# Identification of vortex in unstructured mesh with graph neural networks

Lianfa Wang, Yvan Fournier, Jean-François Wald, Youssef Mesri

# Identification of vortex in unstructured mesh with graph neural networks

Lianfa Wang[a,b], Yvan Fournier[b], Jean-François Wald[b], Youssef Mesri[a]

[a] *Cemef Mines ParisTech — PSL University, CS 10207 rue Claude Daunesse, Sophia Antipolis, 06904, France*

[b] *Dpt. Fluid Mechanics, Energy, Environment (MFEE), Electricité de France, 6 quai Watier, Chatou, 78401, France*

**Abstract**

Deep learning has been employed to identify flow characteristics from Computational Fluid Dynamics (CFD) databases to assist the researcher to better understand the flow field, to optimize the geometry design and to select the correct CFD configuration for corresponding flow characteristics. Convolutional Neural Network (CNN) is one of the most popular algorithms used to extract and identify flow features. However its use, without any additional flow field interpolation, is limited to the simple domain geometry and regular meshes which limits its application to real industrial cases where complex geometry and irregular meshes are usually used. Aiming at the aforementioned problems, we present a Graph Neural Network (GNN) based model with U-Net architecture to identify the vortex in CFD results on unstructured meshes. The graph generation and graph hierarchy construction using algebraic multigrid method from CFD meshes are introduced. A vortex auto-labeling method is proposed to label vortex regions in 2D CFD meshes. We precise our approach by firstly optimizing the input set on CNNs, then benchmarking current GNN kernels against CNN model and evaluating the performances of GNN kernels in terms of classification accuracy, training efficiency and identified vortex morphology. Finally, we demonstrate the adaptability of our approach to unstructured meshes and generality to unseen cases with different turbulence models at different Reynolds numbers.

*Keywords:* Computational Fluid Dynamics (CFD); Vortex identification; Convolutional Neural Network (CNN); Graph Neural Network (GNN); Unstructured mesh.

## 1. Introduction

Vortex identification is of important interest to understand fluid flow characteristics. Deterministic vortex identification methods based on the physical properties of the flow field such as the $Q$-criterion [1], the $\Delta$-criterion [2], and the $\lambda_2$-criterion [3], provide efficient solutions but with many false positives and false negatives as highlighted in [4]. Moreover, these methods require careful selection of appropriate thresholds to obtain valid results. Other methods based on the topological properties of the flow field such as Instantaneous Vorticity Deviation (IVD) [5] are more accurate but computationally expensive. Machine learning methods for vortex identification were introduced recently as an alternative to the existing methods and overcome their shortcomings. It is well known that machine learning methods are good at finding the complex non-linear function between the input and the output if a large amount of data is available. And once trained properly they have better generality than a single criterion to flow cases with different conditions and geometries. However, the current applications of machine learning algorithms on vortex detection are limited to regular mesh and simple flow cases.

Recently, many deep learning algorithms have been employed to identify the flow characteristics in CFD databases. The most successful ones are CNNs [6, 7, 8, 9, 10, 11, 12, 13, 14]. CNN can effectively learn the local features by implicitly embedding the learned translational-invariant features in the sequence of the learnable parameters in convolution kernels. Different CNN architectures were designed to identify the vortexes, including Eddy-Net [15], R-CNN [12], Vortex-Net [9], Vortex-U-Net [16], Vortex-Seg-Net [17] and U-Net CNN [11]. However, the design of the rectangular-like convolution kernel limits the CNNs to only accept the data stored on Cartesian grid which constrains its generality to the data stored on unstructured meshes widely used in the industrial CFD cases. Therefore, most of the current applications of CNNs on detecting flow phenomena are limited to simple flow cases. When dealing with the unstructured meshes and complex geometries, either the mesh deformation [12, 14, 16] or data interpolation [7] is used which inevitably introduces numerical errors.

Compared with CNNs, GNNs taking the data stored on graphs $\mathcal{G}(\mathcal{V}, \mathcal{E})$ as input can naturally adapt to the unstructured meshes used in the real industrial CFD cases to reflect the complex geometries. A graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is formed by a set of nodes $\mathcal{V}$ connected by a set of edges $\mathcal{E}$. The data can be stored on

the edges, nodes or both. Many GNN kernel functions are proposed to gather neighbor information to the center node. One popular type is the isotropic kernel functions such as graph convolutional network (GCN)[18], GraphSage [19]. The isotropic aggregation functions treat the features from different neighbors equally and fail to take into account the spatial distribution of the neighbors. Another type of GNN kernels is anisotropic and distinguishes the edges by assigning each edge an importance. There are many anisotropic kernels using different mechanisms, like attention mechanism [20] or edge gates [21]. However, the above-mentioned kernels are designed to perform regression or classification on networks where there are no Euclidean coordinates, such as social networks, citation networks and biological networks, and only the connections between nodes are important. On the contrary, the coordinates of the nodes or the relative positions of the neighbors to the central nodes are very important on identifying flow characteristics in CFD results which reside on meshes. Therefore, these algorithms are unsuitable to identify the flow phenomena with a certain spatial structure such as the vortex on the CFD meshes.

Inspired by the effectiveness of CNN learning local features in the images, many researchers tried to generalize CNN paradigms to irregular grids which fall into geometric deep learning domain [22, 23]. Masci et al. [24, 25] designed geodesic convolutional neural networks to extract invariant shape features on Riemannian manifolds by constructing the convolution kernel on a local geodesic system of polar coordinates. Boscaini et al. [26] generalized convolutions to non-Euclidean domains by constructing a set of oriented anisotropic heat diffusion kernels whose shape, orientation and scale can be varied by the trainable parameters and achieved state-of-the-art results on learning correspondences between deformable shapes. Monti et al. [27] designed a MoNet algorithm which can learn the edge importance according to the pseudo-coordinates of the two connected nodes using the Gaussian Mixture Model (GMM) as the kernel. GMM learns the mean vector and standard deviation of the edge vector pointing from a central node to neighbor nodes. Therefore, GMM assigns an importance to each edge according to the relative position of the connected neighbor nodes with respect to a center node. Fey et al. [28] proposed a spline-based convolutional neural network (SplineCNN), a generalization of traditional CNN convolution kernel by using B-spline basis functions parameterized by trainable weights which learns on unstructured and geometric input. To the best of our knowledge, GMM and SplineCNN are the two GNN kernels the most equivalent counterparts

of the convolution kernel in traditional CNNs.

In this paper, we propose a GNN framework which can precisely identify the vortex in CFD results as the traditional CNN does but accepts unstructured meshes. The proposed GNN model uses U-Net architecture which utilizes the graph hierarchy generated from AMG method embedded in code_saturne, an open-source, finite volume method (FVM) based CFD software developed by Electricité de France (EDF), to accelerate the training and improve the classification performance. We detail how both the graphs and graph hierarchy are generated from CFD meshes and propose a vortex auto-labeling method based on biased random walking algorithm to generate dataset. We show both advantages and disadvantages of two GNN kernels, GMM and SplineCNN, the two most equivalent counterparts of the convolution kernel in traditional CNN.

The rest of this paper is organized as follows. The relevant convolution kernels, including traditional CNN, GMM, SplineCNN and GCN, are introduced in Section 2. The model architecture, the graph generation details and the algebraic multigrid coarsening method to generate a graph hierarchy for GNN based models are introduced in Section 3. The dataset generation details including CFD case simulation, ground-truth labeling and input sets for machine learning models are introduced in 4. The results of a series of trainings, including training details, evaluation of input sets and GNN kernels, GNNs' adaptability to unstructured meshes and generality analysis are shown in Section 5. Finally, a conclusion is made at the end.

## 2. Related work

During the last several years, CNNs using the convolution to extract the feature embedded have achieved great success in computer vision. Convolution is a specialized type of linear operation used for feature extraction, where a small array of numbers, called a kernel, is applied across the input. An element-wise product between each element of the kernel and the input is calculated at each location of the tensor and summed to obtain the output value in the corresponding position of the output [29]. A typical CNN model normally stacks multiple convolutional layers which use the following equation to update the hidden features from one layer $f^l$ to the next:

$$f^{l+1}(i,j) = \sigma\left(\Theta^l \cdot (f^l \star g)^l)(i,j) + b^l\right), \tag{1}$$

where $(f \star g)$ represents the convolution kernel, $\Theta \in \mathbf{R}^{C_l \times C_{l+1}}$ represents the trainable mapping hidden features between two consecutive layers and have the dimension of $C_l \times C_{l+1}$, $C$ the channel number, $b \in \mathbf{R}^{C_{l+1}}$ the bias trainable, $\sigma$ the non-linear activation function.

While the traditional CNN kernel is only suitable for structured data, like images and data on a cartesian mesh, many researchers have tried to extend its efficiency and accuracy to graph domain including GMM, SplineCNN, GCN and so on. In this section, we introduce the details of these different convolution kernels.

**Traditional CNN.** In traditional CNN, the convolution between the kernel of the size $H \times W$ and the input data located at point $(i, j)$ in the receptive field or image can be expressed as follows:

$$(f \star g)(i, j) = \sum_{h=1}^{H} \sum_{w=1}^{W} f(i + h, j + w) \cdot g_{h,w}, \tag{2}$$

where $g$ is a rectangular trainable kernel with dimension of $H \times W$. As shown in this equation, each neighbor pixel always times the element at the same position in the convolution kernel according to their position relative to the center pixel which enables CNNs to know where the information comes from. And the values in the learned kernels represent the importance of the information from the corresponding directions. Therefore the convolution kernel in traditional CNNs associating the direction with the importance of each gathered features is the key to its success.

