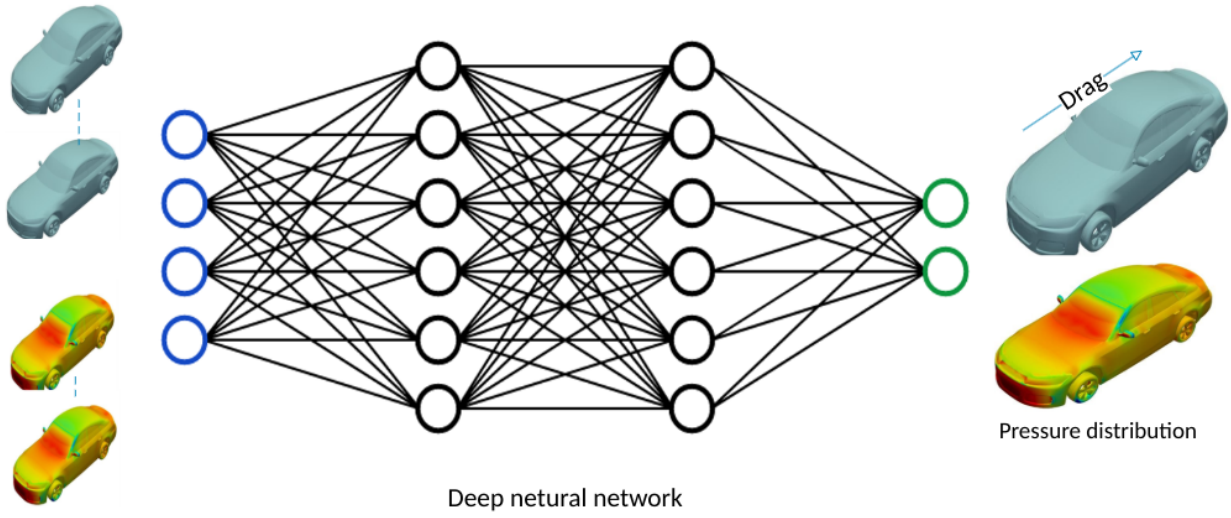
CHALMERS
UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Deep learning for fast aerodynamic estimation of road vehicles

Master's thesis in Computer science and engineering

SHAN SREENIVAS
SANJAYA RAJAMANTRI

Department of Mechanics and Maritime Sciences
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2025

Master's thesis 2025

Deep learning for fast aerodynamic estimation of road vehicles

SHAN SREENIVAS
SANJAYA THILAK BANDARA ASURASINGHE RAJAMANTRILAGE

UNIVERSITY OF GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanics and Maritime Sciences
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2025

Deep learning for fast aerodynamic estimation of road vehicles

SHAN SREENIVAS
SANJAYA THILAK BANDARA ASURASINGHE RAJAMANTRILAGE

Supervisor: Chao Xia(chao.xia@chalmers.se), Mechanics and Maritime Sciences
Examiner: Alexey Vdovin(alexey.vdovin@chalmers.se), Mechanics and Maritime Sciences

Master's Thesis 2025
Department of Mechanics and Maritime Sciences
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Drag and Pressure field prediction using deep learning

Typeset in LaTeX
Gothenburg, Sweden 2025

Deep learning for fast aerodynamic estimation of road vehicles
SHAN SREENIVAS
SANJAYA THILAK BANDARA ASURASINGHE RAJAMANTRILAGE
Department of Mechanics and Maritime Sciences
Chalmers University of Technology and University of Gothenburg

Abstract

The high computational cost and long runtimes of traditional evaluation methods often slow automotive aerodynamic design. Computational Fluid Dynamics (CFD) simulations and wind tunnel tests, while accurate, are resource-intensive and impractical for real-time feedback during iterative design. This thesis addresses the need for faster aerodynamic estimation by developing deep learning-based surrogate models for predicting aerodynamic quantities directly from 3D geometry.

Specifically, the performance of PointNet and Geometry-Informed Neural Operators (GINO) is evaluated for predicting global drag coefficients (C_d) and local pressure distributions over complex automotive geometries. Using the DrivAerNet dataset, systematic experiments investigate the influence of total sample size, point cloud resolution, batch size, and hyperparameters on predictive accuracy.

Results demonstrate that PointNet achieves strong drag prediction performance, reaching an R^2 of 0.957, with a mean error percentage of approximately 1.6% and a maximum error percentage of under 8% for unseen data of around 3500 samples, when the model is trained with 400 samples which is 80% of 500 total samples, 100,000 vertices per sample, and a batch size of 16. However, PointNet shows limited sensitivity to training variations in pressure prediction, with Rel L2 errors consistently within the range of 0.35–0.37. In contrast, GINO significantly outperforms PointNet in pressure prediction tasks, achieving a test R^2 of 0.873, Rel L2 errors below 0.28, and demonstrating robust data efficiency and sensitivity to latent space configurations.

This study establishes a rigorous baseline for deep learning-driven aerodynamic prediction, highlighting the suitability of PointNet for global scalar quantities and the potential of GINO for accurate field-level predictions. The findings support the future development of hybrid models for fast, data-driven aerodynamic design optimization in the automotive industry.

Keywords: Deep learning, PointNet, GINO, DrivAerNet, Automotive aerodynamics.

Acknowledgements

Heartfelt gratitude to Chao Xia for his invaluable supervision and guidance, and for consistently providing direction throughout this thesis. His expertise and insightful feedback played a vital role in shaping this work.

Sincere thanks to Alexey Vdovin for his generous support and for providing a deeper understanding of the fundamentals of aerodynamics, which greatly contributed to the clarity and strength of this research.

Contents

List of Figures

List of Tables

1

Introduction

The aerodynamic design of cars is a critical factor in vehicle performance, as aerodynamic drag significantly affects fuel efficiency, battery range, and overall energy consumption. Traditionally, simulations of computational fluid dynamics (CFD) and wind tunnel testing have been the primary methods for evaluating aerodynamic properties[1]. Although CFD offers detailed and accurate insights, it demands substantial computational resources and time. A single industrial-grade CFD simulation of a complete vehicle can exceed 10^8 control volumes and require several days or even weeks of computation, limiting its practicality for iterative design and real-time applications. Wind tunnel testing is also expensive to build and operate, as well as time-consuming.

These challenges underscore the need for alternative approaches that deliver accurate aerodynamic predictions in seconds rather than days, thereby enabling rapid design iterations and supporting data-driven engineering workflows. Deep learning has emerged as a promising solution for learning from large datasets generated by computational fluid dynamics (CFD) simulations or experiments. By approximating aerodynamic quantities at significantly lower computational costs, deep learning models enable fast predictions, accelerate design cycles, and expand the range of design options available within tight timeframes.

This thesis addresses the challenge of developing a deep learning-based surrogate model for the rapid prediction of aerodynamic characteristics of road vehicles, utilizing computational fluid dynamics (CFD) simulation data from the publicly available DrivAerNet [2] dataset. DrivAerNet provides industry-scale automotive geometries and simulation results, offering a unique opportunity to train models in realistic and complex aerodynamic scenarios.

The proposed approach investigates two state-of-the-art geometric deep learning architectures PointNet[3] and Geometry-Informed Neural Operators (GINO)[4] for predicting key aerodynamic metrics, including the drag coefficient (C_d) and surface pressure distributions. By systematically studying the impact of model architecture, total sample size, geometric resolution, and hyperparameters on predictive accuracy, the goal is to develop a robust and data-efficient model that can provide aerodynamic predictions directly from 3D vehicle geometry in a fraction of the time required for traditional CFD simulations, thereby supporting faster design iterations and informed decision making in automotive engineering. In addition, this investigation aims to provide practical guidelines for future researchers and practitioners on how

to prepare and structure datasets, select suitable model architectures, and balance data quantity with model complexity when using deep learning for aerodynamic prediction tasks.

2

Background

The development of data-driven methods for aerodynamic prediction has been driven by the need to overcome the computational limitations of traditional Computational Fluid Dynamics (CFD) simulations. This chapter reviews key advancements in the field, beginning with a timeline of major deep-learning models and datasets that have shaped the landscape of aerodynamic prediction. It then categorizes state-of-the-art models by their underlying architectures, ranging from reduced order models and convolutional networks to graph-based and point cloud methods, and highlights their strengths and limitations. The chapter further examines the role of geometric representations, including point clouds and mesh-based formats, in enabling efficient learning from 3D vehicle shapes. Publicly available aerodynamic datasets are summarized, outlining their scale, diversity, and relevance for model training and evaluation. Finally, the chapter introduces commonly used evaluation metrics that quantify model performance in predicting aerodynamic properties such as drag coefficients and surface pressure fields. Together, these elements provide a comprehensive foundation for understanding the current state of data-driven aerodynamic prediction.

2.1 Related work

2.1.1 Timeline of the development of deep learning methods

To understand the current state of machine learning in aerodynamics, it is essential to examine the historical progression of methodologies and resources. Figure 2.1 illustrates the evolution of machine learning techniques for aerodynamic applications, highlighting key architectures and datasets. Early models such as Reduced Order Models and UNet laid the foundation, while PointNet and MeshGraphNets introduced point cloud and graph-based processing. Neural operators, such as FNO and GINO, further extended the capabilities by learning mappings over continuous domains. The timeline also marks the public availability of essential datasets, such as DrivAerNet++ and DrivAerML, which support scalable and high-fidelity aerodynamic modeling.

These advances in datasets and model architectures have paved the way for the development of data-driven approaches that can directly predict aerodynamic quantities from geometry, offering a significant reduction in computational time compared

to traditional CFD methods. The following section provides an overview of latest data-driven models and their key design principles.

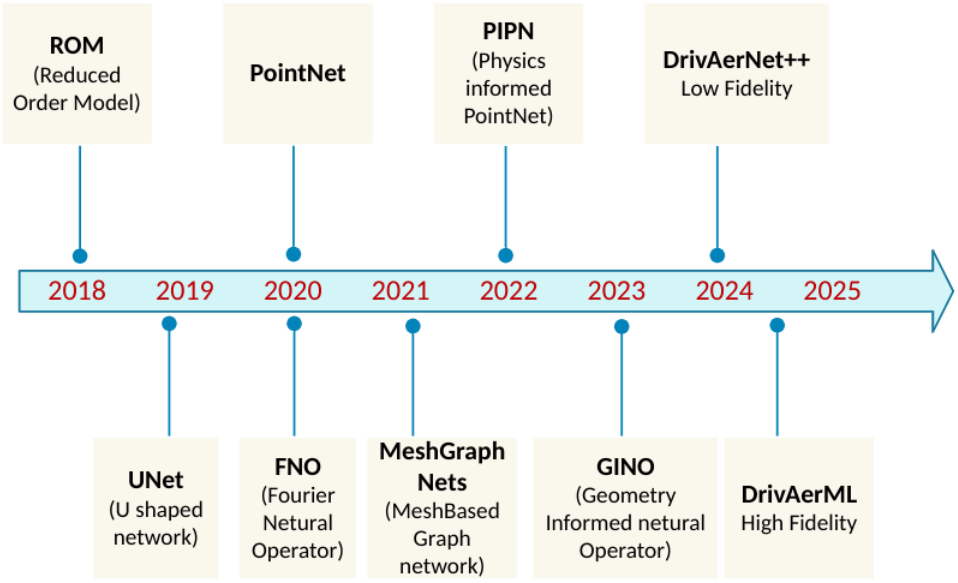


Figure 2.1: Timeline of key machine learning models and datasets for aerodynamic prediction.

2.1.2 Data Driven Models

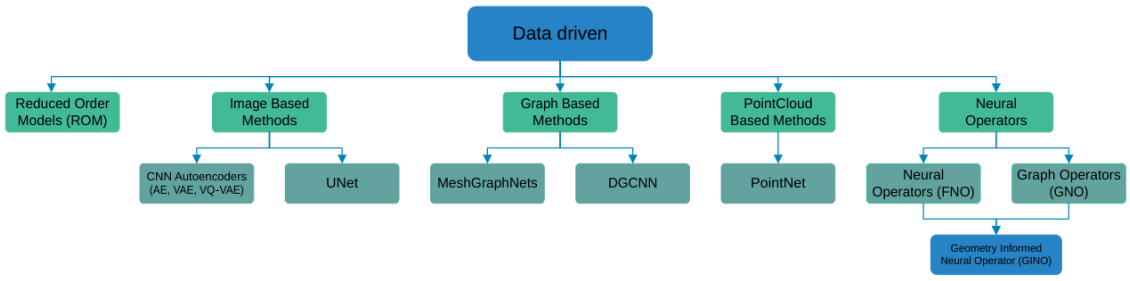


Figure 2.2: Data-driven models for aerodynamic prediction.

The field of data-driven aerodynamic modeling has evolved to encompass diverse architectural approaches, each offering unique advantages for different prediction tasks and computational constraints. Figure 2.2 presents data-driven models for aerodynamic prediction, illustrating the landscape of available architectures and their interconnections. The diagram arranges models into several major categories:

- **Reduced Order Models (ROMs)** [5] This proposes a datadriven multifidelity reduced-order modeling (ROM) framework that marries Proper Orthogonal Decomposition (POD) with Kriging-based response surface models to enable real-time aerodynamic predictions during vehicle styling iterations. By

integrating high-fidelity Detached-Eddy Simulations (DES) and lower-fidelity Reynolds-Averaged NavierStokes (RANS) data, the model effectively interpolates aerodynamic quantitiessuch as drag coefficient, pressure distributions, and wall-shear stressacross the design space. Their findings, based on a real-world passenger car case, demonstrate that using a multi-fidelity ROM achieves the same accuracy as high-fidelity CFD but with a significantly lower computational cost. It performs better than single-fidelity models and provides fast aerodynamic feedback.

- **Image Based Methods**. These approaches utilize convolutional neural networks (CNNs) to interpret flow fields, such as pressure and velocity, as structured grid data. Convolutional autoencoders (CAEs) and variants (VAEs, VQVAEs) enable dimensionality reduction and accurate reconstruction. For example, an encoder-decoder trained on over 11,000 airfoil simulations generalized well, with only 12% performance loss under moderate geometric variations [6]. CNNs have also been used to predict complete RANS flow fields around unseen airfoils orders of magnitude faster than traditional solvers [7]. Extensions that combine CAE with convolutional LSTM (ConvLSTM) capture the spatio-temporal features of 3D turbulence [8]. However, these methods are constrained to structured grids and struggle with complex, unstructured geometries.

- **Neural Operators (FNO & GINO)**. Neural operators learn mappings between infinite dimensional partial differential equations solution spaces, making them ideal for CFD applications involving complex fluid dynamics. The Fourier Neural Operator (FNO) parameterized integral kernel operations in Fourier (spectral) space, enabling resolution and mesh invariance. Building upon this, the Geometry-Informed Neural Operator (GINO) integrates graph-based geometric encoding using signed distance functions and point cloud representations with Fourier based operators to tackle irregular, unstructured domains. GINO accurately predicts surface pressure and drag coefficients on 3D vehicle geometries using only 500 training examples, achieving a remarkable 26,000 times speedup over optimized GPU-based CFD and reducing error relative to standard deep networks[4].

- **Graph Based Methods**. These techniques embed CFD meshes into graph neural networks, modeling mesh vertices as nodes and spatial adjacencies as edges. MeshGraphNets introduced dynamic message passing on mesh graphs, enabling resolution independent flow simulation with 10, 100 times speedups over traditional solvers [9]. An evolution of this paradigm is demonstrated in DrivAerNet / RegDGCNN, which applies a dynamic graph convolutional architecture to high-resolution 3D car meshes. This model directly predicts surface pressure distributions and aerodynamic drag in seconds, bypassing precomputed Signed Distance Fields, and achieves high accuracy on complex automotive geometries using only a few thousand samples [2].

- **PointCloud Based Methods**. These models operate on unordered point sets representing CFD domains, such as geometry surfaces or volumetric mesh

vertices, enabling permutation invariance and avoiding the need for structured grids [10]. A key challenge with basic PointNet is its difficulty in capturing local geometric context. PointNet++ addresses this issue via hierarchical neighborhood grouping and local feature aggregation, thereby enhancing accuracy on irregular domains. Although a direct CFD application of PointNet++ remains limited, its improved local receptive field has been leveraged in recent operator learning architectures to model spatial flow patterns in complex geometries better.

These data-driven aerodynamic prediction methods differ primarily in their geometric representation of of point clouds, graphs, or structured grids and in their learning architecture, such as neural operators or convolutional networks. Point cloud based models (e.g., PointNet) maintain permutation invariance and enable direct processing of 3D surface geometries without meshing. Graph-based methods (e.g., DGCNN, MeshGraphNets) extend this by incorporating mesh topology via nodes and edges, capturing local geometric relationships and improving prediction fidelity. Neural operators particularly Fourier based ones are adept at modeling global, long range dependencies and offer mesh and resolution independence. Choosing between these representations and architectures depends on the specific aerodynamic task surface pressure prediction, drag estimation, or full field flow reconstruction and involves balancing model accuracy, computational cost, and domain generalizability.

