Journal of Computational Physics 524 (2025) 113708

Contents lists available at ScienceDirect

Journal of Computational Physics

journal homepage: www.elsevier.com/locate/jcp

Predicting transonic flowfields in non–homogeneous unstructured grids using autoencoder graph convolutional networks

Gabriele Immordino [a,b,*], Andrea Vaiuso [b], Andrea Da Ronch [a], Marcello Righi [b]

[a] Faculty of Engineering and Physical Sciences, University of Southampton, Southampton, United Kingdom
[b] School of Engineering, Zurich University of Applied Sciences ZHAW, Winterthur, Switzerland

A B S T R A C T

This paper addresses the challenges posed by non-homogeneous unstructured grids, which are commonly used in computational fluid dynamics. The prevalence of these grids in fluid dynamics scenarios has driven the exploration of innovative approaches for generating reduced-order models. Our approach leverages geometric deep learning, specifically through the use of an autoencoder architecture built on graph convolutional networks. This architecture enhances prediction accuracy by propagating information to distant nodes and emphasizing influential points. Key innovations include a dimensionality reduction module based on pressure-gradient values, fast connectivity reconstruction using Mahalanobis distance, optimization of the network architecture, and a physics-informed loss function based on aerodynamic coefficient. These advancements result in a more robust and accurate predictive model, achieving systematically lower errors compared to previous graph-based methods. The proposed methodology is validated through two distinct test cases—wing-only and wing-body configurations—demonstrating precise reconstruction of steady-state distributed quantities within a two-dimensional parametric space.

1. Introduction

In recent years, addressing problems characterized by non-homogeneous and unstructured grids has become a central topic of research in the field of aerospace engineering. A pertinent example lies within the Computational Fluid Dynamics (CFD) field, where the initial step involves mesh generation, entailing the discretization of the fluid domain through the finite volume method. This mesh serves as a computational grid that enables the simulation of fluid flow and related phenomena within a defined space. A non-homogeneous unstructured grid is characterized by irregularly shaped elements (such as triangles or tetrahedras) connected in a non-regular pattern. The spacing between grid points varies across the domain, providing greater resolution in areas of interest, such as regions with complex geometries or flow features, while optimizing computational resources in less critical areas.

The complexities inherent in non-homogeneous unstructured geometries, especially when predicting intricate fluid flow scenarios, have driven the need for innovative approaches in generating reduced–order models (ROMs). Traditionally, methods based on classical numerical analysis, such as Proper Orthogonal Decomposition, Isomap and manifold learning [1–4], have been extensively employed due to their computational efficiency and reliability. However, as the complexity of grids increased, these traditional methods proved inadequate in capturing nonlinear dynamics, handling irregular geometries, and scaling to high-dimensional data.

In response to these challenges, machine learning emerged as a promising avenue, offering new ways to handle complex, non-traditional data structures. Initial efforts focused on deep neural networks, which demonstrated significant success in capturing intricate patterns and relationships within the fluid dynamics domain [5–7]. Despite these successes, the unique challenges posed by

* Corresponding author at: Faculty of Engineering and Physical Sciences, University of Southampton, Southampton, United Kingdom.
E-mail address: G.Immordino@soton.ac.uk (G. Immordino).

https://doi.org/10.1016/j.jcp.2024.113708
Received 26 April 2024; Received in revised form 11 October 2024; Accepted 24 December 2024

Available online 2 January 2025

Nomenclature

Acronyms

GB-AE-GCN	gradient-based autoencoder graph convolutional network
CFD	computational fluid dynamics
GCN	graph convolutional network
GNN	graph neural network
LHS	Latin Hypercube Sampling
ML	machine–learning
MAE	mean absolute error
MAPE	mean absolute percentage error
MSE	mean squared error
MWLSI	moving weighted least squares interpolation
ROM	reduced–order model

Symbols

AoA	angle of attack, deg
c	mean chord, m
C_D	drag coefficient
C_F	skin friction coefficient
C_L	lift coefficient
C_{My}	pitching moment coefficient
C_P	pressure coefficient
M	Mach number
Re	Reynolds number

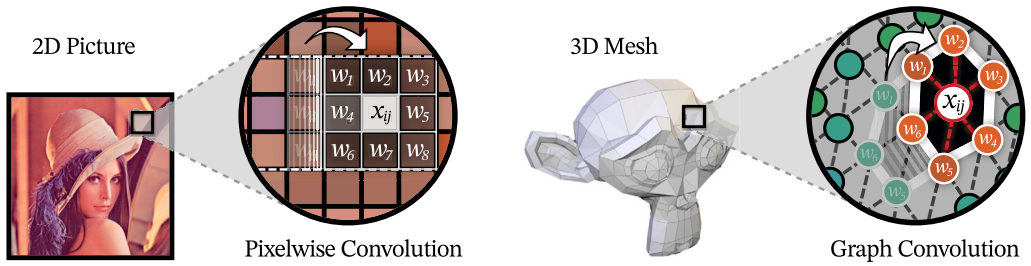


Fig. 1. Visual comparison between pixelwise convolution on a 2D digital image and graph convolution on a 3D mesh.

non-homogeneous unstructured grids—such as irregular node spacing, varying connectivity, and high dimensionality—necessitated the development of more sophisticated architectures to fully leverage the potential of machine learning in this field.

This need for advanced architectures led to the concept of geometric deep learning, which emerged around 2017 [8]. This approach introduced the use of graph-structured data prediction through graph neural network (GNN) architectures [9], specifically designed for applications involving interconnected entities. GNNs excel in capturing intricate relationships and dependencies within graph nodes and their connections [10–12]. The ability of GNNs to consider both local and global context through neighborhood aggregation mechanisms makes them well-suited for tasks where topological information is critical. These versatile networks have found extensive application as a foundation for solving classical artificial intelligence tasks and addressing various challenges in data science and analysis [13–17,10]. Notably, it has been shown that GNNs outperform traditional approaches in handling local nonlinearities [18]. They have demonstrated precise predictions for aerodynamic performances [19] and flowfield properties [20]. Additionally, they are effective in addressing complex time-dependent problems [21] and have proven successful in diverse aerospace applications, including data fusion tasks [22], uncertainty quantification [23], and multi-objective optimization [24].

While Convolutional Neural Networks (CNNs) have demonstrated remarkable accuracy across various domains [25–29], they rely on the assumption that inputs exhibit a Cartesian grid structure. This assumption allows CNNs to leverage three fundamental properties—sparse connection, parameter sharing, and translation invariance—to achieve accurate results. However, this limitation confines CNNs to regular grid data, such as images (2D grids) and texts (1D sequences) [30]. Consequently, our approach involves the adoption of Graph Convolutional Networks [31], which harness the convolutional operation of CNNs and extend it to non–homogeneous unstructured data. It involves a single-element filter swept across the connected nodes and being weighted by the corresponding edge weights, hence the convolutional analogy (refer to Fig. 1). This idea enables the application of convolutional operations to data structures without the regular grid assumption, broadening our predictive capabilities and allowing direct input of raw 3D model mesh data to GCNs. This approach avoids unnecessary pre–computation or feature extraction methods that may introduce bias or loss of information.

In addition to handling unstructured data, reduced space modeling is crucial for large grids, as not only it manages computational resources but also improves accuracy by addressing the curse of dimensionality. High-dimensional data can lead to overfitting and poor generalization, while the autoencoder architecture can help to capture the most important features, enhancing the model performance. Conventional autoencoders are generally designed for structured, regular grid data, and thus are not ideally suited for predictions on irregular or unstructured grids, they have nonetheless been applied to aerodynamic predictions with varying degrees of success. Researchers have adapted these frameworks using various techniques, such as interpolation, grid regularization, and hierarchical models, to extend their applicability to unstructured data [32–35,29]. Although these adaptations can be effective in certain scenarios, they often introduce additional complexities and loss of information.

With the challenges outlined before, our architecture has been specifically designed to address the limitations of traditional methods. We incorporate GCN layers, complemented by pooling and unpooling layers to effectively reduce and expand the dimensionality

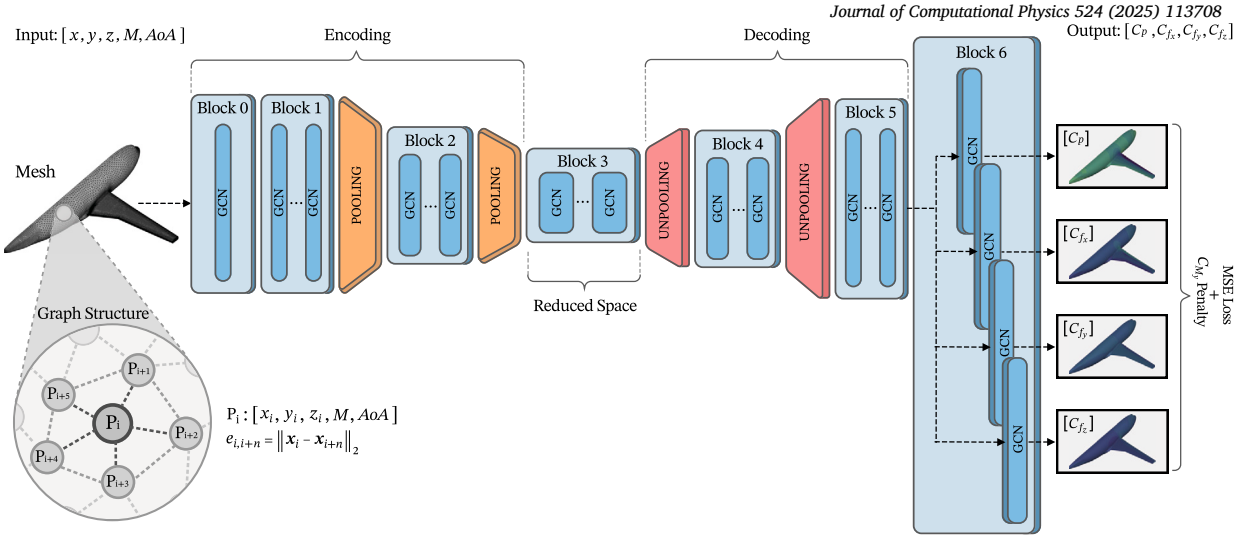


Fig. 2. Schematic of the graph autoencoder architecture.

of the latent spaces while propagating information to more distant nodes. This combination enhances the network ability to identify and emphasize key features, resulting in a more robust and accurate predictive model. We refined our methodology by building on the architecture introduced by Massegur et al. [36], incorporating more complex approaches to improve model accuracy. Our model employs an autoencoder GCN architecture and stands out from other graph-based approaches by introducing several innovations: a dimensionality reduction module based on pressure-gradient values, fast connectivity reconstruction using Mahalanobis distance, Bayesian optimization of the network architecture, and a physics-informed loss function that includes a penalty term for the pitching moment coefficient. Collectively, these enhancements lead to a systematically lower error compared to previous graph-based studies [36] and traditional technique. Two test cases characterized by distinct physical phenomena are proposed to validate the developed methodology: wing–only model and wing–body configuration.

The structure of the paper is as follows: Section 2 outlines the methodology implemented, where a comprehensive explanation of the architecture and its blocks is given, Section 3 presents the results obtained on examples of steady–state prediction of aircraft wing configurations, and Section 4 summarizes the conclusions drawn from the study.

2. Methodology

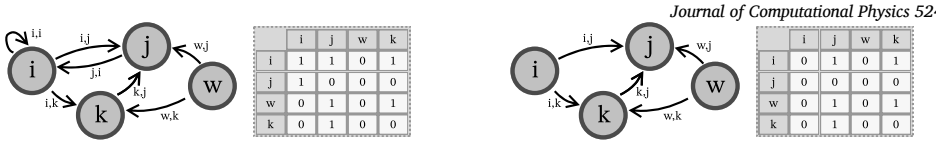
This section explains the methodology that guided the creation of the model at hand. Initially, the general autoencoder graph convolutional network architecture is introduced, followed by a detailed explanation of each component that constitutes every module of the model.