**GMM.** Monti et al. [27] extended the traditional CNN convolution to graph by using Gaussian Mixture Model. GMM kernel gathers the hidden features from neighbors to center node with the following equation:

$$(f \star g)(i) = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} f(j) \cdot g_k(\mathbf{e}_{ij}), \tag{3}$$

where hyperparameter $K$ represents the number of directions learned in the kernel, $|\mathcal{N}(i)|$ the degree of node $i$, the edge $\mathbf{e}_{ij}$ pointing from central node $i$ to neighbor node $j$:

$$\mathbf{e}_{ij} = [x_j, y_j]^T - [x_i, y_i]^T, \tag{4}$$

and $g_k(\mathbf{e}_{ij})$ the alignment between the edge $\mathbf{e}_{ij}$ and the $k$-th direction $\mu_k$ in

the kernel:

$$g_k(\mathbf{e}_{ij}) = \exp\left[-\frac{(\mathbf{e}_{ij}-\mu_k)^T(\mathbf{e}_{ij}-\mu_k)}{2\sigma_k}\right], k \in [1, 2, ..., K], \tag{5}$$

where two trainables $\mu_k$ and $\sigma_k$ represent the $k$-th learned direction in the kernel and its variance, respectively. The better the edge $\mathbf{e}_{ij}$ and the learned direction $\mu_k$ aligned, the higher $g_k(\mathbf{e}_{ij})$.

**SplineCNN.** Fey et al. [28] generalized the traditional CNN convolution kernel to interpolate the edge importance from fixed positions to desired positions using B-spline basis functions with the following equation:

$$(f \star g)(i) = \frac{1}{|\mathcal{N}(i)|} \sum_{j \in \mathcal{N}(i)} f(j) \cdot \sum_{\mathbf{k} \in \mathcal{K}} B_{\mathbf{k}}(\mathbf{e}_{ij}) \cdot g_{\mathbf{k}}, \tag{6}$$

where $\mathcal{K}$ is the Cartesian product of the B-spline bases:

$$\mathcal{K} = N^1_{k_1,p} \times ... \times N^D_{k_D,p},\ \mathbf{k} = (K_1, ..., K_D), \tag{7}$$

$\mathbf{k}$ represents the number of control points on each dimension of the $D-$ dimensional kernel, and $g_{\mathbf{k}}$ is the trainables and the control points as well associated with the corresponding product $B_{\mathbf{k}}(\mathbf{e})$ of the basis functions in $\mathcal{K}$:

$$B_{\mathbf{k}}(\mathbf{e}) = \Pi^D_{d=1} N^d_{k_d,p}(e^d). \tag{8}$$

The $k$-th B-spline basis function of degree $p$, written as $N_{k,p}(e^d)$, is defined recursively as follows:

$$N_{k,0}(e^d) = \begin{cases} 1, & \text{if } u_k \le e < u_{k+1}. \\ 0, & \text{otherwise.} \end{cases} \tag{9}$$

$$N_{k,p}(e^d) = \frac{e - u_k}{u_{k+p} - u_k} N_{k,p-1}(e^d) + \frac{u_{k+p+1} - e^d}{u_{k+p+1} - u_k} N_{k+1,p-1}(e^d). \tag{10}$$

where $u_k$ is the knot vector and $e^d$ is the edge coordinate on $d$-th dimension.

Different from those used in [28], multiple knots are used at the ends of x and y directions to force the interpolated surface to converge to the control points at the ends. Three example convolution kernel surfaces are visualised in Fig. 1.

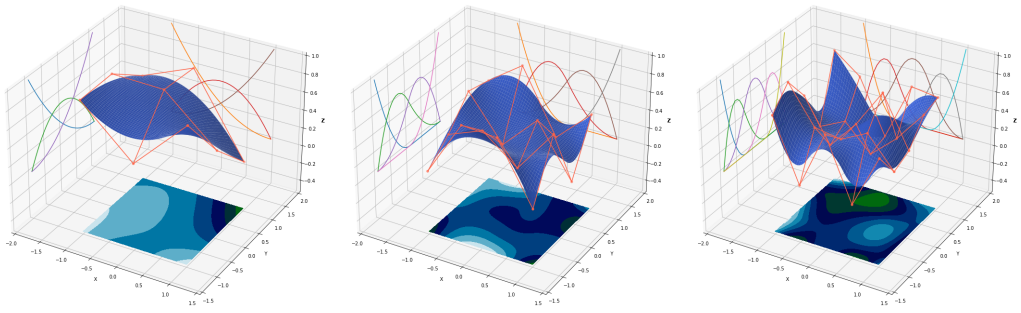


Figure 1: Examples of the convolution kernels for B-spline basis degrees $p = 2$ with control points number $K_1 = K_2 = 3$, 4 and 5, from left to right. The orange points connected by orange wireframe represent the randomly sampled control points. The curves on the yz-plane and xz-plane represent basis functions for each control point on corresponding direction.

**GCN.** A GCN [18] kernel gathers information from all neighbors $\mathcal{N}(i)$ to the center node $i$ with the following equation:

$$(f \star g)(i) = \frac{1}{\sqrt{|\mathcal{N}(i)||\mathcal{N}(j)|}} \sum_{j \in \mathcal{N}(i)} f(j) \cdot g_j \tag{11}$$

where $\mathbf{g}$ is a $\mathcal{N}(i)$ unit vector. GCN kernel is an isotropic kernel and permutation invariant which means the information from all the neighbors is equal no matter in what sequence or from which direction it comes. Therefore, a bad performance of GCN on feature extraction is anticipated. However it shows us where the low bound of the classification performance of a GNN-based algorithm can be, thus GCN kernel is also included in the comparison.

## 3. Architecture and graphs

### *3.1. Architecture*

A 4-depth and 8-layer U-Net architecture, as shown in Fig. 2, is adopted to evaluate all convolution kernels. It consists of a leading contract part, a trailing expansive part, the skip-connections among the corresponding contract and expansive parts, and a bottom level. The contract part has three blocks with each of them having one convolutional layer and a following down-sampling (average pooling) layer. In the contract part, the convolutional layer doubles the channel number except that of the first depth level

where the output channel is fixed to 8 and the input channel is varied, and the down-sampling layer shrinks the hidden feature size from one level to the next deeper level. Correspondingly, the expansive part also has three blocks with each of them having one up-sampling layer and one following convolutional layer. In expansive part, the up-sampling layers up-sample hidden features from a deeper level to the shallower level, and the convolutional layers decrease the channel number by a factor of 4. Each convolutional layer is followed by a rectified linear unit (ReLU) activation function. The successive down-samplings of the input not only reduce the input size thus leading to computational efficiency, but also enable the U-Net architecture to extract local features and global features in the input since a kernel of the same size at deeper levels can cover larger region with respect to the original flow field. The skip-connections alleviate the vanish gradient problem in the sequentially stacked deep architectures. A fully connected (FC) layer is added to the end to map the output to the classification.

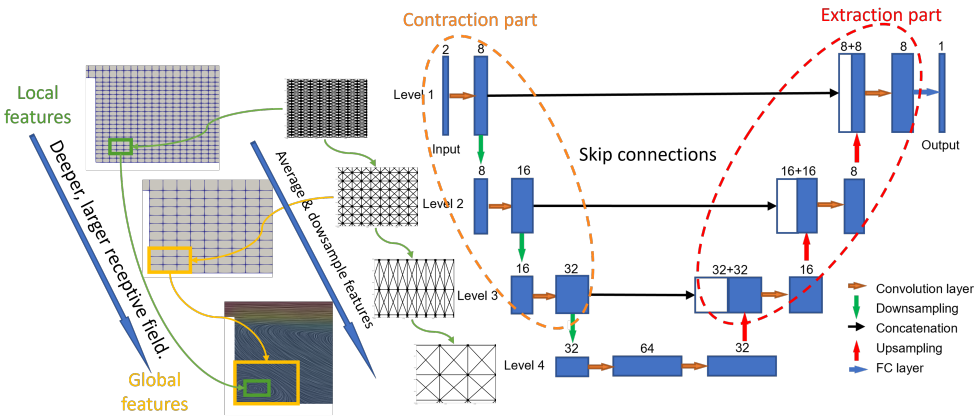



Figure 2: U-Net architecture.

### *3.2. Graph generation*

The graphs fed to the GNNs are derived from the CFD meshes where the cells and the vertices shared by adjacent cells in CFD meshes become the nodes and the edges in graphs. The derived graph is a dual mesh of the CFD mesh. The input data are stored on the nodes. To regularize the graph, the normalized direction $\mathbf{e}_{ij}$ pointing from center node $i$ to neighbor node $j$ is stored on the edge connecting the two nodes:

$$\mathbf{e}_{ij} = \frac{\mathbf{x}_j - \mathbf{x}_i}{||\mathbf{x}_j - \mathbf{x}_i||} \tag{12}$$

where, $\mathbf{x_i}$ and $\mathbf{x_j}$ are the center node and neighbor node coordinates, respectively. There exists one forward edge and one backward edge between two connected nodes since every node can be a center node and a self-loop is added to each node with the distance of $\mathbf{0}$. Therefore the bi-directed graphs with self-loops are generated, as shown in Fig. 3.

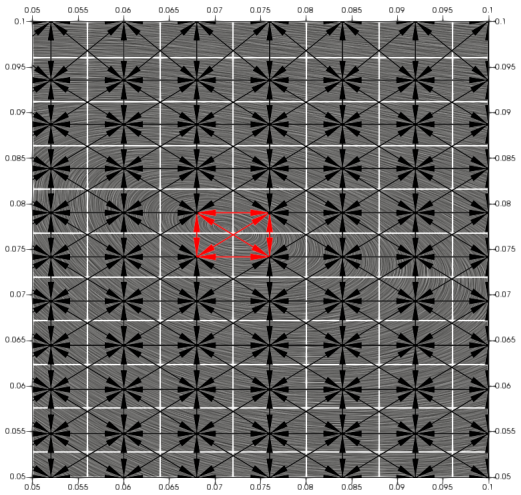


Figure 3: Bi-directed graph superimposed on streamline background. The red arrows connect the vortex core cells. The self-loop of each node is not show here. The white wire frame is the CFD mesh.

