Finite Volume Features, Global Geometry Representations, and Residual Training for Deep Learning-based CFD Simulation

Loh Sher En Jessica [*] [1 2] Naheed Anjum Arafat [*] [1] Wei Xian Lim [1] Wai Lee Chan [1 3] Adams Wai Kin Kong [1 2]

Abstract

Computational fluid dynamics (CFD) simulation is an irreplaceable modelling step in many engineering designs, but it is often computationally expensive. Some graph neural network (GNN)-based CFD methods have been proposed. However, the current methods inherit the weakness of traditional numerical simulators, as well as ignore the cell characteristics in the mesh used in the finite volume method, a common method in practical CFD applications. Specifically, the input nodes in these GNN methods have very limited information about any object immersed in the simulation domain and its surrounding environment. Also, the cell characteristics of the mesh such as cell volume, face surface area, and face centroid are not included in the message-passing operations in the GNN methods. To address these weaknesses, this work proposes two novel geometric representations: Shortest Vector (SV) and Directional Integrated Distance (DID). Extracted from the mesh, the SV and DID provide global geometry perspective to each input node, thus removing the need to collect this information through message-passing. This work also introduces the use of Finite Volume Features (FVF) in the graph convolutions as node and edge attributes, enabling its message-passing operations to adjust to different nodes. Finally, this work extends the use of residual training to improve flow field prediction for a GNN scenario with immersed object, when low resolution data is available. Experimental results on two datasets with five different state-of-the-art GNN methods for CFD indicate that SV, DID, FVF and residual training can effectively reduce the predictive error of current GNN-based methods by as much as 41%. Our codes and datasets are available at `https://github.com/toggled/FvFGeo`

1. Introduction

Computational fluid dynamics (CFD) is a branch of fluid dynamics in which physical phenomena involving fluid flow are modelled mathematically as partial differential equations (PDEs), like the Navier–Stokes (NS) equations, and solved computationally via numerical analysis. CFD is applied to a wide range of scientific and engineering problems that requires the flow of the fluid and its interaction with surfaces to be simulated, including aircraft aerodynamic optimisation (Martins, 2022), combustion engine design (Vijayashree & Ganesan, 2018), marine hydrodynamics prediction (Demirel, 2021), microfluidic device evaluation (Chaves et al., 2020), and urban planning (Zhang et al., 2021). Despite its versatility, CFD simulation is generally slow and/or costly due to the need for both high spatial and temporal resolutions to solve the governing PDEs accurately.

Researchers have exploited deep learning to accelerate CFD simulation. Multilayer perceptron (MLPs) and convolutional neural networks (CNNs) have been considered. However, they both do not fit industrial requirements in many cases because of their restrictions in the input fields and architectures. MLPs cannot handle high dimensional input, which would increase the number of training parameters dramatically and cause overlearning. Thus, the current MLP methods such as PINN consider each spatial location separately and ignore their relationship (Raissi et al., 2019). In other words, they do not have an explicit scheme for information exchange between nodes. It should be emphasised that flow of a fluid in a particular location would influence that of a neighbour region. Although CNNs allow this information exchange, they can apply only to the flow field represented on a fixed, regular grid. In many industrial simulations, the computational domain may contain objects with

[*]Equal contribution [1]Rolls-Royce@NTU Corporate Lab, Nanyang Technological University, Singapore [2]College of Computing and Data Science, Nanyang Technological University, Singapore [3]School of Mechanical & Aerospace Engineering, Nanyang Technological University, Singapore. Correspondence to: Adams Wai Kin Kong <adamskong@ntu.edu.sg>, Wai Lee Chan <chan.wl@ntu.edu.sg>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

a complex geometry, such as a turbine blade, and hence may have an irregular mesh, so CNNs are not suitable. To fit the industrial requirements and bypass the limitations, researchers have recently employed graph neural networks (GNN) for CFD problems, such as laminar and turbulent flow prediction and geometry optimisation (Liu et al., 2020; Baque et al., 2018). Some of these studies have considered physical properties or constraints in their GNN. For example, Horie & Mitsume (2022) designed boundary encoders to impose Dirichlet boundary conditions and Bonnet et al. (2022a) separated the nodes on airfoils and other nodes in their objective for computing drag and lift coefficients.

The current GNN methods neglect the cell characteristics in the mesh, such as cell volume, face surface area, and face centroid, which are core components of the finite volume method, a widely adopted CFD method in industry (Moukalled et al., 2016). CFD simulators based on the finite volume method in general take three steps to compute the flow of the fluid. First, the simulation domain will be discretised into a finite number of small volumes known as cells. Next, the fluid properties, flow models, boundary conditions of the domain, and initial conditions for the flow will be prescribed. Finally, through Gauss's divergence theorem and assuming a constant solution in each cell, the governing PDEs can be written in an integral form, discretised by numerical approximation, and solved as a system of algebraic equations through numerical methods. The cell characteristics play a critical role in the discretisation process because they are used to model the flux between a certain cell and its neighbours for the purpose of conservation of mass, momentum, and energy. However, the current GNN methods do not consider this information.

Another weakness of the current GNN methods is that their input node features, such as signed distance function (SDF), spatial coordinates, and inlet velocity, only provide very limited information about any embedded object and its surrounding environment to the nodes (Bonnet et al., 2022a; Belbute-Peres et al., 2020). Thus, the nodes need to collect this information through message-passing between neighbouring nodes. In fact, this weakness also appears in the traditional finite volume simulators, which propagate object boundary conditions only locally.

To address these weaknesses, in this paper, we make the following three contributions regarding three aspects of graph neural network training for CFD:

- **Input layer.** We propose Shortest Vector (SV) and Directional Integrated Distance (DID) for enhancing the performance of GNN-based methods. The SV and DID extracted from the mesh provide a global geometry perspective to each input node, thus easing the learning by removing the need to collect this information through message-passing.

- **Graph convolution.** We propose the Finite Volume Features (FVF), including cell volume, face area normal vector, and face centroid, to be used as node and edge attributes in graph convolution, such that the convolution filters can be adjusted based on the cell characteristics. A theorem is presented to show that the input mesh can be reconstructed from the FVF.

- **Training scheme.** While existing GNN methods (Belbute-Peres et al., 2020) exploit low-resolution data as prior knowledge, and the "learned correction" approach (Kochkov et al., 2021) corrects the error of very low-resolution simulations without an immersed object using a CNN, we demonstrate that residual training reduces the prediction error of a GNN by helping the model focus more on regions around and downstream from the geometries, where the low-resolution data tends to be less accurate.

The experimental results show that the (i) combined effect of the proposed geometric features and finite volume features reduces predictive errors of MeshGraphNet (Pfaff et al., 2021), BSMS-GNN (Cao et al., 2023), Chen-GCNN (Chen et al., 2021) and Graph U-Net (Bonnet et al., 2022a) by as much as 41%, as well as reduces the predictive error of CFDGCN (Belbute-Peres et al., 2020) by about 24%, and (ii) additional usage of residual training increases the reduction of the error of CFDGCN to 41%. Further investigation of Chen-GCNN and MeshgraphNet models reveals that the residual training reduces the models' predictive errors by 25% and 45%, respectively.

2. Preliminaries and Related Work

Finite volume method. Consider the integral form of the steady state, incompressible turbulent Navier–Stokes equation for x direction over a control volume, which is given by

$$\begin{aligned}&\int \nabla\cdot\left(\bar{u}_x\overline{\boldsymbol{U}}-(\nu+\nu_t)\nabla\bar{u}_x\right)+\frac{\partial\bar{p}}{\partial x}\mathrm{dV}\\&=\oint\left(\bar{u}_x\overline{\boldsymbol{U}}-(\nu+\nu_t)\nabla\bar{u}_x\right)\cdot\mathrm{d}\boldsymbol{S}+\int\frac{\partial\bar{p}}{\partial x}\mathrm{dV}=0\ ,\end{aligned}\tag{1}$$

where $\overline{\boldsymbol{U}}=[\bar{u}_x,\bar{u}_y]^T$ is the velocity vector, ν and ν_t are the dynamic and turbulent viscosities respectively, and $\bar{p}$ is the normalised pressure. The second expression of Equation 1 is obtained from Gauss's theorem, where $\mathbf{S}$ denotes the area normal vector on the surface of the control volume and points outwards by convention (Moukalled et al., 2016). The finite volume method discretises the control volume into cells, as shown in Figure 1(a). For sufficiently small cell volume V like in Figure 1(b), all variables within a cell or along each of its faces are approximately constant (Moukalled et al., 2016), so Equation 1 becomes

$$\begin{aligned}&\sum_f \left(\bar{u}_x\overline{\boldsymbol{U}} - (\nu+\nu_t)\nabla\bar{u}_x\right)_f \cdot \boldsymbol{S}_f + \left(\frac{\partial \bar{p}}{\partial x}\right)\mathrm{V} \\ &= \sum_f \boldsymbol{\Phi}_f \cdot \boldsymbol{S}_f + \Omega\mathrm{V} = 0\ ,\end{aligned} \tag{2}$$

where f is a counter for the discrete faces of the cell. The first and second terms of the first expression of Equation 2 are known as the flux ($\boldsymbol{\Phi}$) and source (Ω) terms, respectively (Moukalled et al., 2016), thus constituting the second expression.

In typical finite volume simulators, variables like $\overline{\boldsymbol{U}}$ and $\bar{p}$ are solved at the cell centroids such as $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ in Figure 1(b). However, from Equation 2, the flux term needs face centroid values, which can be approximately interpolated from the cell centroids by (Tasri & Susilawati, 2021)

$$\boldsymbol{\Phi}_f \approx \left(\boldsymbol{\Phi}_i \frac{||\boldsymbol{c}_{f,ij}-\boldsymbol{x}_i||}{||\boldsymbol{x}_j-\boldsymbol{x}_i||} + \boldsymbol{\Phi}_j \frac{||\boldsymbol{c}_{f,ij}-\boldsymbol{x}_j||}{||\boldsymbol{x}_j-\boldsymbol{x}_i||}\right)\ . \tag{3}$$

The interpolated flux term will then be projected to its respective face area normal vector, $\boldsymbol{S}_{f,ij}$. Note that Equation 3 implies that the face centroid at $\boldsymbol{c}_{f,ij}$ lies along the line connecting $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, which is not necessarily true because cells in a CFD simulation need not be regular and can have different sizes and shapes, as shown in Figure 1(b). In general, cells with small volumes are used in sensitive regions, for instance, the close vicinity of an object or wake region where flow variables may change drastically due to boundary conditions. Therefore, Equation 3 will incur an error that corresponds to the deviation from the face centroid, though a spatial correction scheme can be implemented as mitigation (Tasri & Susilawati, 2021). Also, the error will reduce with smaller cell volume, which is the main reason for the high computational cost of CFD. On the other hand, the source term weighted by the cell volume will be represented exactly by $\Omega_j\mathrm{V}_j$.

Graph construction. In GNN-based CFD methods, both the inputs and outputs of the model are often graphs. The CFD simulation mesh M is represented as a graph $G = (V, E)$, where V and E represent a set of nodes and edges, respectively. There are two methods to do so.

