256, 512, 1024] [128, 128, 256, 256, 512, 512, 1024, 2048]
Encoder Channels (Multi-Decoder Branch)	Array specifying the number of convolutional channels for the encoder layers at each level of the Unet	One of the following: [32, 32, 64, 64, 128, 128, 256, 256] [64, 64, 128, 128, 256, 256, 512, 512] [64, 64, 128, 128, 256, 256, 512, 1024]
Decoder Channels (Single-Decoder Branch)	Array specifying the number of convolutional channels for the decoder layers at each level of the Unet	One of the following: [256, 256, 128, 128, 64, 64, 32, 32] [512, 512, 256, 256, 128, 128, 64, 64] [1024, 512, 256, 256, 128, 128, 64, 64] [2048, 1024, 512, 512, 256, 256, 128, 128]
Decoder Channels (Multi-Decoder Branch)	Array specifying the number of convolutional channels for the decoder layers at each level of the Unet	One of the following: [256, 256, 128, 128, 64, 64, 32, 32] [512, 512, 256, 256, 128, 128, 64, 64] [1024, 512, 256, 256, 128, 128, 64, 64]
Learning Rate	Learning rate for the optimizer	0.00001 - 0.5
Non Linearity	Non-linearity used for all network layers that required a non-linearity	One of the following: [ELU, ReLU]
Train Loss	Loss metric utilized during training	One of the following: [MSE, MAE]

Table A6: Point cloud model hyperparameter ranges

Parameter Name	Description	Range
Batch Size		2 - 8
N Inlet and Outlet Layers	Number of convolutional blocks at the top level of the UNet	1 - 3
Inlet and Outlet Conv Channel	Channelwise depth of convolutional layer at inlet and oulet convolution blocks	16 - 256
N blocks	Number of residual convolution blocks at each encoding/decoder layer of the UNet	1-2 (multi-decoder branch) 1-4 (single-decoder branch)
Encoder Channels (Single-Decoder Branch)	Array specifying the number of convolutional channels for the encoder layers at each level of the Unet	One of the following: [32, 32, 64, 64, 128, 128, 256, 256] [64, 64, 128, 128, 256, 256, 512, 512] [64, 64, 128, 128, 256, 256, 512, 1024]
Encoder Channels (Multi-Decoder Branch)	Array specifying the number of convolutional channels for the encoder layers at each level of the Unet	One of the following [32, 32, 64, 64, 128, 128, 256, 256] [64, 64, 128, 128, 256, 256, 512, 512] [64, 64, 128, 128, 256, 256, 512, 1024]
Decoder Channels (Single-Decoder Branch)	Array specifying the number of convolutional channels for the decoder layers at each level of the Unet	One of the following: [256, 256, 128, 128, 64, 64, 32, 32] [512, 512, 256, 256, 128, 128, 64, 64] [1024, 512, 256, 256, 128, 128, 64, 64]
Decoder Channels (Multi-Decoder Branch)	Array specifying the number of convolutional channels for the decoder layers at each level of the Unet	One of the following: [256, 256, 128, 128, 64, 64, 32, 32] [512, 512, 256, 256, 128, 128, 64, 64] [1024, 512, 256, 256, 128, 128, 64, 64]
N Centers	Number of Points to use in the ConvPoint Convolution	8-24
Learning Rate	Learning rate for the optimizer	0.00001 - 0.1
Non Linearity	Non-linearity used for all network layers that required a non-linearity	One of the following: [ELU, ReLU]
Train Loss	Loss metric utilized during training	One of the following: [MSE, MAE]

Table A7: Voxel model final hyperparameter set for training

Parameter Name	Single-Decoder Branch	Multi-Decoder Branch
Batch Size	8	4
N Inlet and Outlet Layers	3	1
Inlet and Outlet Conv Channel	96	96
N blocks	2	1
Encoder Channels	[32, 32, 64, 64, 128, 128, 256, 256]	[64, 64, 128, 128, 256, 256, 512, 512]
Decoder Channels	[256, 256, 128, 128, 64, 64, 32, 32]	[512, 512, 256, 256, 128, 128, 64, 64]
Learning Rate	0.0003	0.0003
Non Linearity	ELU	RELU
Train Loss	MAE	MAE

Table A8: Point cloud model final hyperparameter set for training

Parameter Name	Single-Decoder Branch	Multi-Decoder Branch
Batch Size	8	4
N Inlet and Outlet Layers	1	3
Inlet and Outlet Conv Channel	168	152
N blocks	4	2
Encoder Channels	[64, 64, 128, 128, 256, 256, 512, 1024]	[64, 64, 128, 128, 256, 256, 512, 1024]
Decoder Channels	[1024, 512, 256, 256, 128, 128, 64, 64]	[1024, 512, 256, 256, 128, 128, 64, 64]
N Centers	12	16
Learning Rate	0.00006	0.000025
Non Linearity	ReLU	ReLU
Train Loss	MAE	MAE