2.1.3 Drag Prediction previous studies

Table 2.1: Drag Prediction Performance on DrivAerNet Dataset

Model	**Mean SE** ($\times 10^{-5}$) ($\downarrow$)	**Mean AE** ($\times 10^{-3}$) ($\downarrow$)	**Max AE** ($\times 10^{-2}$) ($\downarrow$)	$\mathbf{R^2}$ ($\uparrow$)	**Time (sec)** ($\downarrow$)
PointNet++ [11]	7.813	6.755	3.463	0.896	0.200
DeepGCN [12]	6.297	6.091	3.070	0.916	0.151
MeshGraphNet [13]	6.000	6.080	2.965	0.917	0.250
AssaNet [14]	5.433	5.810	2.390	0.927	0.110
PointNeXt [15]	4.577	5.200	2.410	**0.939**	0.239
PointBERT [16]	6.334	6.204	2.767	0.915	0.163
DrivAerNet DGCNN [2]	8.000	6.910	**0.880**	0.901	0.520

Machine learning approaches to drag coefficient estimation have matured considerably, with contemporary models achieving near-CFD accuracy at a fraction of the computational cost. Table 2.1 summarizes the performance of leading point cloud and graph-based models on the DrivAerNet dataset. Among these, PointNeXt and AssaNet achieved the best trade-off between accuracy and inference time, with PointNeXt delivering the lowest mean squared error (MSE) of 4.577×10^{-5} and an R^2 score of 0.939. AssaNet, in turn, demonstrated fast inference at 0.110 seconds while maintaining competitive accuracy. Graph-based models such as DeepGCN and MeshGraphNet also performed strongly, confirming the benefits of incorporating relational structures in aerodynamic prediction tasks. Recent advancements, such as hybrid approaches like FIGConvNet [17], have further advanced the field. Additionally, DoMINO (Decomposable Multi-scale Iterative Neural Operator) presents

a scalable, local, pointcloud-based framework that learns multi-scale geometric encodings and dynamically constructs computational stencils to predict both surface and volumetric flow fields on DrivAerML vehicle datasets [18]. This architecture achieves fast, mesh-independent aerodynamic inference while demonstrating strong in-distribution and out-of-distribution generalization across engineering-specific metrics such as drag, pressure distribution, and wall-shear stress.

These advances in data-driven modeling rely on the availability of high-quality aerodynamic datasets, which are reviewed in the following section.

2.2 Datasets for Aerodynamic Prediction

Accurate aerodynamic prediction relies on datasets that combine 3D vehicle geometries with CFD-generated flow fields. This section outlines key publicly available datasets and their underlying geometric representations.

2.2.1 CFD Methods for Automotive Aerodynamics

Computational Fluid Dynamics (CFD) methods differ in their balance between accuracy, resolution, and computational cost. Techniques such as Large-Eddy Simulation (LES) [19] and Delayed Detached Eddy Simulation (DDES) [20] solve the Navier-Stokes equations on fine meshes with explicit turbulence modeling, resolving small-scale vortex structures, steep pressure gradients, and transient flow behavior. These simulations are computationally intensive, often requiring thousands of CPU hours for a single case. Simpler approaches, such as Reynolds-Averaged Navier-Stokes (RANS) solvers or potential flow models, reduce this computational burden by introducing modeling assumptions, but at the cost of lower accuracy in capturing complex flow phenomena. This trade-off between computational efficiency and accuracy guides the choice of CFD methods depending on the design stage and application requirements.

Regardless of the method used, CFD simulations rely on accurate geometric representations of the vehicle. The following section describes standard file formats and mesh representations used in automotive aerodynamics.

2.2.2 Geometric Representations and File Formats

CFD geometries are typically represented as meshes that discretize the surface or volume:

- **STL (Stereolithography):** Surface meshes composed of non-overlapping triangles, widely used in CAD and preprocessing. Datasets such as AhmedML and WindsorML provide vehicle geometries in STL format. (see Figure 2.3).
- **VTP (VTK PolyData):** Extends STL by storing scalar and vector fields (e.g., pressure, velocity) alongside geometry, facilitating post-processing in tools like ParaView. The DrivAerNet dataset stores surface pressure and velocity fields in VTP files.

- **VTK/VTU:** Supports volumetric meshes and CFD solution fields on nodes or cells, enabling full 3D data storage [21]. DrivAerML, for example, includes high-fidelity volumetric flow fields stored in VTK/VTU format. (see Figure 2.4).

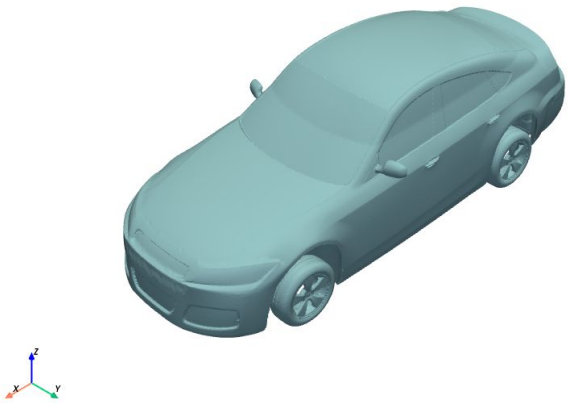


Figure 2.3: Example STL surface mesh of a vehicle geometry, showing only the discretized surface without field data.

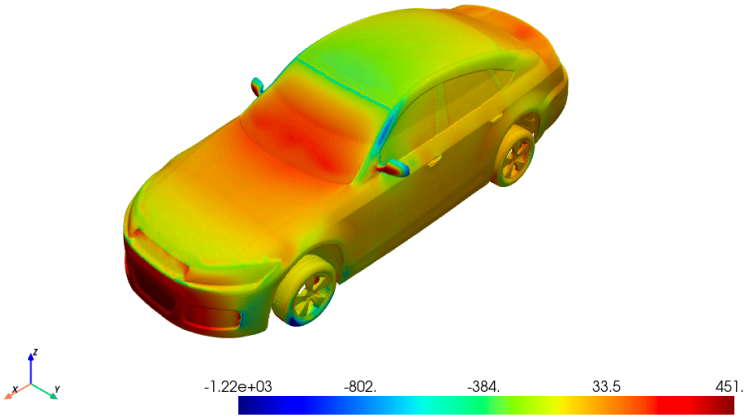
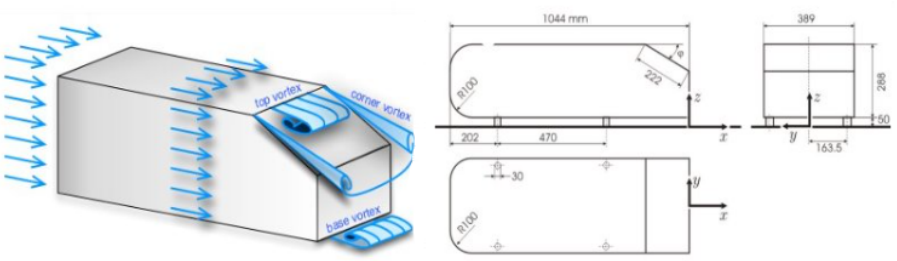


Figure 2.4: VTK visualization of pressure distribution on a car body surface. The color map represents mean pressure values from CFD simulation.

2.2.3 Publicly Available Datasets for Aerodynamic Prediction

AhmedML [22] includes 500 variants of the Ahmed body, a simplified car-like shape, CFD results using a hybrid RANS-LES solver. This dataset supports studies on flow separation and bluff body aerodynamics.

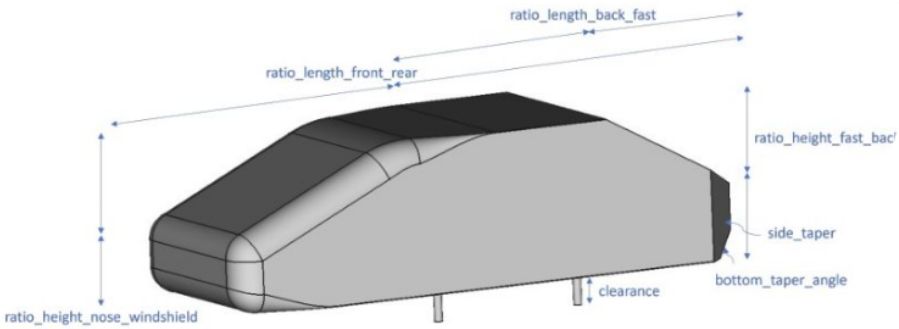


Geometry variants of the Ahmed car body

Part		
Variable	Min	Max
angle of rear-window (degrees)	10	60
tilted surface length (mm)	150	250
length of the body (mm)	800	1200
height of the body (mm)	250	315
width of the body (mm)	300	500
radius of the front body (mm)	80	120

Figure 2.5: AhmedML datasets with parameter details.[22]

WindsorML [23] provides 355 parameterized Windsor body designs CFD results from Wall-Modeled LES. It bridges academic and industrial geometries, offering higher realism than AhmedML.



Geometry variants of the Windsor body

Variable	Min	Max
ratio_length_front_rear	0	0.8
ratio_length_back_fast	0.08	0.5
ratio_height_nose_windshield	0.3	0.7
ratio_height_fast_back	0	0.9
side_taper [mm]	50	100
clearance [mm]	10	200
bottom_taper_angle [°]	1	50

Figure 2.6: WindsorML datasets with parameters.[23]

DrivAerML [24] features 500 DrivAer notchback designs with DDES simulations. Each case includes surface pressures and volumetric flow fields, with meshes exceeding 160 million cells.

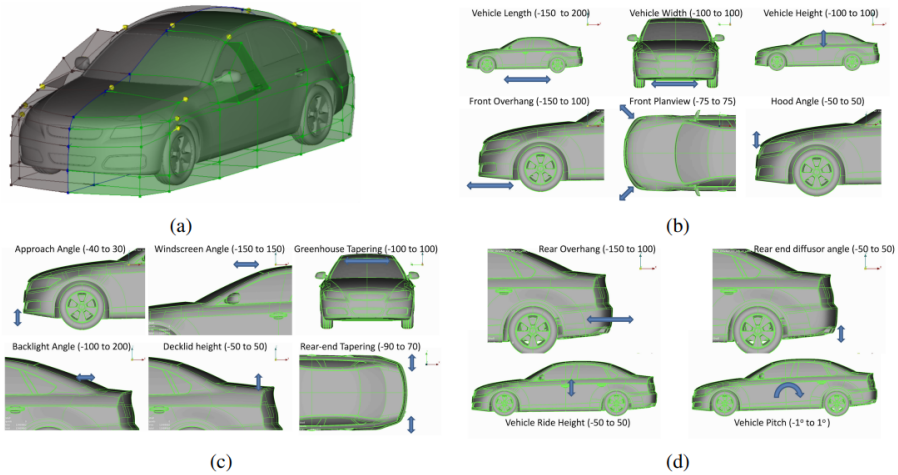


Figure 2.7: DrivAerML datasets with parameters.[24]

DrivAerNet [2] comprises 4000 DrivAer-based geometries generated via systematic parameter variation. RANS simulations provide pressure and velocity fields, supporting large-scale surrogate modeling.

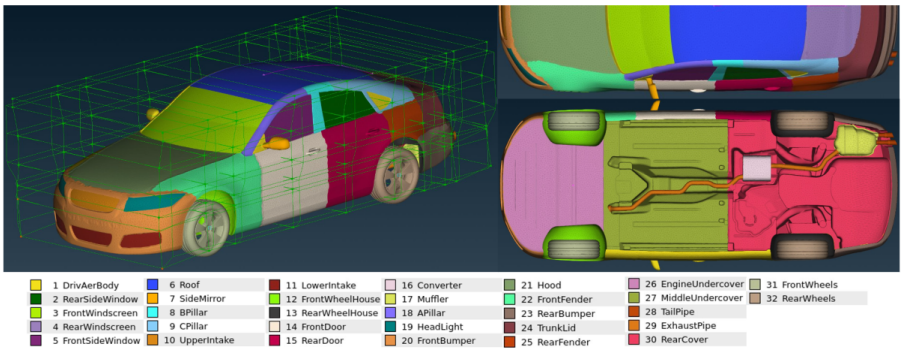


Figure 2.8: DriverNet datasets with parameter details.[2]

These datasets present a trade-off between fidelity, scale, and geometric complexity, providing complementary resources for training and evaluating machine learning models in automotive aerodynamics.

Table 2.2: Summary of Publicly Available CFD Datasets for Vehicle Aerodynamics

Dataset	**Number of Designs**	**Volume Mesh Size**	**Surface Mesh Size**	**Total Data Volume**
AhmedML	500	∼20M	∼150k	∼2 TB
WindsorML	355	-	-	∼8 TB
DrivAerML	500	∼160M	∼750k	∼31 TB
DrivAerNet	4000	∼24M	500-750k	∼16 TB

These datasets underpin the development of deep learning models that predict aerodynamic properties directly from 3D vehicle geometries. The next section provides

an overview of such models and their architectural characteristics.

While model architectures and geometric representations provide the foundations for learning, it is essential to establish robust evaluation metrics to quantify model performance. These metrics assess how accurately the model predicts aerodynamic quantities such as drag coefficients and surface pressure fields, guiding model selection and comparison. The following section outlines the primary metrics used for evaluating aerodynamic prediction models.

2.3 Evaluation Metrics

The performance of prediction models is commonly evaluated using:

- **Mean Squared Error (MSE)** measures the average squared difference between predicted and true values

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2$$

  MSE heavily penalizes large errors, making it sensitive to outliers but effective for identifying poor predictions.

- **Mean Absolute Error (MAE)** computes the average absolute difference between predicted and true values.

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

  MAE provides interpretable error measurements in the same units as the target variable, offering intuitive understanding of prediction accuracy.

- **Coefficient of Determination (R^2)** evaluates the proportion of variance in the true values explained by the model predictions. Values approaching 1 indicate excellent predictive capability, while values near 0 suggest performance no better than predicting the mean.

$$R^2 = 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2}$$

  Here, y_i are true values, $\hat{y}_i$ are predictions, and $\bar{y}$ is the mean of true values.

- **Relative error(Rel L2)** assesses the normalized difference between predicted and true pressure fields across all surface points. This metric proves particularly relevant for evaluating pressure field predictions where spatial distribution matters as much as point-wise accuracy.

$$\text{Relative } L2 = \frac{\|y - \hat{y}\|_2}{\|y\|_2} = \frac{\sqrt{\sum_i (y_i - \hat{y}_i)^2}}{\sqrt{\sum_i y_i^2}}$$

  A lower relative L2 value indicates better predictive accuracy of pressure fields.

3

Methods

This chapter presents the methodology employed for developing a deep learning framework to predict aerodynamic properties, such as drag coefficients and pressure distributions, directly from 3D vehicle geometries. Based on the analysis of datasets and model architectures in the theory chapter, this work focuses on the **DrivAerNet** dataset for its scale and variability. DrivAerNet provides over 4,000 parameterized car shapes, offering a large and diverse collection of geometries that enable robust training and generalization across different aerodynamic designs. The high-volume, low-fidelity nature of DrivAerNet also facilitates the exploration of data-driven surrogate models that can generalize to unseen shapes.

For the modeling task, two complementary architectures are selected: **PointNet** and **GINO**. PointNet is a simple, lightweight architecture designed to process unordered point clouds directly, making it well-suited for global property prediction such as drag coefficients. Its efficiency and directness make it a strong baseline for aerodynamic prediction tasks. In contrast, **GINO (Geometry-Informed Neural Operator)** offers an enhanced capability by combining graph neural networks and Fourier layers to capture both local geometric relationships and global flow interactions. GINO is particularly well suited for predicting high-resolution surface pressure distributions from geometric data, which can subsequently be used to estimate drag coefficients.

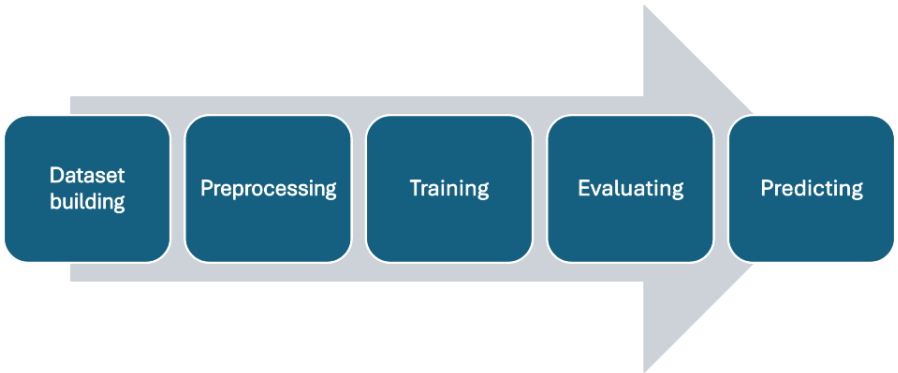


Figure 3.1: Machine learning pipeline for aerodynamic property prediction from vehicle geometries

The research methodology follows a systematic machine learning workflow as con-

ceptually depicted in Figure 3.1. The publicly available **DriveAerNet**[24] dataset is used here in the first stage of dataset building. The workflow then proceeds through data pre-processing to normalize geometries and extract features from vehicle meshes. The training phase optimizes deep neural network parameters using geometric inputs and ground-truth aerodynamic values, followed by evaluation using validation metrics. Finally, the prediction phase applies trained models to estimate drag coefficients and pressure distributions for unseen vehicle geometries. The subsequent sections provide a detailed explanation of the data processing pipelines, architectural implementations, and training configurations developed for each methodological approach.

3.1 PointNet

3.1.1 Data Preprocessing

Figure 3.2: Data preprocessing pipeline

The preprocessing pipeline consists of point cloud generation, augmentation, normalization, and splitting, as illustrated in Figure 3.2.

The first step involves converting the STL 3D surface mesh of each car into a point cloud by uniformly sampling points across the mesh surface. As illustrated in Figure 3.3, this process reduces the original high-resolution mesh (208,490 vertices) to a fixed-size point cloud (100,000 points), striking a balance between computational efficiency and the preservation of geometric detail.

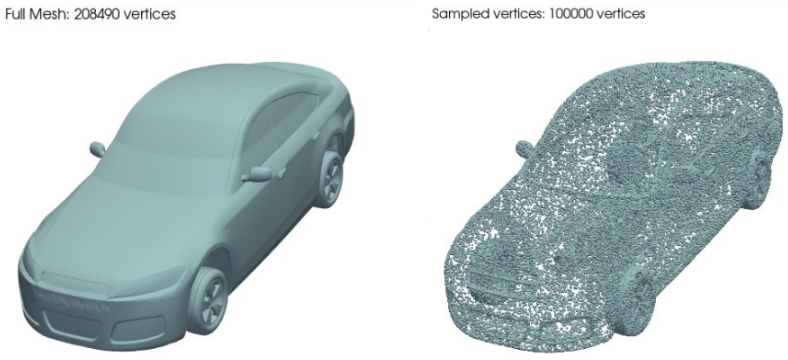


Figure 3.3: Point cloud generation from 3D vehicle mesh. Left: Original mesh. Right: Uniformly sampled point cloud.