Using a 2D CFD mesh for illustration, the first method, as shown in Figure 2(a), is to directly represent the mesh nodes as graph nodes $i \in V$, and the faces between them as edges $(i, j), (j, i) \in E$. We refer to this as the *mesh node-based* method, and it has been used by other researchers (Bonnet et al., 2022a; Belbute-Peres et al., 2020; Chen et al., 2021; Pfaff et al., 2021). An alternate approach proposed by the authors is the following: the graph node $i \in V$ represents the mesh cell centroid $m_i \in M$, and the bi-directional edge $(i, j), (j, i) \in E$ indicates that cells i and j are adjacent, i.e., share a face. We refer to this approach as the *cell centroid-based* method, which is illustrated in Figure 2(b). This method is key to the use of FVF as described in § 3.

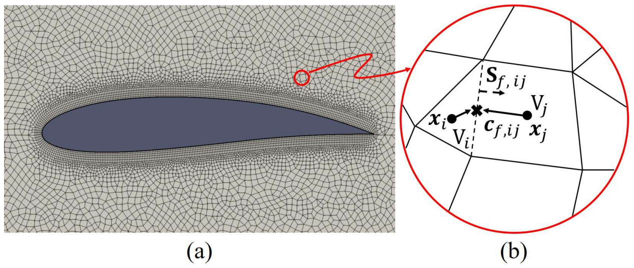

Figure 1. (a) CFD mesh with an airfoil body surrounded by different sizes of cells. (b) Illustration of cell characteristics, namely cell centroids, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, face centroid, $\boldsymbol{c}_{f,ij}$, face area normal vector, $\boldsymbol{S}_{f,ij}$, and cell volumes, V_i and V_j.

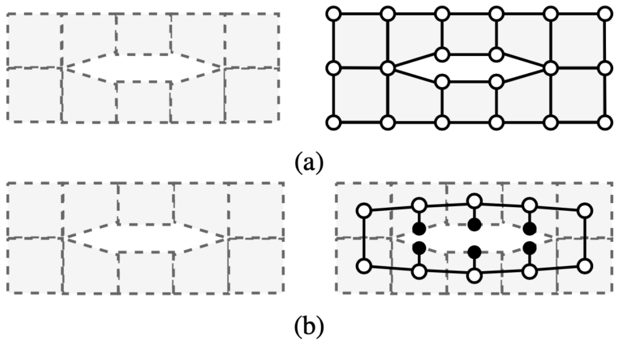

Figure 2. Two graph construction methods: (a) **Mesh node-based:** Graph nodes represent mesh nodes and graph edges represent mesh faces. (b) **Cell centroid-based:** Graph nodes represent either cell centroids (white) or boundary face centroids (black), and graph edges represent the adjacency of the cell centroids with one another or with boundary faces.

Note that in this method, the centroids of the boundary faces are represented as graph nodes as well, despite not being shared by two cells, to capture flow characteristics at the boundary. This representation allows message-passing from and towards boundary faces to be captured by the edges between these nodes and that of the cell adjacent to the boundary face.

Global geometry representations. In a steady simulation, a GNN is trained to predict the velocity vector and pressure for each node. To train a GNN to predict target flow characteristics at each node, the current methods encode some features into the input nodes. The most common approaches include variants of the binary representation (Chen et al.,

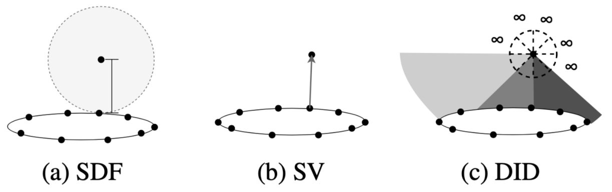

Figure 3. Illustration of three geometry representations: (a) SDF value only indicates presence of the closest boundary point somewhere along the circle's circumference. (b) SV provides both distance and direction from the nearest boundary point. (c) DID gives the average distance of all boundary within several difference angle ranges.

2021) and the Signed Distance Function (SDF) (Bonnet et al., 2022a; Belbute-Peres et al., 2020; Guo et al., 2016). In binary representations, nodes are given discrete values such as 0 and 1, depending on whether they are on the geometry boundary or not. The SDF, proposed by Guo et al. (2016) for CNNs, is defined as

$$SDF(\boldsymbol{x}_i) = \min_{\boldsymbol{x}_b \in B} ||\boldsymbol{x}_i - \boldsymbol{x}_b|| h(\boldsymbol{x}_i), \tag{4}$$

where $\boldsymbol{x}_i$ and $\boldsymbol{x}_b$ denote an internal node and its closest boundary node, respectively. $h(\boldsymbol{x}_i)$ is equal to 1, -1, and 0 if $\boldsymbol{x}_i$ is outside, inside, and on the object boundary, respectively. The SDF representation was shown to be more effective than the simple binary representations in the CNN case (Guo et al., 2016). Since graphs generated from meshes do not have nodes on the inside of the object, the SDF for GNNs is in fact just the shortest Euclidean distance between $\boldsymbol{x}_i$ and the object, estimated in a discrete case. However, it provides each node with very limited information about the object. The SDF value only indicates the existence of an object at the distance of $SDF(\boldsymbol{x}_i)$. No information about the object's size, shape, or direction from the node is given by the SDF, even though all these factors will affect the flow at the node. The boundary of the circle in Figure 3(a) indicates the uncertainty due to incomplete information of $SDF(\boldsymbol{x}_i)$. Although some other properties about the nodes and the flow, such as the spatial location of the nodes, inlet, angle of attack, and Mach number (Belbute-Peres et al., 2020), can also be provided as input node features, they cannot substitute the missing geometric information, which remains to be acquired by the nodes through message-passing. Note that this weakness is also true of typical CFD simulators where each mesh cell carries no global geometry information and the object boundary condition is transmitted only locally.

Prior studies. The GNN-based CFD methods employ on-shell GNNs and exploit the physical knowledge in different ways. Some of them embed physical constraints and properties into the architectures and objective functions and some others directly use numerical simulators as a part of their methods. Belbute-Peres et al. (2020) combined a differentiable CFD simulator and graph convolution network (GCN) for fluid flow prediction. Liu et al. (2020) exploited graph attention network (GAT) for turbulent flow prediction without any physics prior. Ogoke et al. (2020) applied GraphSAGE (Hamilton et al., 2017) to predict drag forces around airfoils of different shapes and angles of attack under laminar flow. They demonstrated that GraphSAGE outperforms non-graph based MLP and CNN methods. Battaglia et al. (2018) proposed a general method called graph network (GN) blocks that can handle graphs with local features such as node features, edge features, and global graph-level features. A GN block first passes messages from nodes to edges through an edge convolution kernel before updating the edge features. The updated edge features are then aggregated to the nodes through sum or other permutation invariant operations as the edge messages. A node convolution kernel then takes the old node features and edge messages to obtain the updated node features. Finally, the updated node features and edge features are aggregated to compute global graph-level features. Sanchez-Gonzalez et al. (2020) used GN blocks to predict the future roll-out of physical systems of particles, including fluids, rigid solids, and deformable materials. Pfaff et al. (2021) used a GNN to predict future roll-outs of unsteady flow and showed better performance than CNN on various simulation scenarios in their proposed MeshGraphNet. More recently, their method was built upon by Libao et al. (2023) to include an RNN-based state encoding and physics loss terms. Bonnet et al. (2022a) released a large-scale high-resolution two-dimensional (2D) Reynolds-averaged Navier-Stokes (RANS) simulation datasets on airfoils and demonstrated good generalisation capabilities of GraphSAGE (Hamilton et al., 2017) and Graph U-Net (Gao & Ji, 2019) to different physical conditions and airfoil geometries. DiscretizationNet (Ranade et al., 2021) used finite volume discretisation to approximate spatio-temporal partial derivatives in its CNN encoder-decoder training. It is a non-data driven method that cannot generalise to new scenarios, requiring it to be trained on every new instance to obtain a solution. Also, its convolutions did not leverage cell characteristics as the finite volume method does. Chen et al. (2021) proposed their own permutation-invariant edge-convolution layer and smoothing layer to predict laminar flow on obstacles of different shapes. Their proposed architecture, which is referred to as *Chen-GCNN* in this paper, showed better performance than the standard U-net model (Ronneberger et al., 2015). Finally, similar concepts behind the MeshGraphNet and Chen-GCNN were further enhanced with multi-scalability in the MS-MGN by Fortunato et al. (2022), MultiScaleGNN by Lino et al. (2021), and BSMS-GNN by Cao et al. (2023), with the latter two using a GU-Net style processor for faster information propagation performance. All these GNN works neglect the cell

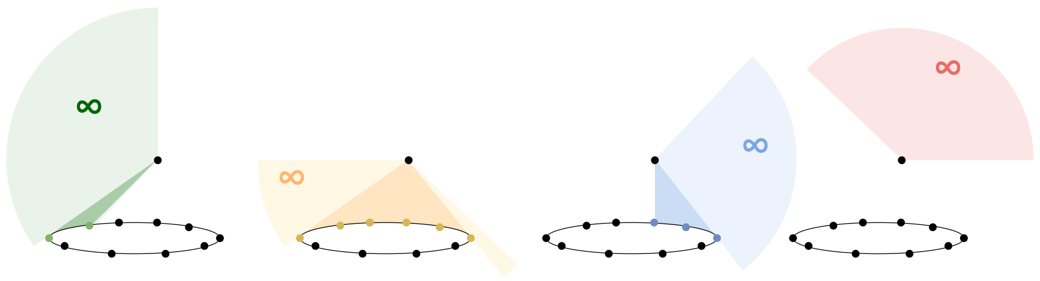

Figure 4. Illustration of the computation of DID. Each angle range is represented by the weighted average of the distance to the boundary (shown by the darker segments) and the ∞ value for the directions where there is no boundary (shown by the lighter segments). Note that ∞ is a predefined number in the implementation.

characteristics, which, in the finite volume method, are used to discretise the governing equations and model the flow among the cells.

3. Proposed Features and Training Scheme

3.1. Global Geometry Perspective

To give each node global geometric perspective about the object and its surrounding environment, we propose two features: Shortest Vector (SV), and Directional Integrated Distance (DID). SV is the vector formed by an internal cell centroid node $\boldsymbol{x}_i$ and its closest boundary face node $\boldsymbol{x}_b$ (Figure 3(b)), and is defined as

$$\phi(\boldsymbol{x}_i) = \boldsymbol{x}_i - \boldsymbol{x}_b \text{ s.t. } \min_{\boldsymbol{x}_b \in B} ||\boldsymbol{x}_i - \boldsymbol{x}_b|| \, . \tag{5}$$

SV is related to SDF in that the length of SV, $||\phi(\boldsymbol{x}_i)|| = SDF(\boldsymbol{x}_i)$. However, SV has more discriminative power than SDF because two nodes, $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, can have the same SDF values (i.e., $SDF(\boldsymbol{x}_i) = SDF(\boldsymbol{x}_j)$) despite the vectors $\phi(\boldsymbol{x}_i)$ and $\phi(\boldsymbol{x}_j)$ not being the same.

