Roznowicz *et al. Advanced Modeling and Simulation in Engineering Sciences(2024)11:6*
https://doi.org/10.1186/s40323-024-00259-1

Advanced Modeling and Simulation in Engineering Sciences

RESEARCH ARTICLE Open Access

Large-scale graph-machine-learning surrogate models for 3D-flowfield prediction in external aerodynamics

Davide Roznowicz[1,3], Giovanni Stabile[2*], Nicola Demo[1,4], Davide Fransos[3] and Gianluigi Rozza[1,4*]

*Correspondence:
giovanni.stabile@uniurb.it;
grozza@sissa.it
[1]Mathematics Area, mathLab, [2]SISSA, Via Bonomea, 265, Trieste 34136, Italy
[2]Department of Pure and Applied Sciences, Informatics and Mathematics Section, University of Urbino Carlo Bo, Piazza della Repubblica, 13, Urbino 61029, Italy, e-mail: giovanni.stabile@uniurb.it
[3]Sauber Motorsport AG, Wildbachstrasse, 9, Hinwil 8340, Switzerland
[4]FAST Computing Srl, Via Mazzini, 20, Trieste 34121, Italy

Abstract

The article presents the application of inductive graph machine learning surrogate models for accurate and efficient prediction of 3D flow for industrial geometries, explicitly focusing here on external aerodynamics for a motorsport case. The final aim is to build a surrogate model that can provide quick predictions, bypassing in this way the unfeasible computational burden of traditional computational fluid dynamics (CFD) simulations. We investigate in this contribution the usage of graph neural networks, given their ability to smoothly deal with unstructured data, which is the typical context for industrial simulations. We integrate an efficient subgraph-sampling approach with our model, specifically tailored for large dataset training. REV-GNN is the chosen graph machine learning model, that stands out for its capacity to extract deeper insights from neighboring graph regions. Additionally, its unique feature lies in its reversible architecture, which allows keeping the memory usage constant while increasing the number of network layers. We tested the methodology by applying it to a parametric Navier–Stokes problem, where the parameters control the surface shape of the industrial artifact at hand, here a motorbike.

Keywords: Computational fluid dynamics, Graph machine learning, External aerodynamics, Large scale model, 3D surrogate model

Introduction

The knowledge of the behavior and the effects of impacting flows around or within target objects is one of the main interests in Aerodynamics modeling, in order to aid the design process in optimizing performance. Computational fluid dynamics (CFD) simulations are the most established ways to gain proper insight into flowfield behavior. However, they tend to have both pros and cons. Their limitations are even more explicit for multi-query problems or when the available computational power is limited. Both these factors are present in several engineering fields, among which the motorsport one. In fact, in order to perform shape optimization, a large number of different geometries are in need of being tested, and the total number of computing hours is usually restricted by computational

availability and, in the case of sports competitions, also by regulations. The main goal of this contribution is indeed the investigation of innovative data-driven solutions in search of a surrogate model capable of tackling the weaknesses of large-scale simulations, such as:

- Non-negligible running time, especially given the need to run large simulations (several millions of cells, enormous memory requirements as a consequence) almost non-stop (often day and night).
- Limitations on the number of simulations that can be performed.

This setting is particularly relevant in the motorsport scenario but can be extended to a variety of different situations where a large number of configurations of the system under study must be performed or a limited computational cost is necessary. This situation occurs, for example, in shape optimization problems, uncertainty quantification, inverse problems, and real-time control. Our surrogate model should be able to perform inferences much faster than the consolidated simulation frameworks—finite volume, finite element—, at the expense of slightly less reliability in accuracy. Overall, a lot of effort is put into developing an efficient and scalable model, capable of handling specific environmental constraints together with a huge amount of data (beyond the size of the case studies displayed in this research paper).

A possible way to reduce the cost of CFD simulations is given by reduced order models (ROMs) [1]. The construction of a ROM is usually based on two different phases, a first one where a set of properly collected high-fidelity solutions are used to determine a low-dimensional representation of the solution manifold; and a second one, which consists of the determination of the solution, in the compressed solution manifold, for any new value of the input parameters. The first stage can be conducted using both linear and nonlinear compression strategies. From what concerns linear approaches, the most used approaches are the proper orthogonal decomposition (POD) [2,3] and the reduced basis method [4]. Nonlinear approaches are usually based on machine learning tools such as autoencoders [5], convolutional autoencoders [6,7], or similar architectures.