### *3.3. Graph coarsening*

On the contrary to the widely used pooling methods in CNNs thanks to their simplicity, there is no widely-accepted sampling method for GNNs because it requires a hierarchy of graphs with different refinement levels whose generation depends on their application domain. There are several different methods to coarsen graphs including but not limited to: 1) spectral graph sparsification [30], 2) algebraic multigrid (AMG) [31, 32], 3) geometric graph coarsening and even 4) GNN-based graph coarsening [33]. In code_saturne, an algebraic multigrid method is used to coarsen the meshes to accelerate the convergence of the linear algebraic equations $Au = b$. The equations resulted from fine grid are iterated once and the residuals are restricted to right hand

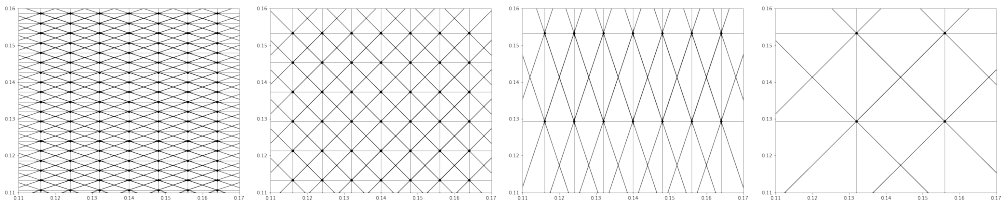
side of the equations at the next level where a coarser grid is constructed by aggregating the strongly coupled adjacent cells at fine levels which is decided by the relative magnitude of the corresponding values in the coefficient matrix. This process can go on and on until a maximum number of coarsening levels is reached or a matrix is small enough to be directed solved by conjugate gradient iteration method. Then the converged solution is successively prolonged from coarser levels to finer levels to correct the solution in the finer level. AMG and U-Net architecture are both designed to separate components of different scales in the signal in order to accelerate the calculation. We construct the graph hierarchy based on the coefficient matrix from the pressure correction step at this stage. In the pressure correction step, the Poisson equation is solved:

$$div(\Delta t \underline{\nabla} p') = div(\rho \underline{\tilde{u}}) \tag{13}$$

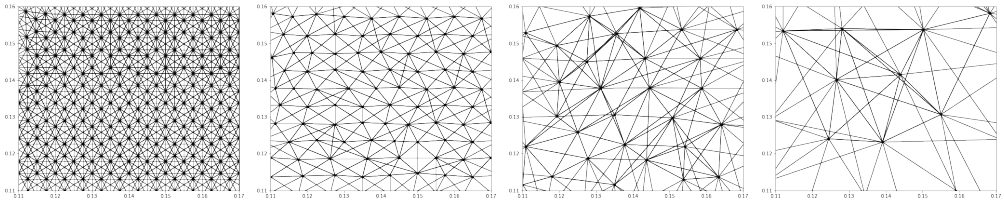
where $\Delta t$ is the time step, $p'$ the pressure increment and $\underline{\tilde{u}}$ the velocity field resulting from the prediction step, $\rho$ density. For the internal node $i$, the integrated form of the above equation is:

$$\sum_{j \in Neigh(i)} [\Delta t (\underline{\nabla} p')_{f_{ij}}] \cdot S_{ij} = \sum_{j \in Neigh(i)} \rho \underline{\tilde{u}} \cdot \underline{S}_{ij} \tag{14}$$

where $j$ is $i$'s neighbor node, $f_{ij}$ the interface between nodes $i$ and $j$, and $S$ the surface vector. Since the AMG method used here is aggregation-based which uses only the information present in the system matrix $A$, which here corresponds to $\Delta t S_{ij}$. It means that the mesh coarsening depends only on the mesh topology which can evenly coarsen the original mesh to much deeper levels. The coarsening based on other variables will be tested in the future. The details of the AMG algorithm are out of the scope of this paper and the interested readers are referred to Y. Notay's work[31]. Two example graph hierarchies generated using AMG method from backward-facing step (BFS) structured and unstructured meshes are shown in Fig. 4. The sizes of images used for CNNs and graphs generated by AMG from different mesh types are summarized in Table 1. Since CNNs only accept a rectangular array, therefore only 240×220 mesh cells behind the step are used as input. By contrary, the graph derived from the entire mesh is used for GNN-based methods, thus leading to the discrepancy between the number of pixels in images and that of nodes in graphs at level 1.



(a) Structured mesh.



(b) Unstructured mesh.

Figure 4: Graph hierarchy generated from BFS meshes for four levels in U-Net architecture. The first graph is generated from original mesh, the following three graphs are generated by AMG.

Table 1: The details of images and graphs.

| Level | Imges ($Pixels_{W\times H}$) | AMG graphs (Structured) | | AMG graphs (Unstructured) | |
|---|---|---|---|---|---|
| | | Nodes | Edges | Nodes | Edges |
| 1 | $52800_{240\times220}$ | 74400 | 666124 | 131118 | 1693758 |
| 2 | $13200_{120\times110}$ | 24720 | 220002 | 50780 | 441429 |
| 3 | $3300_{60\times55}$ | 8231 | 72401 | 17127 | 196325 |
| 4 | $810_{30\times27}$ | 2719 | 23727 | 5596 | 67212 |

## 4. Dataset

### *4.1. CFD case simulation*

The dataset was generated with code_saturne [34] which is an open-source software developed primarily by EDF for CFD applications. It solves the Navier-Stokes equations for 2D, 2D-axisymmetric and 3D flows, steady or unsteady, laminar or turbulent, incompressible or weakly dilatable, isothermal or not, with scalar transport. It provides multiple turbulence models,

including Reynolds-Averaged Navier-Stokes (RANS) models, Reynolds Stress Models (RSM) and Large Eddy Simulation (LES) models, and many of specific physical modules: coal and heavy-fuel oil combustion, semi-transparent radiative heat transfer, particle-tracking with Lagrangian modeling, Joule effect, electrics arcs, weakly compressible flows, atmospheric flows, rotor/stator interaction for hydraulic machines.

The CFD simulation of turbulent flow over BFS [35] is simulated using transient code_saturne solver by solving Navier-Stokes equations:

$$\frac{\partial \rho}{\partial t} + div(\rho \underline{u}) = 0, \tag{15}$$

$$\frac{\partial \rho \underline{u}}{\partial t} + div\left(\rho \underline{u} \otimes \underline{u} + P \underline{\underline{Id}} + \underline{\underline{R}}\right) = 0, \tag{16}$$

and $R_{ij} - \epsilon$ turbulence model [36]:

$$\frac{\partial R_{ij}}{\partial t} + div\left(\rho \underline{u} R_{ij} - \mu \underline{\nabla} R_{ij}\right) = P_{ij} + G_{ij} + \Phi_{ij} + d_{ij} - \rho \epsilon_{ij} + S'_{ij}, \tag{17}$$

$$\frac{\partial \epsilon}{\partial t} + div\left(\rho \underline{u} \epsilon - \mu \underline{\nabla} \epsilon\right) = d_\epsilon + C_{\epsilon 1} \frac{\epsilon}{k} P - \rho C_{\epsilon 2} \frac{\epsilon^2}{k} + S'_\epsilon, \tag{18}$$

where, $P_{ij}, G_{ij}, \Phi_{ij}, d_{ij}, \rho\epsilon_{ij}$ are the generation term, production-destruction term related to gravity effects, pressure-strain term, dissipation term and turbulent diffusion term of $R_{ij}$ respectively, $d_\epsilon$ and $P$ turbulent diffusion term and generation term for $\epsilon$ respectively, $S_{ij}$ and $S_\epsilon$ additional source terms for $R_{ij}$ and $\epsilon$ respectively.

The flow configuration is shown in Fig. 5. The computational domain consists of an inlet section $L_i = 10h$ prior to the sudden expansion. The total length in streamwise direction is $L_x = 30h$ and the vertical height is $L_y = 6h$. The mean velocity with the turbulence intensity of 5% of the mean velocity magnitude is imposed at the inlet boundary. The Neumann condition is imposed at the outlet boundary. The scalable wall function is applied to the bottom wall. The upper boundary is set to symmetry boundary condition. The Reynolds number $Re_h$ based on step height $h$ and bulk velocity is 5100. The total number of computational cells in x direction after the step is 240 and that in y direction is 220. Only one cell is used for z direction.

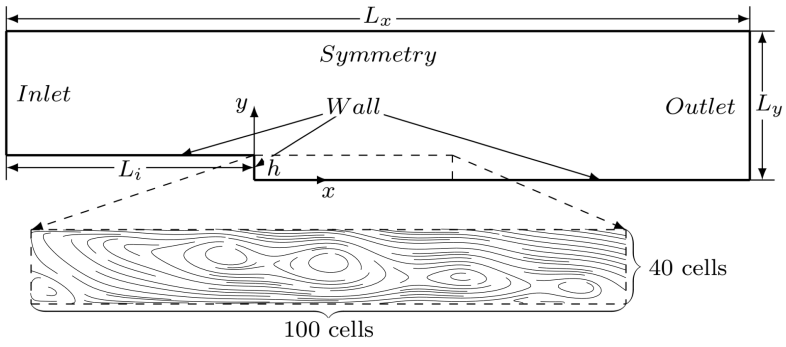



Figure 5: Backward-facing step flow configuration.

At least twenty flow-throughs, flow time from 0s to 10s, were calculated before outputting the result in order to make sure the flow field fully developed. Then the physical fields at 100 time steps with an interval of 0.05s, flow time from 10s to 15s, were output and divided into training, validation and testing cases . The dataset contains results at 100 time steps in total of which the first 80 times steps from 10s to 14s, 10 time steps from 14s to 14.5s, and 10 time steps from 14.5s to 15s were used as training cases, validation cases and test cases, respectively.

### *4.2. Ground-truth labeling*

Since the supervised training of the neural networks is adopted in our study, a binary ground-truth label for each node indicating whether the node is in the vortex region or not should be provided. We propose an auto-labeling method using Depth First Search (DFS) and Biased Random Walking (BRW) on the directed graph derived from the velocity field, as shown in Fig. 6. The edge in the directed graph is the shared face of adjacent cells in the mesh whose direction is from upwind cell to downwind cell according to the velocity field.