One limitation of SV is that only the closest node on the object's boundary is represented. Information in other directions is still missing. DID defined on an angular range is a generalisation of SV. DID has different angular segments to handle different directions. Figure 4 shows four overlapping angular segments, each with a range of $2\pi/3$. To clearly present the concept of DID in the following description, we consider a continuous object boundary. For the j^{th} segment of DID, θ_j and θ'_j represent the starting and ending angles of the segment, respectively, where $\theta_j \leq \theta'_j$. The continuous version of DID is defined as

$$DID(\boldsymbol{x}_i, \theta_j, \theta'_j, B_c) = \int_{\theta_j}^{\theta'_j} w_j(\theta) g(\boldsymbol{x}_i, B_c, \theta) d\theta \, , \tag{6}$$

where $w_j(\theta)$ is a suitable weightage function such as a Gaussian function centred at $(\theta_j + \theta'_j)/2$, B_c is the continuous boundary of the object, and $g(\boldsymbol{x}_i, B_c, \theta)$ is the distance between the object and a node $\boldsymbol{x}_i$ in the direction θ. If there is no object boundary at the direction θ, an appropriately large constant, as denoted in Figure 3(c) as ∞, is given to $g(\boldsymbol{x}_i, B_c, \theta)$. In this continuous version of DID, B_c is represented by a parametric equation rather than a set of nodes. Each DID value provides a weighted average distance between the object and the node in a particular angle range. While shorter angular segments make a more accurate description of the object, it increases the number of input features to train the network with. In this work, we use SV and DID to provide relative global geometry information of the object and its environment to the input nodes. Hence, this information will not have to be collected during propagation, easing the learning process. The algorithmic details of our discrete, non-parametric DID implementation can be found in the appendix.

As stated in the implementation details of DID in Appendix B.1, the weight function used is a uniform distribution over segment range (θ_j, θ'_j). This is better suited for our choice of angle segments, where each segment overlaps with half of the segment preceding it. As a result, every point on the mesh is represented by two successive overlapping segments. In this case, the exact selection of regions is unlikely to significantly change the results, as every point on the boundary is represented the same number of times and with the same weight. On the other hand, if the chosen angle segments meant that some points (in highly overlapping regions) were represented more times than other points (in less overlapping regions), the user may choose to put more weight on the points that are represented fewer times.

3.2. Finite Volume Graph Convolution

As discussed in Section 2 with Equations 2–3 and the accompanying figure 1, the finite volume method uses cell characteristics, such as the cell ($\boldsymbol{x}_i$, $\boldsymbol{x}_j$) and face centroids ($\boldsymbol{c}_{f,ij}$), face area normal vector ($\boldsymbol{S}_f$), and cell volume (V$_i$, V$_j$), extensively. Motivated by the finite volume method, we hence embed these characteristics in GNN. To ease the following discussion, we will take the spatial graph convolution network (SGCN) as an example. First introduced and evaluated on graphs from chemical compounds (Danel et al., 2020), the SGCN has two key features: (i) use of the spatial location of graph nodes as node attributes and (ii) use of multiple filters. One limitation is that the node attributes only consider node positions and ignore other useful information. Hence, we generalise the convolution to be able to take in node attributes, p_j, and edge attributes, q_{ij}. Given node feature $h_i \in \mathbb{R}^{d_{in}}$ at the i^{th} node and its spatial location $\boldsymbol{x}_i \in \mathbb{R}^t$, a convolution using FVF is defined as

$$\bar{h}_i(U, b, N_i) = \sum_{j \in N_i} ReLU(U^T(q_{ij}) + b) \odot (h_j \oplus p_j) \ , \quad (7)$$

where $U \in \mathbb{R}^{3t \times (d_{in}+1)}$ and $b \in \mathbb{R}^{d_{in}+1}$ are trainable parameters, t is the spatial dimension of the CFD simulation, and N_i is an index set indicating the neighbourhood of the node i, $\odot$ is the element-wise multiplication, and $\oplus$ is concatenation (Danel et al., 2020). In our model, the node attributes are its associated cell volume, denoted as $p_j = \mathrm{V}_j \in \mathbb{R}^1$, and the edge attributes are its associated face area normal vector and the relative spatial location of its face centroid to the nodes, denoted as $q_{ij} = \mathbf{S}_{f,ij} \oplus (\boldsymbol{c}_{f,ij} - \boldsymbol{x}_i) \oplus (\boldsymbol{c}_{f,ij} - \boldsymbol{x}_j) \in \mathbb{R}^{3t}$. Note that cell centroids $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ are used as reference points when we use face centroid, $\boldsymbol{c}_{f,ij}$, as with the finite volume method. Finally, as with CNN operations, multiple filters are used and their outputs are concatenated such as

$$\hat{h}_i(\theta, N_i, k) = \bar{h}_i(U_1, b_1, N_i) \oplus \cdots \oplus \bar{h}_i(U_k, b_k, N_i) \ , \quad (8)$$

where $\theta = \{U_1, \cdots, U_k, b_1, \cdots, b_k\}$ are trainable parameters. Finally, an MLP is applied on $\hat{h}_i$ and the final output of node i is obtained, whose dimension is the same as the output dimension of the MLP.

When used directly, we refer to this method as *Finite Volume Graph Convolution (FVGC)* and V_i, $\mathbf{S}_{f,ij}$, $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_i)$, $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_j)$ as *Finite Volume Features* (FVF). Alternatively, the same principles can be incorporated into other graph convolution types indirectly, for instance by using the FVGC as the aggregation function of the SAGE convolution. In convolutions like the invariant edge convolution (Chen et al., 2021), which already employ multiple filters and edge features, just the use of FVF as node and edge attributes in each convolution has to be implemented.

Any common 2D mesh typically used for CFD simulations, such as those with triangular or quadrilateral cells, can be reconstructed from its prescribed FVF. More specifically,

Theorem 3.1 (The Completeness of FVF). *Let M be a 2D mesh such that the cells along its farfield have no more than 2 boundary faces each, and the faces of each of its cells enclose a singular volume. If a graph $G = (V, E)$ is the cell centroid-based graph representation of M, the relative positions of all the nodes and faces of M can be uniquely deduced given V_i, $\forall i \in V$, and $\boldsymbol{S}_{f,ij}$, $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_i)$ and $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_j)$, $\forall (i, j) \in E$.*

The implementation details of the FVF and proof of this theorem are provided in the appendix.

3.3. Residual Training

In the previous subsections, we propose geometric features and FVF to improve the learning performance. In this subsection, we exploit low-resolution data and residual learning further improving the learning performance. Residual training is a well-known approach in image super-resolution (Zhang et al., 2018; Yang et al., 2019). The general idea is to train the network to predict the residual field $F - Upsample(F_{LR})$, where F_{LR} is the low-resolution field, instead of the original field F itself. However, the most common way of utilizing low-resolution data as reference in CFD–AI literature is to concatenate $Upsample(F_{LR})$ to one of the intermediate convolution layers (Belbute-Peres et al., 2020). The prior knowledge from the low-resolution flow field could be further utilised through the residual training scheme. Instead of minimising the loss $\mathcal{L}\left(F_{GT}, \widehat{F}\right)$, where $\widehat{F}$ is the predicted flow variable, F_{GT} is the corresponding ground truth, and $\mathcal{L}$ is an arbitrary loss criterion, the network minimises the residual loss $\mathcal{L}\left(F_{GT}, \widehat{F}_r + Upsample(F_{LR})\right)$. Here, $\widehat{F}_r$ is the predicted residual and the prediction is $\widehat{F} = \widehat{F}_r + Upsample(F_{LR})$. Since the low-resolution field is an approximation of the ground truth, much of the residual field will be close to zero. Thus, training the model to predict the residual field eases the learning, and helps the model to focus on the more nuanced areas where the low-resolution fields tend to be inaccurate. For instance, the areas around and downstream from internal geometries.

Kochkov et al. (2021) utilised a similar training scheme in the method of "learned correction", aimed at correcting a very low-resolution field using a CNN, without internal geometries, to an output of the same resolution. In the next section, this work intends to support the use of residual training to improve GNN performance for flows around immersed objects on a much finer mesh.

4. Experiments

We evaluate our proposed geometric and finite volume features on two databases, 2DSHAPES (Viquerat & Hachem, 2020; Chen et al., 2022) and AirfRANS (Bonnet et al., 2022a), and five state-of-the-art GNN methods and their respective training schemes in the CFD–AI literature, MeshGraphNet (Pfaff et al., 2021), BSMS-GCNN (Cao et al., 2023), Chen-GCNN (Chen et al., 2021), Graph U-Net (Bonnet et al., 2022a), and CFDGCN (Belbute-Peres et al., 2020).

While all state-of-the-art methods utilise mesh node-based graphs, we adapt both datasets to cell centroid-based graphs to train the models with FVF. Models that do not use FVF are trained on mesh node-based graphs to maintain consistency with the baselines. The effectiveness of residual training is demonstrated only on CFDGCN with the AirfRANS dataset because CFDGCN is the only method that takes in low-resolution data.

2DSHAPES is a collection of 2000 random shapes where each shape is generated by connected Bézier curves along with their steady-state velocity vector $[\bar{u}_x, \bar{u}_y]^T$ and pressure $\bar{p}$, at Reynolds number = 10. The training set, validation set, and test set consist of 1600, 200, and 200 shapes, respectively. We use this dataset to demonstrate the effectiveness of our features applied to the Chen-GCNN model.

AirfRANS is a high fidelity aerodynamics dataset of airfoil shapes. Their (RANS) solutions range from Reynolds numbers 2 to 6 million and angle-of-attacks $-5°$ to $+15°$ degrees. In their scarce data regime, the training set, validation set, and test set consist of respectively 180, 20, and 200 airfoils along with their steady-state velocity vector $[\bar{u}_x, \bar{u}_y]^T$, pressure $\bar{p}$, and turbulent viscosity ν_t. This dataset is used as-is to demonstrate the effectiveness of our features applied to the Graph U-Net. An amended dataset, which excludes the turbulent viscosity field, is likewise used for comparing BSMS-GNN and CFDGCN with and without the proposed features. Similarly, the dataset is amended to further exclude the pressure fields for the MeshGraphNet models. These fields are excluded to more closely match those predicted by the original methods. Additionally, we use the authors' codebase (Bonnet et al., 2022b) to generate RANS solutions of the same set of airfoils on coarser meshes ($\sim$ 20 times fewer nodes than the original), which we refer to as *coarse-AirfRANS* and utilise as a database of low-resolution reference flow fields to evaluate the effectiveness of residual training on CFDGCN.

Performance metrics. Our objective is to determine if incorporating the proposed features and residual training into the state-of-the-art methods significantly improves performance. We evaluate different methods with different measures to correspond with the metrics in their respective studies. To compare Chen-GCNN with our models derived from it, we use the MAE loss function, which is the summation of mean-absolute error of flow variables averaged over the test set. For Graph U-Net from Bonnet et al. (Bonnet et al., 2022a), the following three types of metrics are employed:

1. *Test loss:* Loss function by equation 3 in Bonnet et al. (2022a) evaluated on and averaged over the test set.
2. *Volume MSE:* Mean-squared errors of normalised flow field ($[\bar{u}_x, \bar{u}_y]^T$, $\bar{p}$, ν_t) predictions at internal nodes.
3. *Surface MSE:* Mean-squared error of normalised flow field predictions at boundary nodes. Due to boundary constraints, only pressure ($\bar{p}$) is non-zero at airfoil surfaces.

Finally, we evaluate the models derived from MeshGraphNet and CFDGCN using the MSE loss function, which is the mean-squared error of flow field prediction averaged over the test set, and the models derived from BSMS-GNN with the RMSE or root mean-squared error.

4.1. Results

MeshGraphNet (Pfaff et al., 2021). The MeshGraphNet model uses an encoder-processor-decoder architecture. The original convolution types were maintained in the experiments. Table 1 shows that the adoption of the geometric features (Geo) and the Finite Volume Features (FVF) reduced the MSE by $\sim$41% from the baseline.