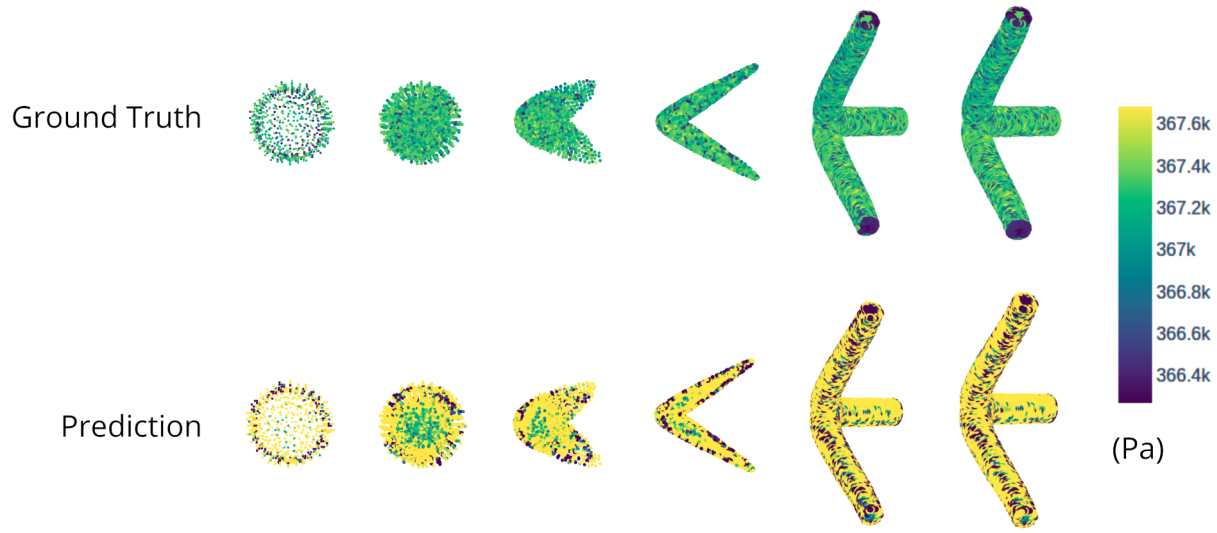


Figure A2: Voxel model pressure profile comparison with ground truth range as scale

REFERENCES

[1] Guo, Xiaoxiao, and Li, Wei, and Iorio, Francesco. "Convolutional Neural Networks for Steady Flow Approximation." *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. Association for Computing Machinery, 2016.

[2] Tompson, Jonathan, and Schlachter, Kristofer, and Sprechmann, Pablo, and Perlin, Ken. *"Accelerating Eulerian Fluid Simulation With Convolutional Networks"*. 2016.

[3] Bridson, Robert. "Fluid Simulation for Computer Graphics". 2nd ed. CRC Press, 2015.

[4] Ribeiro, Mateus D., and Rehman, Abdul and Ahmed, Sheraz, and Dengel, Andreas. "DeepCFD: Efficient Steady-State Laminar Flow Approximation with Deep Convolutional Neural Networks". 2021

[5] Baqué, Pierre, and Remelli, Edoardo, and Fleuret, François, and Fua, Pascal. "Geodesic Convolutional Shape Optimization". 2018.

[6] Pajaziti, Endrit, and Montalt-Tordera, Javier, and Capelli, Claudio, and Sivera Raphaël, and Sauvage, Emilie, and Quail, Michael, and Schievano, Silvia, and Muthurangu,Vivek "Shape-driven deep neural networks for fast acquisition of aortic 3D pressure and velocity flow fields." *PLoS computational biology* vol. 19,4 e1011055. 24 Apr. 2023, .

[7] Hennigh, Oliver. "Automated Design using Neural Networks and Gradient Descent". 2017.

[8] Harsch, Lukas, and Riedelbauch Stefan. "Direct Prediction of Steady-State Flow Fields in Meshed Domain with Graph Networks". 2021.

[9] Ozaki, Hiroto, and Aoyagi, Takeshi. "Prediction of steady flows passing fixed cylinders using deep learning". *Sci Rep 12, 447*, 2022.