2.1. Graph autoencoder architecture

The steady–state prediction ROM developed in this work uses freestream conditions and mesh coordinates as input, and is designed to predict specific values for each point in the graph. Scalar freestream conditions are assigned to each node of the surface alongside their respective coordinates. A Gradient-Based Autoencoder GCN model (GB-AE-GCN) with two custom levels of dimensional reduction/expansion was implemented. The output of the model is generated by four parallel GCN layers. The whole architecture is finally trained for the pointwise prediction of the four desired output C_p, C_{f_x}, C_{f_y} and C_{f_z}. A schematic of the model architecture is illustrated in Fig. 2.

The use of an Encoder-Decoder based architecture aims to reduce the computational effort by reducing the size of the data during the prediction, increasing the scalability of the system, and also allows the model to consider the connection between more distant points of the mesh, which are not directly connected initially. This step has been taken in order to reproduce the CNN behavior used in AI-based computer vision tasks [37,38], with the addition of the information about the distances and connections between the points given by the graph structure.

The pooling module implemented in our approach is a gradient–based point selection and connection reconstruction. The pressure gradient–based point selection task involves two key steps. Firstly, we compute the gradients for each sample and subsequently identify the regions of interest across all samples. This process enables us to pinpoint areas where pressure gradients exhibit significant disparities, thereby identifying points characterized by heightened nonlinearity. Once these critical points are identified, we implement a Moving Weighted Least Squares Interpolation (MWLSI) algorithm [39,40] to seamlessly interpolate values from the source points (fine grid) to the destination points (coarse grid). To reconstruct connections and calculate Euclidean distances between the remaining points, a Mahalanobis distance [41] based method was implemented. This method re-establishes connections of each point with its 5 neighbors in the destination space based on Mahalanobis distances calculated in the original space.



(a) Cyclic (b) Acyclic

Fig. 3. Visual representation of a graph diagram and its connectivity matrix.

The aim of the unpooling module is to reconstruct the original structure of the input to generate an output with the same dimension of the input, but this operation requires a new interpolation matrix computed with the MWLSI algorithm (refer to Section 2.3 for details) to calculate the missing data of the new nodes, moving from a coarser grid back to the finer one. The pooling and unpooling modules are pre-computed in order to save computational resources. An on demand version could be implemented for adding learnable capabilities of space reduction/expansion, especially on time–variant problems.

To enhance the predictive capacity of the model, we adopted two strategies: a Bayesian optimization and a custom loss function. The Bayesian approach has been employed for optimizing the neural network hyperparameters, such as number of layers per block, units per layers and compression ratio of encoding/decoding operations. By leveraging Bayesian optimization, the model systematically explores and adapts these hyperparameters to maximize performance and predictive accuracy. The custom loss function aims to optimize the distribution of C_P and C_F components across the grid by minimizing the mean squared error (`MSE`) between the model predictions and the ground truth. Factors like shock waves and boundary layer separation introduce complexity to predictions, affecting force resultant and, therefore, moment calculation. Incorporating physics-based penalty terms in loss functions has proven effective in improving aerodynamic prediction accuracy [42]. Therefore, a penalty term for the pitching moment coefficient C_{M_y} has been introduced into the `MSE` loss function. This addition, represented as $Loss = \text{MSE} + \lambda \cdot C_{M_y}$, with $\lambda = 0.01$ for dimensional consistency, guides the model towards more precise predictions, particularly in terms of shock wave positioning.

2.2. Graph deep–learning model

Graph Neural Networks (GNNs) are a class of neural networks designed to work with graph-structured data, where graphs consist of nodes and edges—nodes representing entities and edges representing relationships or connections between these entities. GNNs have gained popularity for their effectiveness in tasks involving graph-structured data, such as generating node embeddings that capture essential structural information.

A graph G consists of nodes N and edges E. An edge (i, j) denotes a directional connection from node i to node j, differing from (j, i) when $i \neq j$. Self-loops are possible if $(i, i) \in E$. Graphs are often illustrated with circles for nodes and arrows for connections. In the graph G shown in Fig. 3, with nodes $N = \{i, j, k, w\}$, edges are represented by one-way arrows. These connections can be expressed using an adjacency matrix $\mathbf{A}$, where $\mathbf{A}_{ij} = 1$ if $(i, j) \in E$, and $\mathbf{A}_{ij} = 0$. Graphs can also carry edge costs, denoted as e_{ij}, representing distances or other values. In an adjacency matrix with costs, replace 1 with the cost and use ∞ for absent connections. A path $p(i \to j)$ in a graph is a finite series of steps $\langle n_k, n_{k+1} \rangle$ from i to j. A graph G is acyclic if there are no paths $p(i \to j)$ where $i = j$; otherwise, it is cyclic. Examples of cyclic and acyclic graphs are shown in Fig. 3(a) and 3(b).

In our context, the mesh could be considered as a cyclic graph G wherein each grid point i in the surface mesh G is a node characterized by variables (features), which are positional coordinates $\mathbf{x}_i$, pressure coefficient C_{P_i} and three components of skin fiction coefficient C_{F_i}. The connections between grid points form the edges of the graph, linking target node i with grid points $j \in S$. The nodes features are denoted as y_i, and the weights on edges are denoted as e_{ij}.

Graph connectivity is expressed through the adjacency matrix $\mathbf{A}$, where each entry e_{ij} represents the weight on the edge connecting node j to node i. The weights are determined by the Euclidean distance between adjacent grid points: $e_{ij} = \|\mathbf{x}_i - \mathbf{x}_j\|_2$. To normalize the edge weights within the range $(0, 1]$, including self-loops with $e_{ii} = 1$, the adjacency matrix is augmented by the identity matrix: $\hat{\mathbf{A}} = \mathbf{A} + \mathbf{I}$. Additionally, since $\forall (i, j) \in E \; \exists (j, i) \in E$ and $e_{ij} = e_{ji}$, the adjacency matrix results symmetric: $\hat{\mathbf{A}} = \hat{\mathbf{A}}^T$. The adjacency matrix may become sparse as the number of nodes increases, and there are efficient techniques for storing such matrices.

Considering the sparsity of both the graph connectivity and the adjacency matrix, a more memory-efficient organization in Coordinate List (COO) format is adopted. The edge-index matrix has dimensions $n_e \times 2$ (pairs of node indices), and the edge-weight matrix is $n_e \times 1$, where n_e represents the number of edges in the mesh.

To extract meaningful features from the graph, we employ Graph Convolutional Networks (GCNs), which leverage the graph convolutional operator. This operator was introduced by Duvenaud et al. in 2015 [31] for extracting features from molecular fingerprints. Kipf et al. extended this work in 2016 [43], providing the foundation for the current implementation in the `PyTorch-Geometric` Library [44] used in this paper. GCNs are renowned for their ability to generate node embeddings that capture essential structural information on a graph. This is particularly beneficial for tasks that necessitate an understanding of relationships and connections between entities. GCNs utilize a convolutional operation similar to classical CNNs to aggregate information from neighboring nodes, while also incorporating distance information from the local neighborhood. The scalability of GCNs is facilitated by parameter sharing, as the parameters are uniformly shared across all nodes.

The GCN operator follows the layer-wise propagation rule that is defined by:

$$H^{(l+1)} = \sigma(\tilde{\mathbf{D}}^{-\frac{1}{2}} \tilde{\mathbf{A}} \tilde{\mathbf{D}}^{-\frac{1}{2}} H^{(l)} W^{(l)}) \tag{1}$$

Table 1
Hyperparameters design space.

Hyperparameter	Value	Step size
Compression ratio	1/4 - 1/3 - 1/2	–
Number of Hidden Layers per Block	1 to 3	1
Number of Neurons per Hidden Layer	32 to 512	16

where $H^{(l)}$ and $H^{(l+1)}$ represent the input and output graphs at layers l and $l+1$, respectively. The matrix $\tilde{\mathbf{A}} = \mathbf{A} + \mathbf{I}_N$ is the adjacency matrix with added self-loops, and $\tilde{\mathbf{D}}$ is a diagonal matrix called degree matrix of the graph, defined as $\tilde{\mathbf{D}}_{ii} = \sum_j \tilde{\mathbf{A}}_{ij}$. $W^{(l)}$ is the trainable matrix for layer l, and σ is the activation function. This equation allows the network to capture local graph structure and propagate information between nodes.

Next, the concept of spectral convolution is introduced, which applies a graph filter g_θ to the input graph in the Fourier domain [43]. By expressing the filter in terms of Chebyshev polynomials, the convolution is simplified to a computationally efficient form. To prevent overfitting and reduce complexity, the Chebyshev order is limited to $K = 1$ [45], resulting in a simplified convolution operation:

$$g_\theta * x \approx \theta(\mathbf{I}_N + \mathbf{D}^{-\frac{1}{2}}\mathbf{A}\mathbf{D}^{-\frac{1}{2}})x$$

However, repeated application of this operator can lead to issues like exploding or vanishing gradients. To address this, a renormalization trick is applied [43], stabilizing the gradient flow during training.

Through successive application of pooling operations, information from a node is propagated through increasingly distant neighborhoods. For instance, with k_l concatenated GCN layers, we extend influence to the k_l^{th}-order neighborhood surrounding node i.

Lastly, the output of the GCN layer is fed through an activation function σ to introduce nonlinearities. Thus, the operation at each layer l consists of the GCN operator in Equation (1) with the Rectified Linear Unit (PReLU) [46] operator used as an activation function. A neural network model based on graph convolutions can therefore be built by stacking multiple convolutional layers defined as before.

ADAptive Moment estimation (`Adam`) [47] was adopted during the back–propagation phase for optimizing neural network weights and minimizing `MSE` loss function. An adaptable learning rate has been used, starting from 0.001 and applying a learning rate decay of a factor of 0.9 every 30 epochs. A batch size equals to 1 led to the most accurate results.

Bayesian optimization for hyperparameters tuning

To enhance the predictive accuracy of our model, we employed Bayesian optimization [48] to fine-tune the hyperparameters, following the approach by Immordino et al. [7]. Bayesian optimization iteratively refines the hyperparameters by balancing exploration and exploitation through probabilistic modeling, thus efficiently navigating large design spaces.

We utilized the `Optuna` library [49], which integrates seamlessly with the `PyTorch` framework, to perform this optimization. The primary hyperparameters targeted in this process were the number of layers per block, the number of units per layer, and the dimensionality compression/expansion value. The number of layers per block defines a group of layers before or after a spatial reduction operation in the encoding module, with the decoding module mirroring this structure to save computational resources. The number of units per layer refers to the number of neurons in a single GCN module, optimized initially for the encoding phase and then mirrored for the decoding phase, ensuring both dimensional compatibility and minimized computational cost. Lastly, the dimensionality compression/expansion value controls the ratio between the number of points in coarser and finer meshes during the compression phase, and vice versa during expansion.

The design space for these hyperparameters is presented in Table 1, with ranges carefully selected to allow sufficient exploration while ensuring convergence to optimal values.

To ensure sufficient convergence towards the optimal set of hyperparameters, we conducted 30 trials, each limited to 500 epochs to manage computational costs. Upon completion of the optimization phase, we executed the training procedure for the refined encoder-decoder architecture that minimizes the loss function for 2000 epochs. For a detailed overview of the final optimized architecture, please refer to Table A.6 in Appendix A.