Chen-GCNN (Chen et al., 2021). On the Chen-GCNN model using invariant edge convolutions, Table 2 shows that the adoption of the geometric features and finite volume features reduces MAE by $\sim$27% from the baseline. We also tested GCNN with SAGE convolution and obtained $\sim$82% reduction in MAE with respect to the baseline.

Table 1. Performance evaluation using MeshGraphNet on the AirfRANS dataset.

Models	MSE $\times 10^{-2}$
MeshGraphNet (Baseline)	5.7571
MeshGraphNet w/ Geo	3.4683
MeshGraphNet w/ FVF w/ Geo	**3.3811**

Table 2. Performance evaluation using Chen-GCNN on the 2DSHAPES dataset.

Models	Conv type	MAE $\times 10^{-2}$
Chen-GCNN (Baseline)	Invariant Edge Convolution	1.1590
Chen-GCNN w/ Geo		0.9727
Chen-GCNN w/ FVF w/ Geo		**0.8491**
Chen-GCNN (Baseline)	SAGE Convolution	6.7103
Chen-GCNN w/ Geo		3.8041
Chen-GCNN w/ FVF w/ Geo		**1.1982**

Table 3. Performance evaluation using BSMS-GNN on the AirfRANS dataset.

Models	RMSE $\times 10^{-2}$
BSMS-GNN (Baseline)	7.7589
BSMS-GNN w/ Geo	6.7498
BSMS-GNN w/ FVF w/ Geo	**6.1919**

Table 4. Performance evaluation using Graph U-Net on the AirfRANS dataset.

Models	Conv type	Test loss $\times 10^{-2}$	Volume MSE $\times 10^{-2}$ $\bar{u}_x$	$\bar{u}_y$	$\bar{p}$	ν_t	Surface MSE $\times 10^{-1}$ $\bar{p}$
Graph U-Net (Baseline)	SAGE Convolution	1.816	1.140	1.429	2.190	**2.492**	0.932
Graph U-Net w/ Geo		1.786	1.224	0.599	2.028	3.286	0.828
Graph U-Net w/ FVF w/ Geo		**1.441**	**1.061**	**0.515**	**1.368**	2.724	**0.549**
Graph U-Net (Baseline)	Vanilla Graph Convolution	15.310	19.249	12.845	14.041	15.190	4.633
Graph U-Net w/ Geo		12.709	17.408	11.256	10.106	12.146	3.631
Graph U-Net w/ FVF w/ Geo		**4.544**	**2.968**	**2.286**	**4.546**	**8.287**	**1.505**

BSMS-GNN (Cao et al., 2023). The Bi-Stride Multi-Scale GNN (BSMS-GNN) model also has an encoder-processor-decoder architecture, similar to the MeshGraph-Net. Likewise, only the original convolution types were used in the experiment. The BSMS-GNN is a multi-scale model, and hence the FVF implementations were only applied to the encoder, decoder, and first and last convolutions of the processor, where the graph is at full resolution. As can be seen in Table 3, the adoption of our methods results in a $\sim$20% reduction in the RMSE.

Graph U-Net (Bonnet et al., 2022a). We trained Graph U-Net architecture (baseline) with the same experimental set-up described in Appendix L of Bonnet et al. (Bonnet et al., 2022a). The reason for choosing Graph U-Net is that it is the best-performing model reported. We evaluated this baseline with two convolution types: SAGE convolution and vanilla graph convolution. In Table 4, we observe that, on SAGE convolution, the adoption of the geometric and finite volume features reduces the loss on the test set by about 21% as indicated by the baseline method's loss on the test set (1.816×10^{-2}) and that of Graph U-Net w/ FVF w/ Geo (1.441×10^{-2}). We also observe that our features improve the volume MSE of three out of four flow variables as well as the surface MSE of pressure. Do note that each flow variable was not handled separately by the training loss, but rather the MSE of all four as a whole. We also tested Graph U-Net with vanilla graph convolution and obtained 71% reduction in the loss on the test set w.r.t the baseline.

CFDGCN (Belbute-Peres et al., 2020). CFDGCN uses a CFD simulator in the training and testing scheme to generate low-resolution data and vanilla graph convolution to enhance learning on a high-resolution mesh with varying initial physical conditions, e.g., angle of attack and Mach number. Although CFDGCN generalises well to unknown physical conditions, it requires the geometry to remain fixed for the purpose of optimising the coarse mesh. This makes the model unable to generalise to different geometries and limits the method's practical functionality. Hence, in our adaption, we removed their differentiable CFD simulator component from the training and testing loop and directly passed the coarse-AirfRANS solutions for upsampling. As the experimental set-up assumed that a low-resolution flow field is available, we adopted and evaluated the proposed residual training scheme in addition to Geo and FVF. In Table 5, we observe that the predictive error reduces by about 41%, as indicated by the baseline method's MSE (0.1211×10^{-2}) and that of CFDGCN w/ residual training w/ FVF w/ Geo (0.0719×10^{-2}). Incorporating our proposed features and residual training scheme one by one consistently reduces the error, suggesting that they are generally effective. We also tested CFDGCN with SAGE convolution and obtained a 53% reduction in MSE w.r.t the baseline.

Table 5. Performance evaluation using CFDGCN on the AirfRANS dataset.

Models	Conv type	MSE $\times 10^{-2}$
CFDGCNN (Baseline)	Vanilla Graph Convolution	0.1211
CFDGCN w/ Geo		0.1093
CFDGCN w/ FVF w/ Geo		0.0918
CFDGCN w/ residual training w/ FVF w/ Geo		**0.0719**
CFDGCN (Baseline)	SAGE Convolution	0.1342
CFDGCNN w/ Geo		0.1092
CFDGCNN w/ FVF w/ Geo		0.0631
CFDGCNN w/ residual training w/ FVF w/ Geo		**0.0628**

Table 6. Computational time of SV, DID and FVF for two mesh resolutions.

	Average Computation Time (s) Coarse-AirfRANS	AirfRANS
SV	0.001	0.018
DID	0.031	4.276
FVF	0.001	0.032

4.2. Computation Time of the Geometric Features

We present the running time of computing the proposed features on Coarse-AirfRANS and (Fine) AirfRANS datasets, shown in Table 6. The average computation times of SV and

Table 7. Impact of Residual training on Chen-GCNN (IVE convolution) and MeshGraphNet using AirfRANS dataset.

MODELS	MSE $\times 10^{-2}$
CHEN-GCNN W/ FVF W/ GEO	5.6104
CHEN-GCNN W/ RESIDUAL TRAINING W/ FVF W/ GEO	**4.2166**
MESHGRAPHNET W/ FVF W/ GEO	3.3811
MESHGRAPHNET W/ RESIDUAL TRAINING W/ FVF W/ GEO	**1.8855**

FVF on coarse and fine meshes are quite reasonable. The reason the computational time of DID is relatively higher than that of other features is because the values of each angle segment are calculated consecutively rather than in parallel due to memory constraints. Note that the computational time can be reduced by calculating them in parallel or by using fewer angle segments.

4.3. Effectiveness of Residual Training

In order to evaluate the effectiveness of residual training more thoroughly, we have conducted experiments using the AirfRANS dataset with MeshGraphNet and Chen-GCNN with IVE convolution. The results are shown in Table 7. We observe that the mean-squared error reduces significantly due to the incorporation of residual training in both models. To be precise, we observe a 25% and 45% reduction in error of Chen-GCNN and MeshGraphNet models, respectively. These observations suggest that the residual training generally improves the learning performance.

5. Conclusion

This work presents two novel geometric representations, SV and DID, and the use of FVF in graph convolutions. The SV and DID provide a more complete representation of the geometry to each node. Moreover, the FVF enable the graph convolutions to more closely model the finite volume simulation method. Their effectiveness at reducing prediction error has been shown across two datasets, as well as five different state-of-the-art methods and training scenarios using various types of graph convolution. Additionally, this paper demonstrates the ability of residual training to further improve accuracy in scenarios with low-resolution data.

Limitations. The experiments assumed the availability of sufficient but not overwhelming data to an extent. The benefits of the proposed features over the baseline are likely to decrease as the amount of training data increases. This is because, with a large dataset, the networks can learn all necessary information from the data and thus rely less on the prior knowledge given in the proposed features. However, it should be emphasized that such a database would be extremely costly to simulate. Generalising the geometric features to a 3D case and testing our methods on models designed to predict turbulent flows have been left for future work. Finally, the effectiveness of residual training assumes the availability of additional data from low-resolution simulations as an approximate solution.

Acknowledgement

The research, undertaken in the Rolls-Royce Corporate Lab @ Nanyang Technological University, is supported by the Singapore Government (Industry Alignment fund IAF-ICP Grant - I1801E0033). We thank Yi Wang and Bryce Conduit from Rolls-Royce plc. for their feedback on this work. We also thank the authors of BSMS-GNN and Chen-GCNN for sharing their codes and data.

Impact Statement

This study introduces approaches to enhance AI predictions in CFD simulations. Similar to conventional CFD solvers, these methods find application in the design process of diverse components, encompassing those relevant to weaponry. The team will not actively pursue collaboration or knowledge-sharing with defence or military agencies.

References

Baque, P., Remelli, E., Fleuret, F., and Fua, P. Geodesic convolutional shape optimization. In *International Conference on Machine Learning*, pp. 472–481. PMLR, 2018.

Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Belbute-Peres, F. D. A., Economon, T., and Kolter, Z. Combining differentiable PDE solvers and graph neural networks for fluid flow prediction. In *International Conference on Machine Learning*, pp. 2402–2411. PMLR, 2020.

Bonnet, F., Mazari, J. A., Cinella, P., and Gallinari, P. AirfRANS: High fidelity computational fluid dynamics dataset for approximating Reynolds-averaged Navier–Stokes solutions. In *Neural Information Processing Systems (NeuRIPS 2022)*, 2022a.

Bonnet, F., Mazari, J. A., Cinella, P., and Gallinari, P. Github codebase of AirfRANS. https://github.com/Extrality/AirfRANS, 2022b.

Cao, Y., Chai, M., Li, M., and Jiang, C. Efficient learning of mesh-based physical simulation with bi-stride multi-scale graph neural network. In *Proceedings of the 40th*

International Conference on Machine Learning, pp. 3541–3558. PMLR, 23–29 Jul 2023.

Chaves, I., Duarte, L., Coltro, W., and Santos, D. Droplet length and generation rate investigation inside microfluidic devices by means of CFD simulations and experiments. *Chemical Engineering Research and Design*, 161: 260–270, 2020.

Chen, J., Hachem, E., and Viquerat, J. Graph neural networks for laminar flow prediction around random two-dimensional shapes. *Physics of Fluids*, 33(12):123607, 2021.

Chen, J., Viquerat, J., Heymes, F., and Hachem, E. A twin-decoder structure for incompressible laminar flow reconstruction with uncertainty estimation around 2D obstacles. *Neural Computing and Applications*, pp. 1–17, 2022.

Danel, T., Spurek, P., Tabor, J., Śmieja, M., Struski, Ł., Słowik, A., and Maziarka, Ł. Spatial graph convolutional networks. In *Neural Information Processing: 27th International Conference, ICONIP 2020, Bangkok, Thailand, November 18–22, 2020, Proceedings, Part V*, pp. 668–675. Springer, 2020.