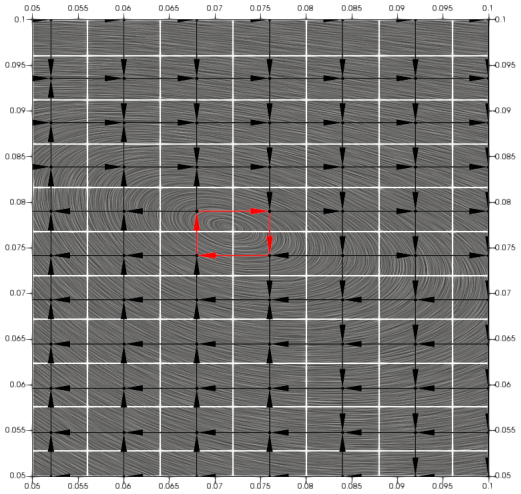


Figure 6: Directed graph superimposed on streamline background. The red arrows connect the vortex core cells. The white wire frame is the CFD mesh.

As shown in Algorithm 1, this vortex auto-labeling algorithm labels the vortex in 2D CFD results based on the closure of the streamline in two steps: a) locating the vortex core with DFS algorithm [37]; b) enlarging the vortex region by BRW. For the first step, we assume that a vortex core exists among limited adjacent cells. The DFS algorithm, as shown in Algorithm 2, is used to traverse all the nodes on the graph and recursively search the downstream nodes until it goes back to the beginning node within predefined steps. Once the vortex core is located, the BRW algorithm, as shown in Algorithm 3, enlarges the vortex region starting from the vortex core. The probability of random walking from central node to a downstream node is proportional to the ratio of mass flux entering the corresponding downstream node from central node to the total mass flux exiting the central node. One walking path either ends at the beginning node leading to the success of enlarging the vortex region or ends at the outlet nodes leading to the failure of enlarging vortex region. To make sure the final labeled vortexes are separate from each another, a single enlarged vortex region shouldn't contain multiple vortex cores. The vortex location labeling process is terminated when the searched vortex boundary is not changed from the last random walking. The auto-labeling algorithms takes 292s and 122s on average to label vortexes on the 2D BFS structured and unstructured meshes, respectively. The streamline plot and ground-truth label obtained with this method at one time instant

from both structured and unstructured meshes are shown in Fig. 7. All the cells in vortex zone are represented by the points over the streamline background. It should be noted that although the vortex boundary labeled in this method doesn't precisely reflect the real vortex shape leading to a noisy dataset, it has the merit of auto-labeling and can label vortex location for massive snapshots. And we expect that the machine learning algorithm outperforms the ground-truth label on identifying the vortex position and shape after trained on this noisy dataset.

**Algorithm 1:** Vortex auto-labeling algorithm in 2D CFD cases

**input :** Nodes $\mathcal{V}$, Mass flux $\{M_{u,v} : u \in \mathcal{V}, v \in \mathcal{V}\}$, Neighborhood $\mathcal{N}(u) = \{v \in \mathcal{V} : M_{u,v} > 0\}$

**output:** Vortex nodes: $\mathcal{V}_{vortex}$

Build weighted directed graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with edge weight $e_{u,v} = \frac{M_{u,v}}{\sum_{v \in \mathcal{N}(u)} M_{u,v}}$;
initialize vortex core set $\mathcal{V}_{core}$;
**for** *each* $u \in \mathcal{V}$ **do**
    add `DFS` $(\mathcal{G}, v, 0)$ to $\mathcal{V}_{core}$;
**end**
create working graph $\mathcal{G}_w \leftarrow \mathcal{G}$;
**while** $\mathcal{G}_w$ *changes* **do**
    **for** *icore in* $\mathcal{V}_{core}$ **do**
        randomly select a node $u$ from *icore*;
        find the loop enclosing $u$: $L_u \leftarrow$`BRW` $(\mathcal{G}_w, u)$;
        remove the nodes enclosed by $L_u$ from $\mathcal{G}_w$;
    **end**
**end**
return $\mathcal{V}_{vortex} = \mathcal{G}.nodes() - \mathcal{G}_w.nodes()$;

**Algorithm 2:** Depth first search algorithm

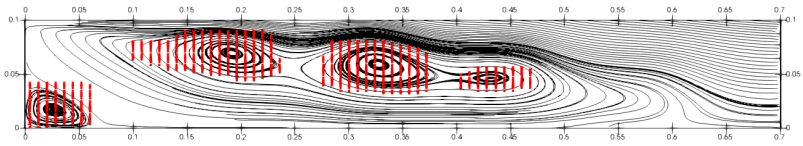
**input :** Graph $\mathcal{G}$, starting node $v$, depth count $d$
**output:** Vortex core nodes

```
d++1;
for w in G.adj[v] do
    if w is unvisited and d < depth then /* Set depth to 4 for
     structured mesh, 8 for unstructured mesh.          */
        if DFS (G, w, d) then
            return list(w, DFS (G, w, d));
        end
    else if w is visited then
        return w;
    else
        return None
    end
end
```

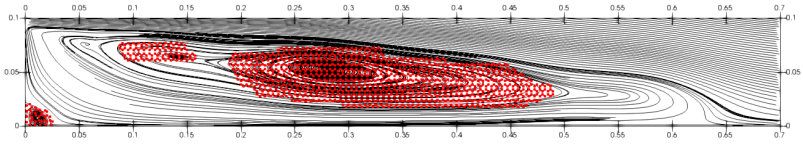
**Algorithm 3:** Biased random walking

**input :** Graph $\mathcal{G}$, Starting point $u$
**output:** Nodes on the loop $\mathbf{V}_{path}$ enclosing $u$

```
while step < threshold do /* Threshold depends on mesh size.
 */
    add u to path;
    select next node v in G.adj[u] with possibility P(v) ∝ e_{u,v};
    if v in V_core then
        return V_path;
    end
    walk to next v : u ← v;
    step++1;
end
```



(a) Structured mesh.



(b) Unstructured mesh.

Figure 7: Ground-truth label points on streamline background.

### *4.3. Input selection*

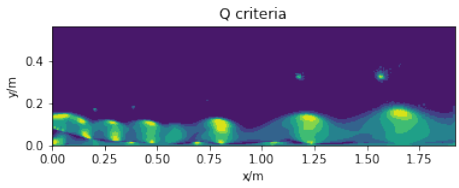
To find out the optimal vortex indicators as the input, five input sets were tested as summarized in Table 2. The velocity field as the baseline input set and their normalized counterpart were tested. Their definitions are summarized in Table 3. The non-dimensional $Q$ criterion and other three non-dimensional vortex indicators which have certain spatial distribution were selected. As shown in Fig. 8, the turbulence intensity overlaps the vortex region. There are characteristic lines traversing through the vortex region for deviation from shear flow field and pressure gradient along streamline field. The combination of input set #1 and input set #4 is also tested.

Table 2: Five input sets.

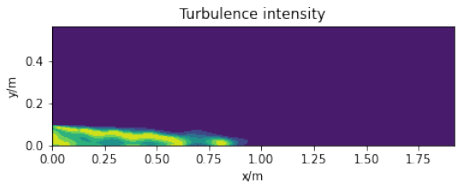
| Input set | Variables |
|---|---|
| 1 | U, V |
| 2 | $U_{norm}, V_{norm}$ |
| 3 | $Q$ |
| 4 | Turbulence intensity<br>Deviation from shear flow<br>Pressure gradient along streamline |
| 5 | U, V<br>Turbulence intensity<br>Deviation from shear flow<br>Pressure gradient along streamline |

Table 3: Input variables

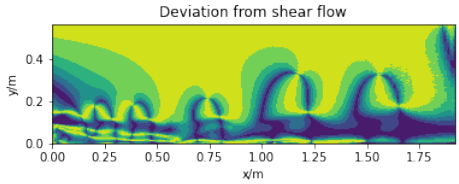
| Description | Formula |
|---|---|
| Normalized velocity | $\mathbf{U}/\|\|\mathbf{U}\|\|$ |
| Q-criterion | $\frac{\|\|\mathbf{R}\|\|^2-\|\|\mathbf{S}\|\|^2}{\|\|\mathbf{R}\|\|^2+\|\|\mathbf{S}\|\|^2}$ with $\mathbf{R}=\frac{1}{2}(\nabla\mathbf{U}-(\nabla\mathbf{U})^T)$ and $\mathbf{S}=\frac{1}{2}(\nabla\mathbf{U}+(\nabla\mathbf{U})^T)$ |
| Turbulence intensity | $k/\left(0.5U_iU_i+k\right)$ |
| Pressure gradient along streamline | $U_k\frac{dP}{dx_k}/\left(\sqrt{\frac{dP}{dx_j}\frac{dP}{dx_j}U_iU_i}+\|U_l\frac{dP}{dx_l}\|\right)$ |
| Deviation from parallel shear flow | $\|U_kU_l\frac{dU_k}{dx_l}\|/\left(\sqrt{U_nU_nU_i\frac{dU_i}{dx_j}U_m\frac{dU_m}{dx_j}}+\|U_iU_j\frac{dU_i}{dx_j}\|\right)$ |



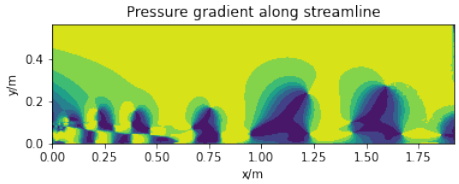

(a) Q criteria



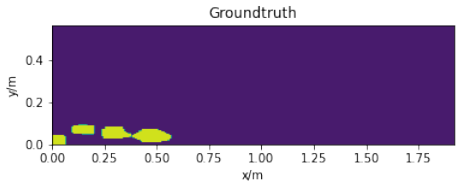

(b) Turbulence intensity.