2.3. Dimensionality reduction/expansion

The core idea behind the use of space reduction/expansion operations is to minimize non–influential information from nodes that do not contribute to the nonlinearity of the system. The aim is to streamline the complexity of hidden layer operations and eliminate redundant information that could potentially mislead the model. The pooling and unpooling modules entail different concepts, which are herein explained. An overview is presented in Fig. 4, where it is possible to distinguish all the processes used for construct and reconstruct hidden spaces. During encoding, we select points based on pressure gradients, creating a reduced–point cloud. We then use a Mahalanobis distance–based method to reconstruct connectivity, resulting in a connected reduced graph. Node values are computed through grid interpolation using the moving weighted least squares method. In decoding, we interpolate on the original fine point map and connectivity using the same method with a new interpolation matrix.

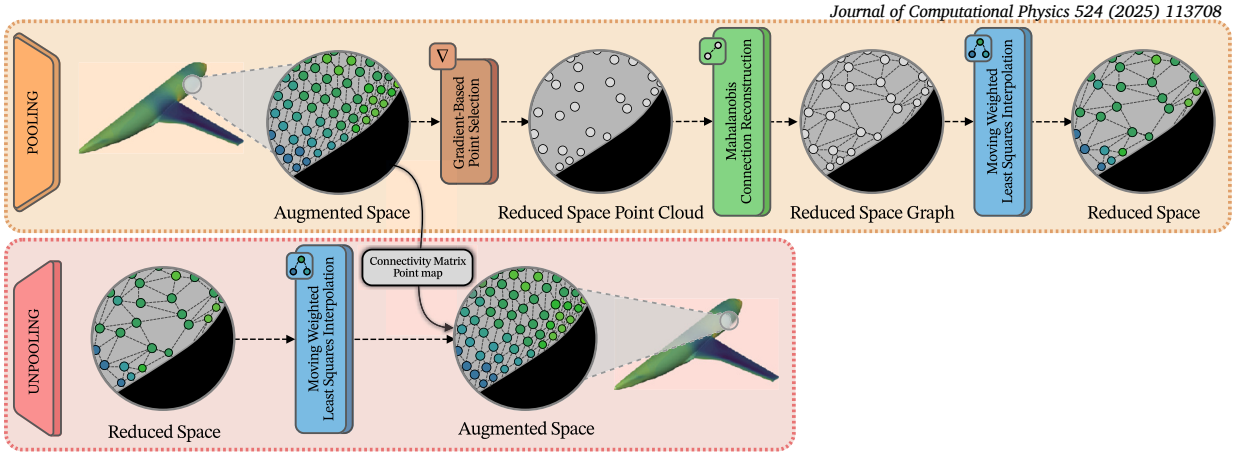


Fig. 4. Pooling and unpooling modules architecture.

Pressure gradient–based point selection

The goal of gradient–based point selection is to find the optimal approach for implementing a pooling phase. During this phase, points are chosen for removal from the mesh graph in the space reduction operation. The general idea is to employ a more advanced point selection method instead of relying solely on the simplistic density–based approach [36]. By doing so, the pooling phase can more effectively consider the primary region where nonlinear phenomena occur.

This method entails two fundamental steps. Initially, gradients on pressure value are computed for each sample. Then, the value of gradient for each example is used for the identification of regions of interest across the entire dataset. This approach facilitates the detection of areas where pressure gradients display notable differences on pressure, thereby identifying points characterized by heightened nonlinearity.

Spatial gradients are computed for each point by considering the pressure value at each node of the graph. To calculate gradients in unstructured grids, it is assumed that the pressure variable varies linearly in all dimensions, yielding:

$$p - p_0 = \Delta p = \Delta x p_x + \Delta y p_y + \Delta z p_z \tag{2}$$

Where p_0 is the pressure in the node. Then, a matrix equation is constructed using the pressure differences among all nodes neighboring the current node. With five connections, the matrix equation results in:

$$\begin{bmatrix} \Delta x_1 & \Delta y_1 & \Delta z_1 \\ \Delta x_2 & \Delta y_2 & \Delta z_2 \\ \Delta x_3 & \Delta y_3 & \Delta z_3 \\ \Delta x_4 & \Delta y_4 & \Delta z_4 \\ \Delta x_5 & \Delta y_5 & \Delta z_5 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = \begin{bmatrix} \Delta p_1 \\ \Delta p_2 \\ \Delta p_3 \\ \Delta p_4 \\ \Delta p_5 \end{bmatrix} \tag{3}$$

Equation (3) is then inverted via the least-squares method to compute the gradient vector.

Starting from the value of the gradients calculated for all the points in the graph, a suitable probability distribution has been employed to determine the number of points retained in the reduced space. The challenge arises in regions of the original mesh with low gradients, potentially resulting in an inadequate number of nodes at the coarsened level and leading to an irreversible loss of information. Conversely, excessive node removal in regions of originally high gradients may result in insufficient accuracy reconstruction of complex physics phenomena. Thus, an appropriate node selection strategy is essential to ensure the proper representation of both high and low gradient regions in the coarsened domain. This is obtained using a probability function applied on the gradient value of each mesh element (node):

$$p(i) = 1 + \frac{1 - e^{-2i/n}}{1 - e^{-2}}(p_1 - p_n) + p_1 \quad \text{for} \quad i = 1, \dots, n \tag{4}$$

Here, i represents the mesh node index, sorted by pressure gradient value in descending order, and n is the total number of nodes. The probabilities p_1 and p_n denote the choices for the highest and lowest gradients, respectively, set to 0.2 and 1.

After each space reduction, an unconnected point cloud is obtained, therefore it is essential to restore the connectivity between neighbors.

Mahalanobis connection reconstruction

To identify the neighbors of each node in the point cloud after the reduction process and thereby restore connectivity, we use a reconstruction method based on the Mahalanobis distance [41], that is widely used in clustering problems and other statistical classification techniques [50,51]. The Mahalanobis distance is a measure of the distance between points in a distribution. Unlike the simple Euclidean distance, the Mahalanobis distance takes into account the spread of points in different directions through the

covariance matrix of the distribution of points. Using this type of distance, it is possible to connect each point to its neighbors by following the distribution of points in the finer mesh by using the covariance matrix calculated in the original space. This method minimizes false connections between opposite faces of the mesh which are considered close according to the simple Euclidean distance. Therefore, the distance between points is calculated using the following equation:

$$D_M(x,y) = \sqrt{(x-y)^T S^{-1}(x-y)} \tag{5}$$

Where x and y are two points of the reduced space and S is the covariance matrix of the distribution of the points in the finer mesh. Additionally, to reduce the searching field of nearest neighbors on the reduced space, we used the K-d tree algorithm [52] to determine for each point a subset of 250 elements using Euclideian distance, and then selected the nearest neighbors by following the Mahalanobis distance calculated only in that subset.

Building on this process, we tested the accuracy of connection reconstruction by removing all connections from the fine grid and using Mahalanobis and Euclidean distances to identify the nearest neighbors for each node, creating new connections. These reconstructed connections were compared to the original grid and the percentage of incorrect or false connections was calculated for each node. The average error across all nodes assessed overall accuracy. For the BSCW model, we observed that the Mahalanobis distance resulted in only 0.4% false connections, compared to 48.6% when using Euclidean distance. Similarly, for the CRM model, Mahalanobis distance led to 1.5% false connections, while Euclidean distance produced 58.4%. These false connections can significantly impact the model performance by introducing noise into the learning process, linking unrelated nodes, and misguiding the model, which reduces its capacity to accurately capture the physical phenomena accurately. This can result in poorer generalization and lower prediction accuracy, particularly in regions with complex flow dynamics.

Moving weighted least squares for grid interpolation

Efficient information transfer between grids is a critical aspect in the proposed methodology. While one option involves using a neural network with learnable weights, this approach could significantly escalate computational requirements. On the contrary, traditional interpolation techniques may yield inaccuracies that are not suitable for our purposes [36]. Consequently, we opted for the Moving Weighted Least Squares (MWLS) technique [39,40], which has proven to deliver accurate results in similar problems [36]. This decision aims to strike a balance between accuracy and computational efficiency, while also ensuring the conservation of the integrated quantity across both grids and maintaining continuity across the domain [39]. MWLS assigns varying weights to neighboring data points based on their proximity to the interpolation point, allowing for a more adaptive and accurate representation of the underlying data. The approach involves fitting a local polynomial to a subset of nearby points, with the influence of each point weighted according to its distance. This adaptability ensures that closer points have a more significant impact on the interpolated value, while those farther away contribute less.

The core idea of MWLS is to generate an interpolation matrix $I_{S_s \to S_d}$ that maps features $\mathbf{y}_i$ from the source grid S_s with n_s nodes to the destination grid S_d with n_d nodes:

$$\mathbf{y}_j = I_{S_s \to S_d} \mathbf{y}_i \quad \forall j \in S_d, \quad \forall i \in S_s$$

To accomplish this, we construct a shape function $u(\mathbf{x}) = \mathbf{p}^T(\mathbf{x})\,\mathbf{a}$ that approximates the grid data y_i by minimizing the least squares error:

$$\min L = \sum_{i \in S_s} \left(\mathbf{p}^T(\mathbf{x_i})\,\mathbf{a} - y_i\right)^2 w(\mathbf{x}_i)$$

where $w(\mathbf{x}_i) = e^{-\|\mathbf{x}-\mathbf{x}_i\|_2}$ is the Gaussian weight function that ensures nodes closer to the interpolation point have a greater influence, $\mathbf{p}(\mathbf{x})$ is a second-order polynomial basis function, and $\mathbf{a}$ is the vector of coefficients. The coefficients $\mathbf{\Phi}(\mathbf{x}_j)$ for each destination node are calculated by:

$$\mathbf{\Phi}(\mathbf{x}_j) = \mathbf{p}^T(\mathbf{x}_j)(\mathbf{P}^T\mathbf{W}\mathbf{P})^{-1}\mathbf{P}^T\mathbf{W}$$

The design matrix $\mathbf{P}$ and the weight matrix $\mathbf{W}$ are formed based on the source nodes, where $\mathbf{W}$ is diagonal with Gaussian weights. The interpolation matrix $I_{S_s \to S_d}$ is then constructed as:

$$I_{S_s \to S_d} = \begin{bmatrix} \mathbf{\Phi}(\mathbf{x}_1) \\ \mathbf{\Phi}(\mathbf{x}_2) \\ \vdots \\ \mathbf{\Phi}(\mathbf{x}_{n_d}) \end{bmatrix}$$

To reduce computational complexity, a local interpolation is applied by considering only the k_n nearest neighbors for each destination node. The optimal number of neighbors was found to be $k_n = 10$, striking a balance between minimizing reconstruction errors and managing computational requirements efficiently.

It is worth remarking that this interpolated matrix is of non-square size $n_s \times n_d$ and largely sparse, with only k_n non-zero values in each row. Consequently, with regards to executing the inverse interpolation in the decoder phase, this matrix is not invertible. Thus, it is necessary to compute two independent interpolation matrices: $I_{S_s \to S_d}$ and $I_{S_d \to S_s}$. The entire interpolation process, from fine grid to coarse grid and back, yields an average error of 1.9%. However, this error does not accumulate during these steps, as the

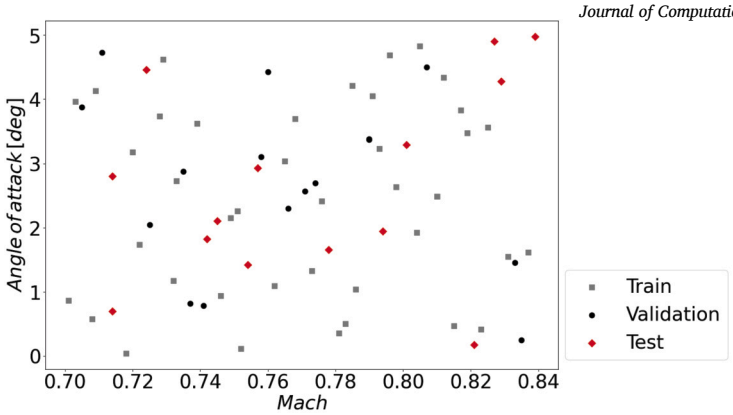


Fig. 5. Training, validation and test samples for Mach number and angle of attack.

subsequent network layers learn to mitigate any potential accumulation, preserving accuracy throughout the process. Introducing errors during training can also act as a regularization technique that enhances the model generalization capabilities, and prevents overfitting to specific patterns.