[10] Le, Thi-Thu-Huong, and Kang, Hyoeun, and Kim, Howon. 2022. "Towards Incompressible Laminar Flow Estimation Based on Interpolated Feature Generation and Deep Learning." *Sustainability*, vol. 14, no. 19, Sept. 2022

[11] Wandel, Nils, and Weinmann, Michael, and Klein, Reinhard. "Fast Fluid Simulations in 3D with Physics-Informed Deep Learning". 2020.

[12] Pfaff, Tobias, and Fortunato, Meire, and Sanchez-Gonzalez, Alvaro, and Battaglia, Peter W. "Learning Mesh-Based Simulation with Graph Networks". 2021.

[13] Thuerey, Nils, and Weißenow, Konstantin, Prantl, Lukas, and Hu, Xiangyu. "Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows". *AIAA Journal, 58(1), 25–36,* 2020.

[14] Ummenhofer, Benjamin, and Prantl, Lukas, and Thuerey, Nils, and Koltun, Vladlen "Lagrangian Fluid Simulation with Continuous Convolutions". *In International Conference on Learning Representations*. 2020.

[15] Kashefi, Ali, and Rempe, Davis, and Guibas, Leonidas J. A point-cloud deep learning framework for prediction of fluid flow fields on irregular geometries. *Physics of Fluids*, 33(2), 2021.

[16] Kummerländer, Adrian, and Avis, Sam, and Kusumaatmaja, Halim, and Bukreev, Fedor, and Crocoll, Michael, and Dapelo, Davide, and Hafen, Nicolas, and Ito, Shota, and Jeßberger, Julius, and Marquardt, Jan E., and Mödl, Johanna, and Pertzel, Tim, and Prinz, František, and Raichle, Florian, and Schecher, Maximilian, and Simonis, Stephan, and Teutscher, Dennis, and Krause, Mathias J.. OpenLB Release 1.6: Open Source Lattice Boltzmann Code. 1.6, Zenodo, 2023, doi:10.5281/ZENODO.7773497.

[17] "OpenFOAM". v2306, ESI Group, 2023, openfoam.com.

[18] "Sailfish". Release 2012.2, Github, 2012, www.github.com/sailfish-team/sailfish/.

[19] "SU2". version 7.5.1, GitHub, 2023, www.su2code.github.io.

[20] Thuerey, Nils, and Tobias Pfaff. "Mantaflow". Version 0.1, 2017, mantaflow.com.

[21] NvidiaGameWorks. "FleX". Nvidia, Release 1.1.0, 2017, www.developer.nvidia.com/flex.

[22] Ansys, Inc. "Fluent". Ansys, Inc., Release 23 R1, 2023, www.ansys.com/products/fluids/ansys-fluent.

[23] Precise Simulation. "FEATool". Precise Simulation, Version 1.16.2, 2023, www.featool.com.

[24] COMSOL AB. "COMSOL Multiphysics". COMSOL AB, Version 6.1, 2022, www.comsol.com.

[25] Djadoudi, Hichem. "Ansys Fluent Expert Review, Pricing and Alternatives - 2023". WorQuick, 09 Aug. 2023, www.worquick.com/post/fluent_review.

[26] Gropp, William D., Kaushik, Dinesh K., Keyes, David E., & Smith, Barry. "Performance Modeling and Tuning of an Unstructured Mesh CFD Application". *In Proceedings of the 2000 ACM/IEEE Conference on Supercomputing* (pp. 34–es), IEEE Computer Society, 2000.

[27] Ansys, Inc. "HPC Solutions". Ansys, Inc., 10 Aug. 2023, www.ansys.com/it-solutions/hpc#tab1-3.

[28] CFD Online. "Price of Star-CD or Star-CCM+?". CFD Online, 11 Aug, 2023, www.cfd-online.com/Forums/siemens/56953-price-star-cd-star-ccm.html.

[29] Ansys Forum. "Limitations to Student License?". Ansys, Inc., 12 Aug,. 2023, www.forum.ansys.com/forums/topic/limitations-to-student-license.

[30] Siemens Community. "How faculty members in academic institutions can get access to Simcenter STAR-CCM+". Siemens, 12 Aug. 2023, www.community.sw.siemens.com/s/article/How-faculty-members-in-academic-institutions-can-get-access-to-Simcenter-STAR-CCM.

[31] Shellabear, Michael C., and Olli Juhani Nyrhilae. “DMLS - DEVELOPMENT HISTORY AND STATE OF THE ART”. 2004.

[32] Reynolds, Osborne. “An Experimental Investigation of the Circumstances Which Determine Whether the Motion of Water Shall Be Direct or Sinuous, and of the Law of Resistance in Parallel Channels”. Philosophical Transactions of the Royal Society of London, vol. 174, 1883, pp. 935–82. JSTOR, http://www.jstor.org/stable/109431. Accessed 16 Aug. 2023.