(c) Deviation from shear flow.




(d) Pressure gradient along streamline.




(e) Ground-truth label.

Figure 8: The contour plot of vortex indicators and ground-truth of the test case at time=12.65s.

## 5. Experimental study

### *5.1. Training details*

Firstly, five CNN models are trained on the structured mesh in order to select the optimal input set. With the selected input set, three GNN kernels are evaluated on graphs generated by AMG method from 2D BFS structured mesh and compared with CNN. The kernel size is set to 9 and $3 \times 3$ for GMM and SplineCNN respectively to make the their trainables comparable to CNN model, while GCN model has much less trainables than other models because of its isotropic nature. We tested GCNs with trainables at the same order as other models, however the results are not very different from the current one. For simplicity, those tests are not included here. The trainables of different models are summarized in Table 4. The GNN models are trained on graphs generated from unstructured mesh to demonstrate their adaptability to unstructured mesh. As last, in order to demonstrate the generality and limitation of the proposed method, the GNN models are trained on BFS structured and unstructured meshes and then applied to unseen cases at different Reynolds numbers simulated by different turbulence models with different meshes.

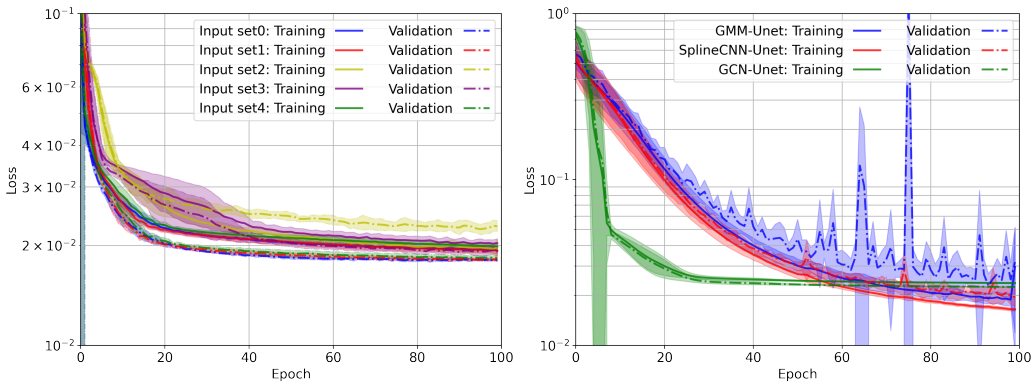
Table 4: Summary of details of different models.

| No. | Model | Layers | Kernel size | Trainables |
|---|---|---|---|---|
| 1 | CNN | 8 | 3×3 | 55633 |
| 2 | GMM | 8 | 9 | 55921 |
| 3 | SplineCNN | 8 | 3×3 | 55633 |
| 4 | GCN | 8 | - | 6353 |

All the trainings are trained for 100 epochs and repeated five times with different random seeds to demonstrate the stability of the model. The mean and standard deviation of the training loss and classification evaluation metrics are calculated. The Adam optimizer is used to minimize the binary cross-entropy loss function. The learning rate and weight decay rate are set to $10^{-3}$ and $10^{-5}$, respectively. The batch size is set to 5. All the GNN models are implemented in Deep Graph Library (DGL) in Pytorch framework and trained on single Intel Xeon Platinum 8260 CPU @ 2.40GHz on the cronos cluster of EDF.

The training loss histories for CNNs with five input set and GNNs on AMG graphs from structured mesh are shown in Fig.9. CNNs give very

good identifications within just 10 epochs while GNNs need more epochs. The training of all GNNs are stable except GMM which has a significant fluctuation for validation loss.




(a) CNNs with five input sets.

(b) GNNs on AMG graphs from structured mesh.

Figure 9: Training loss history. The curve and shaded region represent the mean and standard deviation of five trainings, respectively.

### *5.2. Evaluation of input sets*

Since the dataset is a biased one where only a small portion of points belong to vortex region, therefore the following four classification metrics are used to evaluate the classification performance: accuracy, precision, recall and F1 score:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}, \tag{19}$$

$$Precision = \frac{TP}{TP + FP}, \tag{20}$$

$$Recall = \frac{TP}{TP + FN}, \tag{21}$$

$$F1 = \frac{2 \times Precision \times Recall}{Precision + Recall}, \tag{22}$$

where $TP, TN, FP, FN$ are the numbers of true positives, true negatives, false positives and false negatives, respectively. The performances of five input sets are evaluated in the region behind the step ($x > 0$) with CNN-Unet algorithm and summarized in Table 5. The highest and lowest values of each of four classification evaluation metrics among all input sets are labeled

in green and red color, respectively. It is obvious that input sets #1 and #2 are generally better than other input sets in terms of accuracy, precision and F1 score, while the input sets #4 has the worst performance. Input set #2 outperforms other input sets on training time per epoch by a large margin. Input set #3 (Q-criterion) has the highest standard deviations for all metrics and it also needs the longest training time. Compared with input sets #1 and #2, additional inputs in Input set #5 do not bring performance improvement but the increased training time.

Table 5: Performance of different input sets. (Green: best value; Red: worst value.)

| No. | Accuracy | Precision | Recall | F1 score | Time/epoch |
|---|---|---|---|---|---|
| 1 | 88.03±0.16 | 90.86±0.48 | 61.73±0.81 | 73.51±0.50 | 2.63±0.79s |
| 2 | 88.02±0.27 | 91.82±0.49 | 60.92±0.94 | 73.24±0.74 | 1.85±0.13s |
| 3 | 87.83±1.10 | 89.38±3.97 | 62.74±7.78 | 73.22±4.32 | 4.43±0.34s |
| 4 | 87.53±0.31 | 90.42±0.86 | 60.03±1.23 | 72.15±0.89 | 2.42±0.27s |
| 5 | 87.64±0.12 | 91.44±0.49 | 59.68±0.61 | 72.22±0.37 | 3.32±1.63s |

A receiver operating characteristic (ROC) curve [38] is a graphical plot that illustrates the diagnostic ability of a binary classifier system as its discrimination threshold varies. The ROC curve is created by plotting the true positive rate (TPR) against the false positive rate (FPR) at various threshold settings. The ideal prediction model should yield a point in the upper left corner or coordinate (0,1) in the ROC space, representing no false negatives and no false positives. As shown in Fig. 10, the ROC curves of all input sets are very close and input set #2 is slightly better than the others.

As shown in Fig. 11, although all input sets can be used to precisely identify the vortex position, those identified with input sets #2 have the shapes closest to the real vortexes. Since with input set #2, CNN-Unet achieves a good performance on four classification evaluation metrics, has the best ROC curve and best identified vortexes shape, and shortest training time per epoch, we continue to compare different GNN-based models based on input set #2 - normalized velocity field.

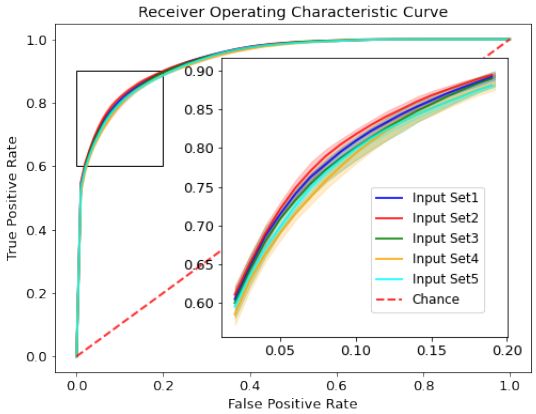



Figure 10: Receiver operating characteristic curves of CNN-Unet with different input sets. (The curve and shaded region represent the mean and standard deviation of five trainings respectively.)

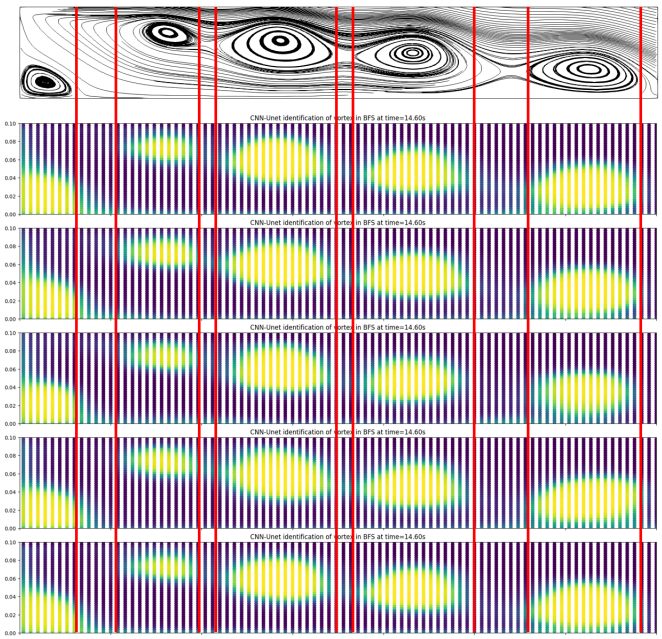


Figure 11: Vortexes at time=14.60s identified by CNN-Unet algorithm with different input sets. From top to bottom: streamline, input sets from #1 to #5.

### *5.3. Evaluation of GNN kernels*

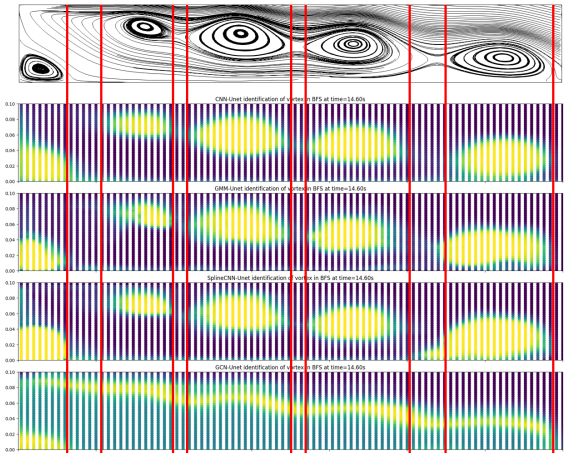
We evaluate the performance of three GNN-based kernels on graphs generated by AMG from structured mesh by comparing them with training No. 2 of CNN models in Table 5 with the same input set - normalized velocity field.