Figure 3.4 shows the augmenataion and normalized point cloud which can be use as the input to the model. Here, to enhance model robustness and generalization,

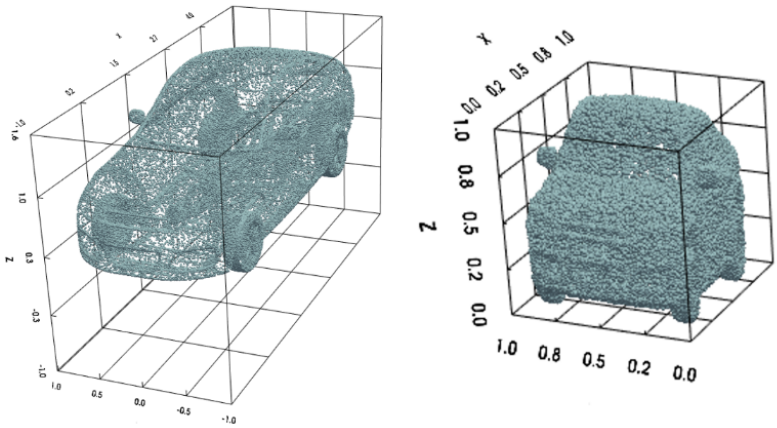


Figure 3.4: Left: Augmented mesh. Right: Normalized point cloud.

several data augmentation strategies are applied during training. These include random translations to simulate small shifts in geometry, Gaussian noise perturbations (jittering) to model measurement or sampling variations, and random point dropping to mimic sparse or incomplete point clouds. Such augmentations introduce variability in the training data, enabling the PointNet model to learn invariant features and improve its predictive performance on unseen geometries.

To ensure consistent geometric scales across all samples, each point cloud undergoes min-max normalization[25]. This technique rescales the data so that each feature lies within the range $[0, 1]$, using the formula:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

where x represents the original coordinate values, and x' denotes the normalized values. This normalization facilitates stable learning and ensures that input distributions remain comparable across the dataset.

After normalization, the dataset is partitioned into 80% training, 10% validation, and 10% test sets, based on the total number of samples. This ensures sufficient data for model training, hyperparameter tuning, and final evaluation.

3.1.2 Training

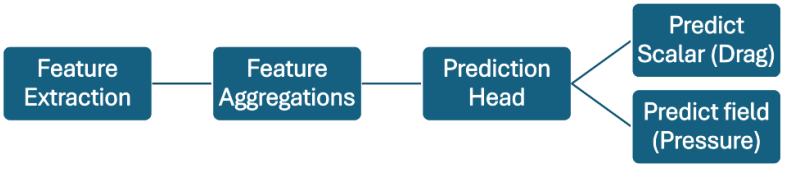


Figure 3.5: Simplified architecture of the PointNet model for aerodynamic property prediction.

Figure 3.5 illustrates the high-level structure of the PointNet model. The model consists of three main stages: Feature Extraction, where shared MLP layers learn point-wise features; Feature Aggregation, where global max pooling combines features into a global vector; and the Prediction Head, which maps the global feature vector to target outputs. Depending on the task, the network either predicts a scalar value, such as the drag coefficient (C_D), or a per point field, such as the pressure distribution.

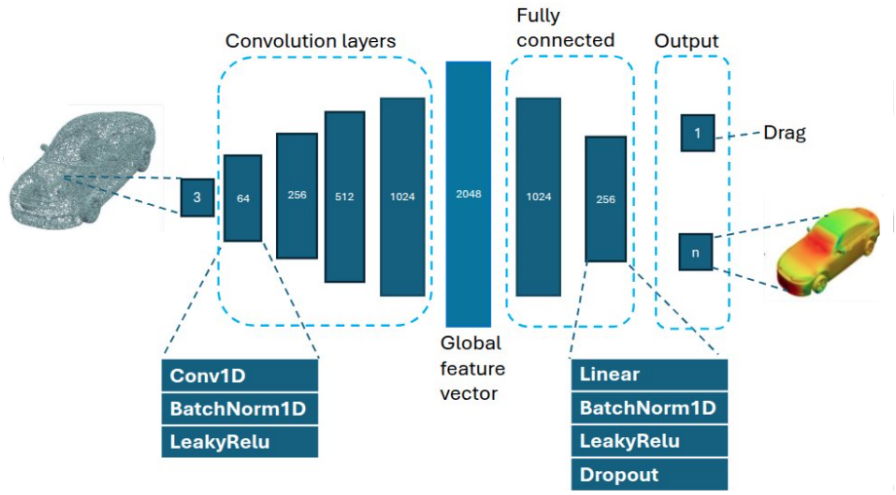


Figure 3.6: PointNet architecture for predicting drag coefficient and pressure from point clouds.

Figure 3.6 illustrates the detailed architecture of the PointNet model employed in this study for aerodynamic prediction. The network consists of two main components: convolutional layers for point-wise feature extraction, and fully connected layers for mapping global features to target aerodynamic properties.

The **convolutional layers** process the input point cloud, which consists of 3D coordinates (x, y, z), using a sequence of 1D convolutional layers. Each convolutional layer applies a shared linear transformation across all points, progressively increasing the feature dimensionality from 3 to 64, 256, 512, and up to 1024. Each convolution is followed by a batch normalization (BatchNorm1D) layer and a leaky ReLU activation function. This structure enables the network to learn complex geometric features from the point cloud while ensuring stable gradients and efficient convergence.

The extracted point-wise features are aggregated into a **global feature vector** using a max-pooling operation across all points, resulting in a fixed-length vector of dimension 2048. This vector captures a comprehensive representation of the input geometry and serves as the basis for subsequent predictions.

The **fully connected layers** take the global feature vector as input and map it to the target aerodynamic output, such as drag coefficient (C_D) or per-point pressure field. This is usually called as **Multilayer perceptron**[26]. The layers progressively reduce the dimensionality (e.g., 2048 to 1024, then 256, and finally 1 or n), with

each intermediate layer followed by batch normalization, leaky ReLU activation, and dropout regularization to improve generalization and prevent overfitting. The final output layer produces a scalar value for drag prediction or a field of values for pressure prediction, depending on the task configuration.

This architecture enables PointNet to efficiently extract and aggregate geometric information from point clouds, making it a suitable and lightweight baseline model for aerodynamic prediction tasks.

3.1.3 Evaluation

This model is trained to minimize the Mean Squared Error (MSE) loss between predicted and ground-truth aerodynamic values. The performance of the models is evaluated using established metrics: for drag prediction, the Coefficient of Determination (R^2) and MSE are reported, while for pressure field prediction, both MSE and relative L_2 error are computed to assess spatial accuracy.

3.2 Geometry Informed Neural Operator (GINO)

3.2.1 Data Preparation and Preprocessing

3.2.1.1 Mesh Processing and Standardization

To ensure consistency across diverse vehicle geometries, the surface meshes undergo several processing steps. The data processing of a vtk file is shown in the figure 3.7.

- **Triangulation**: All mesh elements are converted to triangular faces to guarantee uniform element types throughout the dataset. This standardization simplifies downstream processing and ensures compatibility with the graph construction algorithms.
- **Mesh Decimation**: A topology-preserving decimation algorithm reduces mesh complexity while maintaining essential geometric features. The reduction factor strikes a balance between computational efficiency and geometric fidelity, employing feature-angle preservation to retain sharp edges and surface discontinuities that are critical for aerodynamic analysis.
- **Regularization**: To address variable mesh densities across different vehicle models, surface data is resampled onto a uniform grid using nearest-neighbor interpolation. This regularization maps irregular vertex distributions to a consistent grid structure, enabling batch processing while preserving the spatial distribution of flow quantities.

3.2.1.2 Geometric Representations

The methodology generates two complementary geometric representations for each vehicle:

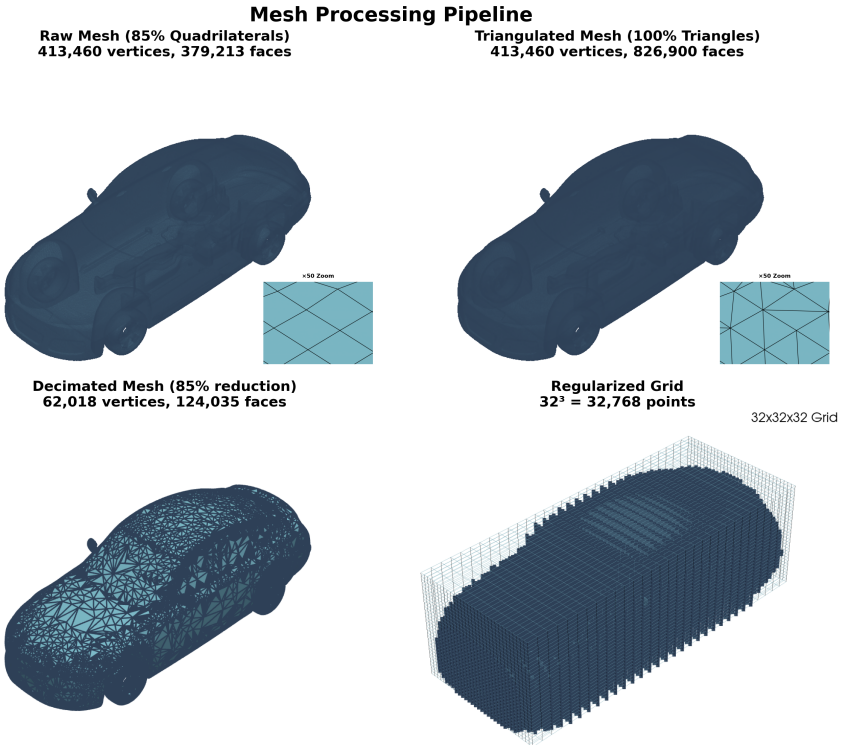


Figure 3.7: VTK file data processing for GINO

- **Surface Point Cloud**: The vertices extracted from the decimated mesh form an unstructured point cloud that represents the vehicle's outer boundary. Each point maintains its three-dimensional coordinates and associated CFD-computed values, primarily surface pressure. This representation preserves the exact surface geometry while providing flexibility in spatial discretization.
- **Signed Distance Field**: A volumetric signed distance field is computed on a regular Cartesian grid encompassing the vehicle geometry. Using Open3D's RayCastingScene for efficient distance queries [27], the SDF encodes the signed orthogonal distance from each grid point to the nearest surface. Negative values indicate points inside the vehicle, while positive values indicate exterior points; zero defines the surface boundary. This implicit representation provides continuous geometric information throughout the computational domain, which is essential for the neural operator's spatial reasoning.

3.2.1.3 Feature Normalization

Different normalization strategies have been used to enhance training stability and convergence:

- **Spatial Normalization**: Geometric coordinates and signed distance values are range normalization to the interval [0, 1]. This leverages the bounded nature of spatial data while ensuring consistent scaling across vehicles of different sizes. The global bounding box computed across all training samples defines the normalization parameters.

- **Pressure Normalization**: Surface pressure values are standardized using unit Gaussian normalization, which transforms the data to have a zero mean and unit variance. This approach accommodates the wide dynamic range typical in aerodynamic simulations, where pressure can vary by orders of magnitude between stagnation and separation regions. Normalization parameters are computed exclusively from the training set and consistently applied during validation and inference.

3.2.1.4 Data Structure Preparation

The preprocessed data is structured as a dictionary that includes surface vertices, which represent the normalized 3D coordinates of the mesh points; query points, corresponding to regular grid positions used for SDF evaluation; the distance field, containing signed-distance values computed on the regular grid; and pressure values, which denote the target pressure at each surface point.

3.2.2 GINO Architecture

GINO [4] integrates two powerful components: a Graph Neural Operator (GNO)[28], which processes irregular geometric inputs and unstructured data such as surface mesh points or unstructured CFD grids, and a Fourier Neural Operator (FNO) [29], which operates on a structured latent representation to capture global interactions via Fourier transforms. The GINO architecture comprises three interconnected modules: a GNO-based encoder, an FNO core, and a GNO-based decoder, which collectively transform data between irregular geometric domains and regular computational grids. By leveraging message-passing operations, GINO effectively captures both local and global dependencies, enhancing spatial awareness on complex surfaces. The figure 3.8 represents the architecture and the components of the GINO.

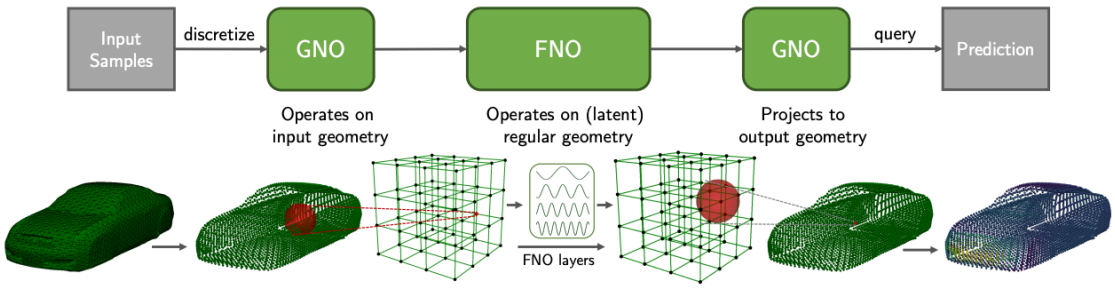


Figure 3.8: GINO architecture[4]

3.2.2.1 Graph Neural Operator Encoder

The encoder serves as the interface between irregular surface meshes and the regular computational grid required for spectral processing [28]. It accepts surface point cloud positions along with optional geometric features and transforms them into a structured latent representation. The transformation process relies on localized

kernel integration, where each point in the regular latent grid aggregates information from nearby surface points through learned kernel functions.

The implementation employs dynamic neighborhood construction using radius-based search with a radius of 0.033 in normalized coordinates. This approach adapts naturally to varying mesh densities, as the spatial hashing algorithm efficiently identifies neighbors for each query point regardless of the local point distribution. The kernel functions themselves are parameterized by neural networks that process relative positions and features to determine the influence of each surface point on the latent representation.

The encoder architecture consists of three Graph Neural Operator layers that progressively build spatial features with increasing receptive fields. Each layer performs localized message passing using the learned kernels, followed by pointwise transformations through multilayer perceptrons. The inclusion of normalization and nonlinear activations between layers ensures stable gradient flow during training. The final output is a regular three-dimensional feature grid with 64 channels that encodes the irregular surface geometry in a format amenable to spectral processing.

3.2.2.2 Fourier Neural Operator Core

The FNO core processes the regular latent grid to capture global interactions and long-range dependencies in the flow field. This component leverages the efficiency of spectral convolutions, which transform spatial features into the frequency domain, where global operations become pointwise multiplications. The architecture stacks FNO blocks [29], each performing spectral convolution followed by local processing.

The spectral convolution implementation retains only the first 16 Fourier modes per spatial dimension from the full 64-mode spectrum. This deliberate truncation serves multiple purposes: it reduces computational complexity, provides implicit regularization by filtering high-frequency noise, and focuses the model on the smooth, large-scale features characteristic of aerodynamic pressure fields. The learnable weights in the frequency domain are parameterized using Tucker tensor decomposition with a specified rank, which reduces the parameter count while maintaining representational capacity.

Each FNO block also includes a parallel channel-wise multilayer perceptron that processes features locally, complementing the global spectral operations. Instance normalization between layers stabilizes training, which is particularly important given the wide dynamic range of pressure values in aerodynamic simulations. To handle non-periodic boundary conditions inherent in vehicle aerodynamics, the implementation applies domain padding before spectral operations, which is removed after processing.

The concatenation of the signed distance field with the latent features provides explicit geometric context throughout the spectral processing. This design choice ensures that the global operations remain geometrically informed, preventing the model from learning physically implausible pressure distributions that violate the boundary conditions imposed by the vehicle surface.

3.2.2.3 Graph Neural Operator Decoder

The decoder reverses the encoding process, mapping features from the regular FNO output grid back to pressure values at each surface vertex. This component mirrors the encoder architecture but operates with surface points as queries and grid cells as sources of information. For each surface point, the decoder identifies nearby grid cells within the same radius used in the encoder and aggregates their features through learned interpolation kernels.

The reverse kernel integration process employs three GNO layers that progressively refine the pressure predictions [28]. The kernel networks in the decoder learn to weight grid features based on their spatial relationship to each surface point, effectively performing learned interpolation that adapts to the local geometry. This approach proves more flexible than traditional interpolation methods, as the learned kernels can account for geometric features like surface curvature and proximity to edges.

A critical property of both encoder and decoder is their discretization invariance. The radius-based construction means that the same trained model can process meshes of vastly different resolutions by simply adjusting the neighbor search without any changes to the model parameters. This enables zero-shot super-resolution, where models trained on coarse meshes with approximately 50,000 vertices can accurately evaluate fine meshes containing over 500,000 vertices.

3.2.3 Training Setup

The model training follows a purely supervised learning approach with L2 error loss computed between predicted and ground-truth pressure values at each surface point. This pointwise loss function ensures the model learns to reconstruct accurate pressure distributions rather than merely capturing aggregate statistics. The choice of L2 loss aligns with the continuous nature of pressure fields, providing stable gradients throughout training.

Optimization employs the AdamW algorithm[30], which improves upon standard Adam by decoupling weight decay from gradient-based updates. This separation proves particularly beneficial for the spectral convolution weights in the FNO layers, where proper regularization is crucial for generalization.

The training process spans 100 epochs; due to GPU memory constraints, training proceeds with a batch size of 1. No data augmentation is applied during training to preserve the physical consistency of the CFD simulations. The model instead relies on the natural diversity of vehicle geometries in the dataset for generalization.

3.3 Experiment Setup

The experiments were conducted using a flexible and modular deep learning pipeline implemented in PyTorch, with additional libraries such as PyVista for visualization, Trimesh for geometry processing, and YAML for configuration management. The

pipeline allows full configurability through YAML files and a `hyper_parameters.csv` file, enabling systematic exploration of different model architectures, datasets, and training configurations.