Proper orthogonal decomposition

One of the models used to benchmark our GB-AE-GCN framework is the Proper Orthogonal Decomposition with interpolation (POD-I), a classic dimensionality reduction approach. Despite its linear formulation, POD has been applied to study nonlinear problems [53–55] as it resulted particularly effective in extracting dominant patterns from high-dimensional data by identifying a set of orthonormal modes that minimize the error between the original data and its projection onto a lower-dimensional space. This is achieved by applying Singular Value Decomposition (SVD) to the snapshot matrix $\mathbf{X}$, yielding:

$$\mathbf{X} = \mathbf{U}\boldsymbol{\Sigma}\mathbf{V}^* = \sum_{i=1}^{n} \sigma_i u_i v_i^*$$

Here, $\mathbf{U}$ represents the spatial modes, $\boldsymbol{\Sigma}$ contains the singular values, and $\mathbf{V}$ comprises the temporal modes. We truncate the basis to retain the k modes that capture 95% of the system energy. Subsequently, the POD coefficients are interpolated in the reduced space using radial basis functions (RBF) across the parameter space, resulting in the POD-I method.

3. Test cases

Two test cases, characterized by different physics and complexity, were employed for assessing the model prediction capability. Angle of attack and Mach number were chosen as the two independent parameters for the ROM. The chosen ranges for the angle of attack (AoA) and Mach number (M) are $[0, 5]$ [deg] and $[0.70, 0.84]$, respectively. These ranges are specifically chosen for the transonic regime, where shock wave formation occurs on the wing, and high angles of attack, that lead to boundary–layer separation. To generate the required number of samples, Latin hypercube sampling (LHS) [56] is employed, resulting in a total of 70 points as illustrated in Fig. 5. Sixty percent of these samples (40 flight conditions denoted by circles) are designated for training, 20% (15 flight conditions marked with squares) for validation, and the remaining 20% (15 conditions represented by diamonds) are reserved for testing.

The dataset has been generated through CFD simulations. Reynolds-averaged Navier–Stokes (RANS) equations are discretized using SU2 v7.5.1 [57] software. The closure of RANS equations is achieved using the one–equation Spalart–Allmaras turbulence model. Convergence method is set to Cauchy method, specifically applied to the lift coefficient, considering a variation of 10^{-7} across the last 100 iterations. A $1v$ multigrid scheme is adopted for accelerating the convergence of CFD simulations. The discretization of convective flows involves the use of the Jameson-Schmidt-Turkel (JST) central scheme with artificial dissipation. Flow variable gradients are computed through the Green Gauss method. The selected linear solver is the biconjugate gradient stabilization, with an ILU preconditioner.

Following dataset generation, preprocessing and normalization to the range $[-1, 1]$ were performed before inputting the data into the GB-AE-GCN model. The rest of the section explores the model predictive capabilities for distributed quantities and integral loads across different test cases. Optimized architectures are detailed in Appendix A.

3.1. Wing–only model

The first test case is the Benchmark Super Critical Wing (BSCW), which is a transonic rigid semi–span wing with a rectangular planform and a supercritical airfoil shape from the AIAA Aeroelastic Prediction Workshop [58]. This wing is elastically suspended on a flexible mount system with two degrees of freedom, pitch and plunge, and it has been developed for flutter analysis. However, for our case study, we focus solely on the wing, excluding the aeroelastic system. The BSCW is characterized by shock wave motion,

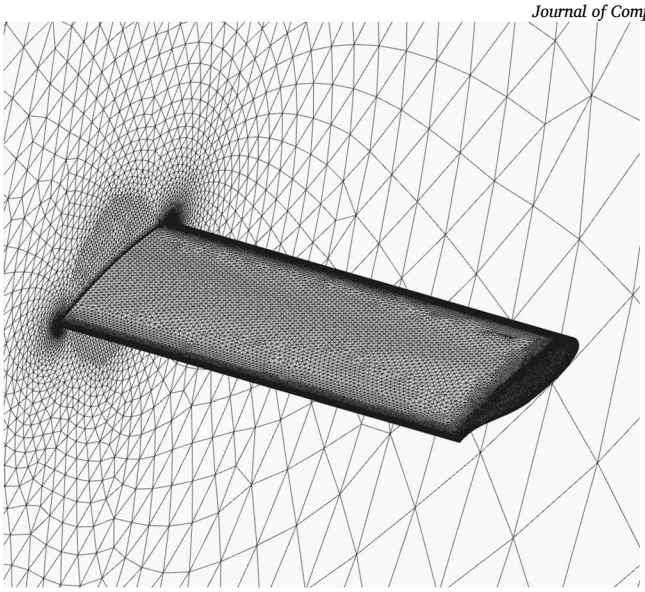


Fig. 6. Impression of the BSCW CFD grid.

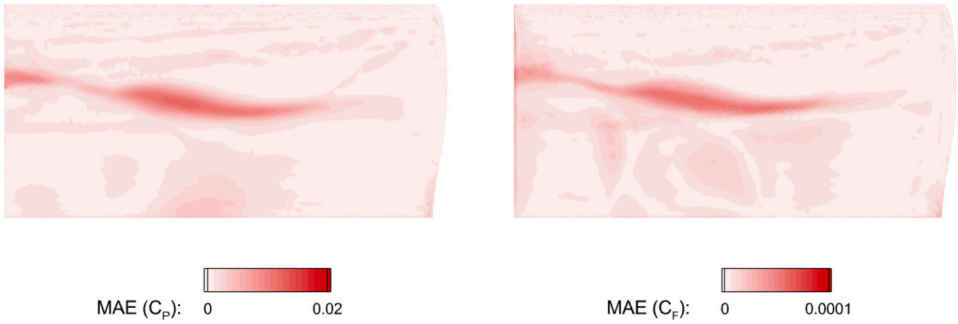


Fig. 7. Mean absolute error (MAE) of C_P and C_F computed across every point in the mesh of the test set for BSCW test case using the GB-AE-GCN model. (For interpretation of the colors in the figure(s), the reader is referred to the web version of this article.)

shock–induced boundary–layer separation and interaction between shock wave and detached boundary–layer. These three types of nonlinearity are challenging for the ROM predictions.

An unstructured grid configuration with $8.4 \cdot 10^6$ elements and 86,840 surface elements was generated. A $y^+ = 1$ is adopted, after a preliminary mesh convergence study that ensured an adequate resolution of the boundary–layer and shock wave. The computational domain extends 100 chords from the solid wall to the farfield. An impression of the grid can be obtained from Fig. 6.

The performance of the GB-AE-GCN model on the BSCW test case is shown through the Mean Absolute Error (MAE) calculated for C_P and C_F at each surface point in the test set mesh. As depicted in Fig. 7, the errors for both predictions are considerably small, with the selected ranges serving solely to offer a visual depiction of areas where the model faces challenges in prediction. The errors are minimal across the entire surface, except for a localized region near the shock wave.

Fig. 8 illustrates the percentage errors in $[C_L, C_D, C_{My}]$ across different Mach numbers and angles of attacks. Remarkably, the GB-AE-GCN model predictions exhibit high accuracy for all coefficients, even for data points located far from the training set. This indicates the model robustness in extrapolating beyond the provided data points. Aerodynamic coefficients were calculated using a reference chord length of 0.4064 m and surface of 0.3303 m^2, and derived by integrating the pressure coefficient distribution and the skin friction coefficient distribution over the entire wing surface. C_{My} was calculated with respect to 30% of the chord, accounting for the rigid mounting system of the BSCW, which induces pitch oscillations around this specific location.

The comparison of pressure coefficient contours between the CFD data and the GB-AE-GCN predictions, along with the POD-I results, is presented in Fig. 9. This comparison is for the test case with the highest error of the GB-AE-GCN model at $M = 0.714$ and $AoA = 2.807$ [deg]. The GB-AE-GCN model demonstrates excellent agreement with CFD, particularly in predicting the position and intensity of shock waves. Minor discrepancies are visible near the low-pressure regions, but overall, the GB-AE-GCN captures the nonlinearities more effectively than POD-I, especially in complex shock-dominated areas. Similarly, Fig. 10 compares the skin friction contours, where again the GB-AE-GCN model outperforms POD-I, particularly in boundary layer regions where high gradients occur. The POD-I method tends to oversmooth the flow features, missing key aerodynamic phenomena like boundary layer separation,

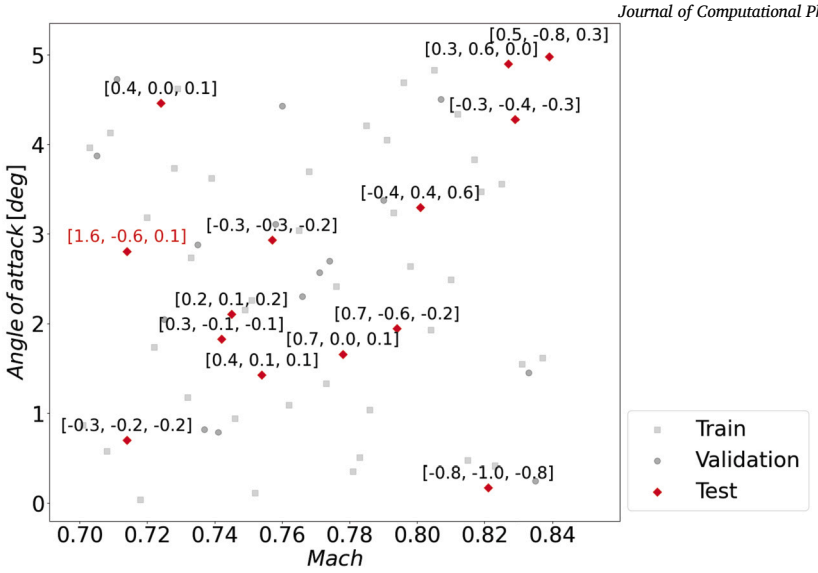


Fig. 8. Errors % in $[C_L, C_D, C_{My}]$ on the test samples across varying Mach numbers and angle of attacks for BSCW test case using the GB-AE-GCN model.

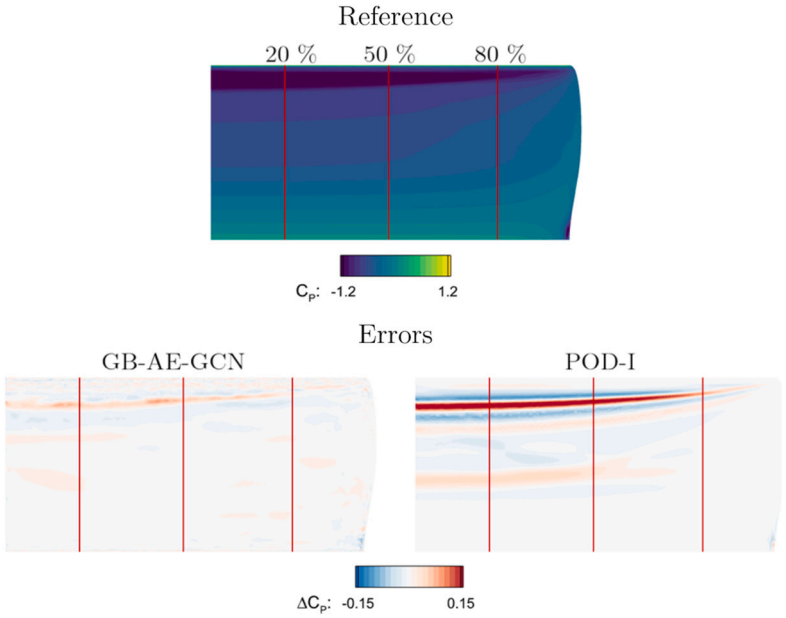


Fig. 9. Prediction of the pressure coefficient contour on the upper surface for BSCW test case at $M = 0.714$ and $AoA = 2.807$ [deg].

whereas GB-AE-GCN maintains more accurate fidelity to the CFD data, reflecting its enhanced capability in capturing fine-scale aerodynamic phenomena.