**Classification performance.** The classification performance of four models is summarized in Table 6. GMM kernel performance is very close to that of CNN kernel. SplineCNN kernel outperforms CNN kernel with higher mean values of all classification evaluation metrics except for precision and its standard deviations are smallest among four GNN kernels although higher than CNN. Among all models, GCN model has the worst classification performance, which is expected.

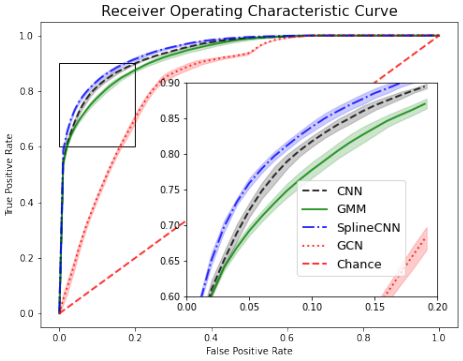
Regarding the computational efficiency, it should be pointed out that GCN has disproportionately longer training time compared to its significantly smaller trainables, which suggests that it struggles to back-propagate the gradients since it cannot identify the features with spatial distribution. Except GCN, CNN training time is one order lower than those of GNN-based models. The training time of SplineCNN is the highest and 1.5 times longer than that of GMM.

Table 6: Comparison of classification performances of GNN kernels on graphs generated by AMG with CNN. (Among three GNN-based models: Green: best value; Red: worst value.)

| Model | Accuracy | Precision | Recall | F1 score | Training time/epoch | Inference time/case |
|---|---|---|---|---|---|---|
| CNN | 88.02±0.27 | 91.82±0.49 | 60.92±0.94 | 73.24±0.74 | 1.85±0.13s | 0.008s |
| GMM | 88.13±0.14 | 87.95±1.41 | 64.81±1.69 | 74.59±0.65 | 44.14±0.11s | 0.291s |
| SplineCNN | 89.28±0.37 | 91.70±0.63 | 66.17±2.03 | 76.84±1.15 | 67.59±0.16s | 0.553s |
| GCN | 76.47±0.23 | 62.17±1.42 | 32.29±2.12 | 42.43±1.71 | 10.94±0.03s | 0.097s |



(a) Identified vortex at time=14.60s. From top to bottom: streamline, CNN, GMM, SplineCNN and GCN.




(b) Receiver operating characteristic curves. (The curve and shaded region represent the mean and standard deviation of five trainings respectively.)

Figure 12: Comparison of GNN kernels against CNN kernel.

**Vortex morphology.** As shown in Fig. 12a, the shapes and positions of the vortexes identified by CNN and SplineCNN kernels are very close to each other, both are very clear and accurate. By comparison, the vortexes' boundary identified by GMM kernel is diffused. GCN kernel fails to identify neither the vortex shape nor position.

**ROC curve.** As shown in Fig. 12b, among all models, GCN kernel has the worst ROC and SplineCNN kernel has the ROC curve closest to the left upper corner which means it's classification can obtain the highest positive rate and the lowest false positive rate. The ROC curves of CNN and SplineCNN kernel are very close to CNN's ROC and slightly better than that of GMM.

### *5.4. Adaptability to unstructured mesh*

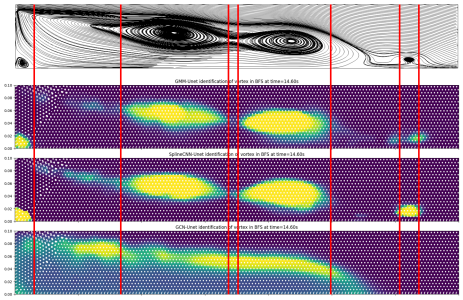
To demonstrate their adaptability to unstructured meshes, the three GNN kernels are trained on graphs generated by AMG method from unstructured mesh, whose edge and node information is summarized in Table 1. As shown in Table 7, SplineCNN outperforms GMM on four classification evaluation metrics, higher mean values and smaller standard deviations, especially for recall and F1 score whose mean values exceed those of GMM by a considerable margin. However, the better classification performance of SplineCNN is also accompanied by considerable computational overhead, about 66.6% higher than GMM which is a disadvantage dealing with 3D CFD cases with

much more mesh cells in the future. Three of five GCN trainings did not converge resulting in the Nan±Nans in the table.

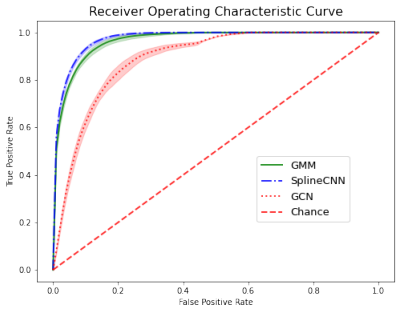
Table 7: Classification performance of GNN kernels on graphs generated by AMG from unstructured mesh. (Between GMM and SplineCNN: Green: best value; Red: worst value.)

| Kernel | Accuracy | Precision | Recall | F1 score | Training time/epoch | Inference time/case |
|---|---|---|---|---|---|---|
| GMM | 92.76±0.27 | 81.00±1.58 | 67.27±3.22 | 73.43±1.55 | 104.05±0.34s | 0.755s |
| SplineCNN | 93.54±0.26 | 82.21±1.11 | 72.41±2.42 | 76.97±1.23 | 173.32±0.20s | 1.530s |
| GCN | 85.60±0.68 | Nan±Nan | 9.73±12.14 | Nan±Nan | 19.72±0.11s | 0.223s |

As shown in Fig. 13a, GMM well identifies the shape of the small vortexes and fails on elongated vortexes while SplineCNN does the opposite. GCN can neither identify the vortex shape nor the position. As shown in Fig. 13b, the ROC curve of SplineCNN is very close to but slightly better than that of GMM and both ROC curves of two anisotropic kernels are far better than that of isotropic kernel - GCN.



(a) Identified vortex.




(b) Receiver operating characteristic curves. (The curve and shaded region represent the mean and standard deviation of five trainings respectively.)

Figure 13: GNNs on unstructured mesh.

### *5.5. Generality analysis*

To further evaluate the generality of the proposed approach on detecting vortexes with respect to different meshes in terms of mesh type, mesh density and mesh aspect ratio for structured mesh, different turbulence models and different Reynolds numbers, the GMM-Unet and SplineCNN-Unet models

are trained again on the dataset formed by the BFS results obtained using both BFS structured and unstructured meshes and then tested on the three of code_saturne validation cases: lid-driven cavity flow, heat transfer in a cooling channel with periodic ribs (RIBS)[39, 40], asymmetric plane diffuser flow[41, 42]. The main simulation details of these cases are summarized in Table 8. The configuration of three cases are shown in Fig. 14.

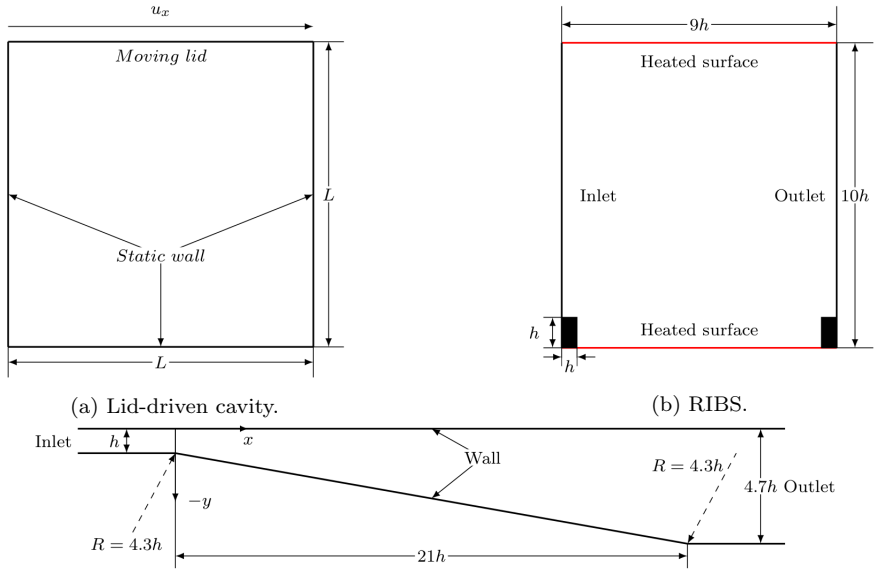
**Lid-driven cavity.** The lid-driven cavity flow configuration is shown in Fig. 14a. The top lid moves towards right direction and other walls are static. The no-slip conditions are applied on the walls. The Reynolds number is 5000. The $k-\omega$ $SST$ turbulence model was used to simulated the cavity flow on the finest mesh which contains $N_x \times N_y = 300 \times 300$ cells. The velocity field is then interpolated to coarser meshes of different refinement levels from 2500 to 40000 cells of the same aspect ratio $AR = 1$, and two meshes of different aspect ratios from 1 to 9 with the cell number around 2500. The aspect ratio is defined as the ratio of the cell's length on the x direction to its width on the y direction.

**RIBS.** The RIBS configuration is shown in Fig. 14b. Air flows from the left to the right, at the atmospheric pressure and temperature. The left and right boundaries are set to be periodic. The top and bottom walls are heated while the two ribs are not. The Reynolds number and Prandtl number are 30000 and 0.71, respectively. The RIBS case was simulated using three turbulence models with both structured and unstructured meshes.

**Diffuser.** The 2D flow inside a planar asymmetric diffuser, as shown in Fig. 14c, is simulated using $k-\omega$ $SST$ turbulence model with fully-developed turbulent inlet at $Re = 18000$ based on the bulk inlet velocity and the inlet channel height with two types of meshes: high-Reynolds (HR) mesh and low-Reynolds (LR) mesh.