The methodology used to perform the second stage distinguishes intrusive ROMs from non-intrusive ROMs. Intrusive ROMs are based on the idea of projecting the set of equations on the compressed solution manifold. This type of model has the advantage of being intrinsically physics-based since the evolution of the latent coordinates into the compressed solution manifold is retrieved through the minimization of the residual associated with the governing equations. On the other side, knowledge of the discretized equations is necessary. Therefore, access to the source code of the solver used to compute the high-fidelity solutions is required (hence the intrusive epithet). The speedup that can be achieved by this type of method is usually limited by the efficiency of the full order solver and drastically decreases in case of high dimensional problems and high nonlinearities [8].

Non-intrusive methods directly use different regression or interpolation methods to approximate the map between the evolution of the latent coordinates in the compressed solution manifold and the input parameters. Possible approaches are based on multivariate interpolation methods like radial basis function techniques [9], or other regression strategies such as Gaussian progress regression [10] or machine learning methods [11,12]. In the present article, our attention is devoted to the development of a surrogate model that is purely data-driven and that could directly work on motorsport simulations (i.e.

3D high-dimensional full-order simulations on unstructured grids). In what follows, we briefly summarize the most promising approaches that fall into this category:

- Gaussian Processes (GP): they can be defined as a (potentially infinite) collection of random variables (RVs) such that the joint distribution of every finite subset of RVs is multivariate Gaussian. The main use case is related to the capability to compute the posterior distribution in a Bayesian setting and infer a function that is expected to approximate some variables of interest in a region of missing training data. Broadly speaking, this method tends to have an edge in the context of little amount of training data available. However, fitting too much data also makes the model extremely slow. Relying, instead, on a subset of the data would not let us exploit the full capabilities of our dataset, which is again against our main interest given our need for accuracy. In spite of that, some variants blending GP and deep learning have recently been studied in some papers and further research is underway [13–15].
- CNN-like architectures: when dealing with 3D-flowfield prediction, these methods tend to exploit the fact that the 3D shapes being used are quite easy to model. Therefore, they often perform some sort of function-mapping from the 3D space into a 2D one in order to be able to apply the 2D convolutions afterward [16]. Nevertheless, the primary challenge lies in the mapping process itself, which lacks clear guidelines for addressing complex 3D geometries.
- Encoder-Decoder-based architectures: generally speaking, these deep learning algorithms encode the input in a low-dimensional space, in order to learn the fundamental structure of the data. Afterward, they use this synthetic and dense description to reconstruct the output. Multiple forms of this architecture have been researched [17,18], also in the context of autoencoders [19], allowing a generative-like process. In the context of large fluid dynamics simulations, characterized by high Reynolds number, a lot of attention should be put into making sure that convergence does not fail and that the model does not overfit the training data either.
- Physics-informed neural networks (PINN): they consist of a deep learning model enforcing physics constraints in the loss function, often via a differential equation. Several variants of these methods often try to address these kinds of physics-related problems where the underlying behavioral laws are known but an analytical solution is not available as well as easy computations of an approximate one. In our case, PINN tries to enforce the behavior of the *Navier–Stokes* Equations [20,21]; nevertheless, the complexity of these equations makes it hard to work in this context, too. In fact, current attempts only deal with very simple applications, both in terms of geometry and equations.
- Graph Neural Networks (GNN): this recent and yet not-widely adopted deep learning methodology (mostly applied to 2D mesh data in aerodynamics-related literature [22,23]) allows to deal also with highly heterogeneous and unstructured 3D data by training directly on the graph made up of nodes and edges (and related features) without any further constraint about the space topology like in the CNNs. In this research article, our proposed methodology will entail the utilization of graph neural networks to construct a surrogate model for 3D flowfield prediction.

Over the course of academic research, several neural network models have been built on top of graphs, with the first powerful neighborhood aggregation model being the *graph convolutional networks (GCN)* [24]

In the context of fluid dynamics, impressive physics-driven models have lately been implemented [25]: they show how complex particle-based physical systems, with no more than a few thousand particles, can be modeled by blending the latest deep learning breakthroughs in graph machine learning with the physics underpinning the phenomenon of interest. DeepMind also experimented with mesh-based [26] underlying structures, presenting innovative and inspiring ideas for the field of aerodynamics.