All experiments were executed on a high-performance computing (HPC) cluster equipped with eight NVIDIA A800 GPUs, each with 80 GB of high-bandwidth memory (HBM), to accelerate training and evaluation. The full hardware configuration of the cluster is summarized in Table 3.1.

Table 3.1: Cluster Hardware Configuration

Component	**Specification**
Processor	2 × Intel Xeon Platinum 8375C (32 cores each)
Main Frequency	2.0 GHz
Memory	512 GB DDR4
Computing Network	200 Gb/s InfiniBand (IB)
Acceleration Cards	2 × NVIDIA A800 GPUs
Video Memory per GPU	80 GB HBM

3.3.1 Finding the suitable dataset and model

In this section, different datasets are evaluated using PointNet model architectures for aerodynamic prediction tasks. Performance metrics across the AhmedML, WindsorML, and DrivAerNet datasets are compared. The objective is to identify the most suitable dataset and the best-performing model configuration. This selection will form the basis for evaluating other parameters in the following sections.

Following experiments carried out with configuration described in Table 3.2.

Table 3.2: Training Configuration for Experiments

Parameter	**Value**
Model Architecture	PointNet
Number of Points	100,000
Optimizer	Adam
Learning Rate	0.001
Batch Size	16
Epochs	300
Dropout	0.1
Loss Function	Mean Squared Error (MSE)
Target Variable	Drag Coefficient (C_d)

3.3.1.1 Results: AhmedML

The experiment on AhmedML dataset consists of 350 training samples, 75 validation samples and 75 test samples. The results, summarized in Table 3.3, show that moderate-depth PointNet architectures provide slightly better performance than deeper networks. However, the overall test R^2 scores are relatively low, with the

best architecture achieving a test R^2 of 0.4473. Given the limited performance, AhmedML is not selected for further analysis.

Table 3.3: AhmedML: PointNet Architectures and Performance Summary

Convolution Layers	Connected Layers	Test R^2	Train R^2	Test MSE ($\times 10^{-3}$)	Train MSE ($\times 10^{-3}$)	Test MAE ($\times 10^{-3}$)	Test Max MAE ($\times 10^{-3}$)
[3, 128, 256, 512]	[512, 256, 1]	0.4473	0.6627	1.34	0.73	26.22	30.02
[3, 256, 512, 512]	[512, 256, 1]	0.3044	0.7066	1.65	0.70	28.69	35.36
[3, 512, 1024, 2048]	[2048, 1024, 256, 1]	0.3264	0.6679	1.66	0.70	28.16	32.32
[3, 32, 64, 128, 256]	[256, 128, 64, 32, 1]	-0.1713	0.2040	2.16	1.23	35.84	43.50
[3, 64, 128, 128]	[128, 128, 64, 1]	0.1963	0.4020	2.40	1.33	38.71	44.66

3.3.1.2 Results: WindsorML

The experiment on WindsorML dataset, with 284 training samples, 35 validation samples and 36 test samples, was evaluated using the same experimental setup. The results, summarized in Table 3.4, consistently show negative test R^2 scores, indicating poor generalization despite relatively low test MSE values. Due to this lack of generalization, the WindsorML dataset was excluded from further analysis.

Table 3.4: WindsorML: PointNet Architectures and Performance Summary

Convolution Layers	Connected Layers	Test R^2	Train R^2	Test MSE ($\times 10^{-3}$)	Train MSE ($\times 10^{-3}$)	Test MAE ($\times 10^{-3}$)	Test Max MAE ($\times 10^{-3}$)
[3, 128, 256, 512]	[512, 256, 1]	-1.85	0.663	1.330	0.261	20.40	22.47
[3, 256, 512, 512]	[512, 256, 1]	-1.79	0.720	1.654	0.256	24.09	26.97
[3, 512, 1024, 2048]	[2048, 1024, 256, 1]	-1.89	0.699	1.613	0.218	23.23	25.70
[3, 32, 64, 128, 256]	[256, 128, 64, 32, 1]	-3.05	0.660	1.484	0.247	21.24	22.89

3.3.1.3 Results: DrivAerNet

The experiment on DrivAerNet dataset consisting of 350 training samples, 75 validation samples, and 75 test samples, was selected for further experiments. As shown in Table 3.5, the best performance was achieved by the architecture with convolution layers `[3,64,128,256,512,1024,2048]` and fully connected layers `[2048,1024,256,1]` with a test R^2 score of 0.938.

Table 3.5: DrivAerNet: PointNet Architectures and Performance Summary

Convolution Layers	Connected Layers	Test R^2	Train R^2	Test MSE ($\times 10^{-3}$)	Train MSE ($\times 10^{-3}$)	Test MAE ($\times 10^{-3}$)	Test Max MAE ($\times 10^{-3}$)
[3, 128, 256, 512]	[512, 256, 1]	0.753	0.823	0.133	0.146	8.82	10.55
[3, 256, 512, 512]	[512, 256, 1]	0.642	0.642	0.134	0.134	9.19	10.56
[3, 512, 1024, 2048]	[2048, 1024, 256, 1]	0.920	0.920	0.044	0.044	5.28	6.88
[3, 32, 64, 128, 256]	[256, 128, 64, 32, 1]	0.839	0.839	0.153	0.153	9.45	12.26
[3, 64, 128, 256, 512, 1024, 2048]	[2048, 1024, 256, 1]	0.938	0.937	0.060	0.036	6.04	6.85

Taking into account the above results, the **DrivAerNet** dataset was selected for subsequent analyses. The DriveNet dataset also features realistic car geometries and provides a collection of 4,000 samples, making it suitable for studying various parameters that may affect model performance.

The best performing PointNet architecture, consisting of convolution layers `[3, 64, 128, 256, 512, 1024, 2048]` and fully connected layers `[2048, 1024, 256, 1]`, is used as the foundation for all experiments in this study.

4

Results

This chapter explains the results obtained using the methods selected from the pre-study on aerodynamic prediction. Two model architectures are tested: **PointNet**, which is used to predict both drag coefficients and surface pressure distributions, and **GINO**, which is used to predict only surface pressure distributions on the **DrivAerNet** dataset. The chapter also explores how different hyper-parameters affect the performance of the models. These findings offer useful insights for building surrogate models in computational fluid dynamics (CFD).

4.1 PointNet Model Results

This section presents results from the use of the PointNet model for two aerodynamic prediction tasks: predicting drag coefficients and surface pressure distributions. The experiments test how factors such as the number of total samples, the density of points, and the batch size affect the accuracy of the model. These results help provide practical guidelines for the use of PointNet in CFD applications.

4.1.1 Drag Coefficient (C_d) Prediction

Table 4.1 summarizes the main configuration used for predicting the drag coefficient (C_d) with the PointNet model. The dataset is divided into 80% for training, 10% for validation, and 10% for testing. The model is trained for 300 epochs using the Adam optimizer with a learning rate of 0.001. A dropout rate of 0.3 is applied to prevent overfitting. These settings form the baseline for evaluating the performance of PointNet in aerodynamic drag prediction tasks.

4.1.1.1 Effect of Total Sample Size

This section investigates how the number of total samples affects the accuracy of drag coefficient prediction. The number of samples was varied from 200 to 3966, while the number of input points was fixed at 100,000 and the batch size at 16. The results are shown in Table 4.2 and illustrated in Figures 4.1–4.2.

With an 80%/10%/10% split for training, validation, and testing, at least 160 total samples are required to ensure that the validation set contains more samples than the batch size. Using fewer samples results in a validation subset that is smaller than the batch size, which causes errors during training.

Table 4.1: PointNet Configuration for Drag Coefficient Prediction

Parameter	**Value**
Model Architecture	PointNet
Training/Validation/Test Split	80% / 10% / 10%
Optimizer	Adam
Learning Rate	0.001
Epochs	300
Dropout	0.3
Embedding Dimension	2048
Loss Function	Mean Squared Error (MSE)
Target Variable	Drag Coefficient (C_D)
Random Seed	1
Trainable Parameters	5,165,121

It is important to note that this study focuses on the effect of the total number of available samples, not just the training subset. Since only 80% of the total samples is used for training, the actual number of training samples is lower than the total. The goal of this analysis is to provide practical insight into how much CFD data a company would need to generate, in order to achieve reliable prediction performance on a specific model. This is particularly relevant in industrial settings where generating CFD data is expensive and time-consuming, and understanding the minimum required dataset size can help optimize resources.

Table 4.2: PointNet Drag Prediction - Effect of Total Sample Size

Samples	**Test R^2**	**Train R^2**	**Test MSE** ($\times 10^{-5}$)	**Train MSE** ($\times 10^{-5}$)	**Test MAE** ($\times 10^{-3}$)	**Max MAE** ($\times 10^{-3}$)	**Train Time (s)**	**Inference** Time (s)
200	0.901	0.919	8.855	7.184	7.307	7.307	10288.04	0.28
300	0.921	0.935	7.044	4.692	5.272	5.272	11449.11	0.32
400	0.934	0.953	5.048	2.506	6.744	8.074	15021.79	0.32
500	0.955	0.955	3.069	2.564	4.503	5.083	16364.14	0.33
1000	**0.960**	**0.965**	**2.968**	2.389	4.317	6.046	28972.40	0.64
2000	0.930	0.943	4.923	3.260	5.521	7.916	51423.15	1.39
3000	0.889	0.911	6.741	5.217	6.480	9.667	69365.88	1.44
3966	0.872	0.895	8.628	7.430	7.075	10.465	89732.20	1.84

The best performance was observed when using 1,000 total samples, with the test R^2 score reaching **0.960**. A relatively close result was also observed with 500 samples ($R^2 = 0.955$), indicating that the model can generalize well with a relatively small dataset. As shown in Figure 4.1, performance declined gradually when using more than 1000 samples. This drop in R^2 suggests that the model began to overfit and could not effectively learn from the increased variation present in the larger datasets.

This observation is further supported by the MSE trend in Figure 4.2, where the test MSE increases as the sample size grows beyond 1000. For example, the lowest test MSE (2.97×10^{-5}) occurs at 1000 samples, while the error increases notably at 3000 and 3966 samples. This pattern reinforces the idea that the models current configuration, with limited capacity, may be insufficient to fully capture the added complexity of larger datasets.

To improve generalization at higher sample sizes, it may be necessary to increase the

Figure 4.1: PointNet Drag Prediction - Test and Train R^2 scores vs Total Samples Size.

model's capacity such as by using a deeper architecture, increasing the embedding dimension, or optimizing other hyper-parameters. A more expressive model could better utilize the richer information in the extended dataset and become more robust against overfitting.

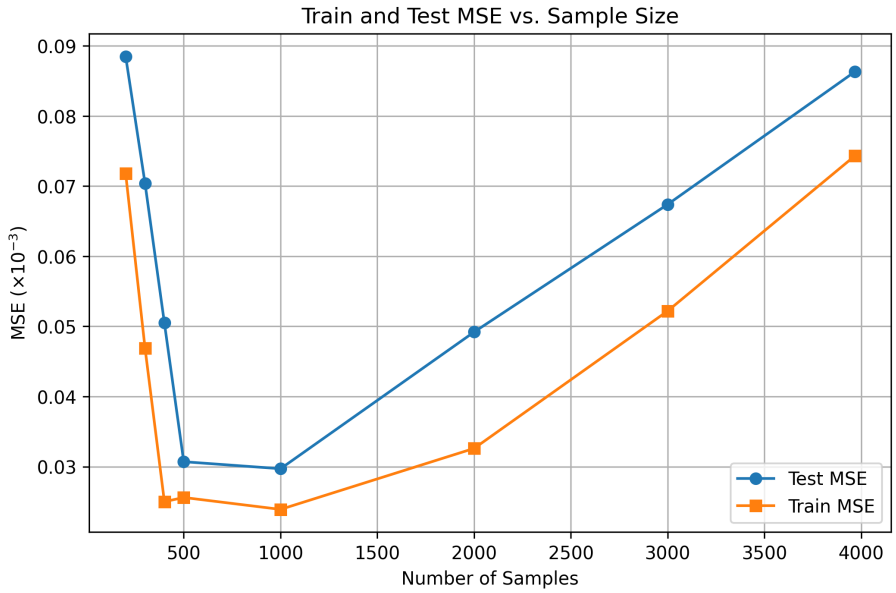


Figure 4.2: PointNet Drag Prediction - Test and Train MSE vs Total Sample Size.

4.1.1.2 Effect of Number of Input Points

This section investigates the effect of the number of input points per sample on drag coefficient prediction. The number of points was varied from 25,000 to 125,000.

Although the original mesh contains approximately 350,000 points, the maximum number of input points was limited to 125,000 due to GPU memory constraints. The results are presented in Table 4.3, and the performance trends are illustrated in Figures 4.3–4.4.

Table 4.3: PointNet Drag Prediction - Effect of Number of Input Points

Points per Sample	Test R^2	Train R^2	Test MSE ($\times 10^{-5}$)	Train MSE ($\times 10^{-5}$)	Test MAE ($\times 10^{-3}$)	Test Max MAE ($\times 10^{-3}$)	Train Time (s)	Inference Time (s)
25,000	0.952	0.964	3.662	2.208	4.284	5.336	15984.54	0.39
50,000	0.942	0.940	3.800	3.336	5.095	5.524	15859.33	0.29
75,000	0.941	0.951	4.998	2.746	5.199	5.765	15998.15	0.34
100,000	0.955	0.955	3.069	2.564	4.504	5.083	17092.18	0.28
125,000	**0.955**	**0.958**	**3.019**	2.406	4.291	5.230	17092.32	0.33

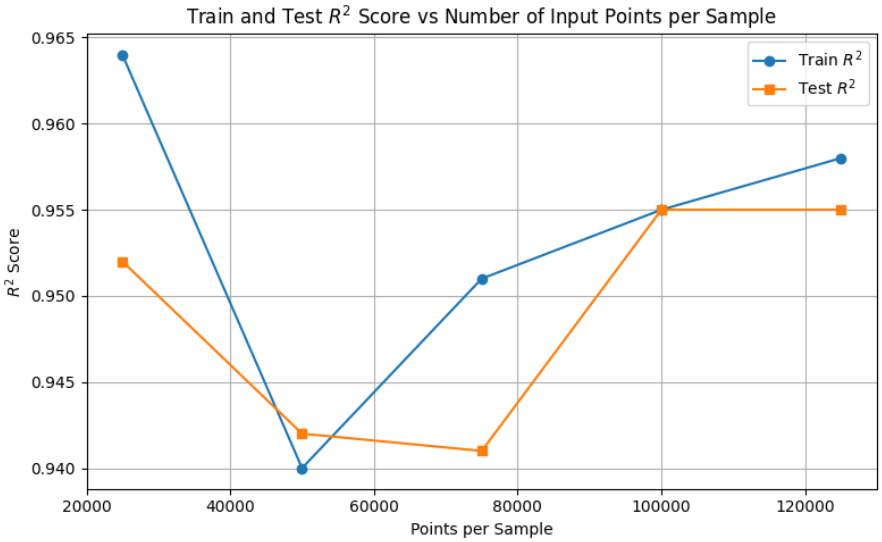


Figure 4.3: PointNet Drag Prediction - Test and Train R^2 scores vs Number of Input Points.

Although the model trained with 25,000 points achieved a high test R^2 score of 0.952, it showed signs of overfitting. This is evident from the notable gap between training and test MSE values (2.21 vs. 3.66 $\times 10^{-5}$), suggesting that the model performed well on the training set but struggled to generalize. Performance fluctuated slightly for point counts of 50,000 and 75,000, where both the test R^2 and MSE metrics showed instability. As the number of points increased to 100,000, performance stabilized, and both the train and test errors were aligned more closely, indicating an improved generalization. The best overall result was achieved at 125,000 points, with the lowest test MSE (3.02 $\times 10^{-5}$) and highest test R^2 (0.955); however, the performance gain compared to 100,000 points was marginal.

Considering accuracy, model stability, and computational cost, using 100,000 input points per sample provides a good balance for the given PointNet model. This configuration avoids the instability observed with lower point counts and does not lead to the increased memory usage and training time associated with larger input sizes. Therefore, 100,000 points is recommended as the optimal input size for this PointNet setup in the context of CFD drag prediction.

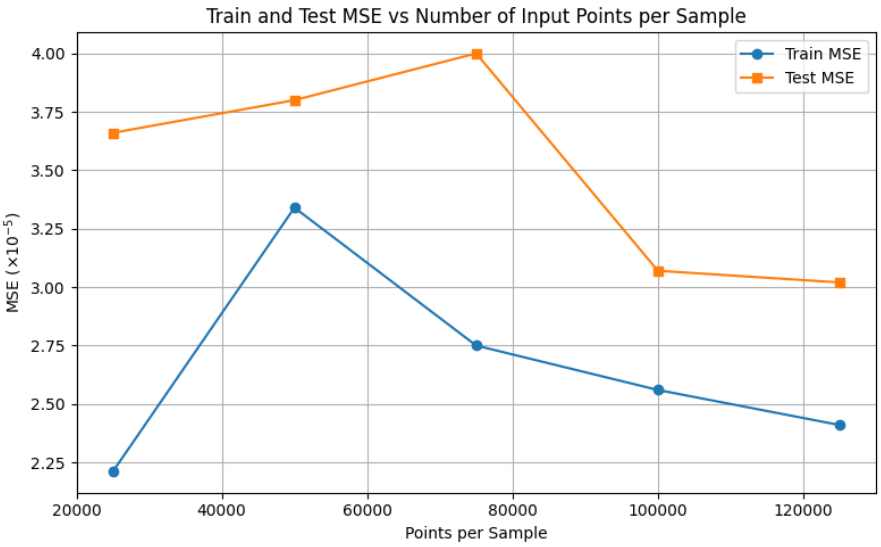


Figure 4.4: PointNet Drag Prediction - Test and Train MSE vs Number of points.

4.1.1.3 Effect of Batch Size

Different batch sizes ranging from 4 to 20 were evaluated to investigate their effect on training stability and model convergence. The experiments were conducted using 500 total samples and 100,000 input points per sample. The results are presented in Table 4.4 and illustrated in Figures 4.5–4.6. Batch sizes larger than 20 were not considered due to GPU memory limitations.