Table 8: Simulations details of four cases.

| Case | $Re$ | Turbulence model | Mesh | | |
|---|---|---|---|---|---|
| | | | Type | AR | $No.\ cells_{(N_x \times N_y)}$ |
| BFS | 5100 | $R_{ij} - \epsilon\ SSG$ | Structured | 3.2 | 74400 |
| | | | Unstructured | - | 131118 |
| Cavity | 5000 | $k - \omega\ SST$ | Structured | 1 | $90000_{300\times300}$ |
| | | | Structured | 1 | $40000_{200\times200}$ |
| | | | Structured | 1 | $10000_{100\times100}$ |
| | | | Structured | 1 | $2500_{50\times50}$ |
| | | | Structured | 3 | $2494_{29\times86}$ |
| | | | Structured | 6 | $2500_{20\times125}$ |
| | | | Structured | 9 | $2499_{14\times147}$ |
| RIBS | 30000 | $k - \epsilon\ LP$<br>$k - \omega\ SST$<br>$R_{ij} - \epsilon\ SSG$ | Structured | 3 | 18296 |
| | | | Unstructured | - | 7735 |
| Diffuser | 18000 | $k - \omega\ SST$ | HR structured | $0.3 \sim 47.7$ | $21648_{328\times66}$ |
| | | | LR structured | $0.3 \sim 119.8$ | $31488_{328\times96}$ |




(a) Lid-driven cavity.

(b) RIBS.

(c) Diffuser.

Figure 14: Configurations of three cases.

**Mesh density influence.** As show in Fig. 15, the vortex regions identified by both GMM-Unet and SplineCNN-Unet in the coarsest mesh are the largest in the coarsest mesh. The identified regions become smaller as the mesh refinement level increases. The capability of recognizing a certain pattern of all the pure convolution-based machine learning models is limited to the size of effective receptive field (ERF). A trained model fails to correlate two points separated by a distance larger than the ERF which is determined by both the hyper-parameters of the model and the dataset. As indicated by one typical snapshot of streamline plot in Fig. 5, for the BFS structured mesh, $100 \times 40$ cells are distributed in the vortex region where normally four to five vortexes exist. Thus, a single vortex in the training dataset covers no more than 25 cells on one specific direction. As a result, these two models trained on this dataset can only detect the vortexes of comparable sizes in terms of how many mesh cells they span.

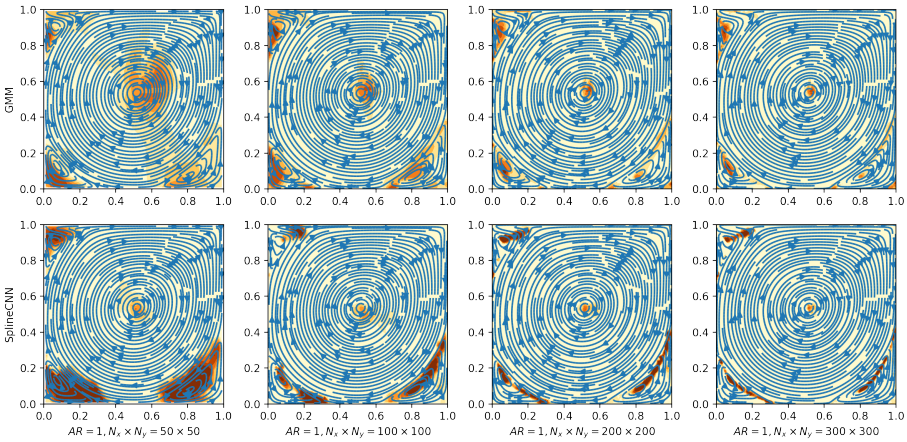



Figure 15: Identifications of vortexes of GMM-Unet and SplineCNN-Unet on lid-driven cavity meshes of different refinement levels.

**Mesh aspect ratio influence.** The aspect ratio of the BFS structured mesh included in the training dataset is $AR = 3.2$. As a result, the two models trained on this dataset well identified the vortexes on the mesh of $AR = 3$ as shown in Fig. 16. As the aspect ratio deviates from 3.2, their identification performances deteriorate which is more evident for SplineCNN-Unet. The SplineCNN-Unet model successfully identified three secondary vortexes in the corners on the mesh of $AR = 1$, all vortexes on the mesh of $AR = 3$, and the primary vortex center on the mesh of $AR = 6$, but failed on the mesh

of $AR = 9$. Compared to SplineCNN-Unet, the GMM-Unet model identified more or less the same vortex region on the four meshes and thus has a better generality to the variation of the mesh aspect ratio. However, the vortex regions identified by the GMM-Unet model do not cover the vortex center and have diffuse boundaries compared with the SplineCNN-Unet.

**Mesh type influence.** The mixture of both structured and unstructured meshes in the dataset poses no problem to the training. As shown in Fig. 17, while the SplineCNN-Unet model better identified the vortex center and shape on the structured mesh compared with the GMM-Unet model, but degraded more on the unstructured mesh where it only identified the lower half of the vortexes. The GMM-Unet model once again has better generality to the mesh type.

**Turbulence models influence.** To test the sensitivity of the proposed approach to the turbulence models, three commonly used models, $k-\epsilon$ linear production, $k-\omega$ $SST$ and $R_{ij}-\epsilon$ $SSG$, were selected because we intend to detect the vortexes generated by RANS turbulence models. As shown in Fig. 17, the turbulence models have no visible influence on the identification performance of the proposed approach. The robustness of our models to different turbulence models is explainable since they identify the vortexes based on the topological distribution of the velocity field which is universal among different turbulence models.

**Mesh size scaling influence.** As shown in Fig. 18, the cell size along the wall normal direction in the low-Reynolds diffuser mesh increases continuously, while in high-Reynolds diffuser mesh, the cell size from the first to the second layer perpendicular to the wall decreases abruptly. As shown in Fig. 19, both GMM-Unet and SplineCNN-Unet models can capture the vortexes on two meshes. On the high-Reynolds mesh, the identified vortex regions by both models are non-connected while those on the low-Reynolds mesh are closer to the real vortex topology. Therefore the proposed approach is quite robust to the mesh size scaling but sensitive to the abrupt scaling jump.

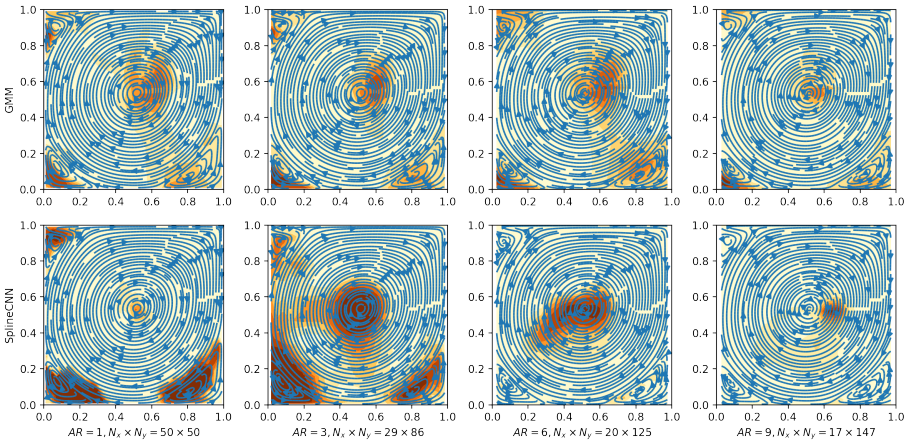
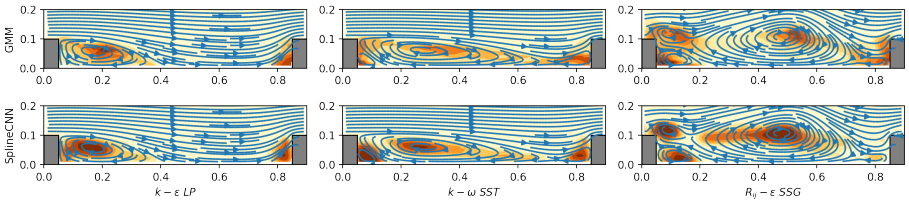
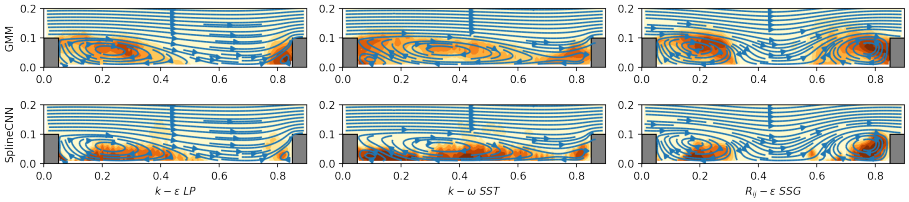



Figure 16: Identifications of vortexes of GMM-Unet and SplineCNN-Unet on lid-driven cavity meshes of different aspect ratios.

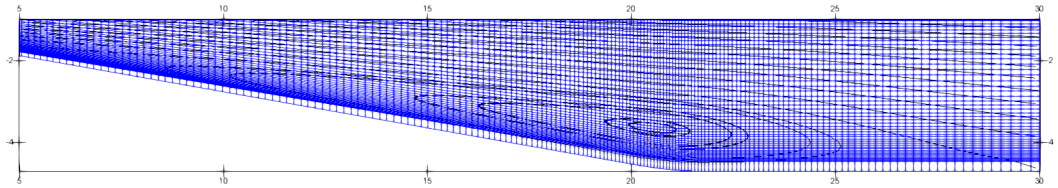



(a) Structured mesh.