However, additional restraints come into play when working within a specific research domain coupled with peculiar constraints, such as managing huge graphs.. In fact, real cases often involve an enormous amount of nodes and edges, an aspect which is hardly taken into account when dealing, for example, with 2D images. In the majority of cases, at least a whole 2D image can fit into GPU; for a big 3D graph, it is completely different as this is not guaranteed at all. Possible solutions to this impediment come from sampling subgraphs within the original graph and learning their local structure. Transductive learning would not make it reasonable to proceed in this way; instead, the inductive properties within the GraphSAGE framework [27] make it feasible. Thus, a fixed-size neighborhood is sampled before learning in order to properly control the memory footprint of the graph loaded into the GPU. More recent ways of sampling the nodes for a subgraph have been experimented in the GraphSAINT sampler [28]. Depending on the research area, the act of sampling might be a close approximation of reality or a very far one: a relatively small subgraph will limit the ability of the model to learn long-distance relationships among the nodes; instead, if we have large enough GPU memory, we can store a sufficiently big subgraph and use a suitable deep model. Simply adding many GCN layers does not help: in fact, similarly to the CNNs, we need a mix of ingredients to make the long-distance relationships worthwhile. For example, skip-connection mechanisms help the CNN not to disperse the information flow among layers; a similar message-forwarding procedure has been imagined for GNNs, too: [29] shows how this approach allows adding many layers while achieving SOTA performances. Nevertheless, memory tends to grow exponentially as the number of layers increases. That is why most of the authors of this just-cited research paper later joined forces again and tried to tackle the problem: in [30] they conceive and explain how revertible connections allow extremely deep architectures by keeping the memory usage constant while varying the number of layers. This is the main reason leading us to adopt this graph-based model, called REV-GNN, while also integrating useful approaches or methodologies presented in other research articles.

A deeper overview of recent cutting-edge and impactful journals deserves attention. Some have shaped the graph machine learning landscape by proving to be real breakthroughs in recent years. Others have focused on providing valuable insights and inspiring approaches for specific real-world domains or niche research areas.

Nonetheless, the vast majority of the models we studied share some limitations when we think about our specific application domain. First of all, the memory tends to explode when designing deep architectures. In addition, they are mostly focused on node classification, link prediction, or graph regression: thus, their priority is building 'few hopes' models that do not need many layers, unlike our case in which we focus on node regression. Lastly, most models are focused on learning some form of short-distance node relationship, which is

fundamentally different from our needs. Many articles exhibit this pattern because the motivation behind constructing Graph Neural Networks (GNNs) often stems from real-world graph structures prevalent across various industry domains. These structures align with problems that demand improved solutions, such as those found in social networks, protein structures, transport networks, and more.

Often, in these contexts, the vast majority of the important information is distributed within the most neighboring nodes; the information residing in farther areas tends to be much more superficial and noisy. This is not our case, as we claim that some long-distance effects might have a mutual strong impact.

Given the fact that the original GCN architecture poses a lot of limitations because of its inductive behavior and lack of capability to scale beyond small graphs, we study the *graph attention networks (GAT)* [31], which exploit the attention mechanism to tackle some of the weaknesses of the GCN. They show improved predictive and generalization capabilities but impose severe memory issues when the underlying dataset gets bigger and bigger. Moreover, GAT only handles homogeneous graphs.

Graph Transformers [32], on the other hand, can overcome this limitation by specializing in heterogeneous graphs. They also employ the attention mechanism, by integrating more deeply the idea of the Transformer architecture, famously hyped within the NLP community. Unfortunately, the benchmark datasets used for testing are small compared to our graph size, as the model is not necessarily meant to work well with huge graphs. In addition, as already said, our purpose is directed toward homogeneous graph modeling, not heterogeneous approaches.

Inductive behavior and high-quality unsupervised node embedding generation are put in place thanks to GraphSAGE, one of the milestone models within the GNN landscape. However, empirically, only a few layers of depth are admissible before memory and training time explode: thus, it cannot scale to very deep models like our REV-GNN model is meant to.

Similarly to GraphSAGE, PinSAGE [33] shares both strong and weak points while also extending some functionalities. It is characterized by huge engineering efforts by Pinterest to build on top of the original GraphSAGE algorithm, making the model more practical and flexible for web-scale recommendation tasks: this is in line with our need to scale to a huge number of nodes. Another limitation, which was also affecting GraphSAGE, is the fact that it struggles with long-range node dependencies.

Useful inspiration for the feature engineering side of the REV-GNN model comes from the Position-aware GNN (P-GNN) [34], which demonstrates enhanced skills in capturing positions/locations of nodes, after fixing a set of reference anchor nodes. However, as with previous models, the P-GNN is only benchmarked on small datasets: as a consequence, its priority is not to address the memory bottlenecks of standard GNNs.