Demirel, Y. *CFD Simulations of Marine Hydrodynamics*. MDPI AG, 2021. ISBN 9783036523354.

Fortunato, M., Pfaff, T., Wirnsberger, P., Pritzel, A., and Battaglia, P. Multiscale MeshGraphNets, 2022.

Gao, H. and Ji, S. Graph U-Nets. In *International Conference on Machine Learning*, pp. 2083–2092. PMLR, 2019.

Guo, X., Li, W., and Iorio, F. Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pp. 481–490, 2016.

Hamilton, W., Ying, Z., and Leskovec, J. Inductive representation learning on large graphs. *Advances in Neural Information Processing Systems*, 30, 2017.

Horie, M. and Mitsume, N. Physics-embedded neural networks: Graph neural PDE solvers with mixed boundary conditions. *Advances in Neural Information Processing Systems*, 35:23218–23229, 2022.

Kochkov, D., Smith, J. A., Alieva, A., Wang, Q., Brenner, M. P., and Hoyer, S. Machine learning-accelerated computational fluid dynamics. *Proceedings of the National Academy of Sciences*, 118(21):e2101784118, 2021.

Libao, E., Lee, M., Kim, S., and Lee, S.-H. Meshgraphnetrp: Improving generalization of gnn-based cloth simulation. pp. 1–7, 11 2023. doi: 10.1145/3623264.3624441.

Lino, M., Cantwell, C. D., Bharath, A. A., and Fotiadis, S. Simulating continuum mechanics with multi-scale graph neural networks. *CoRR*, abs/2106.04900, 2021. URL https://arxiv.org/abs/2106.04900.

Liu, B., Tang, J., Huang, H., and Lu, X.-Y. Deep learning methods for super-resolution reconstruction of turbulent flows. *Physics of Fluids*, 32(2):025105, 2020.

Martins, J. R. Aerodynamic design optimization: Challenges and perspectives. *Computers & Fluids*, 239: 105391, 2022.

Moukalled, F., Mangani, L., and Darwish, M. *The finite volume method in computational fluid dynamics: An advanced introduction with OpenFOAM® and Matlab®*. Springer, 2016.

Ogoke, F., Meidani, K., Hashemi, A., and Farimani, A. B. Graph convolutional neural networks for body force prediction. *arXiv preprint arXiv:2012.02232*, 2020.

Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A., and Battaglia, P. Learning mesh-based simulation with graph networks. In *International Conference on Learning Representations*, 2021.

Raissi, M., Perdikaris, P., and Karniadakis, G. E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019.

Ranade, R., Hill, C., and Pathak, J. DiscretizationNet: A machine-learning based solver for Navier–Stokes equations using finite volume discretization. *Computer Methods in Applied Mechanics and Engineering*, 378:113722, 2021.

Ronneberger, O., Fischer, P., and Brox, T. U-Net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pp. 234–241. Springer, 2015.

Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J., and Battaglia, P. Learning to simulate complex physics with graph networks. In *International Conference on Machine Learning*, pp. 8459–8468. PMLR, 2020.

Tasri, A. and Susilawati, A. Accuracy of compact-stencil interpolation algorithms for unstructured mesh finite volume solver. *Heliyon*, 7:e06875, 2021.

Vijayashree and Ganesan, V. Application of CFD for analysis and design of IC engines. *Advances in Internal Combustion Engine Research*, pp. 251–306, 2018.

Viquerat, J. and Hachem, E. A supervised neural network for drag prediction of arbitrary 2D shapes in laminar flows at low Reynolds number. *Computers & Fluids*, 210: 104645, 2020.

Yang, W., Zhang, X., Tian, Y., Wang, W., Xue, J.-H., and Liao, Q. Deep learning for single image super-resolution: A brief review. *IEEE Transactions on Multimedia*, 21 (12):3106–3121, 2019.

Zhang, S., Kwok, K. C., Liu, H., Jiang, Y., Dong, K., and Wang, B. A CFD study of wind assessment in urban topology with complex wind flow. *Sustainable Cities and Society*, 71:103006, 2021.

Zhang, Y., Tian, Y., Kong, Y., Zhong, B., and Fu, Y. Residual dense network for image super-resolution. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2472–2481, 2018.

Technical Appendix

A. Relation Between a CFD Mesh and its FVF

In this section, we show that any common 2D mesh can be reconstructed from the FVF. Specifically, assuming that in a 2D mesh,

1. the cells along the farfield have no more than two boundary faces each, and
2. the faces of each cell enclose a singular volume,

the relative positions of all the nodes and the faces of a mesh can be uniquely deduced given

1. the volumes of all cells V_i,
2. the face area normal vector $\mathbf{S}_{f,ij}$ of all internal and geometry boundary faces, and
3. the vector from the cell centroids to the face centroids $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_i)$ and $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_j)$ of all internal and geometry boundary faces.

By the "relative positions" $\boldsymbol{a}_i'$ of mesh nodes that have the absolute positions $\boldsymbol{a}_i$, we mean that there exists some constant α such that $\boldsymbol{a}_i = (\boldsymbol{a}_i' - \alpha), \forall i$. By the principle of translational invariance, the absolute positions of the mesh points are inconsequential to the flowfield as long as their relative positions remain the same. We make no further notational distinction between the two in this section.

Lemma A.1. *Given the (relative) position of the face centroid $\boldsymbol{c}_{f,ij}$ and face area normal vector $\boldsymbol{S}_{f,ij} = (S_x, S_y)$ of a face in a 2D mesh, the unique (relative) positions of the mesh nodes $\boldsymbol{a}, \boldsymbol{b}$ of the face, and the existence of the face, can be deduced.*

Proof. From the (relative) position of the face centroid $\boldsymbol{c}_{f,ij}$, we know that $(\boldsymbol{a} + \boldsymbol{b})/2 = \boldsymbol{c}_{f,ij}$. Likewise, we can assume without loss of generality that $(\boldsymbol{b} - \boldsymbol{a})$ is the face area normal vector $\mathbf{S}_{f,ij}$ rotated by a right angle clockwise, or $(\boldsymbol{b} - \boldsymbol{a}) = (S_y, -S_x)$. The unique (relative) positions of the mesh nodes $\boldsymbol{a}, \boldsymbol{b}$ can be solved from this system of two linear equations. The existence of a face shared by them is obvious. □

Lemma A.2. *Given the (relative) position of a cell's centroid $\boldsymbol{x}_i$ and the (relative) positions of all but one of its mesh nodes $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}$, the unknown (relative) position of its remaining mesh node $\boldsymbol{a}$ can be found to be $n\boldsymbol{x}_i - \sum_{j=1}^{n-1} \boldsymbol{b}_j$.*

Proof. From the (relative) position of the cell centroid $\boldsymbol{x}_i$, we know that $\frac{1}{n}\left(\boldsymbol{a} + \sum_{j=1}^{n-1} \boldsymbol{b}_j\right) = \boldsymbol{x}_i$. The rest is obvious. □

Lemma A.3. *If and only if $\boldsymbol{x}_i$ is the (relative) position of a cell centroid with the mesh nodes of (relative) positions $\boldsymbol{b}_1, \ldots, \boldsymbol{b}_{n-1}$, then $\boldsymbol{x}_i = n\boldsymbol{x}_i - \sum_{j=1}^{n-1} \boldsymbol{b}_j$.*

Proof.

$$\begin{aligned}\frac{1}{n-1}\sum_{j=1}^{n-1}\boldsymbol{b}_j = \boldsymbol{x}_i &\Leftrightarrow \sum_{j=1}^{n-1}\boldsymbol{b}_j = (n-1)\boldsymbol{x}_i \\ \Leftrightarrow \sum_{j=1}^{n-1}\boldsymbol{b}_j = n\boldsymbol{x}_i - \boldsymbol{x}_i &\Leftrightarrow \boldsymbol{x}_i = n\boldsymbol{x}_i - \sum_{j=1}^{n-1}\boldsymbol{b}_j\end{aligned}$$

□

Theorem A.4 (The Completeness of FVF)**.** *Let M be a 2D mesh such that the cells along its farfield have no more than 2 boundary faces each, and the faces of each of its cells enclose a singular volume. If a graph $G = (V, E)$ is the cell centroid-based graph representation of M, the relative positions of all the nodes and faces of M can be uniquely deduced given $\mathrm{V}_i,\ \forall i \in V$, and $\boldsymbol{S}_{f,ij}$, $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_i)$ and $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_j)$, $\forall (i,j) \in E$.*

Proof. By the cell-centroid based graph construction method, each cell of the input mesh corresponds to a node $i \in V$, and each internal face and geometry boundary face of the input mesh corresponds to an edge $(i, j) \in E$.

The vectors $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_i)$ and $(\boldsymbol{c}_{f,ij} - \boldsymbol{x}_j)$ are given in the edge attributes of the FVF. It is obvious that the positions of all face centroids $\boldsymbol{c}_{f,ij}$ and cell centres $\boldsymbol{x}_i, \boldsymbol{x}_j$ relative to one another in a connected mesh can be deduced from this. The face area vector $\mathbf{S}_{f,ij}$ is also included as an edge attribute. Hence, from Lemma A.1, all internal faces, geometry adjacent boundary faces, and their respective mesh nodes can be found, shown in Figure 5. This already accounts for the significant majority of the original mesh. All that remains are the edges along the farfield boundary, as well as the relative position of any mesh nodes that did not abut an internal face (such as those at corners).

As mentioned before, the relative positions of the cell centroids $\boldsymbol{x}_i$ can be deduced by the edge attributes. Assuming that these cells do not have more than 2 boundary faces along the farfield each, they will have at most 1 mesh node not associated with an internal edge. This is a fair assumption, considering the typical meshes with triangular or quadrilateral cells, commonly used for 2D CFD simulations. If there is a node with an unknown relative position, by Lemma A.2, it can be found as $n\boldsymbol{x}_i - \sum_{j=1}^{n-1} \boldsymbol{b}_j$ where $b_1, \ldots, b_{n-1}$ are the known mesh positions. Alternatively, if $n\boldsymbol{x}_i - \sum_{j=1}^{n-1} \boldsymbol{b}_j$ is equal to the cell centroid as in Lemma A.3, it can be concluded that $\{b_1, \ldots, b_{n-1}\}$ is the complete set of mesh nodes. Finally, the unique faces connecting these mesh nodes along the farfield can be deduced from the assumption that the faces of each cell along the farfield should enclose a singular volume. This is illustrated in Figure 6. $\square$

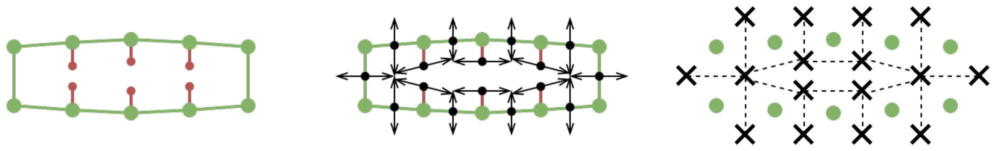
Figure 5. Obtaining the mesh nodes and faces of all internal and geometry boundary faces.