Figs. 11 and 12 show detailed pressure and skin friction distributions at various sections along the wing span, evaluated at $M = 0.714$ and $AoA = 2.807$ [deg]. The GB-AE-GCN model consistently captures the peaks and variations with high precision, particularly around the 20% wing span where the shock occurs. In contrast, the POD-I model exhibits larger deviations, failing to reproduce the sharp gradients and flow complexities, especially near the shock and separation regions.

To provide a quantitative evaluation of the models, we computed Mean Absolute Percentage Error (MAPE) and R^2 scores on the test set, which are reported in Table 2 for the GB-AE-GCN and POD-I models. The MAPE was computed by averaging the absolute error of each prediction calculated by our GB-AE-GCN architecture within the test set. This prediction error was determined by weighted averaging the errors at each grid point, considering the corresponding cell area and normalizing with respect to it. The results show the good performance of the GB-AE-GCN model, with low MAPE values of 0.7712 for C_P and 0.3828 for C_F. In comparison, the POD-I method produced higher errors of 0.8958 for C_P and 2.2753 for C_F. The high R^2 scores for GB-AE-GCN (0.9728 for C_P and 0.9745 for C_F) further confirm its accuracy in predicting flow features on unstructured grids. This superior performance is attributed

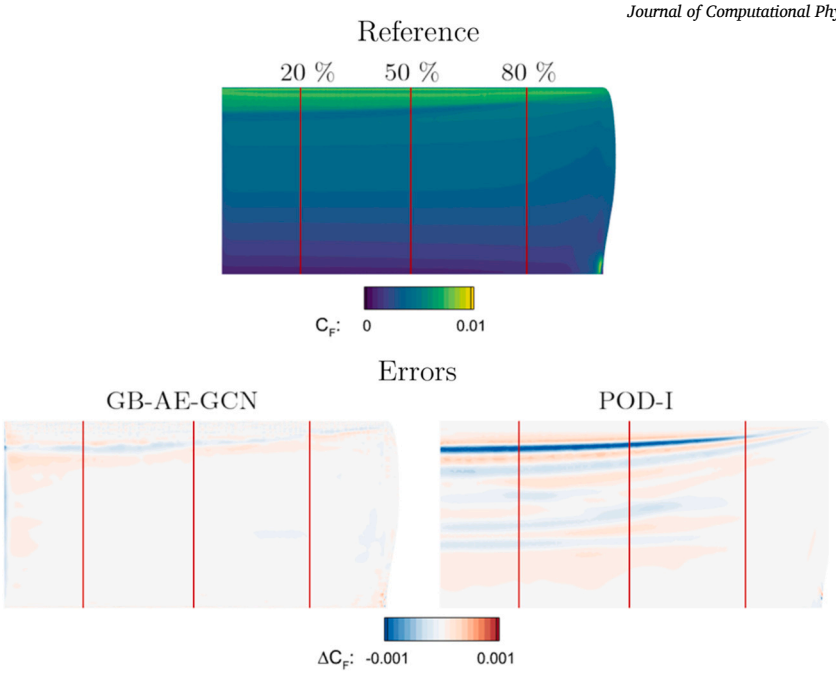


Fig. 10. Prediction of the skin friction magnitude coefficient contour on the upper surface for BSCW test case at $M = 0.714$ and $AoA = 2.807$ [deg].

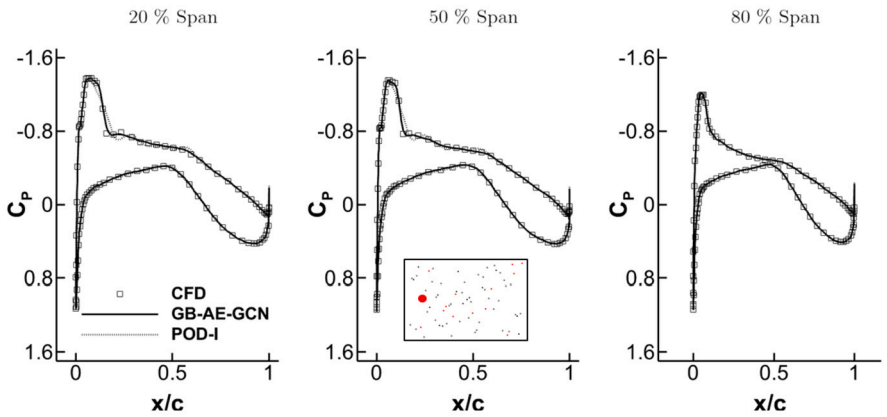


Fig. 11. Pressure coefficient sections of BSCW at $M = 0.714$ and $AoA = 2.807$ [deg].

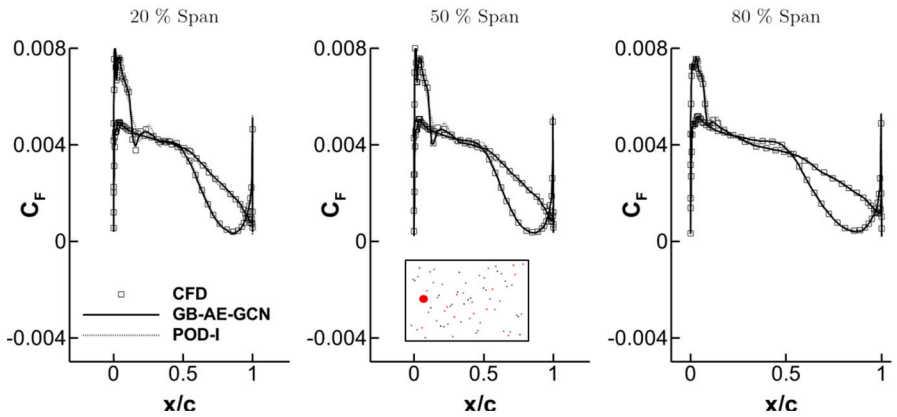


Fig. 12. Skin friction coefficient sections of BSCW at $M = 0.714$ and $AoA = 2.807$ [deg].

Table 2
MAPE [%] and R^2 on the test set for wing-only model.

	GB-AE-GCN		POD-I	
	MAPE	R^2	MAPE	R^2
C_P	0.7712 ± 0.011	0.9728 ± 0.0039	0.8958	0.9039
C_F	0.3828 ± 0.008	0.9745 ± 0.0031	2.2753	0.8909

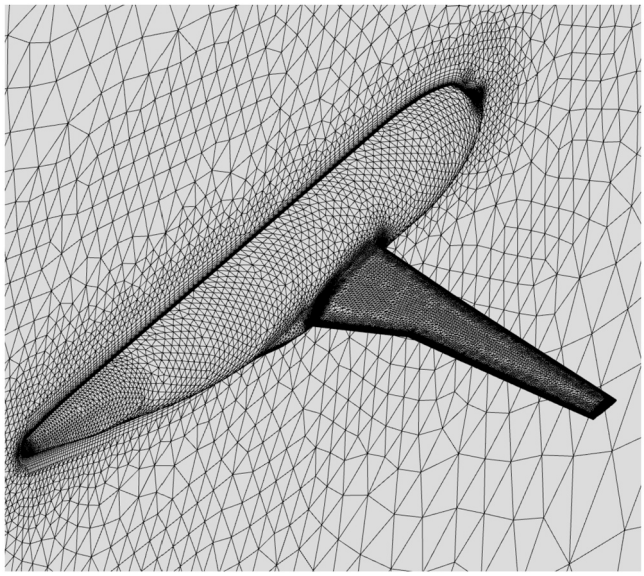


Fig. 13. Impression of the CRM CFD grid.

to the ability of the GB-AE-GCN to capture nonlinearities in complex flow fields more effectively than the linear-based POD-I method, particularly in regions with shock waves and boundary layer separations.

To ensure the robustness of the results, we studied the influence of random initialization of the network weights on the variance of MAPE and R^2 scores. The GB-AE-GCN model was trained 30 times with the same hyperparameters, and the resulting variance was minimal, with 0.011 for MAPE on C_P and 0.008 for C_F. The variance for R^2 was similarly low, at 0.0039 for C_P and 0.0031 for C_F. These results indicate that the model performance is stable and consistent across different initializations.

3.2. Wing–body model

The second test case is the NASA Common Research model (CRM), a transonic wing–body model featured in the AIAA CFD Drag Prediction Workshop [59]. This model encompasses a conventional low–wing configuration and a fuselage typical of wide–body commercial aircraft. The computational grid utilized for this case was adapted from the DLR grid developed for the AIAA Drag Prediction Workshop [60]. This unstructured grid comprises 8.8×10^6 elements, including 78,829 surface elements. The computational domain extends 100 chords from the fuselage to the farfield. A $y^+ = 1$ condition is employed. For a visual representation of the grid, refer to Fig. 13.

This test case poses a complex challenge for our GB-AE-GCN model due to the complex geometry, physics and grid configuration. The MAE of C_P and C_F computed across every point in the mesh of the test set is depicted in Fig. 14. This visualization provides insight into the regions where the model struggles most to accurately represent the flow physical behavior. Interestingly, the errors are generally minimal across the entire surface, except for a localized region near the wing-fuselage junction and between the kink of the wing and its tip. Nonetheless, the broadly distributed small errors suggest that the model effectively captures the nonlinearities inherent in the system.

Fig. 15 shows the percentage errors in $[C_L, C_D, C_{My}]$ on the test samples across varying Mach numbers and angle of attacks. Notably, the predictions demonstrate overall accuracy across all coefficients, even for points distant from the training samples. This suggests a robust performance of the model in extrapolating beyond the known data points. A chord of 0.1412 m and a scaling area of 0.1266 m^2 were considered for aerodynamic coefficients calculation. C_{My} was computed with respect to 25% of the wing mean aerodynamic chord.

To provide a comparative analysis, Figs. 16 and 17 present the surface pressure and skin friction contours for the test set with the highest prediction error in the GB-AE-GCN model at $M = 0.839$ and $AoA = 4.975$ [deg]. For reference, the results from Massegur model (AE-MM-GCN) [21] are also included, providing a useful comparison of performance across models. Despite being the worst-performing case, the GB-AE-GCN model still shows excellent agreement with the CFD data, capturing the flow features across the

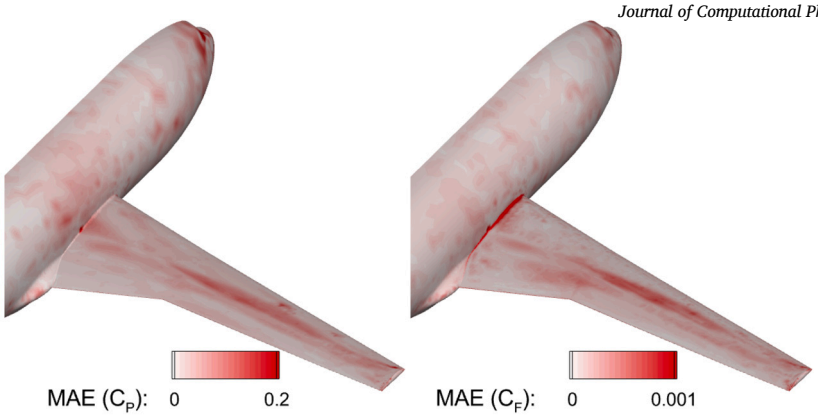


Fig. 14. Mean absolute error (MAE) of C_P and C_F computed across every point in the mesh of the test set for CRM test case using the GB-AE-GCN model.