Meaningful help to overcome such issues can be found with the Cluster-GCN [35] and GraphSAINT models. Both approaches aim at clustering/sampling by breaking down a large graph into several smaller subgraphs. This approach has the goal of making the training of big graphs easier; as a result, this method reduces the memory burden. GraphSAINT, in particular, is flexible thanks to its subgraph sampling approach which we decide to combine together with the current REV-GNN model, so that we do not have to store the whole graph in the GPU memory at the same time. Both methods can scale to more

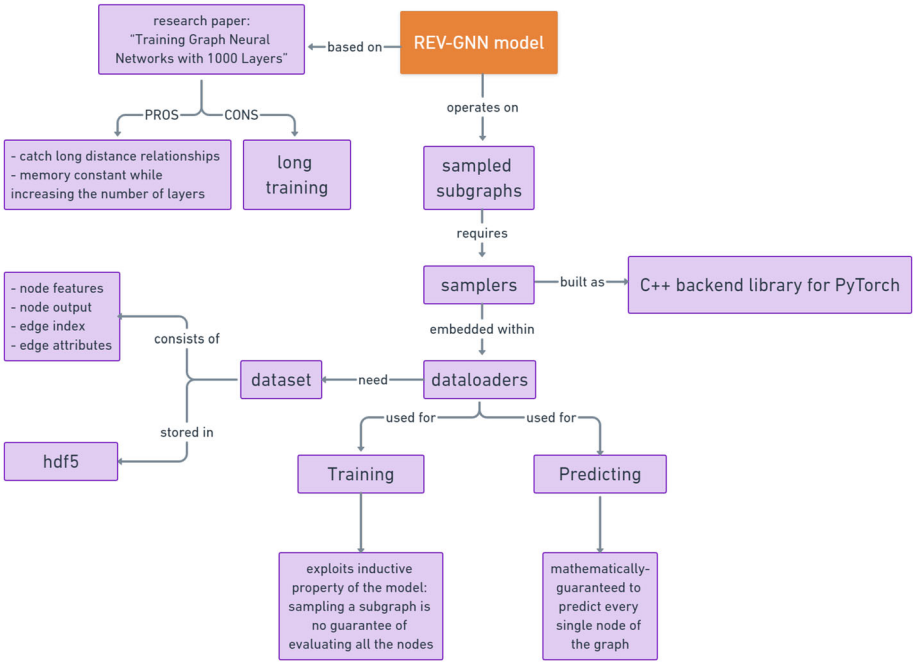


Fig. 1 Logical schema of the computational pipeline

than a few layers; though, they still lack the depth that REV-GNN is capable of. Therefore, they might not be able to capture very long-term relationships.

To address this difficulty, Graph ViT/MLP-Mixer [36] comes into play. It is a very recent model which captures long-range node and edge dependencies, while at the same time offering better speed and memory efficiency. Despite competitive results and focusing on apparently similar needs, the Graph ViT/MLP-Mixer has a very different scope in terms of model architecture and benchmark datasets. All the datasets used for testing are made of several graphs, but each of these graphs is extremely small (in the order of hundreds of nodes). Moreover, the main predictive tasks involve node classification and graph regression. None of the tasks is comparable to the node regression we need to deal with. At the same time, the assertions made in the associated article regarding memory pertain to the efficient training of thousands of small and independent graphs, as opposed to the context of large graphs.

Therefore, the goal of this research paper is to develop a suitable graph machine learning surrogate model in order to greatly improve the time to obtain a prediction compared to a standard CFD simulation (i.e. RANS).

The dataset consists of academic cases regarding a motorbike. Synthetic but relevant descriptions of the adopted pipeline framework (summarized in Fig. 1) are provided as well. Consequently, the structure of the research paper looks like this:

- Section "Creation of the training set": a brief description of the adopted parametric high-fidelity model, whose simulations will be assumed as ground truth and so used as the training set.
- Section"Graph Neural Network application": main information about dataset, pre-processing, and sampling methodology. Afterward, the model architecture is explained, highlighting the aspects of strengths and weaknesses.

- Section "Numerical results and discussion": an overview of the main hyperparameters, thorough explanation of the training and inference phase. The final results are displayed, comparing several models (with different hidden channels and layers) with the ground truth provided by the CFD simulation.
- Section "Conclusions and outlooks": a wrap-up of what has been done and accomplished throughout the presented article. Aspects to improve, future developments, and interesting trends are eventually assessed.