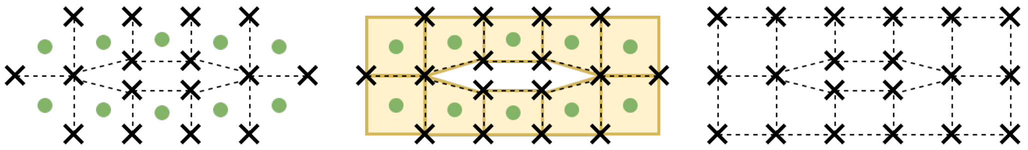
Figure 6. Obtaining corner mesh nodes and farfield boundary faces.

B. Implementation Details

B.1. Numerical DID Implementation

While the continuous DID has already been defined, the geometry boundary had to be represented as a parametric equation, which may not always be available. Instead, we calculate a discrete DID using numerical integration, described in detail in Algorithm 1. Alternatively, the numerical DID calculation of an angle range can be simplified as:

1. Obtain the mean distance to the node from every unobstructed boundary point: d.
2. Find the weighted proportion of the angle segment which faces the object boundary: w_θ.
3. The DID value is calculated to be the weighted average of the mean distance to the boundary and the ∞ parameter: $w_\theta \cdot d + (1 - w_\theta) \cdot \infty$.

It can be seen from Algorithm 1 that the process runs in $\mathcal{O}(|K| * |V|)$ time, where K is the set of boundary nodes and V is the set of all nodes. However, as $|K| << |V|$ in most practical CFD scenarios, the computation cost is almost linear to $|V|$.

Also note that a singular geometry was assumed. For multiple-geometry scenarios, it has to be considered that w_θ may not be represented by a single continuous angle range like $(\theta_{min}, \theta_{max})$. The rest of the algorithm may remain unchanged.

In all experiments, the starting and ending angles of the DID segments were $\left(\theta_j, \theta_j'\right) = \left(-\frac{\pi}{4}, \frac{\pi}{4}\right)$, $\left(0, \frac{\pi}{2}\right)$, $\left(\frac{\pi}{4}, \frac{3\pi}{4}\right)$, $\left(\frac{\pi}{2}, \pi\right)$, $\left(\frac{3\pi}{4}, \frac{5\pi}{4}\right)$, $\left(\pi, \frac{3\pi}{2}\right)$, $\left(\frac{5\pi}{4}, \frac{7\pi}{4}\right)$, $\left(\frac{3\pi}{2}, 2\pi\right)$. These were chosen to give 8 overlapping arcs each spanning $\frac{\pi}{2}$ degrees, centred at $\frac{\pi}{4}$ intervals. The respective weight functions $w_j(\theta)$ were uniform distributions over the $\left(\theta_j, \theta_j'\right)$ range. Finally, 4 was used as the large number ∞, as this was larger than any finite distance from a point in the mesh to the geometry boundary in any direction.

Algorithm 1 Calculation of the DID of a field

```
1: Input: Set of nodes $V$, positions of each node $pos = [(x_i, y_i) : i \in V]$ and boundary indices $bd = [k \in V : k$ is on the geometry boundary], starting and ending angles $\left[\left(\theta_j, \theta_j'\right) : 0 \le j < J\right]$, weight functions $[w_j : 0 \le j < J]$, and large value $\infty$.
2: Initialise $DID \leftarrow [\,]$
3: for $j \in [0, ..., J-1]$ do
4:     Initialise $DID_j \leftarrow [\,]$
5:     for $i \in V$ do
6:         Initialise $\theta_{min}, \theta_{max} \leftarrow \theta_j', \theta_j$
7:         Initialise $DID_i \leftarrow 0$
8:         Initialise $w \leftarrow 0$
9:         for $k \in bd$ do
10:            Initialise $\theta_{i,k} \leftarrow \tanh\left((y_j - y_i)/(x_k - x_i)\right)$
11:            if $(\theta_j < \theta_{i,k} < \theta_j')$ and $(k$ is unobstructed from $i)$ then
12:                $DID_i \leftarrow DID_i + w_j(\theta_{i,k}) * \min(||(x_i, y_i) - (x_k, y_k)||, \infty)$
13:                $w \leftarrow w + w_j(\theta_{i,k})$
14:                $\theta_{min} \leftarrow \theta_{i,k}$ if $\theta_{i,k} < \theta_{min}$
15:                $\theta_{max} \leftarrow \theta_{i,k}$ if $\theta_{i,k} > \theta_{max}$
16:            end if
17:        end for
18:        $w_\theta \leftarrow \left(\int_{\theta_{min}}^{\theta_{max}} w_j(\theta)\, d\theta\right) / \left(\int_{\theta_j}^{\theta_j'} w_j(\theta)\, d\theta\right)$
19:        $DID_i \leftarrow DID_i / w$
20:        $DID_i \leftarrow w_\theta * DID_i + (1 - w_\theta) * \infty$
21:    end for
22:    $DID_j.append(DID_i)$
23: end for
24: $DID.append(DID_j)$
25: Return: $DID$
```

B.2. FVF implementation

Vanilla Graph Convolution and SAGE Convolution For models using vanilla graph convolutions, the convolutions were entirely replaced by the FVGC earlier described when FVF was used. For models using SAGE convolutions, FVF was used by implementing the FVGC as the aggregation function of the SAGE convolution. In both cases, the hidden dimension, or the number of filters, used in each convolution was always 3 for all convolutions in all experiments.

Invariant Edge Convolution Following the notation of the original paper, the Invariant Edge (IVE) convolutions used by Chen et al. (2021) update the edge features $\boldsymbol{x}_e$ and node features $\boldsymbol{x}_v$ into $\boldsymbol{x}_e'$ and $\boldsymbol{x}_v'$ respectively, according to the following rules

$$\boldsymbol{x}_e' = f_e\left(\frac{\boldsymbol{x}_{v_1} + \boldsymbol{x}_{v_2}}{2}, \frac{|\boldsymbol{x}_{v_1} - \boldsymbol{x}_{v_2}|}{2}, \boldsymbol{x}_e\right) , \tag{9}$$

$$\boldsymbol{x}_v' = f_v\left(\boldsymbol{x}_v, \sum_{e_i \in N(v)} \boldsymbol{x}_{e_i}'\right) , \tag{10}$$

where v_1 and v_2 are the two nodes connected by edge e, $N(v)$ is the set of neighbouring edges around node v, and f_e and f_v are both MLPs with 1 hidden layer of 128 neurons. The number of neurons in the output layer, and hence output dimension of the kernel, can be customised. This is similar to the use of multiple filters in FVGC. Hence, when using FVF with IVE convolutions, only the node and edge attributes have to be implemented, resulting in the following rules

$$\boldsymbol{x}_e^* = \boldsymbol{q}_e \oplus \boldsymbol{x}_e \ , \quad \boldsymbol{x}_v^* = \boldsymbol{x}_v \oplus \boldsymbol{p} \ , \tag{11}$$

$$\boldsymbol{x}_e' = f_e\left(\frac{\boldsymbol{x}_{v_1}^* + \boldsymbol{x}_{v_2}^*}{2}, \frac{|\boldsymbol{x}_{v_1}^* - \boldsymbol{x}_{v_2}^*|}{2}, \boldsymbol{x}_e^*\right) \ , \tag{12}$$

$$\boldsymbol{x}_v' = f_v\left(\boldsymbol{x}_v^*, \sum_{e_i \in N(v)} \boldsymbol{x}_{e_i}'\right) \ , \tag{13}$$

where $\boldsymbol{p}$ is the node attributes and $\boldsymbol{q_e}$ is the edge attributes of the edge e. Note that the new edge and node features are still $\boldsymbol{x}_e'$ and $\boldsymbol{x}_v'$, while $\boldsymbol{x}_e^*$ and $\boldsymbol{x}_v^*$ are just intermediate calculations.

Unlike in vanilla graph convolutions or SAGE convolutions, the hidden dimension is not fixed at 3, but will instead follow the number of intermediate edge features used in the original work as described in the next section.

MeshGraphNet Convolution Still following the notation of the original paper, the MeshGraphNet processor designed by Pfaff et al. (2021) used convolutions that updated the edge features $\boldsymbol{e}^M$ and node features $\boldsymbol{v}$ into $\boldsymbol{e}'^M$ and $\boldsymbol{v}'$ according to the following

$$\begin{aligned} \boldsymbol{e}_{ij}'^M &= f^M(\boldsymbol{e}_{ij}^M, \boldsymbol{v}_i, \boldsymbol{v}_j) \ , \\ \boldsymbol{v}_i' &= f^V(\boldsymbol{v}_i, \sum_j \boldsymbol{e}_{ij}'^M) \ . \end{aligned} \tag{14}$$

Where v_i and v_j are the features of two nodes connected by an edge with edge features e_{ij}^M, and f^M and f^V are MLPs with 2 hidden layers of size 128 each. In all MeshGraphNet implementations, they have an output of size 128 as well. When using FVF, it was changed to the following

$$\begin{aligned} \boldsymbol{e}_{ij}^{*M} &= \boldsymbol{e}_{ij}^M \oplus \boldsymbol{q}_{ij}, \quad \boldsymbol{v}_i^* = \boldsymbol{v}_i \oplus \boldsymbol{p}_i \ , \\ \boldsymbol{e}_{ij}'^M &= f^M(\boldsymbol{e}_{ij}^{*M}, \boldsymbol{v}_i^*, \boldsymbol{v}_j^*) \ , \\ \boldsymbol{v}_i' &= f^V(\boldsymbol{v}_i^*, \sum_j \boldsymbol{e}_{ij}'^M) \ . \end{aligned} \tag{15}$$

We also appended the FV node attributes and FV edge attributes to the inputs of the encoder and decoder convolutions accordingly.

BSMS-GNN Convolution Likewise, the original BSMS-GNN processor designed by Cao et al. (2023) used convolutions to update the edge features $\boldsymbol{e}_l^s$ and node features $\boldsymbol{v}_l$ at level l for a problem involving S edge sets like so

$$\begin{aligned} \boldsymbol{e}_{l,ij}^s &= f_l^s(\Delta\boldsymbol{x}_{l,ij}, \boldsymbol{v}_{l,i}, \boldsymbol{v}_{l,j}), \quad s = 1, \dots, S \ , \\ \boldsymbol{v}_{l,i}' &= \boldsymbol{v}_{l,i} + f_l^V(\boldsymbol{v}_{l,i}, \sum_j \boldsymbol{e}_{l,ij}^1, \dots, \sum_j \boldsymbol{e}_{l,ij}^S) \ . \end{aligned} \tag{16}$$

Where $\Delta\boldsymbol{x}_{l,ij} = \boldsymbol{x}_i - \boldsymbol{x}_j$ is the relative positions of the nodes i and j, and f_l^s and f_l^V are MLPs with 2 hidden layers of a hidden dimension of 128. For all BSMS-GNN implementations, an output of size 128 was used as well. With the FVF, it was changed into

$$\begin{aligned} \boldsymbol{v}_{l,i}^* &= \boldsymbol{v}_{l,i} \oplus p_i \quad , \\ \boldsymbol{e}_{l,ij}^s &= f_l^s(\Delta x_{l,ij}, \boldsymbol{v}_{l,i}^*, \boldsymbol{v}_{l,j}^*, q_{ij}), \quad s = 1, \dots, S, \\ \boldsymbol{v}_{l,i}' &= \boldsymbol{v}_{l,i}^* + f_l^V(\boldsymbol{v}_{l,i}^*, \sum_j \boldsymbol{e}_{l,ij}^1, \dots, \sum_j \boldsymbol{e}_{l,ij}^S) \end{aligned} \tag{17}$$

As mentioned before, this was only done for the first and last convolutions of the processor where the graph is at full resolution. Similar to the MeshGraphNet convolution implementation, the FV node attributes were appended to the inputs of the encoder and decoder convolutions as well.