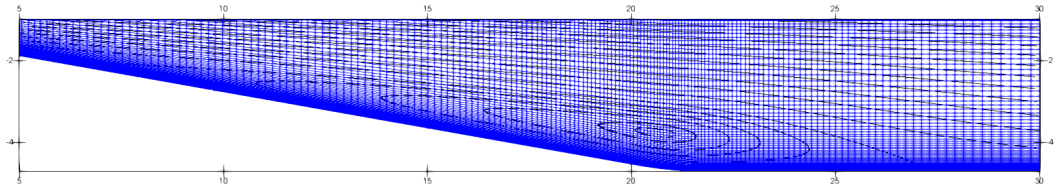



(b) Unstructured mesh

Figure 17: Identifications of vortexes of GMM-Unet and SplineCNN-Unet on RIBS structured and unstructured meshes.



(a) High-Reynolds mesh.



(b) Low-Reynolds mesh

Figure 18: Two diffuser meshes.

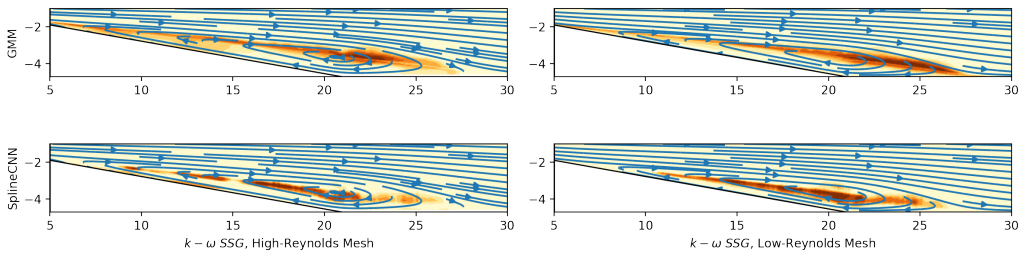



Figure 19: Identifications of vortexes of GMM-Unet and SplineCNN-Unet on two diffuser meshes.

## 6. Conclusion

In this paper, we proposed a machine learning approach for identifying vortexes in unstructured mesh-based CFD results using a Graph Neural Network with U-Net architecture, which includes constructing graphs from CFD meshes, the generation of a hierarchy of graphs using algebraic multigrid method and the labeling of the dataset with the biased random walking

algorithm. We demonstrated that the current machine learning approach combined with either Gaussian mixture model or SplineCNN as convolution kernel can achieve similar identification performance to that of traditional CNN while directly accepting embedded data on unstructured meshes. While the SplineCNN kernels achieves better vortex identification performance on cases similar to those included in the dataset, it degrades more significantly than GMM on unseen cases and meshes. Compared with other influence factors, such as different turbulence models and different Reynolds numbers, the mesh density has the biggest influence on the vortex identification of the GNN models. The mesh sensitivity analysis shows that the trained GNN models can only identify vortexes of the scale similar to those included in the dataset. It means that the region of the same vortex identified by the GNN model shrinks in denser meshes. This problem can be partially alleviated by including larger vortexes in the dataset but can not be fundamentally resolved. In the future, we will try to solve this problem by introducing other modules to the model architecture to exploit the invariant vortex features in much deeper graph hierarchies. The next step is to extend our approach to 3D cases and to other flow phenomena. Since we are working on graphs, this approach is extensible to 3D case without extra effort on the GNN model side but requires much efforts on generating a comprehensive 3D dataset.

**Acknowledgements**

This work has been supported by French National Association of Technical Research (CIFRE 2020/0791).

**Bibliography**

[1] Hunt J. C. R. Vorticity and vortex dynamics in complex turbulent flows. *Transactions of the Canadian Society for Mechanical*, 11(1):21–35, 1987.

[2] Chong M. S., Perry A. E., and Cantwell B. J. A general classification of three-dimensional flow fields. *Physics of Fluids A*, 2(5):765–777, 1990.

[3] Jeong J. and Hussain F. On the identification of a vortex. *Journal of Fluid Mechanics*, 285(1):69, 1995.

[4] Wu J. Z., A. K. Xiong, and Yang Y. T. Axial stretching and vortex definition. *Physics of Fluids*, 17(3):038108, 2005.

[5] George Haller, Alireza Hadjighasem, Mohammad Farazmand, and Florian Huhn. Defining coherent vortices objectively from the vorticity. *Journal of Fluid Mechanics*, 795:136–173, 2016.

[6] Lecun Y., Bottou L., Bengio Y., and P. Haffner. Gradient-based learning applied to document recognition. In *Proc. IEEE*, volume 86, page 2278–2324. IEEE, 1998.

[7] Shuran Ye, Zhen Zhang, Xudong Song, Yiwei Wang, Yaosong Chen, and Chenguang Huang. A flow feature detection method for modeling pressure distribution around a cylinder in non-uniform flows by using a convolutional neural network. *Scientific reports*, 10(1):1–10, 2020.

[8] Mathew Monfort, Timothy Luciani, Jonathan Komperda, Brian Ziebart, Farzad Mashayek, and G Elisabeta Marai. A deep learning approach to identifying shock locations in turbulent combustion tensor fields. In *Modeling, Analysis, and Visualization of Anisotropy*, pages 375–392. Springer, 2017.

[9] Liang Deng, Yueqing Wang, Yang Liu, Fang Wang, Sikun Li, and Jie Liu. A cnn-based vortex identification method. *Journal of Visualization*, 22(1):65–78, 2019.

[10] Xue Bai, Changbo Wang, and Chenhui Li. A streampath-based rcnn approach to ocean eddy detection. *IEEE Access*, 7:106336–106345, 2019.

[11] Marzieh Berenjkoub, Guoning Chen, and Tobias Günther. Vortex boundary identification using convolutional neural network. In *2020 IEEE Visualization Conference (VIS)*, pages 261–265. IEEE, 2020.

[12] Carlos Michelén Ströfer, Jinlong Wu, Heng Xiao, and Eric Paterson. Data-driven, physics-based feature extraction from fluid flow fields. *arXiv preprint arXiv:1802.00775*, 2018.

[13] Byungsoo Kim and Tobias Günther. Robust reference frame extraction from unsteady 2d vector fields with convolutional neural networks. In *Computer Graphics Forum*, volume 38, pages 285–295. Wiley Online Library, 2019.

[14] Jun Wang, Lei Guo, Yueqing Wang, Liang Deng, Fang Wang, and Tong Li. A vortex identification method based on extreme learning machine. *International Journal of Aerospace Engineering*, 2020, 2020.

[15] Franz K., Roscher R., Milioto A., Wenzel S., and Kusche J. Ocean eddy identification and tracking using neural networks. In *in IGARSS 2018 - 2018 IEEE International Geoscience and Remote Sensing Symposium*, volume 86, page 6887–6890. IEEE, 2018.

[16] Liang Deng, Wenchun Bao, Yueqing Wang, Zhigong Yang, Dan Zhao, Fang Wang, Chongke Bi, and Yang Guo. Vortex-u-net: An efficient and effective vortex detection approach based on u-net structure. *Applied Soft Computing*, 115:108229, 2022.

[17] Yueqing Wang, Liang Deng, Zhigong Yang, Dan Zhao, and Fang Wang. A rapid vortex identification method using fully convolutional segmentation network. *The Visual Computer*, 37(2):261–273, 2021.

[18] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[19] Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

[20] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[21] Xavier Bresson and Thomas Laurent. Residual gated graph convnets. *arXiv preprint arXiv:1711.07553*, 2017.

[22] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[23] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

[24] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015.

[25] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Shapenet: Convolutional neural networks on non-euclidean manifolds. Technical report, 2015.

[26] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. *Advances in neural information processing systems*, 29, 2016.

[27] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodola, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5115–5124, 2017.

[28] Matthias Fey, Jan Eric Lenssen, Frank Weichert, and Heinrich Müller. Splinecnn: Fast geometric deep learning with continuous b-spline kernels. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 869–877, 2018.

[29] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. *Insights into imaging*, 9(4):611–629, 2018.

[30] David I Shuman, Mohammad Javad Faraji, and Pierre Vandergheynst. A multiscale pyramid transform for graph signals. *IEEE Transactions on Signal Processing*, 64(8):2119–2134, 2015.

[31] Yvan Notay. An aggregation-based algebraic multigrid method. *Electronic transactions on numerical analysis*, 37:123–146, 2010.

[32] Michael Edwards and Xianghua Xie. Graph based convolutional neural network. *arXiv preprint arXiv:1609.08965*, 2016.

[33] Chen Cai, Dingkang Wang, and Yusu Wang. Graph coarsening with neural networks. *arXiv preprint arXiv:2102.01350*, 2021.

[34] saturne support@edf.fr. *code_saturne 7.1 Theory Guide*. https://www.code-saturne.org/documentation/7.1/theory.pdf, 2021.

[35] Hung Le, Parviz Moin, and John Kim. Direct numerical simulation of turbulent flow over a backward-facing step. *Journal of fluid mechanics*, 330:349–374, 1997.

[36] Frédéric Archambeau, Namane Méchitoua, and Marc Sakiz. Code saturne: A finite volume code for the computation of turbulent incompressible flows-industrial applications. *International Journal on Finite Volumes*, 1(1):http–www, 2004.

[37] Dieter Jungnickel and D Jungnickel. *Graphs, networks and algorithms*, volume 3. Springer, 2005.

[38] Tom Fawcett. An introduction to roc analysis. *Pattern recognition letters*, 27(8):861–874, 2006.

[39] Guido Rau, Moeller C¸ akan, D Moeller, and Tony Arts. The effect of periodic ribs on the local aerodynamic and heat transfer performance of a straight cooling channel. 1998.

[40] Tony Arts, Carlo Benocci, and Patrick Rambaud. Experimental and numerical investigation of flow and heat transfer in a ribbed square duct. In *3rd International symposium on integrating CFD and experiments in aerodynamics*, pages 20–21, 2007.

[41] Carl U Buice. *Experimental investigation of flow through an asymmetric plane diffuser*. Stanford University, 1997.

[42] Shinnosuke Obi, K Aoki, and S Masuda. Experimental and computational study of turbulent separating flow in an asymmetric plane diffuser. In *Ninth symposium on turbulent shear flows*, volume 305, pages 305–312, 1993.