Table 4.4: PointNet Drag Prediction - Effect of Batch Size

Batch Size	Test R^2	Train R^2	Test MSE ($\times 10^{-5}$)	Train MSE ($\times 10^{-5}$)	Test MAE ($\times 10^{-3}$)	Test Max MAE ($\times 10^{-3}$)	Train Time (s)	Inference Time (s)
4	0.685	0.636	67.801	71.091	21.117	31.769	13694.17	0.74
8	0.918	0.922	5.472	4.260	5.695	6.933	14247.83	0.37
12	0.917	0.946	5.160	2.707	5.318	7.828	16546.49	1.21
16	**0.955**	0.955	**3.069**	2.564	4.503	5.083	16969.16	0.39
20	0.943	0.946	4.745	3.798	5.423	5.484	19463.06	0.35

The size of the batch significantly affected the performance, as shown in Figures 4.5 and 4.6. Smaller batch sizes, such as 4, led to poor results ($R^2 = 0.685$), while the best performance ($R^2 = 0.955$) was achieved with a batch size of 16. This configuration also maintained stable training and avoided GPU memory limitations. While larger batch sizes, such as 20, remained feasible, they offered only marginal improvements and risked exceeding memory capacity. Therefore, a batch size of 16 is identified as a robust and practical choice for the PointNet model used here.

Figure 4.7 presents the scatter plot of predicted versus true drag coefficient (C_d) values for the DrivAerNet dataset, using the best performing PointNet model configuration. The optimal setup includes 100,000 input points per sample, a batch size of 16, and 500 training samples. The model achieved a test R^2 score of 0.957, a mean squared error (MSE) of 3.072×10^{-5}, and a mean absolute error (MAE) of 0.0045. The average prediction error was 1.44%, with a maximum error of 4.64% for the test set of 48 samples.

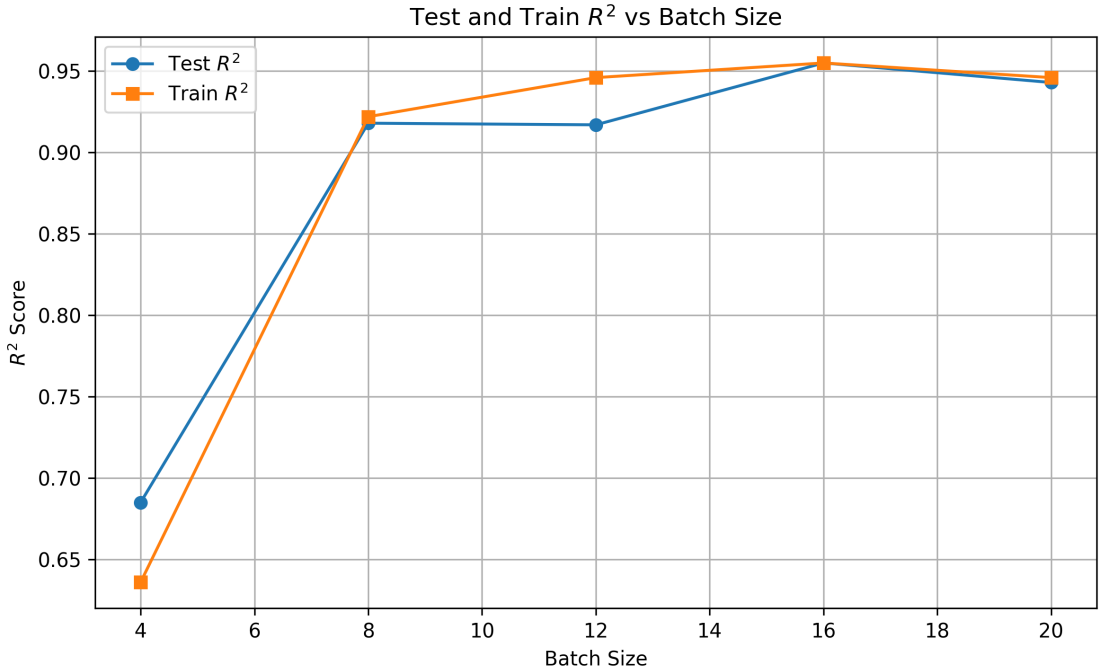


Figure 4.5: PointNet Drag Prediction - Test and Train R^2 scores vs Batch Sizes.

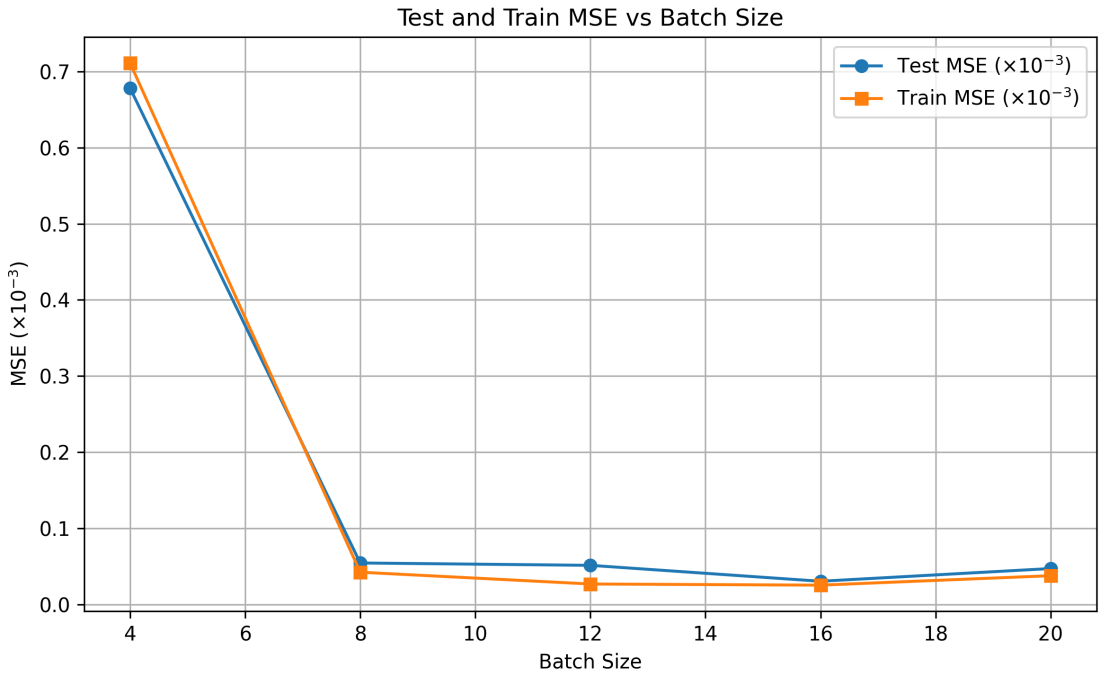


Figure 4.6: PointNet Drag Prediction - Test and Train Mean Squared Error (MSE) vs. Batch Size.

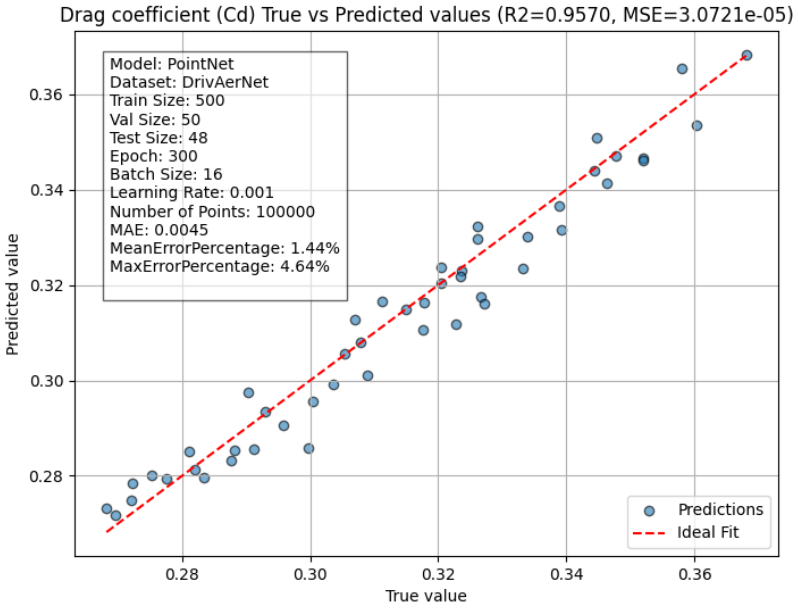


Figure 4.7: PointNet Drag Prediction - True vs Predicted on test set.

4.1.1.4 Drag Predict on unseen data

The DrivAerNet dataset comprises approximately 4,000 samples, representing a diverse range of vehicle geometries. The PointNet model was trained on a subset of 500 samples, using 100,000 points per sample and a batch size of 16, as determined to be the optimal configuration from previous experiments. The remaining 3,466 (after remove invalid samples) unseen samples were reserved for evaluation to assess the model's generalization capability on new, unobserved geometries.

Figure 4.8 presents a scatter plot of the predicted drag coefficients (C_d) against the corresponding ground truth values. The red dashed diagonal line in the plot represents the ideal scenario where predictions perfectly match the true values. Points that lie close to this line indicate accurate predictions, whereas deviations from the line reflect discrepancies between predicted and actual values.

The results indicate that the PointNet model captures the overall trends in aerodynamic drag with high accuracy, as evidenced by the close alignment of most of the points along the diagonal. The model achieves a mean error percenrage of **1.69%** and a maximum error percentage of **7.74%**, demonstrating strong predictive performance. Slight deviations are observed for samples exhibiting extreme C_d values, which may be attributed to the under representation of such cases in the training set. These findings confirm that the PointNet architecture, when optimally configured, is capable of providing reliable aerodynamic drag estimates for complex automotive geometries.

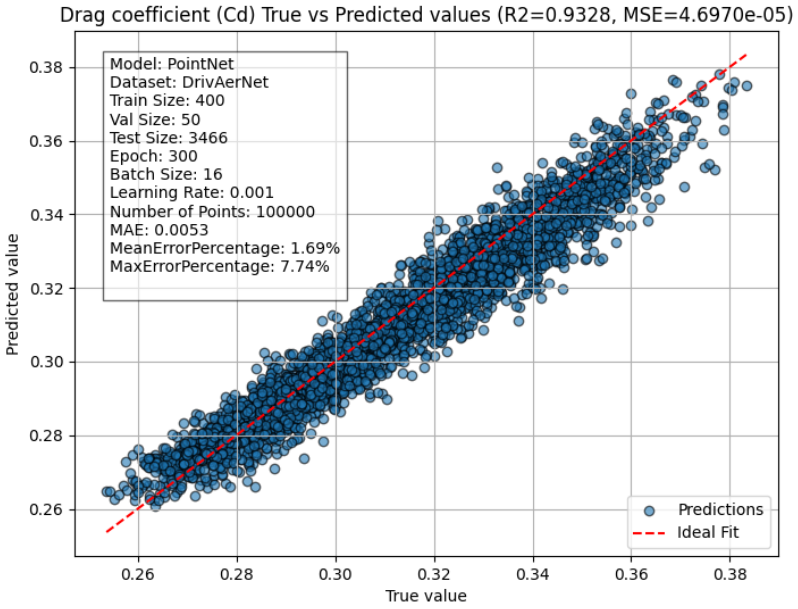


Figure 4.8: True vs. predicted drag coefficients (C_d) for 3,466 test samples from the DrivAerNet dataset, using the PointNet model trained on 500 samples.

4.1.2 Pressure Prediction

For the surface pressure prediction experiments, the PointNet model was trained using the configuration summarized in Table 4.5. The dataset was split into 80% for training, 10% for validation, and 10% for testing. The model was trained for 300 epochs using the Adam optimizer with a learning rate of 0.001. A dropout rate of 0.3 was applied to prevent overfitting, and the embedding dimension was set to 2048. The loss function used was Mean Squared Error (MSE), with pressure field as the target variable. The model had a total of 5,165,121 trainable parameters and was initialized with a random seed of 1 for reproducibility.

4.1.2.1 Effect of Total Sample Size

This section investigates how the total number of available samples affects the accuracy of surface pressure prediction. The number of samples was varied from 200 to 500, while keeping the number of input points fixed at 100,000 and the batch size at 16. Here also, the study focuses on the total sample size, not only the training sample size which is same as the drag prediction.

The results are summarized in Table 4.6, and the corresponding performance trends are visualized in Figures 4.9–4.10. Due to the high computational cost and increased training times associated with larger datasets, the experiments were limited to a maximum of 500 total samples.

The results show relatively stable performance across different training sample sizes, with Rel L2 errors ranging from 0.354 to 0.365. Unlike drag prediction, increasing

Table 4.5: Training Static configuration

Parameter	**Value**
Model Architecture	PointNet
Training ratio	80%
Validation ratio	10%
Test ratio	10%
Optimizer	Adam
Learning Rate	0.001
Epochs	300
Dropout	0.3
Embedding Dimension	2048
Loss Function	Mean Squared Error (MSE)
Target Variable	Pressure
Random seed	1
Trainable Parameters	5165121

Table 4.6: PointNet Pressure Prediction - Effect of Training Sample Size

Samples	**Test Rel L2**	**Train Rel L2**	**Test MSE** ($\times 10^{-3}$)	**Train MSE** ($\times 10^{-3}$)	**Test Max MAE** ($\times 10^{-3}$)	**Train Time** (s)	**Inference Time (s)**
200	0.354	0.344	3.407	3.239	30.027	20230.32	0.18
300	0.364	0.370	3.605	3.684	31.109	21826.40	0.42
400	0.353	0.358	3.400	3.446	30.155	27303.10	0.85
500	0.358	0.359	3.478	3.489	30.029	28085.33	0.38

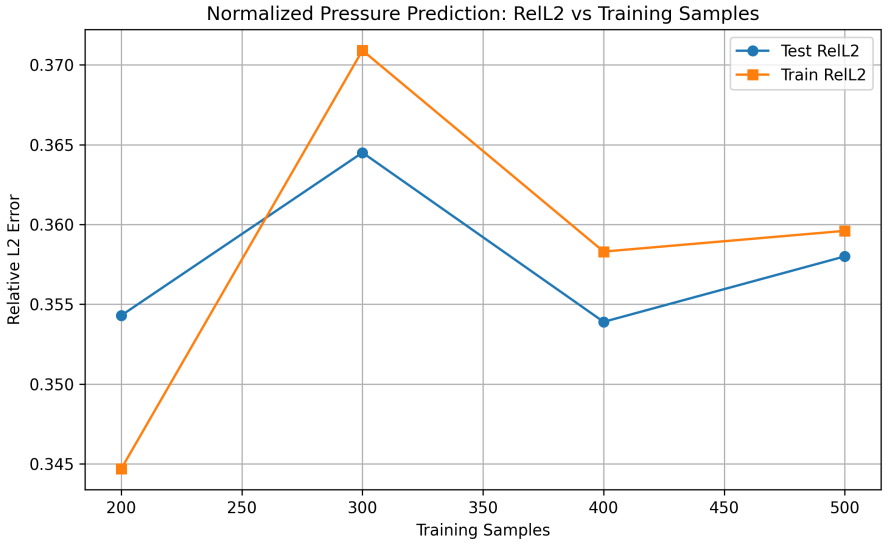


Figure 4.9: PointNet Pressure Prediction - Test and Train Relative L2 Error (Rel L2) vs Training Samples Size.

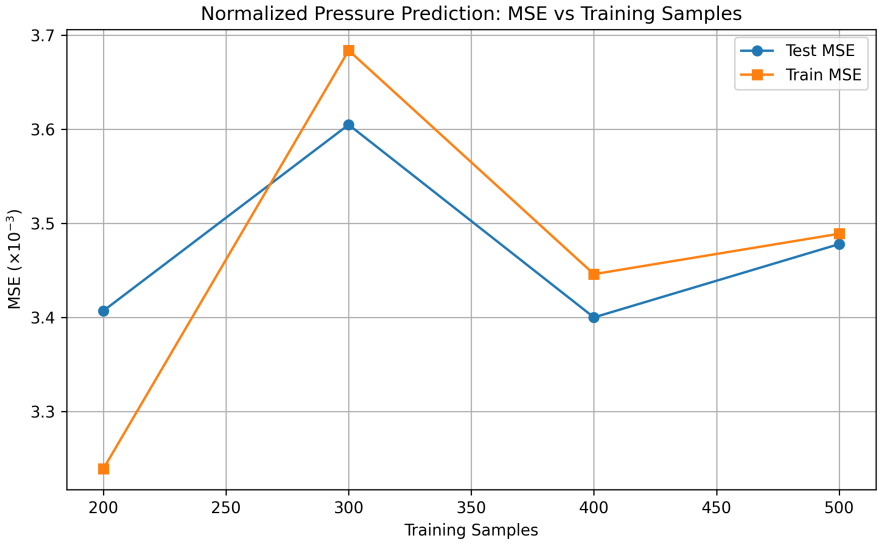


Figure 4.10: PointNet Pressure Prediction - Test and Train MSE vs Training Samples Size.

the total samples did not significantly improve the accuracy of pressure prediction.

4.1.2.2 Effect of Number of Input Points

This section examines how the number of input points per sample influences the accuracy of surface pressure prediction. The number of points was varied from 25,000 to 125,000, while keeping the total number of samples fixed at 500 and the batch size at 16. Increasing the number of points allows the model to capture finer geometric details, which may lead to improved prediction accuracy. However, a higher point count also increases memory usage and training time, which can be a limiting factor in practical applications.

The experimental results are summarized in Table 4.7, and the corresponding performance trends are shown in Figures 4.11–4.12. This analysis helps identify a suitable point resolution that balances accuracy and computational efficiency for the given PointNet model configuration.