Creation of the training set

This section is dedicated to introducing the parametric model used to create the simulations dataset. We describe here the deformation law applied to the industrial artifact—here a motorbike—, the equations of the model and the method to compute its numerical solution. We highlight here that the proposed approach is agnostic to physics: in principle, any model can be used to create the solutions database to train the GNN.

Geometry deformation

In order to have a parametric domain, we use Free-Form Deformation applied on a box on the rear part of the motorbike domain [37]. Such a technique allows to manipulate the geometry by embedding it into a lattice of points, then moving some of these points to perturbate the object. In particular, the lattice is defined in $[1.6, 1.8] \times [-0.2, 0.2] \times [0.85, 1.05]$ using just two points per FFD dimension (see Fig. 2).

The deformed configurations are obtained by moving the rear vertices of the box of the quantities $\mu_1, \mu_2, \mu_3, \mu_4$, where μ_1 and μ_2 are the displacements of the upper and bottom part along the x coordinates, while μ_3 and μ_4 are the displacements of the upper and bottom portion along the z coordinates. 90 parameters $\boldsymbol{\mu} = [\mu_1, \mu_2, \mu_3, \mu_4]$ have been selected using a uniform distribution in the range $[-0.5, 0.5]^4$, leading to the creation of 90 deformed geometries.

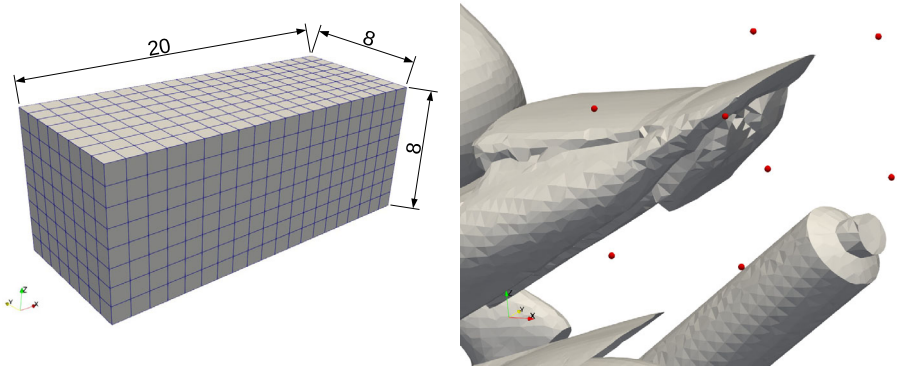


Fig. 2 On the left: base mesh used to embed the motorbike geometry. On the right: the lattice of points used to perform free-form deformation of the rear part of the motorbike indicated by the red dots

Full order model

Once the original geometry has been perturbed in order to create several deformed shapes, full-order simulations are carried out in order to numerically approximate the quantities of interest. The CFD simulation is performed using the finite volume method on an unstructured grid composed by $\approx$ 350,000 cells. It is important to note that, since the deformed motorbikes, the resulting discrete grids may have different topology. We describe the flow behavior using the Navier–Stokes equations:

$$\begin{cases} \frac{\partial \mathbf{u}}{\partial t} = -\nabla \cdot (\mathbf{u} \otimes \mathbf{u}) + \nabla \cdot \nu \left(\nabla \mathbf{u} + (\nabla \mathbf{u})^T \right) - \nabla p, \\ \nabla \cdot \mathbf{u} = \mathbf{0}. \end{cases} \tag{1}$$

To compute the solutions, the OpenFOAM open-source library is used [38]. This is a steady-state solver that employs the SIMPLE algorithm to handle pressure–velocity coupling. The turbulence model is k$-\omega$ SST model. Since the computational mesh changes depending on the deformation parameters, the number of cells is not fixed and varies among the different geometries. The computational mesh in the undeformed configuration counts 353,996 cells made of polyhedra with different numbers of faces and is obtained using the snappyHexMesh utility by embedding the stl file of the motorbike geometry inside a structured mesh. The initial mesh is reported on the left side of Fig. 2 and counts 20 elements along the x-direction and 8 elements along the y and z-directions. The gradient term is discretized using cellLimited Gauss linear scheme, convective terms are discretized using a bounded second-order unwinding scheme for what concerns the velocity field and a first-order bounded scheme for what concerns the κ and ω fields. Laplacian terms are discretized using a Gauss linear scheme. For more details on the numerical setting, the interested reader might check [39].