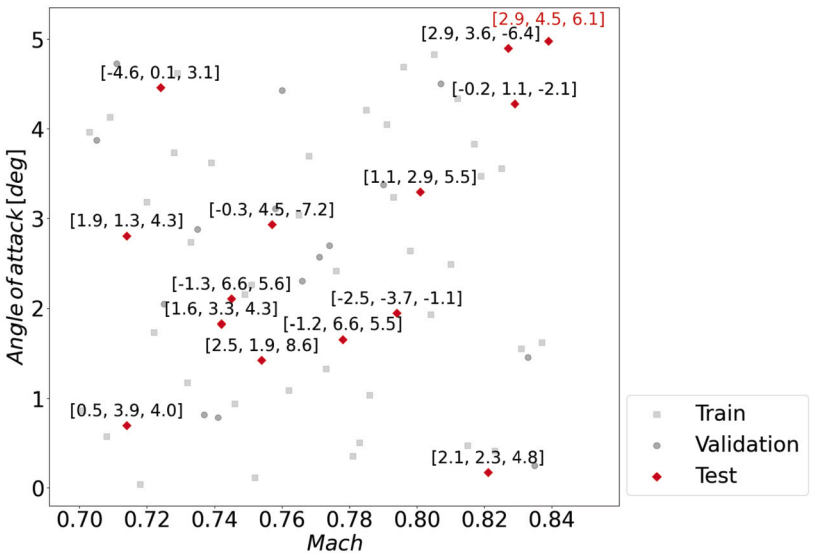


Fig. 15. Errors % in $[C_L, C_D, C_{My}]$ on the test samples across varying Mach numbers and angle of attacks for CRM test case using the GB-AE-GCN model.

aircraft with high accuracy. In contrast, the POD-I method shows notable discrepancies, especially in regions with sharp gradients like shock waves and boundary layer separations, where it struggles to capture complex flow phenomena. The AE-MM-GCN model [21] offers better performance than POD-I but still falls short of the accuracy achieved by GB-AE-GCN, particularly in high-gradient regions.

Figs. 18 and 19 further explore the pressure and skin friction distributions across several spanwise sections of the CRM wing at $M = 0.839$ and $AoA = 4.975$ [deg]. The GB-AE-GCN model consistently aligns closely with the reference CFD data, especially in shock-dominated regions. In comparison, both POD-I and AE-MM-GCN show limitations. While AE-MM-GCN [21] performs better than POD-I, it still lags behind GB-AE-GCN in terms of capturing sharp flow gradients and complex features. The methodological advancements in our model, particularly the use of pressure-gradient-based coarsening and a physics-informed loss function, enable it to handle nonlinearities more effectively, leading to higher overall accuracy in these complex flow conditions.

Building on this detailed analysis of our model performance under its most challenging prediction, we now provide an overview of the performance across the entire test set. To evaluate the models comprehensively, we computed `MAPE` and R^2 scores exclusively for the wing region, where the flow exhibits strong nonlinearities. The CRM model includes large areas with mostly linear flow, particularly over the fuselage, which we excluded from the analysis. Table 3 presents the MAPE and R^2 results for the GB-AE-GCN, POD-I, and AE-MM-GCN [21] models. The GB-AE-GCN model achieved a MAPE of 0.8876 for C_P and 0.2402 for C_F, outperforming both the POD-I and AE-MM-GCN [21] models. The lower MAPE for C_F is especially significant, indicating the model ability to capture skin friction details more accurately, particularly in regions with complex flow gradients. The R^2 values further confirm this performance, with the GB-AE-GCN obtaining R^2 scores of 0.9353 for C_P and 0.9650 for C_F, reflecting its precision in predicting aerodynamic features. In comparison, the POD-I and AE-MM-GCN [21] models show lower R^2 values, highlighting their struggles in capturing sharp flow features.

To further assess the robustness of the GB-AE-GCN model, we evaluated how random initialization affects the variance in MAPE and R^2 scores. After training the model 30 times with the same hyperparameters, we observed a variance of 0.07 in MAPE for C_P and

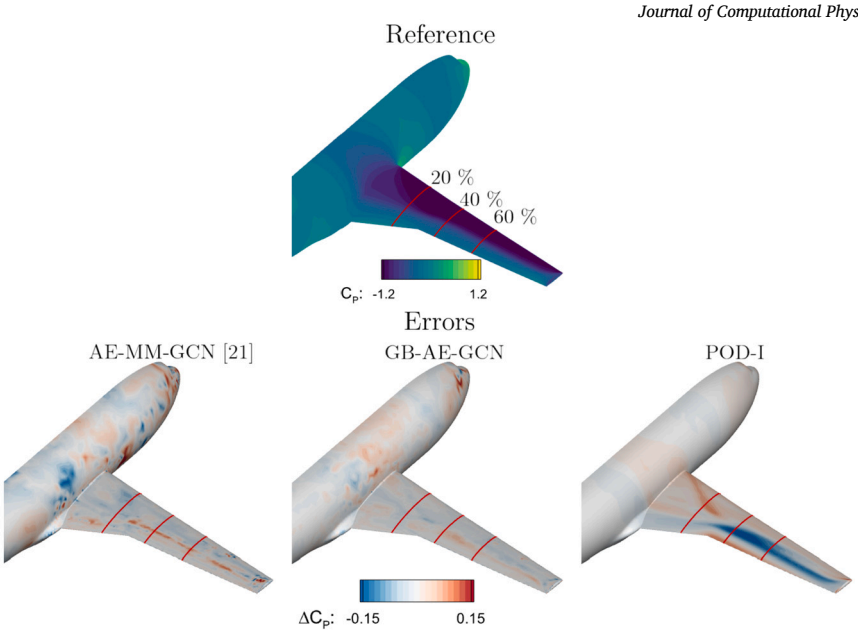


Fig. 16. Prediction of the pressure coefficient contour on the upper surface for CRM test case at $M = 0.839$ and $AoA = 4.975$ [deg].

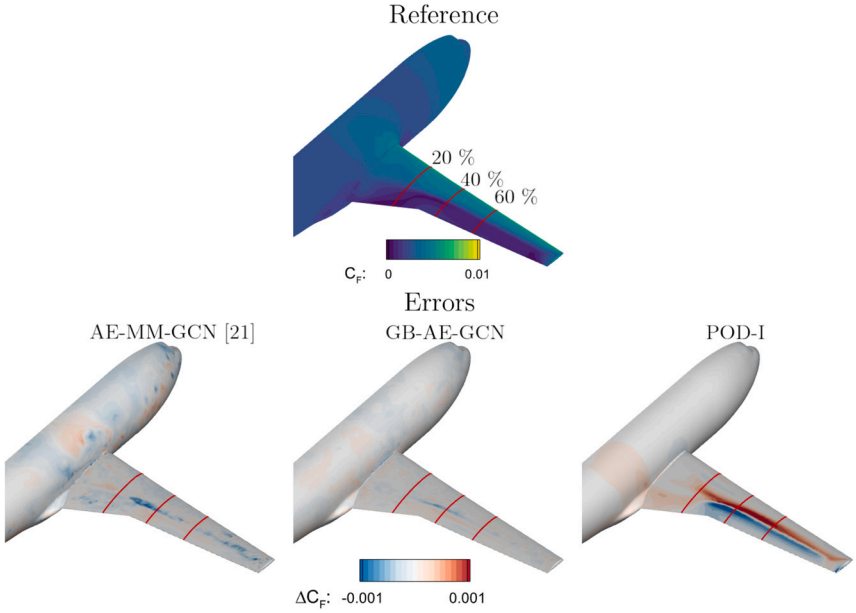


Fig. 17. Prediction of the skin friction magnitude coefficient contour on the upper surface for CRM test case at $M = 0.839$ and $AoA = 4.975$ [deg].

Table 3
MAPE [%] and R^2 on the test set for wing-body model.

	GB-AE-GCN		GCN-MM-AE [21]		POD-I	
	MAPE	R^2	MAPE	R^2	MAPE	R^2
C_P	0.8876 ± 0.07	0.9674 ± 0.014	1.4051	0.9353	3.7136	0.8740
C_F	0.2402 ± 0.05	0.9737 ± 0.019	0.5737	0.9550	1.2142	0.9202

0.05 for C_F. For R^2, the variance was 0.014 for C_P and 0.019 for C_F. These results indicate that, despite the inherent randomness in initialization, the model consistently produces reliable and accurate predictions, reinforcing its robustness across different training runs.

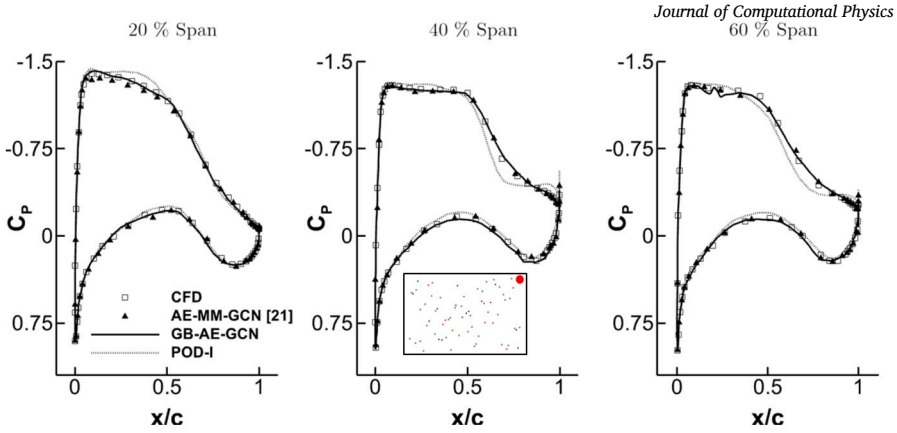


Fig. 18. Pressure coefficient sections of CRM at $M = 0.839$ and $AoA = 4.975$ [deg].

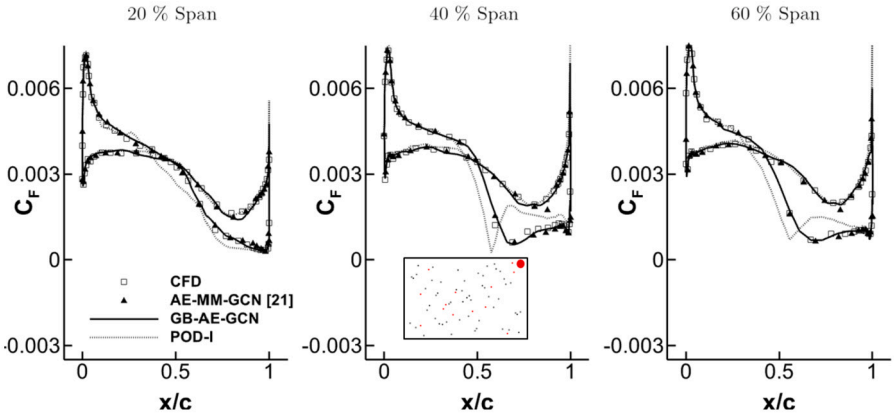


Fig. 19. Skin friction coefficient sections of CRM at $M = 0.839$ and $AoA = 4.975$ [deg].