Table 8. Sizes of the MeshGraphNet models compared.

Models	Number of trainable parameters
MeshGraphNet (Baseline)	2282370
MeshGraphNet w/ Geo	2283650
MeshGraphNet w/ FVF w/ Geo	2302470
MeshGraphNet w/ Residual Training w/ FVF w/ Geo	2302470

Table 9. Sizes of the BSMS-GNN models compared.

Models	Number of trainable parameters
BSMS-GNN (Baseline)	2578947
BSMS-GNN w/ Geo	2580227
BSMS-GNN w/ FVF w/ Geo	2582787

B.3. Model Architectures and Hyperparameter Choices

MeshGraphNet The MeshGraphNet used an encoder-processor-decoder structure. The encoder consisted of two MLPs, for encoding node and edge features respectively. The input node and edge dimensions were both 3 for the baseline model. The number of input node channels increased to 13 if SV and DID were being used. Likewise, the input node channels increased again to 14, and the input edge features to 9, if FVF was implemented. It had a hidden and output size of 128. The decoder consisted of just one MLP to decode the node features only, with an input and hidden size of 128. If FVF was being used, the input size increased to 129. The ouput size was 2, for the velocity fields.

The processor had 15 layers of the MeshGraphNet convolutions as earlier described. The input node and edge sizes were 128 unless FVF was implemented when the number of node channels increased to 129, and likewise, the edge channels increased to 134. The number of output nodes and edge channels was consistently 128. All activations used in the model were ReLU. The resulting number of trainable parameters in each model is shown in Table 8.

Each model was trained for 250 epochs on half-precision, using a mean squared error (MSE) training loss with a learning rate of 0.0001 and a batch size of 1.

BSMS-GNN The BSMS-GNN also had an encoder-processor-decoder structure. The encoder only consisted of one MLP of input size 1 that represented the node type. When SV and DID were used, this increased to 11. Likewise, when FVF was implemented, it increased again to 12. It had two hidden layers with a hidden size of 128 and an output of size 128 as well. The decoder was similarly an MLP with an input of size 128, or 129 if FV was used, and had 2 hidden layers with hidden size 128 and an output size of 3 for the velocity and pressure fields.

The processor had an architecture similar to that of the Graph U-Net, with 9 different scales or levels and only 1 convolution per level. The number of nodes would decrease progressively from the bi-stride pooling method explained in the original paper. The input and output node and edge feature dimensions were 128 each, except when FVF was implemented, and the input node and edge features increased to 129 and 134, respectively. All activations used in the model were ReLU. The resulting number of trainable parameters in each model is shown in Table 9.

Each model was trained on half-precision, using a mean squared error (MSE) training loss with a starting learning rate of 0.0001 that decayed exponentially at a decay rate of 0.7943 until the learning rate hit a minimum of 0.000001. All BSMS-GNN models were trained for 200 epochs with a batch size of 1.

Chen-GCNN The input node features to the baseline model consisted of a boolean representation of the geometry and the spatial coordinates. For models using IVE convolution, there were also input edge features, which were simply the average of the node features of each node in the edge. Thus, the input node features and input edge features both have a size of 3. When geometry features were used, SV and DID replaced the boolean representation, making the feature sizes 12 instead.

The model itself had eight convolution and smoothing layers, and finally a 1×1 convolution output layer. When IVE convolutions were used, the intermediate edge and node feature dimensions in each layer were $(4, 8, 16, 32, 64, 64, 32, 16)$

Table 10. Sizes of the Chen-GCNN models compared.

Models	Conv type	Number of trainable parameters
Chen-GCNN (Baseline)	Invariant Edge Convolution	217853
Chen-GCNN w/ Geo		222461
Chen-GCNN w/ FVF w/ Geo		227185
Chen-GCNN (Baseline)	SAGE Convolution	20193
Chen-GCNN w/ Geo		20337
Chen-GCNN w/ FVF w/ Geo		58067

Table 11. Sizes of the Graph U-Net models compared.

Models	Conv type	Number of trainable parameters
Graph U-Net (Baseline)	SAGE Convolution	65820
Graph U-Net w/ Geo		66396
Graph U-Net w/ FVF w/ Geo		160465
Graph U-Net (Baseline)	Vanilla Graph Convolution	107667
Graph U-Net w/ Geo		108243
Graph U-Net w/ FVF w/ Geo		105201

and $(8, 16, 32, 64, 64, 32, 16, 8)$ respectively, as they were in the original work. When SAGE convolutions were used, there were no intermediate edge feature dimensions. Instead, the number of filters used in the convolutions was fixed at 3, and the node feature dimensions remained $(8, 16, 32, 64, 64, 32, 16, 8)$ as in the IVE convolutions. The final output layer produced 3 node features representing predicted velocity and pressure. There are also skip connections from the input graph to the output of every convolutional and smoothing block, where the spatial coordinates of the nodes were concatenated to the node features. This is similar to the node attributes concatenated to the node features in the convolutions using FVF, although the cell volume was not used as a node attribute in the original work.

Table 10 shows the number of trainable parameters in each model. The model depth and width were kept the same as the original Chen-GCNN model (Chen et al., 2021) as it was already relatively small.

In training, the loss function chosen was the mean absolute error (MAE). We implemented early stopping, where training terminated if the MAE on the validation set did not improve after 60 epochs, or if it reached the maximum of 1000 epochs. A batch size of 64 was used, and the initial learning rate and decay rate were both 0.002 to match the original work, with the decay schedule also following the same formula. Further details can be found in the original codebase[1]. The only difference in our training scheme implementation would be that we used half-precision training, and clipped the gradients to 5.

Graph U-Net Our implementation of the Graph U-Net was adapted from the original codebase[2]. The baseline model started and ended with an MLP encoder and decoder of layer sizes [7, 64, 64, 8] and [8, 64, 64, 4] respectively. The models that used geometry features had an encoder with layer sizes [16, 64, 64, 8] instead, as the SDF input feature was replaced with SV and DID.

For the U-Net itself, the models that did not use FVF both followed the original Graph U-Net architecture exactly. The downward and upward passes had five scales each. At each scale of the downward pass, the number of node features doubled, while the graph nodes were down-sampled by half to create radius graphs of radii 5 cm, 20 cm, 50 cm, 1 m and 10 m respectively. In the upward pass, skip connections concatenated the node feautures of the respective scale in the downward pass to that of the previous scale. In models that used FVF, however, no down-sampling or up-sampling was done to preserve the mesh structure and its corresponding mesh characteristics. Nevertheless, the depth of the model and the presence of skip-links remained the same.

Table 11 shows the number of trainable parameters in each model. Models that used SAGE convolutions kept the original

[1]`https://github.com/cfl-minds/gnn_laminar_flow`
[2]`https://github.com/Extrality/AirfRANS`

Table 12. Sizes of the CFDGCN models compared.

Models	Conv type	Model width	#Trainable parameters
CFDGCN (Baseline)	Vanilla Graph Convolution	512	1057283
CFDGCN w/ Geo		512	1061891
CFDGCN w/ FVF w/ Geo		284	1021780
CFDGCN w/ Residual Training w/ FVF w/ Geo		284	1021780
CFDGCN (Baseline)	SAGE Convolution	512	2112003
CFDGCN w/ Geo		512	2121219
CFDGCN w/ FVF w/ Geo		315	2153241
CFDGCN w/ Residual Training w/ FVF w/ Geo		315	2153241

Graph U-Net model width of 8 (Bonnet et al., 2022a), causing the model sizes to increase from the use of SV, DID, and FVF. This is to be consistent with the experimental practices of Bonnet et al. (2022a), who allowed their model sizes to vary. On the other hand, when vanilla graph convolutions were used, the width of the models without FVF was increased to 15 to make the model sizes similar to the model with FVF. This is because the models with only vanilla graph convolution without FVF were too small to train effectively when using the original model parameters.

The learning rate for the experiments was set with a one-cycle cosine rate capped at 0.001, with the number of epochs fixed at 1600 as in the scarce data regime of the original work (Bonnet et al., 2022a). Half precision was used, and gradients clipped to 5.

For models that did not use FVF, mesh node-based graphs were used. However, the total number of nodes was fixed at 32000 via random subsampling of the input. Edges were formed between nodes within 5 cm of each other, with each node having a maximum of 64 neighbours. In training, each input graph was only subsampled once, and the training loss found against the subsampled ground-truth. In testing, subsampling was done repeatedly till a value was found for all the nodes in the original graph, and nodes with multiple values were assigned the average, before the test losses found against the unsampled ground-truth. On the other hand, on FVF models, an unsampled cell centroid-based graph was used in both training and testing. As before, this was to maintain the mesh's structure and characteristics represented in the FVF.

CFDGCN As explained earlier, to adapt the CFDGCN model to become generalisable to different geometries, we used the coarse mesh of each flow scenario in the data as a fixed input to the model, rather than as a trainable feature of the model. On all other aspects, however, the architecture of the CFDGCN was largely preserved.

The coarse mesh was upsampled once using squared distance-weighted, k-nearest neighbours interpolation to size of the fine mesh. The input node features of the baseline model were the spatial coordinates, angle of attack, mach number and SDF. If geometry features were used, the SDF was replaced by SV and DID. The graph was passed through 3 graph convolutions before the upsampled coarse mesh was appended to the output of the 3rd layer, and another 3 convolutions was performed to generate the final prediction.

All convolutions in models that did not use FVF had 512 hidden channels, just as the original CFDGCN did (Belbute-Peres et al., 2020). However, the number of hidden channels was adjusted for the FV and residual models to keep model size similar for comparability. Table 12 shows the number of trainable parameters in each model. A batch size of 1 was used on all experiments, and half precision was used. All other training parameters were kept the same as the original work. More details can be found from the codebase[3].

C. Coarse AirfRANS Dataset

AirfRANS (Bonnet et al., 2022a) provides 1000 simulated airfoil cases but with a consistent mesh resolution. Having simulations of the same airfoil geometry but from a lower resolution is important for learning tasks such as super-resolution, which is needed for CFDGCN (Belbute-Peres et al., 2020). Since the residual training scheme is intended, albeit not limited, to demonstrate superior performance on super-resolution task, a lower-resolution variant of the AirfRANS cases were needed. The AirfRANS authors released the mesh generation script, which made it possible.

[3]`https://github.com/locuslab/cfd-gcn`

Table 13. The new values of different cell grading for coarse AirfRANS dataset.

CELL GRADING	NEW VALUES
yGRADING	1000
yUGRADING	1000
yDGRADING	1000
xUUGRADING	10
xDUGRADING	10
xUMAEROGRADING	2
xDMAEROGRADING	2
xMGRADING	1
xDTRAILGRADING	0.0001
xUDGRADING	0.5
xDDGRADING	0.5
leadUGRADING	0.05
leadDGRADING	0.05

Table 14. Extrapolation and full data results of the CFDGCN models with vanilla graph convolution

MODELS	DATA REGIME	MSE $\times 10^{-2}$
CFDGCN (BASELINE)	REYNOLD'S NUMBER	0.3210
CFDGCN W/ GEO		0.0889
CFDGCN W/ FVF W/ GEO		0.0678
CFDGCN W/ RESIDUAL TRAINING W/ FVF W/ GEO		**0.0661**
CFDGCN (BASELINE)	AOA	0.9098
CFDGCN W/ GEO		0.1846
CFDGCN W/ FVF W/ GEO		0.1772
CFDGCN W/ RESIDUAL TRAINING W/ FVF W/ GEO		**0.1595**
CFDGCN (BASELINE)	FULL	0.0737
CFDGCN W/ GEO		0.0668
CFDGCN W/ FVF W/ GEO		0.0655
CFDGCN W/ RESIDUAL TRAINING W/ FVF W/ GEO		**0.0499**

The majority of settings in the original AirfRANS dataset remain unchanged, except for the mesh resolution. In this modified dataset, the number of cells in all directions has been reduced to one-quarter of the original settings. Additionally, the gradings of the cell expansion have been reset to ensure a smooth transition of the cell thickness as listed in Table 13.