[33] Munson, Bruce. R., and Okiishi, Theodore. H., and Huebsch, Wade W., and Rothmayer, Alric P. *Fundamentals of fluid mechanics*. Wiley, 2012.

[34] Moody, Lewis F. "Friction Factors for Pipe Flow". Transactions of the ASME, vol. 66, 1944, pp. 671-684.

[35] Buckingham, Edgar. “On Physically Similar Systems; Illustrations of the Use of Dimensional Equations”. *Physical Review*, vol. 4, no. 4, 1 Oct. 1914, pp. 345–376.

[36] Bahrami, Majid. “Dimensional Analysis and Similarity”. ENSC 283 - Introduction to Fluid Mechanics, Simon Fraser University, www.sfu.ca/~mbahrami/ENSC%20283/Notes/Dimensional%20Analysis%20and%20Similarity.pdf. June 26, 2023.

[37] Clough, Ray W. "The Finite Element Method in Plane Stress Analysis". *Proceedings of the 2nd ASCE Conference on Electronic Computation*, 1960.

[38] Hestenes, Magnus R., and Stiefel, Eduard. "Methods of Conjugate Gradients for Solving Linear Systems". *Journal of Research of the National Bureau of Standards*, vol. 49, no. 6, 1952, pp. 409-436.

[39] Dassault Systemes. “SolidWorks”. SolidWorks 2023, Dassault Systemes, 2023.

[40] Dassault Systemes. “Catia”. R2013, Dassault Systemes, 2023.

[41] Braid, Ian C. "The Synthesis of Solids Bounded by Many Faces." *Communications of the ACM*, vol. 18, no. 10, 1975, pp. 209-216.

[42] Library of Congress. "STL (STereoLithography) File Format Family". Library of Congress, 12 Sep. 2019, www.loc.gov/preservation/digital/formats/fdd/fdd000504.shtml.

[43] Prescient Technologies. "Mesh Generation Algorithms”. Prescient Technologies, www.pre-scient.com/knowledge-center/product-development-by-reverse-engineering/meshing-algorithms.html. 13 Aug. 2023.

[44] Allison, Chloe. “Meshing in FEA: Structured vs Unstructured meshes”. OnScale, 1 Apr. 2020, www.onscale.com/meshing-in-fea-structured-vs-unstructured-meshes/.

[45] Ansys, Inc. “ANSYS Meshing User's Guide”. Ansys, Inc. Release 13.0, 2010.

[46] Joshi, Saumitra. “Re: How does a highly skewed cell cause convergence issue while solving equations numerically?”. ResearchGate, 26 Aug. 2014, www.researchgate.net/post/How_does_a_highly_skewed_cell_cause_convergence_issue_while_solving_equations_numerically.

[47] SimScale. "Why Do We Need Inflation Layers on Walls?". SimScale. 2020. www.simscale.com/knowledge-base/why-do-we-need-inflation-layers-on-walls.

[48] SimScale. “What is y+ (yplus)?”. SimScale, Nov 2020, https://www.simscale.com/forum/t/what-is-y-yplus/82394

[49] Ansys, Inc. “Ansys Fluent Theory Guide”. Ansys, Release 14.0, 2011

[50] NASA, “Uncertainty and Error in CFD Simulations”, NASA, 2021, www.grc.nasa.gov/www/wind/valid/tutorial/errors.html

[51] Skill Lync, “All About The Convergence Criteria”, Skill Lync, 2022, https://skill-lync.com/blogs/technical-blogs/cfd-all-about-the-convergence-criteria

[52] Diane Sofranec. "How simulation helps accelerate the design process" 3D CAD World, 3D CAD World, 17 Dec. 2015, https://www.3dcadworld.com/simulation-helps-accelerate-design-process/.

[53] Wikipedia, “Moody Chart”, Wikipedia, 13 Mar. 2023, https://en.wikipedia.org/wiki/Moody_chart

[54] Saumitra Joshi, “How does a highly skewed cell cause convergence issue while solving equations numerically”, ResearchGate, 26 Aug. 2014, https://www.researchgate.net/post/How_does_a_highly_skewed_cell_cause_convergence_issue_while_solving_equations_numerically.

[55] Ansys, Inc. “ANSYS Fluent Tutorial Guide”. Ansys, Inc., Release 13.0, 2009.

[56] Ansys, Inc. “Ansys SpaceClaim 3D Modeling Software”. Ansys Inc., www.ansys.com/products/3d-design/ansys-spaceclaim. 15 Aug. 2023.