Table 4
Computing cost comparison between GB-AE-GCN model and CFD for the two test cases.

Test case	CFD (CPU hours)		GB-AE-GCN (GPU hours)		
	Simulation (70 runs)	(1 run)	Optimization (30 trials)	Training (1 model)	Prediction (1 sample)
BSCW	35,000	500	27	1.4	0.0003 ($\sim 1s$)
NASA CRM	28,000	400	28	1.5	0.0003 ($\sim 1s$)

3.3. Computing cost saving

A detailed computational cost analysis was conducted to evaluate the efficiency of the implemented GB-AE-GCN model in comparison to the high–order approach, as outlined in Table 4. Each CFD simulation was performed on a high-performance computing system with an Intel Skylake-based architecture utilizing 3 nodes with 40 CPU cores each, typically consuming around 450 CPU hours per run, with the entire dataset generation requiring approximately 31,500 CPU hours. Conversely, employing the ROM enables prediction for a single sample in approximately 1 second on a local machine, resulting in a computational saving exceeding 99%.

Therefore, it is essential to consider the high computational cost associated with each high–fidelity simulation used for generating the dataset. Adopting a philosophy aimed at minimizing the amount of training data necessary for developing an accurate model is crucial.

The training process was executed on an Intel XEON W-2255 CPU with a NVIDIA RTX A4000 GPU, ensuring efficient utilization of computational resources.

4. Conclusions

This study highlighted the effectiveness and robustness of the implemented GB-AE-GCN model in delivering precise predictions within a parameter space for various test scenarios, featuring complex geometries and diverse physical phenomena. Through convolutional and pooling operations, the model efficiently influences predictions at individual nodes based on their neighbors, while also enabling information propagation to distant nodes during spatial reduction. Additionally, the model can directly process input grids without requiring preprocessing, greatly simplifying the modeling process.

In terms of performance, the GB-AE-GCN model consistently outperformed both the POD-I approach and AE-MM-GCN model from Massegur et al. [21] across various test cases, particularly in capturing complex nonlinear flow phenomena such as shock waves and boundary layer separations. The GB-AE-GCN superior ability to handle sharp flow gradients and complex geometries, especially in high-gradient regions, is reflected in its lower MAPE and higher R^2 scores compared to the other models. The pressure-gradient-based coarsening, fast connectivity reconstruction, physics-informed loss function, and network optimization played key roles in enabling our model to better capture nonlinearities and critical flow features that the other methods struggled with.

A key strength of GCNs is their inherent flexibility to handle various graph structures, extending them beyond aerospace applications to any non-homogeneous, unstructured data. This flexibility is driven by features like local neighborhood aggregation, parameter sharing, and invariance to node ordering, ensuring consistent performance across a wide variety of graph configurations. Since each input graph is represented by an adjacency matrix and node attributes, the model can theoretically process graphs with any arrangement of nodes. However, when trained on a fixed mesh, as in this study, applying the model to new spatial structures may result in unexpected behavior. To address this, the network could be trained on meshes with varying features or sizes using methods like subgraph splitting or padding, similar to those used in traditional convolutional neural networks.

In addition to flexibility, the model is designed to ensure scalability, enabling it to efficiently operate on larger grids due to the localized nature of GCN operations. As the grid size increases, certain adjustments become necessary, particularly regarding the number of layers in the network. Larger grids may demand deeper networks to capture the complex interactions across the expanded spatial domain. Additionally, increasing the number of pooling operations is crucial to ensure information propagation across more distant nodes, enabling them to communicate more effectively. This increased pooling facilitates the model ability to capture long-range dependencies, but it also requires careful tuning to balance computational efficiency with model accuracy.

Looking ahead, integrating temporal dynamics into the framework would further improve the model ability to capture transient behaviors and dynamic changes over time. This extension would broaden its applicability, making it highly relevant for analyzing dynamical systems where time-dependent phenomena are crucial.

CRediT authorship contribution statement

Gabriele Immordino: Writing – review & editing, Writing – original draft, Visualization, Validation, Methodology, Investigation, Formal analysis, Data curation, Conceptualization. **Andrea Vaiuso:** Writing – original draft, Methodology, Conceptualization. **Andrea Da Ronch:** Writing – review & editing, Supervision, Resources, Project administration. **Marcello Righi:** Writing – review & editing, Supervision, Resources, Project administration, Funding acquisition.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgement

This work was supported by Digitalization Initiative of the Zurich Higher Education Institutions (DIZH) grant 9710.Z.12.P.0003.05 from Zurich University of Applied Sciences (ZHAW). The authors also acknowledge the University of Southampton for granting access to the IRIDIS High Performance Computing Facility and its associated support services.

Appendix A. Optimized GB-AE-GCN architecture

This section provides a comprehensive overview of the optimized architectures and highlights the systematic reduction of loss throughout the optimization trials for each test case.

Table A.5 provides detailed information about the optimized architecture designed specifically for the wing–only test case. This architecture consists of 17 layers and a total of 711,493 parameters, carefully balanced to capture the complexities of this aerodynamic setup. Similarly, Table A.6 displays the optimized architecture for the wing–fuselage test case. With 15 layers and a total of 633,731 parameters, this configuration is tailored to accurately model the interaction between the wing and fuselage, capturing the subtle aerodynamic interactions between these components.

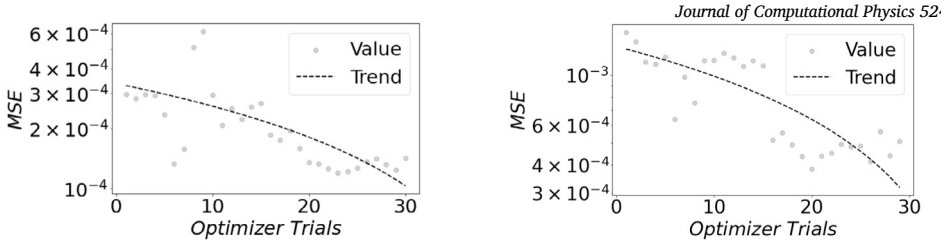
Fig. A.20 illustrates the optimization history of GB-AE-GCN hyperparameters using Bayesian optimization. Each trial is represented by a set of transparent points indicating the `MSE` at the end of training. The dashed black line indicates the trend of error reduction during optimization. The graph underscores a continual decrease in error during the optimization, underscoring the efficacy of the tuning process in discovering the hyperparameters combination that minimizes `MSE` on the validation dataset.

Table A.5
Optimal architecture for Test Case I - Wing–Only Model.

	Block	Layer	Activation	Output Size
Input				m × 86840 × 5
Encoding	Block 0	GCN	PReLU	m × 86840 × 64
	Block 1	GCN	PReLU	m × 86840 × 112
		GCN	PReLU	m × 86840 × 192
		GCN	PReLU	m × 86840 × 256
	Pooling 1			m × 28600 × 256
	Block 2	GCN	PReLU	m × 28600 × 256
		GCN	PReLU	m × 28600 × 288
	Pooling 2			m × 9600 × 288
Reduced Space	Block 3	GCN	PReLU	m × 9600 × 496
		GCN	PReLU	m × 9600 × 288
Decoding	Unpooling 2			m × 28600 × 288
	Block 4	GCN	PReLU	m × 28600 × 256
		GCN	PReLU	m × 28600 × 256
	Unpooling 1			m × 86840 × 256
	Block 5	GCN	PReLU	m × 86840 × 256
		GCN	PReLU	m × 86840 × 192
		GCN	PReLU	m × 86840 × 160
Output	Block 6	GCN	PReLU	m × 86840 × 1
		GCN	PReLU	m × 86840 × 1
		GCN	PReLU	m × 86840 × 1
		GCN	PReLU	m × 86840 × 1
	Concatenate Block 6			
	Prediction			m × 86840 × 4

Table A.6
Optimal architecture for Test Case II - Wing–Fuselage Model.

	Block	Layer	Activation	Output Size
Input				m × 78829 × 5
Encoding	Block 0	GCN	PReLU	m × 78829 × 224
	Block 1	GCN	PReLU	m × 78829 × 192
		GCN	PReLU	m × 78829 × 192
	Pooling 1			m × 26000 × 192
	Block 2	GCN	PReLU	m × 26000 × 240
		GCN	PReLU	m × 26000 × 304
	Pooling 2			m × 8000 × 304
Reduced Space	Block 3	GCN	PReLU	m × 8000 × 432
		GCN	PReLU	m × 8000 × 304
Decoding	Unpooling 2			m × 26000 × 304
	Block 4	GCN	PReLU	m × 26000 × 240
		GCN	PReLU	m × 26000 × 192
	Unpooling 1			m × 78829 × 192
	Block 5	GCN	PReLU	m × 78829 × 192
		GCN	PReLU	m × 78829 × 64
Output	Block 6	GCN	PReLU	m × 78829 × 1
		GCN	PReLU	m × 78829 × 1
		GCN	PReLU	m × 78829 × 1
		GCN	PReLU	m × 78829 × 1
	Concatenate Block 6			
	Prediction			m × 78829 × 4



(a) Test Case I - Wing–only Model (b) Test Case II - Wing–fuselage Model

Fig. A.20. Hyperparameter optimization history of the GB-AE-GCN model for each test case.

Data availability

Data will be made available on request.