D. Computational Environment

All experiments are run on a server with 32 Intel(R) Xeon(R) Silver 4110 CPU @ 2.10GHz processors, 256 GB RAM, and 3 dedicated Nvidia V100 GPU cards each with 32GB memory. All models are trained on a single GPU.

E. Additional Results

E.1. Extrapolation and Full Data Regimes

While the experiments using the AirfRANS dataset were done using the scarce data regime, more CFDGCN models were trained using the Reynold's Number and AoA extrapolation regimes and the full data regime. The results are shown in Table 14. They demonstrate that the proposed methods remain effective under a variety of scenarios with different data availability.

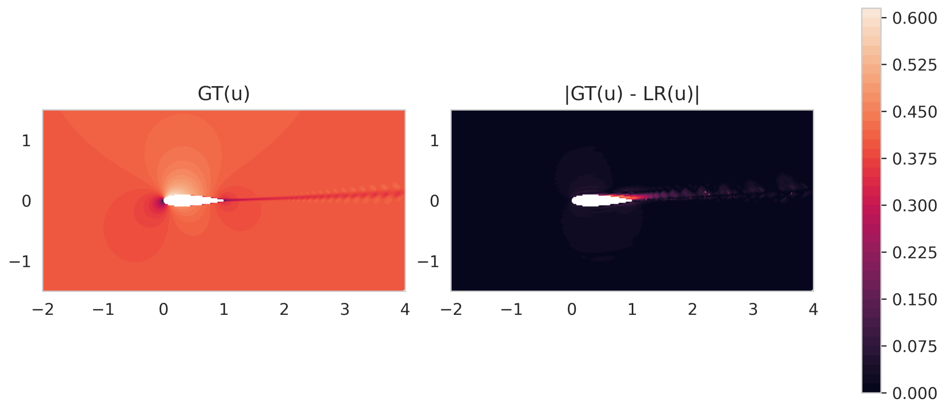

Figure 7. Velocity x-component (u)

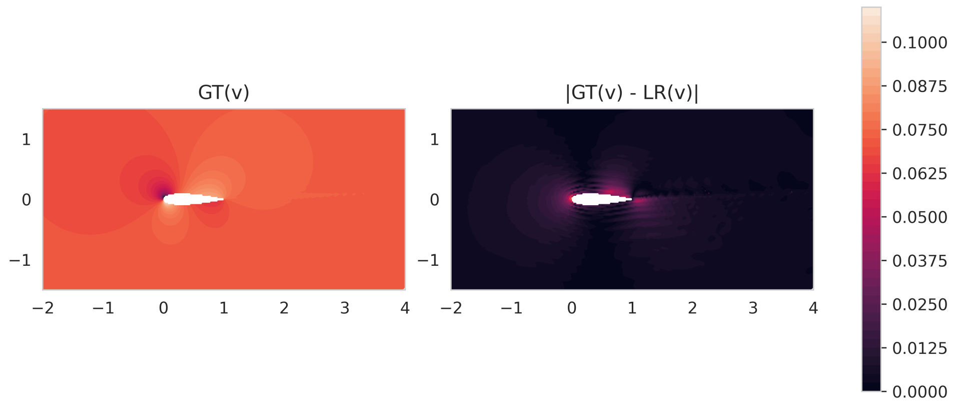

Figure 8. Velocity y-component (v).

E.2. Rationale for residual training

As mentioned earlier, the residual field helps the model to focus on the more nuanced areas where the low-resolution fields tend to be inaccurate. In Figures 7 and 8, we illustrate this phenomenon to emphasise the importance of residual training. We show ground truth velocity components (u,v) and pressure (p) corresponding to the ground truth flow field on the left, while the corresponding residual fields are on the right. In most of the regions, the residual is close to zero. Hence, the model has less difficulty learning flow field values in these regions, which are coloured in darker shades.

F. Predictive Error Visualisation

Figure 10 shows a comparison between the predicted velocity x-component flow fields, u, of the baseline method Chen-GCNN and Chen-GCNN w/ FVF w/ Geo. Both models used invariant edge convolutions. We observe that the predictions of both methods capture the flow features present in the Ground truth, GT(u). A close inspection of the absolute error of the predicted flow field in figure 11 reveals that Chen-GCNN w/ FVF w/ Geo produces relatively small regions with high error, whereas there are many regions with high error in the flow field predicted by Chen-GCNN. This observation indicates that incorporating our proposed features reduces the predictive error across different regions in the domain.

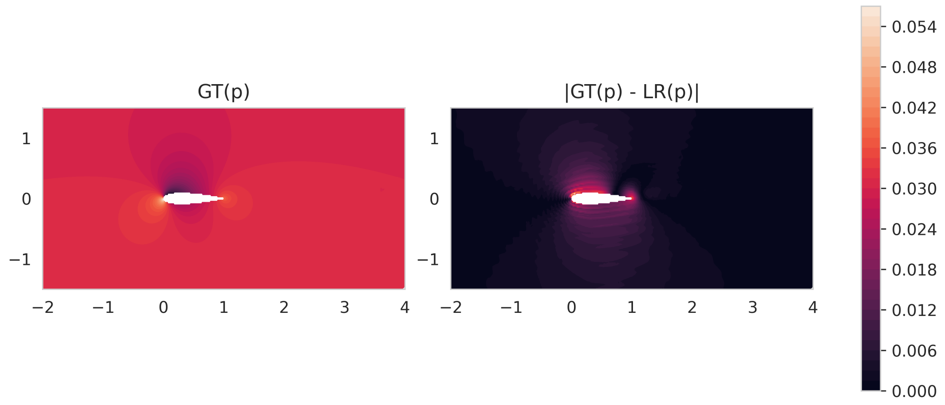

Figure 9. Pressure (p).

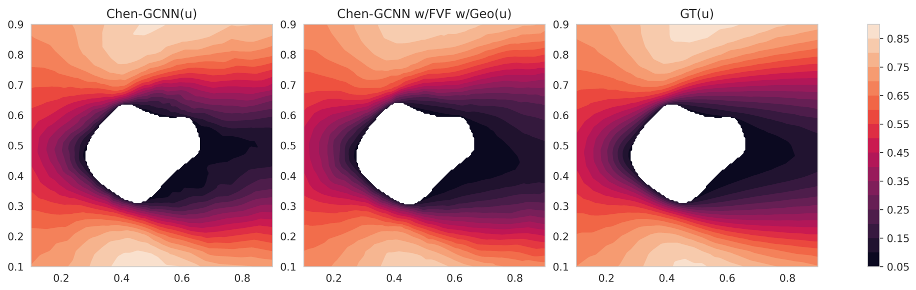

Figure 10. Velocity x-component (u) visualisation.

G. Boundary Layer Profile

We analyse the velocity profile at the boundary layer in Figure 12. We plot the normalised distance from the airfoil boundary (y/c) vs the normalised velocity x-component u/U_∞. Figure 12 shows that GUNet w/ Geo predicts velocity x-component near the airfoil boundary more accurately compared to the baseline GUNet.

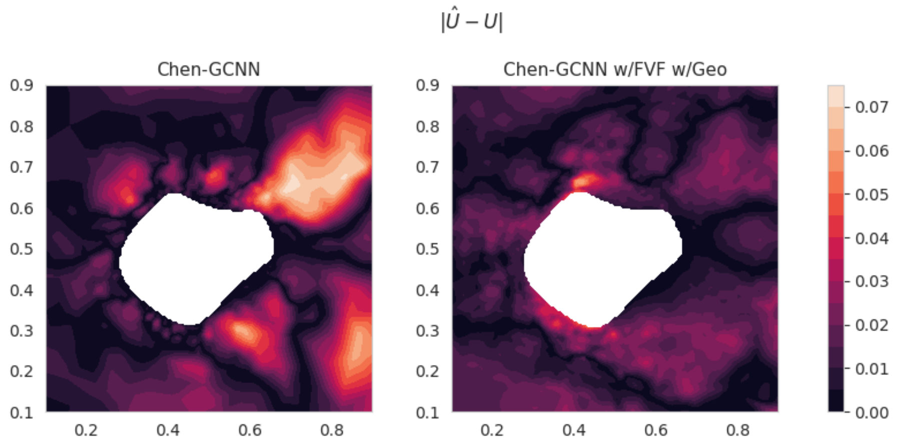

Figure 11. Absolute error of predictions.

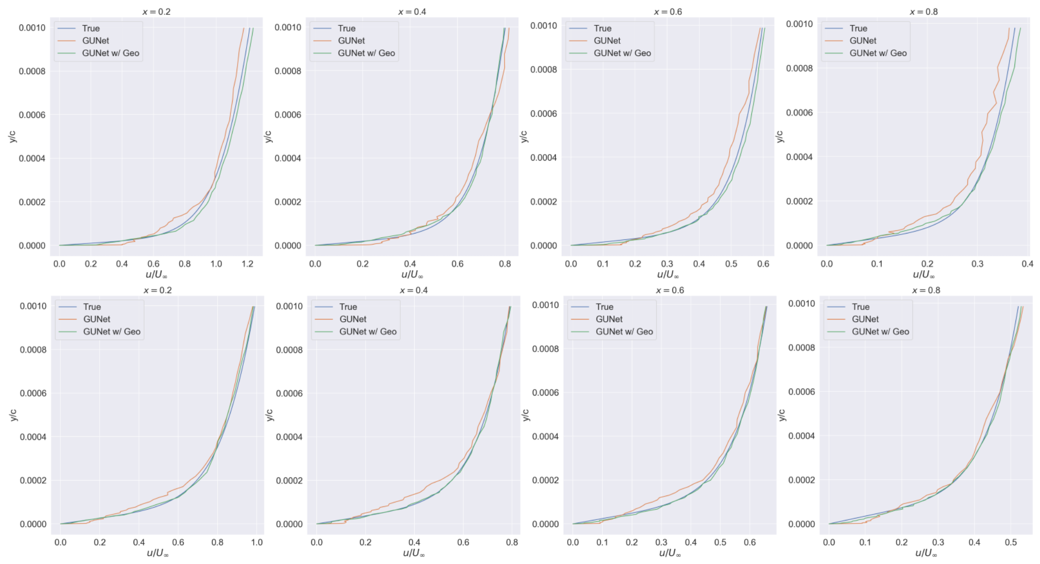

Figure 12. Comparison of the predicted boundary layers profiles on two random test airfoils at four different abscissas in the scarce data regime with respect to the true ones. Each row of plots represents a different airfoil, and each column of plots represents a different abscissa. The x component of the velocity is denoted by u, and the inlet velocity is denoted by U_∞.