[57] The Engineering ToolBox. “Liquids - Densities”. The Engineering ToolBox, www.engineeringtoolbox.com/liquids-densities-d_743.html, 22 Aug. 2023.

[58] Ridgid. “200 psi 4.5 Gal. Electric Quiet Compressor”. Emerson Electric Co, www.ridgid.com/us/en/electric-quiet-compressor, 22 Aug. 2023.

[59] Dixon Valve. “Viscosity Chart”. Dixon Valve, https://www.dixonvalve.com/sites/default/files/product/files/brochures-literature/viscosity%20chart.pdf, 22 Aug. 2023.

[60] Geuzaine, Christophe, and Remacle, Jean-Francoise. “*Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities*”. *International Journal for Numerical Methods in Engineering* 79(11), pp. 1309-1331, 2009.

[61] “meshio”. v0.1.6, Github, 2022, www.github.com/nschloe/meshio/.

[62] Turk, Greg. “The PLY Polygon File Format”. Internet Archive, https://web.archive.org/web/20161204152348/http://www.dcs.ed.ac.uk/teaching/cs4/www/graphics/Web/ply.html, 26, Aug. 2023.

[63] Van Rossum, Guido. “The Python Library Reference”. release 3.8.2. Python Software Foundation, 2020.

[64] Choy, Christopher, and Gwak, JunYoung, and Savarese, Silvio. “4D Spatio-Temporal ConvNets: Minkowski Convolutional Neural Networks.” *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[65] Boulch, Alexandre. “ConvPoint: Continuous Convolutions for Point Cloud” Processing. 2020.

[66] Akiba, Takuya, and Sano, Shotaro, and Tanase, Toshihiko, and Ohta, Takeru, and Koyama, Masanori. “Optuna: A Next-Generation Hyperparameter Optimization Framework”. 2019.

[67] FreeCAD Documentation. “Manual:Traditional modeling, the CSG way”. FreeCAD, https://wiki.freecad.org/Manual:Traditional_modeling,_the_CSG_way, 21 Oct. 2021.

[68] manchestercfd. “All there is to know about different mesh types in CFD!”. ManchesterCFD team | University of Manchester, https://www.manchestercfd.co.uk/post/all-there-is-to-know-about-different-mesh-types-in-cfd, 4 Sep. 2023.

[69] Wikipedia. “Entrance length (fluid dynamics)”. Wikimedia Foundation, 30 Jan. 2023, https://en.wikipedia.org/wiki/Entrance_length_(fluid_dynamics).

[70] “OpenSCAD”. OpenSCAD 2021.01, Github, 6 Feb. 2023, github.com/openscad/openscad/.

[71] “OCCT”. V7_7_2, Github, Aug. 2023, github.com/Open-Cascade-SAS/OCCT

[72] Ansys, Inc. “Ansys Fluent Theory Guide”. Ansys, Release 23.0, 2023

[73] Ansys, Inc. “Mechanical”. Ansys, Inc., Release 23 R1, 2023, www.ansys.com/products/structures/ansys-mechanical.

[74] Sofialidis, Dimitrios. “Fluid Flow and Heat Transfer in a Mixing Tee”, Industry Oriented HPC Simulations, PRACE Autumn School, 21-27 Sep. 2013, University of Ljubljana, Ljubljana, Slovenia. Lecture. events.prace-ri.eu/event/156/contributions/14/attachments/70/102/Fluent-Intro_14.5_WS01_Mixing_Tee.pdf.

[75] He, Kaiming, and Zhang, Xiangyu, and Ren, Shaoqing, and Sun, Jian. “Deep Residual Learning for Image Recognition.” *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[76] Apsley, David. "Friction Laws." University of Manchester, Spring 2009, Manchester. Slides.

[77] Swamee, Prabhata K., and Jain, Akalank K. “Explicit Equations for Pipe Flow Problems”. *Journal of the Hydraulics Division,* 102, 657-664. 1976.

[78] QUESTIONS AND ANSWERS IN MRI. “Classification of Flow”. AD Elster, ELSTER LLC, mriquestions.com/laminar-v-turbulent.html, 10 Sep. 2023.

[79] Xiang, Xu, and Willis, Karl D.D., and Lambourne, Joseph G, and Cheng, Chin-Yi, and Kumar, Pradeep J., and Furukawa, Yasutaka. "SkexGen: Autoregressive Generation of CAD Construction Sequences with Disentangled Codebooks." *Proceedings of the 39th International Conference on Machine Learning*. PMLR, 2022.