References

[1] R. Zimmermann, A. Vendl, S. Görtz, Reduced-order modeling of steady flows subject to aerodynamic constraints, AIAA J. 52 (2014) 255–266, https://doi.org/10.2514/1.J052208.
[2] T. Franz, R. Zimmermann, S. Görtz, N. Karcher, Interpolation-based reduced-order modelling for steady transonic flows via manifold learning, Int. J. Comput. Fluid Dyn. 28 (2014) 106–121, https://doi.org/10.1080/10618562.2014.918695.
[3] T. Franz, Reduced-order modeling for steady transonic flows via manifold learning, Ph.D. thesis, Deutsches Zentrum für Luft-und Raumfahrt, 2016.
[4] M. Ripepi, M.J. Verveld, N. Karcher, T. Franz, M. Abu-Zurayk, S. Görtz, T. Kier, Reduced-order models for aerodynamic applications, loads and mdo, CEAS Aeronaut. J. 9 (2018) 171–193, https://doi.org/10.1007/s13272-018-0283-6.
[5] C. Sabater, P. Stürmer, P. Bekemeyer, Fast predictions of aircraft aerodynamics using deep-learning techniques, AIAA J. 60 (2022) 5249–5261, https://doi.org/10.2514/1.J061234.
[6] R. Castellanos, J.B. Varela, A. Gorgues, E. Andrés, An assessment of reduced-order and machine learning models for steady transonic flow prediction on wings, in: ICAS 2022, 2022.
[7] G. Immordino, A. Da Ronch, M. Righi, Steady-state transonic flowfield prediction via deep-learning framework, AIAA J. (2024) 1–17, https://doi.org/10.2514/1.J063545.
[8] M.M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, P. Vandergheynst, Geometric deep learning: going beyond Euclidean data, IEEE Signal Process. Mag. 34 (2017) 18–42, https://doi.org/10.1109/MSP.2017.2693418.
[9] M. Gori, G. Monfardini, F. Scarselli, A new model for learning in graph domains, in: Proceedings, 2005 IEEE International Joint Conference on Neural Networks, 2005, vol. 2, IEEE, 2005, pp. 729–734.
[10] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, S.Y. Philip, A comprehensive survey on graph neural networks, IEEE Trans. Neural Netw. Learn. Syst. 32 (2020) 4–24, https://doi.org/10.1109/TNNLS.2020.2978386.
[11] Z. Zhang, P. Cui, W. Zhu, Deep learning on graphs: a survey, IEEE Trans. Knowl. Data Eng. 34 (2020) 249–270, https://doi.org/10.1109/TKDE.2020.2981333.
[12] J. Zhou, G. Cui, S. Hu, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, M. Sun, Graph neural networks: a review of methods and applications, AI Open 1 (2020) 57–81, https://doi.org/10.1016/j.aiopen.2021.01.001.
[13] E. Choi, M.T. Bahadori, L. Song, W.F. Stewart, J. Sun, Gram: graph-based attention model for healthcare representation learning, in: Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2017, pp. 787–795.
[14] J. Kawahara, C.J. Brown, S.P. Miller, B.G. Booth, V. Chau, R.E. Grunau, J.G. Zwicker, G. Hamarneh, Brainnetcnn: convolutional neural networks for brain networks; towards predicting neurodevelopment, NeuroImage 146 (2017) 1038–1049, https://doi.org/10.1016/j.neuroimage.2016.09.046.
[15] J. Qiu, J. Tang, H. Ma, Y. Dong, K. Wang, J. Tang, Deepinf: social influence prediction with deep learning, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2110–2119.
[16] T. Nguyen, R. Grishman, Graph convolutional networks with argument-aware pooling for event detection, Proc. AAAI Conf. Artif. Intell. 32 (2018), https://doi.org/10.1609/aaai.v32i1.12039.
[17] D. Zügner, A. Akbarnejad, S. Günnemann, Adversarial attacks on neural networks for graph data, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2018, pp. 2847–2856.
[18] D. Hines, P. Bekemeyer, Graph neural networks for the prediction of aircraft surface pressure distributions, Aerosp. Sci. Technol. 137 (2023) 108268, https://doi.org/10.1016/j.ast.2023.108268.
[19] P. Juangphanich, J. Rush, N. Scannell, Predicting Two-Dimensional Airfoil Performance Using Graph Neural Networks, Technical Report, NASA, 2023.
[20] F. Ogoke, K. Meidani, A. Hashemi, A.B. Farimani, Graph convolutional networks applied to unstructured flow field data, Mach. Learn.: Sci. Technol. 2 (2021) 045020, https://doi.org/10.1088/2632-2153/ac1fc9.
[21] D. Massegur Sampietro, A. Da Ronch, Recurrent multi-mesh convolutional autoencoder framework for spatio-temporal aerodynamic modelling, in: AIAA AVIATION 2023 Forum, 2023, p. 3845.
[22] J. Li, Y. Li, T. Liu, D. Zhang, Y. Xie, Multi-fidelity graph neural network for flow field data fusion of turbomachinery, Energy 285 (2023) 129405, https://doi.org/10.1016/j.energy.2023.129405.
[23] J. Li, T. Liu, G. Zhu, Y. Li, Y. Xie, Uncertainty quantification and aerodynamic robust optimization of turbomachinery based on graph learning methods, Energy 273 (2023) 127289, https://doi.org/10.1016/j.energy.2023.127289.
[24] T. Li, J. Yan, X. Chen, Z. Wang, Q. Zhang, E. Zhou, C. Gong, J. Liu, Accelerating aerodynamic design optimization based on graph convolutional neural network, Int. J. Mod. Phys. C 35 (2024) 2450007, https://doi.org/10.1142/S0129183124500074.
[25] X. Jin, P. Cheng, W.-L. Chen, H. Li, Prediction model of velocity field around circular cylinder over various Reynolds numbers by fusion convolutional neural networks based on pressure on the cylinder, Phys. Fluids 30 (2018) 047105, https://doi.org/10.1063/1.5024595.
[26] K. Fukami, K. Fukagata, K. Taira, Super-resolution reconstruction of turbulent flows with machine learning, J. Fluid Mech. 870 (2019) 106–120, https://doi.org/10.1017/jfm.2019.238.
[27] N. Omata, S. Shirayama, A novel method of low-dimensional representation for temporal behavior of flow fields using deep autoencoder, AIP Adv. 9 (2019) 015006, https://doi.org/10.1063/1.5067313.
[28] J.-Z. Peng, S. Chen, N. Aubry, Z.-H. Chen, W.-T. Wu, Time-variant prediction of flow over an airfoil using deep neural network, Phys. Fluids 32 (2020) 123602, https://doi.org/10.1063/5.0022222.

[29] V. Rozov, C. Breitsamter, Data-driven prediction of unsteady pressure distributions based on deep learning, J. Fluids Struct. 104 (2021) 103316, https://doi.org/10.1016/j.jfluidstructs.2021.103316.
[30] Q. Zhao, X. Han, R. Guo, C. Chen, A computationally efficient hybrid neural network architecture for porous media: integrating cnns and gnns for improved permeability prediction, arXiv preprint arXiv:2311.06418, 2023, https://doi.org/10.48550/arXiv.2311.06418.
[31] D.K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, R.P. Adams, Convolutional networks on graphs for learning molecular fingerprints, Adv. Neural Inf. Process. Syst. 28 (2015), https://doi.org/10.48550/arXiv.1509.09292.
[32] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2016, pp. 481–490.
[33] S. Bhatnagar, Y. Afshar, S. Pan, K. Duraisamy, S. Kaushik, Prediction of aerodynamic flow fields using convolutional neural networks, Comput. Mech. 64 (2019) 525–545, https://doi.org/10.1007/s00466-019-01740-0.
[34] R. Han, Y. Wang, Y. Zhang, G. Chen, A novel spatial-temporal prediction method for unsteady wake flows based on hybrid deep neural network, Phys. Fluids 31 (2019), https://doi.org/10.1063/1.5127247.
[35] K. Fukami, T. Nakamura, K. Fukagata, Convolutional neural network based hierarchical autoencoder for nonlinear mode decomposition of fluid field data, Phys. Fluids 32 (2020), https://doi.org/10.1063/5.0020721.
[36] D. Massegur Sampietro, A. Da Ronch, Graph convolutional multi-mesh autoencoder for steady transonic aircraft aerodynamics, Mach. Learn.: Sci. Technol. (2023), https://doi.org/10.1088/2632-2153/ad36ad.
[37] J. Gu, Z. Wang, J. Kuen, L. Ma, A. Shahroudy, B. Shuai, T. Liu, X. Wang, G. Wang, J. Cai, et al., Recent advances in convolutional neural networks, Pattern Recognit. 77 (2018) 354–377, https://doi.org/10.1016/j.patcog.2017.10.013.
[38] Z. Li, F. Liu, W. Yang, S. Peng, J. Zhou, A survey of convolutional neural networks: analysis, applications, and prospects, in: IEEE Transactions on Neural Networks and Learning Systems, 2021.
[39] G. Quaranta, P. Masarati, P. Mantegazza, A conservative mesh-free approach for fluid structure problems in coupled problems, in: International Conference for Coupled Problems in Science and Engineering, Santorini, Greece, 2005, pp. 24–27.
[40] G.R. Joldes, H.A. Chowdhury, A. Wittek, B. Doyle, K. Miller, Modified moving least squares with polynomial bases for scattered data approximation, Appl. Math. Comput. 266 (2015) 893–902, https://doi.org/10.1016/j.amc.2015.05.150.
[41] R. De Maesschalck, D. Jouan-Rimbaud, D.L. Massart, The Mahalanobis distance, Chemom. Intell. Lab. Syst. 50 (2000) 1–18, https://doi.org/10.1016/j.patcog.2008.05.018.
[42] M.D. Ribeiro, M. Stradtner, P. Bekemeyer, Unsteady reduced order model with neural networks and flight-physics-based regularization for aerodynamic applications, Comput. Fluids 264 (2023) 105949, https://doi.org/10.1016/j.compfluid.2023.105949.
[43] T.N. Kipf, M. Welling, Semi-supervised classification with graph convolutional networks, arXiv preprint arXiv:1609.02907, 2016, https://doi.org/10.48550/arXiv.1609.02907.
[44] M. Fey, J.E. Lenssen, Fast graph representation learning with pytorch geometric, https://doi.org/10.48550/arXiv.1903.02428, arXiv:1903.02428, 2019.
[45] D.K. Hammond, P. Vandergheynst, R. Gribonval, Wavelets on graphs via spectral graph theory, Appl. Comput. Harmon. Anal. 30 (2011) 129–150, https://doi.org/10.1016/j.acha.2010.04.005.
[46] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.
[47] D.P. Kingma, J. Ba, Adam: a method for stochastic optimization, arXiv preprint arXiv:1412.6980, 2014, https://doi.org/10.48550/arXiv.1412.6980.
[48] J. Snoek, H. Larochelle, R.P. Adams, Practical Bayesian optimization of machine learning algorithms, in: F. Pereira, C. Burges, L. Bottou, K. Weinberger (Eds.), Advances in Neural Information Processing Systems, vol. 25, Curran Associates, Inc., 2012.
[49] T. Akiba, S. Sano, T. Yanase, T. Ohta, M. Koyama, Optuna: a next-generation hyperparameter optimization framework, in: Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining, 2019, pp. 2623–2631.
[50] S. Xiang, F. Nie, C. Zhang, Learning a Mahalanobis distance metric for data clustering and classification, Pattern Recognit. 41 (2008) 3600–3612, https://doi.org/10.1016/j.patcog.2008.05.018.
[51] H. Ghorbani, Mahalanobis distance and its application for detecting multivariate outliers, Facta Univ., Ser. Math. Inform. (2019) 583–595, https://doi.org/10.22190/FUMI1903583G.
[52] J.H. Friedman, J.L. Bentley, R.A. Finkel, An algorithm for finding best matches in logarithmic expected time, ACM Trans. Math. Softw. 3 (1977) 209–226, https://doi.org/10.1145/355744.355745.
[53] T. Lassila, A. Manzoni, A. Quarteroni, G. Rozza, Model order reduction in fluid dynamics: challenges and perspectives, in: Reduced Order Methods for Modeling and Computational Reduction, 2014, pp. 235–273.
[54] D.J. Lucia, P.S. Beran, W.A. Silva, Reduced-order modeling: new approaches for computational physics, Prog. Aerosp. Sci. 40 (2004) 51–117, https://doi.org/10.1016/j.paerosci.2003.12.001.
[55] R. Zimmermann, S. Görtz, Improved extrapolation of steady turbulent aerodynamics using a non-linear pod-based reduced order model, Aeronaut. J. 116 (2012) 1079–1100, https://doi.org/10.1017/S0001924000007491.
[56] W.-L. Loh, On Latin hypercube sampling, Ann. Stat. 24 (1996) 2058–2080, https://doi.org/10.1214/aos/1069362310.
[57] T.D. Economon, F. Palacios, S.R. Copeland, T.W. Lukaczyk, J.J. Alonso, Su2: an open-source suite for multiphysics simulation and design, AIAA J. 54 (2016) 828–846, https://doi.org/10.2514/1.J053813.
[58] J. Heeg, Overview of the aeroelastic prediction workshop, in: 51st AIAA Aerospace Sciences Meeting Including the New Horizons Forum and Aerospace Exposition, 2013, p. 783.
[59] J.C. Vassberg, E.N. Tinoco, M. Mani, O.P. Brodersen, B. Eisfeld, R.A. Wahls, J.H. Morrison, T. Zickuhr, K.R. Laflin, D.J. Mavriplis, Abridged summary of the third aiaa computational fluid dynamics drag prediction workshop, J. Aircr. 45 (2008) 781–798, https://doi.org/10.2514/1.30572.
[60] J. Vassberg, E. Tinoco, M. Mani, O. Brodersen, B. Eisfeld, R. Wahls, J. Morrison, T. Zickuhr, K. Laflin, D. Mavriplis, Summary of dlr-f6 wing-body data from the third aiaa cfd drag prediction workshop, RTO AVT-147 Paper 57, https://doi.org/10.2514/6.2007-260, 2007.