Table 4.7: PointNet Pressure Prediction - Effect of Number of Input Points

Points per Sample	**Test Rel L2**	**Train Rel L2**	**Test MSE** ($\times 10^{-3}$)	**Train MSE** ($\times 10^{-3}$)	**Test Max MAE** ($\times 10^{-3}$)	**Train Time** (s)	**Inference Time(s)**
25,000	0.352	0.362	3.362	3.532	30.673	25417.81	0.44
50,000	0.371	0.369	3.773	3.671	31.419	27725.94	0.55
75,000	0.361	0.360	3.549	3.512	30.266	27164.62	0.43
100,000	0.358	0.359	3.478	3.489	30.029	29160.35	0.73
125,000	0.357	0.360	3.466	3.499	30.441	30123.18	0.57

While the stable performance was observed at 100,000 points (Rel L2 = 0.358), results across different point densities remained within a relatively narrow range. Notably, 25,000 points also yielded unexpectedly good results, likely due to random variations. Given the small differences in performance, it is challenging to draw

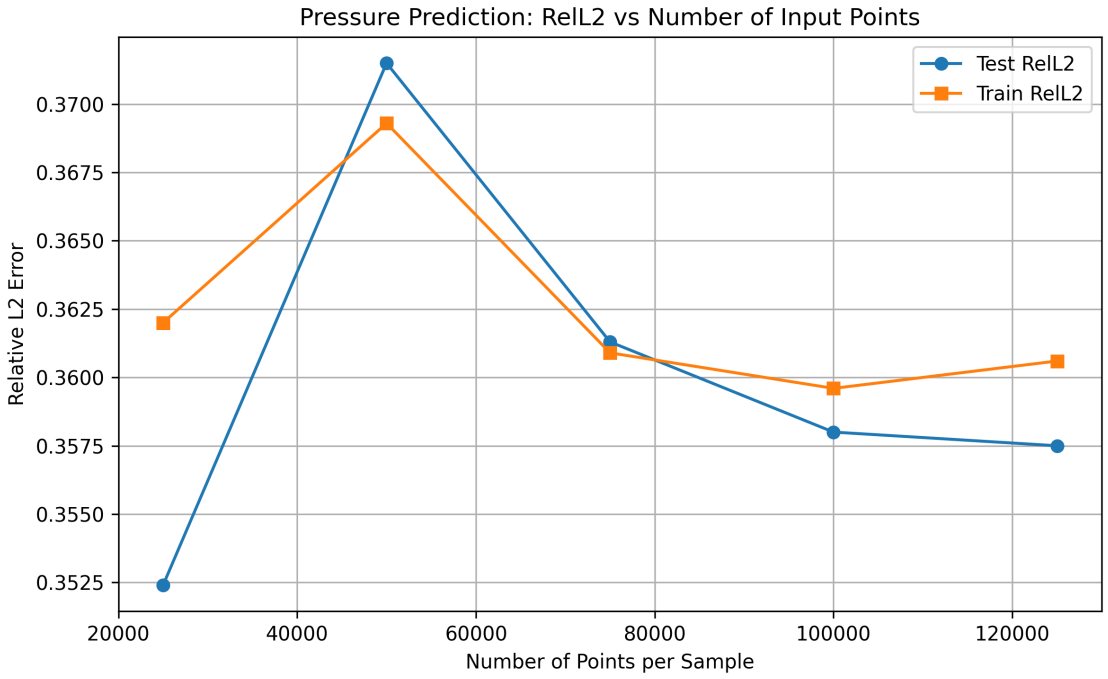


Figure 4.11: PointNet Pressure Prediction - Test and Train Relative L2 Error vs Numbers of Input Points.

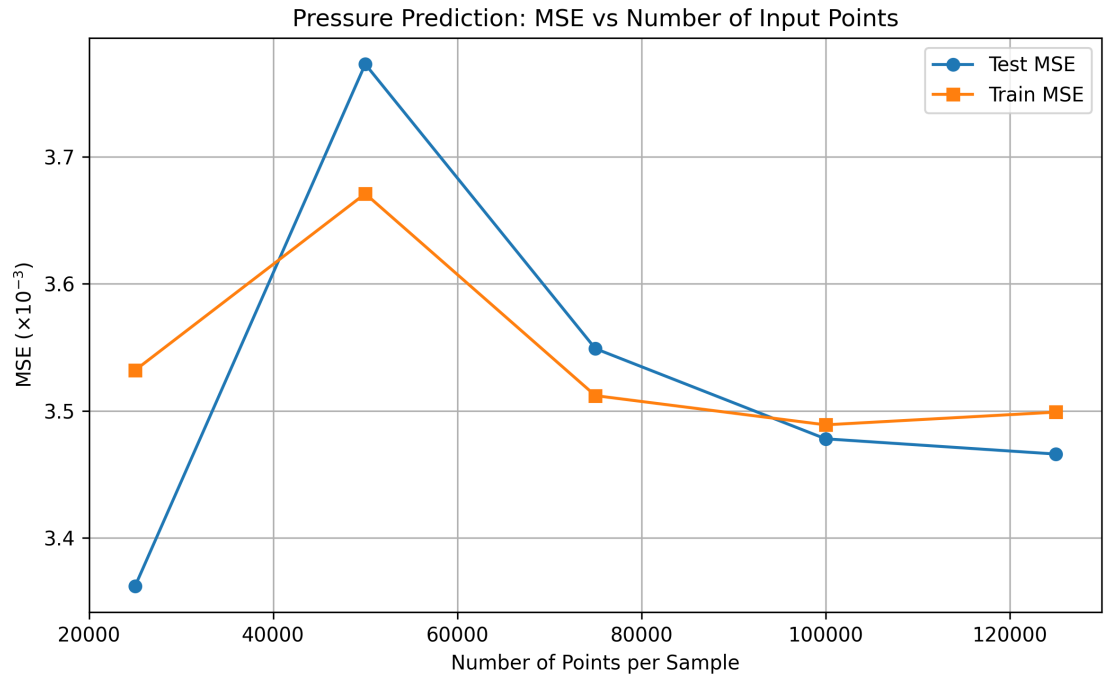


Figure 4.12: PointNet Pressure Prediction - Test and Train MSE vs Numbers of Input Points.

definitive conclusions regarding the impact of point density on pressure field prediction accuracy. However, 100,000 points emerges as a practical and balanced choice, offering a trade-off between accuracy and computational efficiency but overall no much improvement with number of points.

4.1.2.3 Effect of Batch Size

This section investigates how the batch size affects the training performance for surface pressure prediction. The batch size was varied from 4 to 20, while using 500 total samples and 100,000 input points per sample. The experimental results are shown in Table 4.8, and the corresponding performance trends are illustrated in Figures 4.13 and 4.14.

The results show that changes in batch size lead to only minor fluctuations in the Rel L2 error and MSE. Overall, the performance remained consistent across the tested range. A batch size of 16 provided a good balance between training stability and memory usage, with no significant improvement observed beyond that point. Due to GPU memory constraints, batch sizes larger than 20 could not be evaluated.

Table 4.8: PointNet Pressure Prediction - Effect of Batch Size

Batch Size	**Test Rel L2**	**Train Rel L2**	**Test MSE** ($\times 10^{-3}$)	**Train MSE** ($\times 10^{-3}$)	**Test Max MAE** ($\times 10^{-3}$)	**Train Time** (s)	**Inference Time(s)**
4	0.366	0.365	3.645	3.599	33.490	23050.65	0.66
8	0.353	0.361	3.384	3.508	30.803	24650.55	0.86
12	0.357	0.362	3.456	3.520	30.413	27225.79	0.81
16	0.358	0.359	3.478	3.489	30.029	28878.18	0.51
20	0.352	0.355	3.358	3.389	29.887	33151.89	0.37

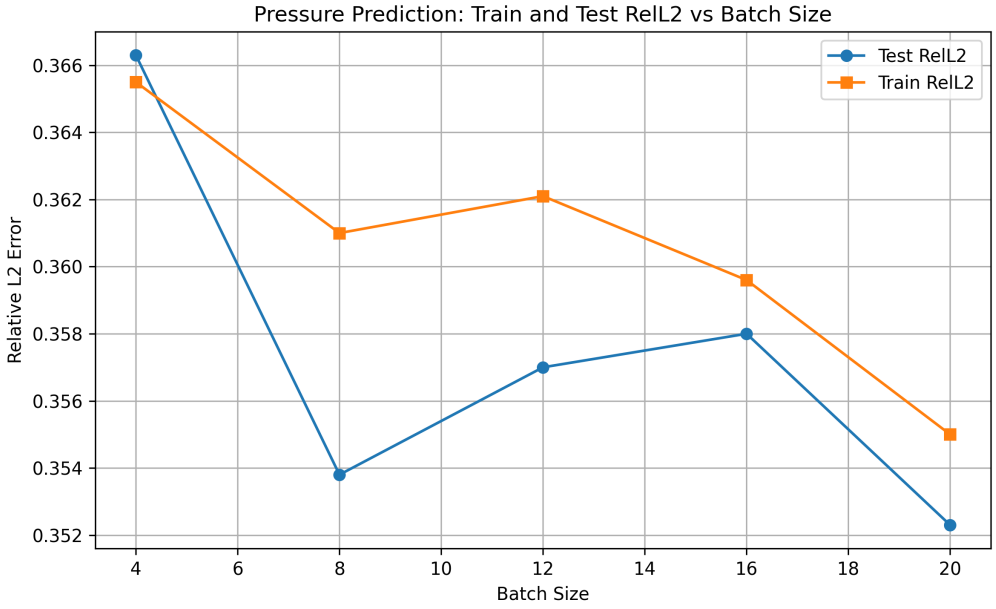


Figure 4.13: PointNet Pressure Prediction - Test and Train Relative L2 Error (Rel L2) vs Batch Size.

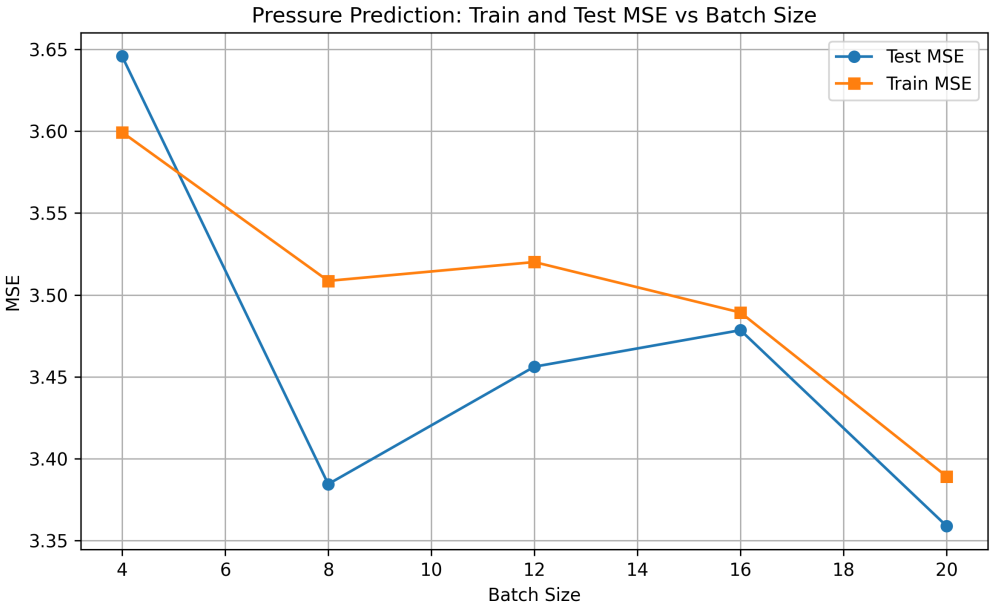


Figure 4.14: PointNet Pressure Prediction - Test and Train Mean Squared Error (MSE) vs Batch Size.

The best performance was observed with a batch size of 20, achieving a Rel L2 error of 0.352; however, the differences in performance across all batch sizes were minimal.

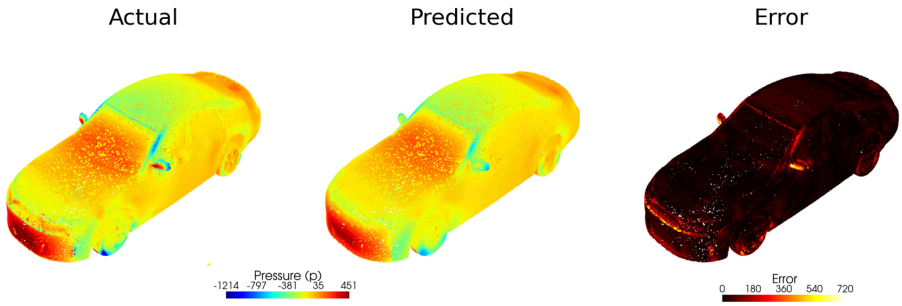


Figure 4.15: PointNet Pressure Prediction - 500 Samples, Batch Size=16. Number of Points=100000, File=DrivAer_F_D_WM_WW_0001.vtk

Figures 4.15 and 4.16 illustrate the PointNet model's pressure prediction results for two DrivAerNet geometries. Each figure displays (from left to right) the ground truth pressure distribution, the predicted pressure distribution, and the corresponding absolute error map.
Overall, the pressure prediction experiments demonstrated limited variance across different total sample sizes, point cloud resolutions, and batch sizes. Rel L2 errors remained in a narrow range between 0.35 and 0.37, indicating that the PointNet architecture was not highly sensitive to these variations. These results suggest that, while PointNet serves as a useful baseline, it may not fully capture the spatial complexity of pressure distributions on automotive geometries, highlighting the need for more advanced architectures in future studies.

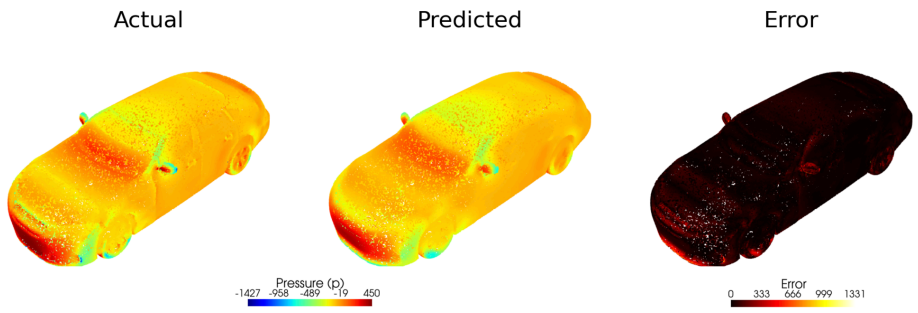


Figure 4.16: PointNet Pressure Prediction - 500 Samples, Batch Size=16. Number of Points=100000, File=DrivAer_F_D_WM_WW_0004.vtk

4.2 GINO Model Results

4.2.1 Pressure Prediction

GINO experiments focused on pressure prediction tasks, exploring the effects of the training sample size, radius parameters, and latent space dimensions. For these experiments, the configuration shown in Table 4.9 was used. For data management, an 80-10-10 split was implemented for training, validation, and testing respectively. The GNO component processes 3D coordinates using NERF-style positional embeddings, while the FNO component operates with 16 frequency modes per dimension and employs Tucker decomposition with a rank of 0.4 for computational efficiency. The training spans 100 epochs with an initial learning rate of 1e-3, which decreases progressively through the StepLR scheduler to ensure smooth convergence. Domain padding of 12.5% was applied to handle boundary conditions effectively.

4.2.1.1 Effect of Total Sample Size

Evaluated GINO's data efficiency by testing different training set sizes ranging from 200 to 1000 samples with a fixed configuration of latent space of [32,32,32], radius of 0.033 and decimation factor of 0.85. The results are presented in Table 4.10, with performance trends illustrated in Figures 4.17-4.18.

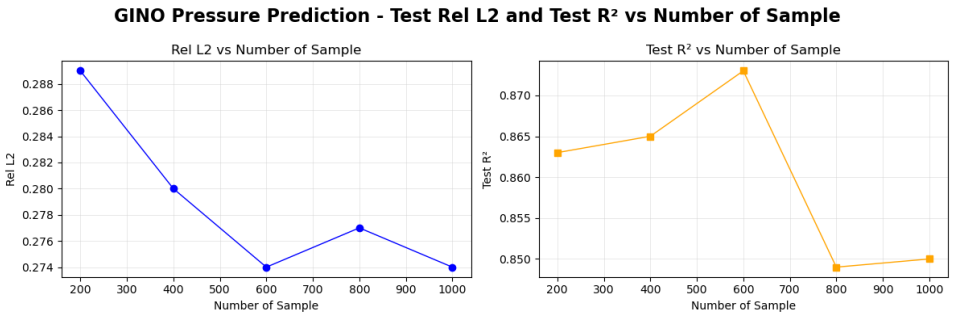


Figure 4.17: GINO Pressure Prediction - Rel L2 and R^2 vs Total Sample Size

GINO achieved strong performance even with 200 training samples (R^2= 0.863),

Table 4.9: GINO Training Static configuration

Parameter	**Value**
Training ratio	80%
Validation ratio	10%
Test ratio	10%
Batch Size	1
Data Channels	0
Output Channels	1
Latent Feature Channel	1
GNO Coordinate Dimension	3
GNO Coordinate Embedded Dimension	16
Input GNO Transform Type	Linear
Output GNO Transform Type	Linear
GNO Positional Embedding Type	nerf
FNO N Modes	[16, 16, 16]
FNO Hidden Channel	64
FNO Ada Input Feature	32
FNO Factorization	tucker
FNO Rank	0.4
FNO Domain Padding	0.125
FNO channel MLP Expansion	1.0
Epochs	100
Learning Rate	1e-3
Training Loss	L2
Scheduler	StepLR
Step Size	10
Gamma	0.5
Trainable parameter	13,786,108

Table 4.10: GINO Pressure Prediction: Effect of Total Sample Size

Samples	**Test R^2**	**Test MSE** ($\times 10^{-3}$)	**Test MAE** ($\times 10^{-3}$)	**Test Max Error** ($\times 10^{-3}$)	**Test Rel L2**
200	0.863	2.404	28.881	4.403	0.289
400	0.865	2.375	26.769	14.524	0.280
600	0.873	2.226	26.031	14.509	0.274
800	0.849	2.732	25.692	19.857	0.277
1000	0.850	2.685	25.192	19.864	0.274

demonstrating excellent data efficiency. The best results were obtained with 600 samples ($R^2 = 0.873$, Rel L2 = 0.274), with performance slightly degrading beyond this point.

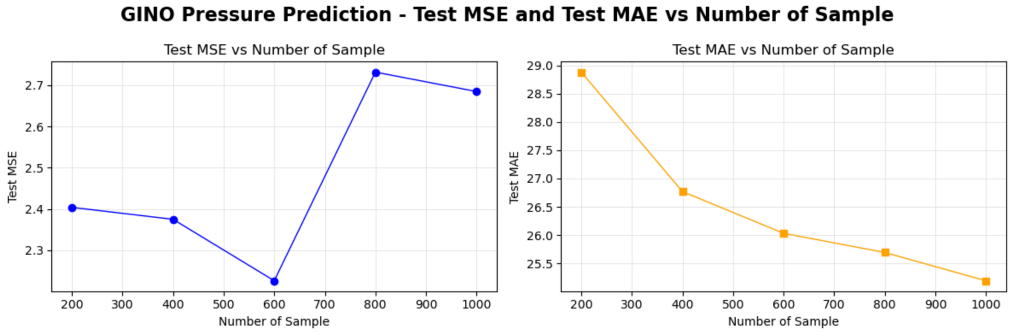


Figure 4.18: GINO Pressure Prediction - Test MSE and Test MAE vs Total Sample Size

This data efficiency has significant practical implications for industrial applications. Traditional CFD simulations for generating training data are computationally expensive. Therefore, achieving optimal GINO performance with 600 samples represents substantial computational savings compared to the potentially massive computational requirements of less efficient architectures.

4.2.1.2 Effect of Radius

Tested various radius values (0.025 to 0.056) while keeping latent space dimensions fixed at [32,32,32] and using a decimation factor of 0.90. Table 4.11 summarizes the experimental results, with Figures 4.19-4.20 showing the performance metrics.

Table 4.11: GINO Pressure Prediction - Effect of radius

Radius	**Test R^2**	**Test MSE** ($\times 10^{-3}$)	**Test MAE** ($\times 10^{-3}$)	**Test Max Error** ($\times 10^{-3}$)	**Test Rel L2**
0.025	0.851	2.672	27.769	14.510	0.292
0.035	0.854	2.616	27.455	14.552	0.288
0.045	0.856	2.569	27.255	14.478	0.286
0.056	0.856	2.569	27.060	14.537	0.285

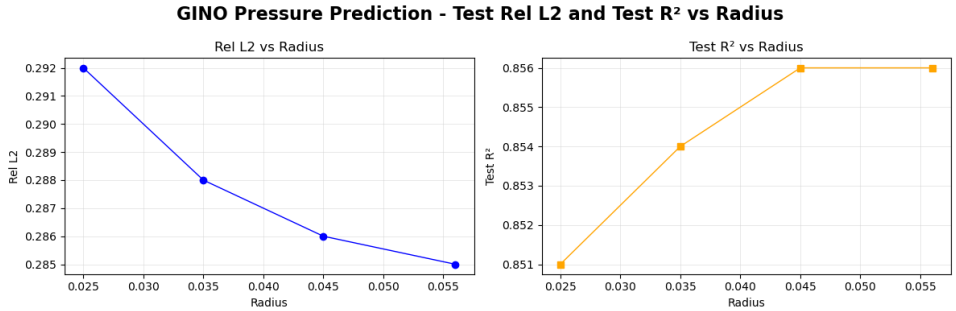


Figure 4.19: GINO Pressure Prediction - Rel L2 and R^2 vs Radius

The radius parameter in GINO controls the spatial extent of local neighborhood aggregation in the GNO layers. This parameter determines how far geometric influences propagate in the pressure field prediction. Increasing radius from 0.025 to 0.056 led to gradual performance improvements. The test R^2 increased from 0.851 to 0.856, while Rel L2 error decreased from 0.292 to 0.285. Performance gains were most significant up to radius 0.045, with minimal improvements beyond this value. Greater radius needs more computational memory, limiting exploration of large configurations due to GPU constraints.

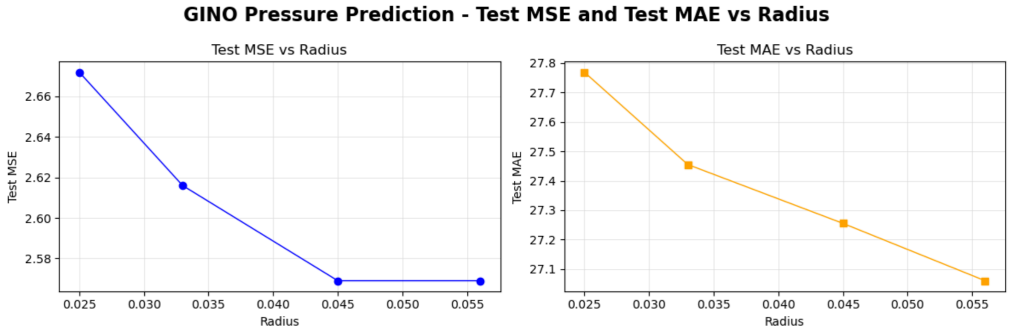


Figure 4.20: GINO Pressure Prediction - Test MSE and Test MAE vs Radius

4.2.1.3 Effect of Latent Space Dimensions

Latent space dimensions determine the model's representational capacity. Three configurations ([8,8,8], [16,16,16], and [32,32,32]) were tested while keeping the radius constant at 0.033 and the decimation factor fixed at 0.90. The results are presented in Table 4.12 and in Figures 4.21–4.22.

Table 4.12: GINO Pressure Prediction: Effect of Latent Space

Latent space	**Test R^2**	**Test MSE ($\times 10^{-3}$)**	**Test MAE ($\times 10^{-3}$)**	**Test Max Error ($\times 10^{-3}$)**	**Test Rel L2**
[8,8,8]	0.032	17.349	80.717	14.547	0.773
[16,16,16]	0.393	10.874	58.183	14.565	0.611
[32,32,32]	0.853	2.623	27.486	14.560	0.289

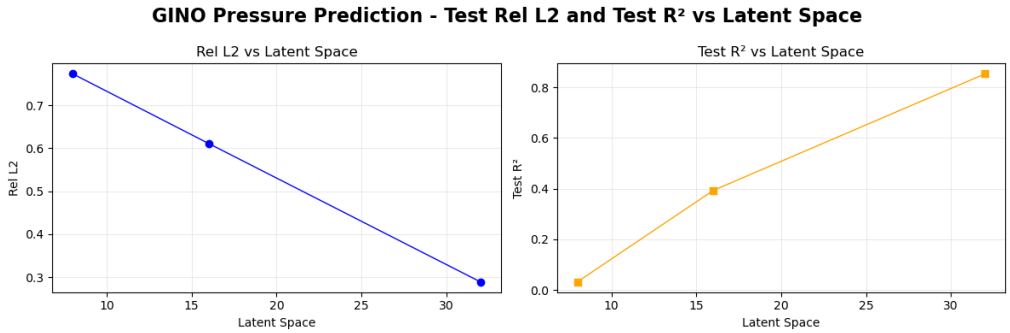


Figure 4.21: GINO Pressure Prediction - Rel L2 and R^2 vs Latent Space

Latent space dimensions dramatically affected performance. The smallest configuration [8,8,8] achieved only $R^2 = 0.032$, while [32,32,32] reached $R^2 = 0.853$, representing a 26-fold improvement. The Rel L2 error decreased from 0.773 to 0.289,

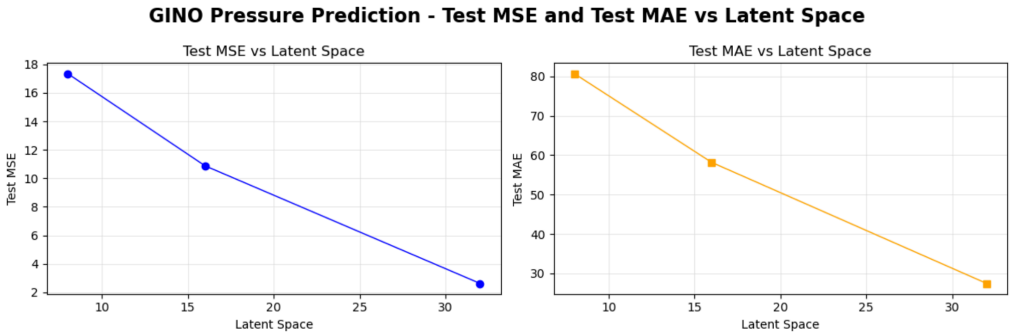


Figure 4.22: GINO Pressure Prediction - Test MSE and Test MAE vs Latent Space

demonstrating that sufficient latent dimensions are critical for capturing pressure field complexity.

GINO demonstrates superior capability for pressure field prediction compared to PointNet, achieving R^2 values exceeding 0.87 and Rel L2 values exceeding 0.275 with optimal configurations. The architecture's strength lies in its ability to process information at multiple scales through the GNO-FNO-GNO pipeline, making it particularly suitable for spatially varying aerodynamic quantities. However, this comes at the cost of increased architectural complexity of learning parameter of 13 million parameters . The model shows remarkable data efficiency, performing well even with 200 training samples. GINO is highly sensitive to latent space dimensions and radius. At least the Latent space dimension of [32, 32, 32] is crucial for capturing the complexity of pressure fields.

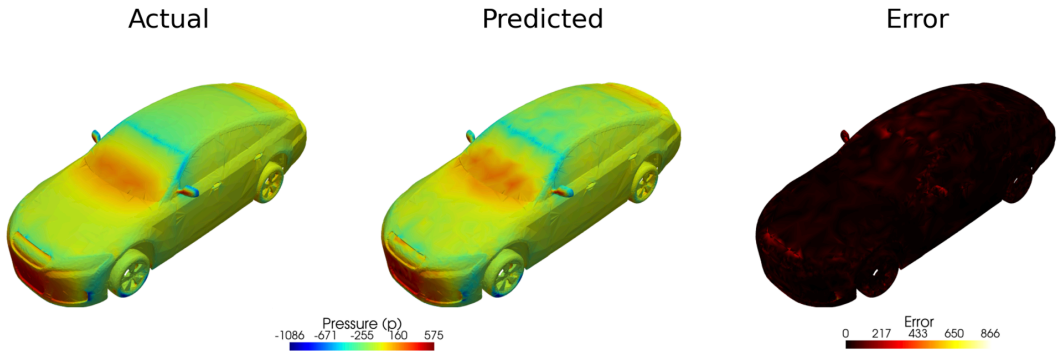


Figure 4.23: GINO pressure prediction on 600 samples with latent space : [32,32,32] and radius : 0.033

Figure 4.23 shows GINO model pressure predictions with 600 samples, using latent space dimensions of [32,32,32] and radius 0.033. The color mapping indicates pressure distribution across the vehicle's geometry, with warmer colors representing higher-pressure regions. Figure 4.24 displays pressure predictions from the same GINO model trained with 400 samples and a larger radius of 0.045 while maintaining the same latent space configuration. The visualization illustrates how various hyperparameters impact the model's pressure field predictions.

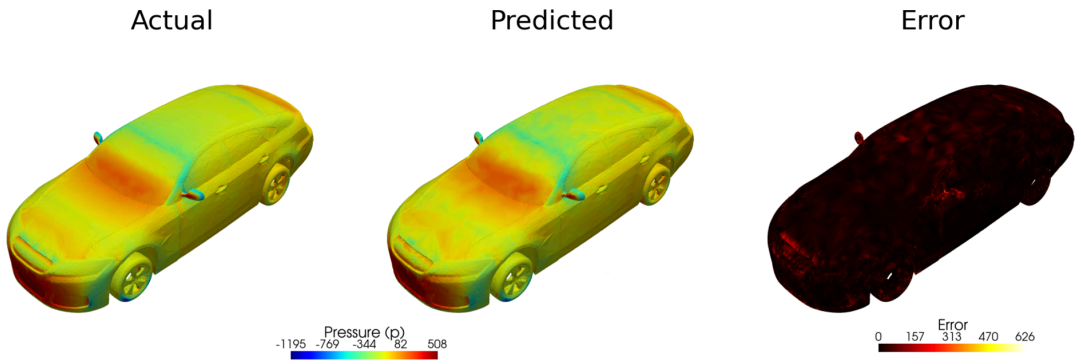


Figure 4.24: GINO pressure prediction on 400 samples with latent space : [32,32,32] and radius : 0.045

5

Conclusion

5.1 Discussion

The results of this experiment reveal distinct differences in the performance of PointNet and GINO in predicting drag coefficient and pressure distribution.

PointNet achieved excellent performance for drag coefficient prediction, with a R^2 value of 0.955 with 500 total samples, 100,000 points per sample and batch size 16; however, its pressure prediction performance was significantly lower, with a Rel L2 value of approximately 0.35. This significant difference arises due to the architecture's global pooling mechanism (batch size 16), which combines local geometric features into a single global representation. Since the drag coefficient is calculated by integrating over the entire vehicle surface, this aggregation approach is effective, with the max pooling operation identifying the most significant geometric features that impact overall drag, a process similar to how drag is physically calculated through surface integration.

GINO performed better for pressure field prediction, with an R^2 of 0.87 and Rel L2 of 0.27, because the architecture maintains local geometric relationships during processing. GINO employs a three-part structure, utilizing Graph Neural Operators (GNO) for local feature extraction, Fourier Neural Operators (FNO) for global pattern recognition, and an additional GNO layer for local reconstruction, thereby enabling it to accurately capture spatially varying phenomena. This capability is significant for pressure distributions that have sharp gradients near geometric features, complex boundary layer effects, and flow separation regions.

The differences between PointNet and GINO stem from their distinct approaches to processing geometric data. PointNet processes each point independently before combining them through global aggregation, which loses spatial context and becomes a problem when predicting spatially continuous fields, as the architecture has no way to capture the relationships between neighboring points that control fluid flow behavior. In contrast, GINO features a GNO layer that enables the network to learn how local geometric changes impact nearby regions. This is crucial for capturing pressure gradients, which depend on spatial derivatives. The GNO-FNO-GNO pipeline can process information on both local flow features and global flow patterns.

The hyperparameter studies showed different sensitivity patterns for each architecture. PointNet model used here achieved the best drag prediction performance with

500-1000 total samples ($R^2 = 0.955$ with 500 samples, $R^2 = 0.960$ with 1000 samples) and was robust to changes in input point density from 25,000 to 125,000 points but showed strong sensitivity to batch size with best performance at batch size 16 ($R^2 = 0.955$) and significant drops at batch size 4 ($R^2 = 0.685$). Performance decreased when using more than 2,000 total samples, with the R^2 score dropping to 0.872 at 3,966 samples. Around 1,000 samples, the current model configuration began to show signs of overfitting. To achieve better performance with larger datasets, it is necessary to increase the model capacity, for example, by adding more layers or increasing the embedding dimension. For pressure prediction, PointNet demonstrated limited performance, with Rel L2 errors remaining relatively stable across configurations, ranging from 0.352 to 0.371, regardless of changes in hyperparameters.

GINO demonstrated excellent data efficiency, achieving ($R^2 = 0.863$ and Rel L2 = 0.289 with only 200 total samples, and its best performance was reached with 600 samples ($R^2 = 0.873$, Rel L2 = 0.274). The architecture was susceptible to latent space dimensions, with performance improving dramatically from [8,8,8] configuration ($R^2 = 0.032$, Rel L2 = 0.773) to [32,32,32] configuration ($R^2 = 0.853$, Rel L2 = 0.289), representing a 26-fold improvement in R^2. GINO showed only minor sensitivity to the radius parameter, with (R^2 varying from 0.851 to 0.856 across the tested range of 0.025 to 0.056. This stable performance across different training set sizes and radius values demonstrates that GINO can effectively learn pressure field patterns once it has sufficient representational capacity through adequate latent dimensions. GPU memory constraints limited our experiments, preventing the testing of larger radius values and latent space configurations for GINO.

5.2 Conclusion

This study demonstrates that deep learning architectures can effectively predict key automotive aerodynamic properties, including drag coefficient and surface pressure. A systematic comparison was conducted between the PointNet and GINO models, using the same datasets and evaluation criteria. The results show that the choice of architecture should depend on the specific aerodynamic quantity being predicted.

PointNet performed best for globally integrated quantities, such as the drag coefficient, achieving an R^2 score in the range of 0.955 to 0.960. This strong performance is attributed to its global feature aggregation mechanism, which enables effective learning of overall flow behavior. For the specific PointNet configuration used in this study, optimal results were obtained with a batch size of 16. To support this batch size while maintaining a valid validation set, a minimum of 200 total samples was required to obtain better results for drag prediction. This highlights the importance of aligning data availability with model design when applying deep learning to CFD tasks.

In contrast, GINO showed better performance for spatially distributed fields, such as surface pressure distributions, achieving an R^2 score of 0.87 with a relative L_2 error in the range of 0.27 to 0.29. This improvement is due to GINOs ability to preserve local geometric relationships throughout the prediction process, making it

more suitable for tasks involving fine-grained spatial details.

Both architectures demonstrate good data efficiency compared to traditional CFD approaches, with GINO achieving an (R^2 of 0.863 and Rel L2 of 0.289 using only 200 total samples. This suggests that the expensive process of generating CFD training data can be reduced significantly. The computational efficiency of these approaches is particularly noteworthy. For drag prediction, PointNet requires less than one second for inference, while GINO takes approximately 70 seconds for pressure prediction. Both methods are significantly faster than traditional CFD simulations, which typically require several hours to complete. Training times are also reasonable, with PointNet requiring 4-6 hours and GINO requiring 7-9 hours on average. With server costs of approximately 8 SEK per hour, this translates to training costs of 200-300 SEK for PointNet and 350-450 SEK for GINO, making these approaches economically viable for industrial applications. Combined with rapid inference times, these approaches open up new possibilities for automotive design workflows, including early-stage design exploration, real-time optimization, and reduced computational costs.

The performance characteristics of each architecture match the physical understanding of fluid mechanics. PointNet's success with global quantities reflects how drag coefficients are calculated through integration, while GINO's better performance for field prediction (improving from Rel L2 0.35 with PointNet to Rel L2 0.27 with GINO) matches the local nature of pressure gradients and flow phenomena. This alignment between architectural design and physical phenomena suggests that these approaches are capable of generalizing beyond the specific datasets that were studied.

Through organized hyperparameter optimization, practical configurations were identified that balance accuracy and computational efficiency. These configurations provide useful starting points for practitioners aiming to implement such methods in industrial settings, potentially accelerating the adoption of machine learning approaches in automotive aerodynamic design.

5.3 Future Work

The findings of this study open several important directions for future research. While PointNet achieved good results for drag prediction, its limited performance for pressure field prediction (Rel L2 0.35) suggests the need for enhanced architectures. Future implementations could explore PointNet++, Dynamic Graph CNN (DGCNN), self attention-based architectures or transform that incorporate hierarchical feature extraction and local neighborhood aggregation to capture spatial relationships more effectively for field predictions.

The next phase of GINO development should focus on expanding its predictive scope to encompass multiple quantities. Extending the model to predict both pressure and wall-shear stress in a single forward pass would enable comprehensive aerodynamic analysis and allow the drag coefficient to be calculated using the physical formula.

This would allow direct comparison with PointNet for drag prediction and potentially highlight GINO's advantages when using physically complete data.

Memory limitations posed a significant constraint in the GINO experiments, particularly in evaluating larger radius values and latent space dimensions. Due to GPU capacity, the radius was restricted to 0.056 and the latent space dimension was limited to $[32, 32, 32]$. As a result, it was not possible to investigate whether larger configurations could lead to improved predictive performance. This represents an important area for future research. To address these limitations, future work should explore memory-efficient strategies such as neighborhood sampling methods (e.g., GraphSAGE), mixed-precision training, or distributed computation across multiple GPUs. These approaches would enable the use of larger models capable of capturing more complex flow physics while maintaining feasible training times and memory usage.

The complementary strengths of PointNet (global quantities) and GINO (local fields) suggest significant potential for hybrid architectures. Such models could use PointNet's global aggregation for drag prediction while leveraging GINO's spatial processing for pressure fields, potentially achieving superior performance on both tasks within a single framework. Attention mechanisms could dynamically balance between global and local processing based on the prediction target.

Extension to more comprehensive datasets beyond DrivAerNet is crucial for validating generalization capabilities. Future datasets should encompass a diverse range of vehicle types (Notchback, Estateback, SUV), various flow conditions (different Reynolds numbers and yaw angles), and comprehensive flow field data, including velocity and turbulence quantities.

Ultimately, the practical implementation of industrial workflows requires addressing several key challenges. Integration with CAD software for direct geometry processing, real-time visualization of predictions, and uncertainty quantification for understanding prediction confidence are all necessary for industrial adoption. Developing standardized benchmarks and evaluation protocols would help establish these methods as reliable alternatives to traditional computational fluid dynamics (CFD) in appropriate applications. These advances would bridge the gap between research demonstrations and practical tools that can accelerate automotive design processes.

Bibliography

[1] J. H. Ferziger and M. Peric, *Computational Methods for Fluid Dynamics*. Springer, 2002.

[2] M. Elrefaie, A. Dai, and F. Ahmed, "Drivaernet: A parametric car dataset for data-driven aerodynamic design and graph-based drag prediction," *arXiv preprint*, 2024.

[3] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "Pointnet: Deep learning on point sets for 3d classification and segmentation," *arXiv preprint arXiv:1612.00593*, 2017.

[4] Z. Li, N. B. Kovachki, C. Choy, *et al.*, "Geometry-informed neural operator for large-scale 3d pdes," *arXiv preprint arXiv:2309.00583*, 2023.

[5] A. Bertram, C. Othmer, and R. Zimmermann, "Towards realtime vehicle aerodynamic design via multifidelity datadriven reduced order modeling," in *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 2018.

[6] K. Tangsali, V. R. Krishnamurthy, and Z. Hasnain, "Generalizability of convolutional encoderdecoder networks for aerodynamic flowfield prediction across geometric and physicalfluidic variations," *ASME J. Mech. Des.*, 2021, Trained on a dataset of 11,000+ airfoil simulations.

[7] Y. Afshar, S. Bhatnagar, S. Pan, K. Duraisamy, and S. Kaushik, "Prediction of aerodynamic flow fields using convolutional neural networks," *Computational Mechanics*, 2019, CNNs used to predict RANS velocity and pressure fields orders of magnitude faster than solvers.

[8] A. Mohan, D. Daniel, M. Chertkov, and D. Livescu, "Compressed convolutional lstm: An efficient deep learning framework to model highfidelity 3d turbulence," *arXiv*, 2019, CAE + ConvLSTM model for spatio-temporal turbulence emulation.

[9] T. Pfaff, M. Fortunato, A. SanchezGonzalez, and P. W. Battaglia, "Learning meshbased simulation with graph networks," *arXiv preprint*, 2020.

[10] A. Kashefi, D. Rempe, and L. J. Guibas, "A pointcloud deep learning framework for prediction of fluid flow fields on irregular geometries," *Physics of Fluids*, vol. 33, no. 2, p. 027 104, 2021.

[11] C. R. Qi, L. Yi, H. Su, and L. J. Guibas, "Pointnet++: Deep hierarchical feature learning on point sets in a metric space," in *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[12] G. Li, M. Muller, A. Thabet, and B. Ghanem, "Deepgcns: Can gcns go as deep as cnns?" In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 2019, pp. 9267–9276.

[13] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, and P. W. Battaglia, "Learning mesh-based simulation with graph networks," *arXiv preprint arXiv:2010.03409*, 2020.

[14] G. Qian, H. A. A. K. Hammoud, G. Li, A. Thabet, and B. Ghanem, "Assanet: An anisotropic separable set abstraction for efficient point cloud representation learning," in *Advances in Neural Information Processing Systems*, vol. 34, 2021, pp. 28 119–28 130.

[15] G. Qian, Y. Li, H. Peng, *et al.*, "Pointnext: Revisiting pointnet++ with improved training and scaling strategies," in *Advances in Neural Information Processing Systems*, vol. 35, 2022, pp. 23 192–23 204.

[16] X. Yu, L. Tang, Y. Rao, T. Huang, J. Zhou, and J. Lu, "Point-bert: Pre-training 3d point cloud transformers with masked point modeling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022, pp. 19 313–19 322.

[17] C. Choy, A. Kamenev, J. Kossaifi, M. Rietmann, J. Kautz, and K. Azizzadenesheli, "Factorized implicit global convolution for automotive computational fluid dynamics prediction," *arXiv preprint arXiv:2502.04317*, 2025.

[18] R. Ranade, M. A. Nabian, K. Tangsali, *et al.*, "Domino: A decomposable multi-scale iterative neural operator for modeling largescale engineering simulations," *arXiv preprint*, 2025.

[19] S. B. Pope, *Turbulent Flows*. Cambridge University Press, 2000.

[20] P. R. Spalart, "Comments on the feasibility of les for wings, and on a hybrid rans/les approach," *Advances in DNS/LES*, vol. 1, pp. 4–8, 1997.

[21] W. Schroeder, K. Martin, and B. Lorensen, *The VTK User's Guide*. Kitware, Inc., 2006.

[22] N. Ashton, D. Maddix, S. Gundry, and P. Shabestari, "Ahmedml: High-fidelity computational fluid dynamics dataset for incompressible, low-speed bluff body aerodynamics," *arxiv.org*, 2024.

[23] N. Ashton, J. Angel, A. Ghate, *et al.*, "Windsorml: High-fidelity computational fluid dynamics dataset for automotive aerodynamics," *Advances in Neural Information Processing Systems 37*, 2024.

[24] N. Ashton, C. Mockett, M. Fuchs, *et al.*, "Drivaerml: High-fidelity computational fluid dynamics dataset for road-car external aerodynamics," *arXiv preprint*, 2024.

[25] Wikipedia contributors, "Feature scaling," *Wikipedia, The Free Encyclopedia*, 2024.

[26] S. Agarwal, *Multilayer perceptron explained with a real-life example and python code: Sentiment analysis*, Accessed: 2025-06-17, Jan. 2020.

[27] Q.-Y. Zhou, J. Park, and V. Koltun, "Open3d: A modern library for 3d data processing," *arXiv preprint arXiv:1801.09847*, 2018.

[28] Z. Li, N. Kovachki, K. Azizzadenesheli, *et al.*, "Neural operator: Graph kernel network for partial differential equations," *arXiv preprint arXiv:2309.00583*, 2021.

[29] Z. Li, N. Kovachki, K. Azizzadenesheli, *et al.*, "Fourier neural operator for parametric partial differential equations," *arXiv preprint arXiv:2010.08895*, 2023.

[30] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

A

Appendix 1

A.1 PointNet Model Code dynamic layers

```
class Model(nn.Module):
    config: Config
    def __init__(self, config):
        super(Model, self).__init__()
        self.config = config
    def forward(self, x):
        pass

class SimplePointNet(Model):
    def __init__(self, config):
        super(SimplePointNet, self).__init__(config)

        emb_dims = config.parameters.model.emb_dims
        dropout = config.parameters.model.dropout

        # **Dynamically create Conv1D layers**
        conv_layers = config.parameters.model.conv_layers
        self.conv_layers = nn.ModuleList()
        self.bn_layers = nn.ModuleList()

        for i in range(len(conv_layers) - 1):
            self.conv_layers.append(nn.Conv1d(conv_layers[i], conv_layers[i + 1], kernel_size=1, bias=False))
            self.bn_layers.append(nn.BatchNorm1d(conv_layers[i + 1]))

        # **Dynamically create Fully Connected layers**
        fc_layers = config.parameters.model.fc_layers
        self.fc_layers = nn.ModuleList()
        self.fc_bn_layers = nn.ModuleList()

        dataset_conf = self.config.datasets.get(self.config.parameters.data.dataset)
        for i in range(len(fc_layers) - 1):
            self.fc_layers.append(nn.Linear(fc_layers[i], fc_layers[i + 1]))
            if i < len(fc_layers) - 2:  # No BatchNorm on final layer
                if dataset_conf.target_col_alias == "Pressure":
```

```
                    self.fc_bn_layers.append(nn.LayerNorm(fc_layers
    [i + 1]))
                else:
                    self.fc_bn_layers.append(nn.BatchNorm1d(
    fc_layers[i + 1]))
        self.dropout = nn.Dropout(dropout)

    def forward(self, x):
        dataset_conf = self.config.datasets.get(self.config.
    parameters.data.dataset)
        # (B, 3, N)  Conv layers
        for conv, bn in zip(self.conv_layers, self.bn_layers):
            x = F.leaky_relu(bn(conv(x)))  # x: (B, C, N)
        if dataset_conf.target_col_alias == "Pressure":
            #  pointwise MLP
            x = x.transpose(1, 2)  # (B, N, C)
            for i, fc in enumerate(self.fc_layers):
                x = fc(x)  # (B, N, C')
                if i < len(self.fc_layers) - 1:
                    x = F.leaky_relu(self.fc_bn_layers[i](x))
                    x = self.dropout(x)
            return x  # (B, N, 1)
        else:
            #  global pooling + FC
            x = F.adaptive_max_pool1d(x, 1).squeeze(-1)  # (B, C)
            for i, fc in enumerate(self.fc_layers):
                x = fc(x)
                if i < len(self.fc_layers) - 1:
                    x = F.leaky_relu(self.fc_bn_layers[i](x))
                    x = self.dropout(x)
            return x  # (B, 1)
```

A.2 YAML Configuration for PointNet

```
model_arch: PointNet  # can be override by hyper_parameters.csv
    file
model_name: SimplePointNet # can be override by hyper_parameters.
    csv file
experiment_batch_name: training_size_constant
base_path: /home/<user>/research/workspace/cfd-aero-pytorch

datasets:
  AhmedML:
    stl_path: inputs/ahmed_ml/stl
    target_data_path: inputs/ahmed_ml/targets.csv
    id_col: "run"
    target_col: " cd"
    target_col_alias: "Drag"  # use to keep same name in target
    subset_dir: inputs/ahmed_ml/subset
  WindsorML:
    stl_path: inputs/windsor_ml/stl
    target_data_path: inputs/windsor_ml/targets.csv
    id_col: "run"
    target_col: " cd"
```

```
    target_col_alias: "Drag"  # use to keep same name in target
    subset_dir: inputs/windsor_ml/subset
  DrivAerML:
    stl_path: inputs/drivaer_ml/stl
    target_data_path: inputs/drivaer_ml/targets.csv
    id_col: "Design"
    target_col: "Average Cd"
    target_col_alias: "Drag"  # use to keep same name in target
    subset_dir: inputs/drivaer_ml/subset
  DrivAerNet:
    stl_path: inputs/drivaer_net/stl
    target_data_path: inputs/drivaer_net/targets.csv
    id_col: "Design"
    #target_col: "Average Cd"
    #target_col_alias: "Drag"  # use to keep same name in target
    target_col: "p"
    target_col_alias: "Pressure"  # use to keep same name in target
    subset_dir: inputs/drivaer_net/subset

environment:
  cuda: false
  device_id: [ 0, 1 ]
  seed: 7
  num_workers: 1

# below parameters can be override by hyper_parameters.csv file
parameters:
  data:
    dataset: DrivAerNet
    data_id_load_random: true  # load ids based on the dataset
   randomly
    max_total_samples: 10 # use when data_id_load_random: true
    train_ratio: 0.6 # use when data_id_load_random: true
    val_ratio: 0.2 # use when data_id_load_random: true and k_folds
   =1
    test_ratio: 0.2 # use when data_id_load_random: true
    k_folds: 1 # use when data_id_load_random: true
    training_size: 6  # use when data_id_load_random: false
    validation_size: 2 # use when data_id_load_random: false
    test_size: 2 # use when data_id_load_random: false
    num_points: 100000
  model:
    lr: 0.001
    batch_size: 2
    epochs: 3
    dropout: 0.0
    emb_dims: 1024
    k: 40
    optimizer: adam
    conv_layers:
      - 3
      - 64
      - 128
      - 256
      - 512
      - 1024
```

```
      - 2048
    fc_layers:
      - 2048
      - 1024
      - 256
      - 1
    output_channels: 1

outputs:
  log_path: outputs/logs
  preprocessed_data: outputs/preprocessed_data
  model:
    best_model_path: outputs/models
    best_scores_path: outputs/scores

predictor:
  enable: true
  best_model_path: outputs/models/20250506
    _pressure_l2_sr_DrivAerNet_PointNet_SimplePointNet_ds500_bs16_epochs300_pts100
    .001_drop0.3_[3_64_128_256_512_1024_2048]_[2048_1024_256_1]
    _best_model.pth
  test_file_path: outputs/predictions/test_sample.txt
  test_output_path: outputs/predictions/test_sample_output.txt
```

A.3 Cluster Submission Script (SLURM)

```
#!/bin/bash

# Directory for logs
LOG_DIR="logs"
mkdir -p $LOG_DIR

# Load Conda environment
source ~/miniconda3/bin/activate python3

# Function to print usage
print_usage() {
    echo "Usage: $0 [batch_name] [hyper_parameters_file]"
    echo "Example: $0 Batch1 hyper_parameters.csv"
    exit 1
}

# Check for required arguments
if [[ $# -ne 2 ]]; then
    echo "Error: Invalid number of arguments."
    print_usage
fi

# Extract arguments
BATCH_NAME=$1
CSV_FILE=$2

# Ensure CSV file exists
if [[ ! -f "$CSV_FILE" ]]; then
```

```
    echo "Error: File '$CSV_FILE' not found!"
    exit 1
fi

# Count total experiments
TOTAL_EXP=$(($(tail -n +2 "$CSV_FILE" | wc -l)))
if [[ "$TOTAL_EXP" -eq 0 ]]; then
    echo "Error: No experiments found in '$CSV_FILE'."
    exit 1
fi

# Initialize counter
RUNNING_COUNT=0

# Skip the header and iterate through each row
tail -n +2 "$CSV_FILE" | while IFS=, read -r dataset_name
   model_arch model_name sample_size batch_size epochs num_points
   lr dropout conv_layers fc_layers
do
    # Increment running experiment counter
    ((RUNNING_COUNT++))

    # Generate an experiment name dynamically
    EXP_NAME="$(date +%Y%m%d)_${BATCH_NAME}_${dataset_name}_${
   model_arch}_${model_name}_ds${sample_size}_bs${batch_size}
   _epochs${epochs}_pts${num_points}_lr${lr}_drop${dropout}_${
   conv_layers}_${fc_layers}"

    # Set unique log files per run
    OUTPUT_LOG="${LOG_DIR}/${EXP_NAME}.out"
    ERROR_LOG="${LOG_DIR}/${EXP_NAME}.err"

    # Print experiment progress
    echo "Submitting experiment $RUNNING_COUNT/$TOTAL_EXP:
   $EXP_NAME"
    echo "Output Log: $OUTPUT_LOG"
    echo "Error Log: $ERROR_LOG"

    # Submit job using SBATCH
    sbatch <<EOF
#!/bin/bash
#SBATCH -J $EXP_NAME
#SBATCH -p dzagnormal
#SBATCH -N 1
#SBATCH --ntasks-per-node=16
#SBATCH --gres=gpu:2
#SBATCH --output=$OUTPUT_LOG
#SBATCH --error=$ERROR_LOG

source ~/miniconda3/bin/activate python3

python3 -u ../source/runner.py \
    --experiment-batch-name "$BATCH_NAME" \
    --dataset-name "$dataset_name" \
    --model-arch "$model_arch" \
    --model-name "$model_name" \
```

```
    --sample-size "$sample_size" \
    --batch-size "$batch_size" \
    --epochs "$epochs" \
    --num-points "$num_points" \
    --lr "$lr" \
    --dropout "$dropout" \
    --exp-name "$EXP_NAME" \
    --conv-layers "$conv_layers" \
    --fc-layers "$fc_layers"

EOF

    echo "Submitting experiment $RUNNING_COUNT/$TOTAL_EXP completed: $EXP_NAME"
    sleep 5
done
```