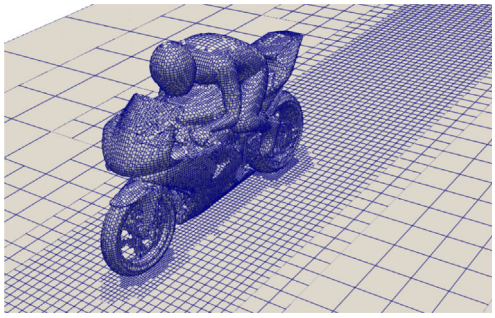
From simulations to graphs

The results of the simulations need a pre-processing step in order to feed the GNN model. This paragraph is dedicated to describing how the training graphs are constructed starting from the simulation output.

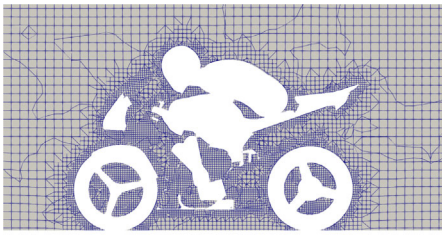
In our work scenario, the conversion from the original mesh-based dataset into a graph relies on the following action: each cell center is converted into a node of the graph; each node is then connected to the neighbouring nodes via bi-directional edges (roughly 3–4 edges per node in our datasets) Fig. 3. For any node and edge, additional information needs to be saved in order to learn and extrapolate the actual physics from the data. Such information should comprise the distances from a specific point of the domain or a boundary, (some of) the output values computed during the simulations, or again—for the edges—the distance between the two nodes linked by the edge itself. The nodes and edges features used in this contribution are listed in Table 1. Each node has associated 13 features, which are divided into 6 distance measures and 7 output values, while each edge contains only 1 feature.

Graph Neural Network application

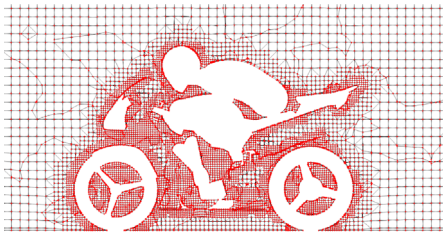
In this section, we focus on the details of the GNN application, firstly providing a sampling strategy to work on large, real-world test cases, and then describing the employed architecture.



(a) 3D mesh



(b) 2D slice of the 3D mesh



(c) 2D slice of the 3D graph

Fig. 3 An example of the motorbike computational grid (top), a related 2D slice (bottom left), and a 2D slice of the 3D graph (bottom right). The graph is 3D and is computed from the whole 3D mesh: here we show a 2D slice just to improve visual clarity

Table 1 Features of the graph and predicted variables

	Node features		**Edge features**
	Distances	**Output values**	
Total tensor dimension	$\|Nodes\| \times 6$	$\|Nodes\| \times 7$	$\|Edges\| \times 1$
Description	• $\|Nodes\| \times 3 \rightarrow$ node distance from the origin (0,0,0) • $\|Nodes\| \times 3 \rightarrow$ wall distance from the closest wall face center	• $\|Nodes\| \times 1 \rightarrow p$ • $\|Nodes\| \times 3 \rightarrow U$ • $\|Nodes\| \times 1 \rightarrow k$ • $\|Nodes\| \times 1 \rightarrow \nu_t$ • $\|Nodes\| \times 1 \rightarrow \omega$	• $\|Edges\| \times 1 \rightarrow$ edge length

Sampling subgraphs

The huge amount of data at our disposal makes it necessary to train only part of the dataset at each iteration. This goal is achieved by sampling techniques which tend to approximate the full-batch training by extracting subsets, i.e. subgraphs, of the original graph. The major issues encountered in doing so are the following:

- the suitability of a graph-based sampling technique depends on the characteristics of the dataset and the model as well
- few already-implemented backend routines are available in PyTorch-Geometric

Tackling these issues is not easy because of the remarkable amount of time needed to research and fully evaluate the overall context.
These are some relevant considerations:

- our graph dataset is fully connected, featuring 3–4 edges per node
- correctly predicting the value of a cell (node) depends on extracting enough insight from the surrounding neighborhood. However, in the aerodynamics context, the information is probably not highly concentrated around the node itself; instead, it is spread around as in a heavy-tail normal distribution
- since the dataset is very big and very diversified, few sampled subgraphs are not enough to learn about the motorbike geometry: a proper exploration of the dataset must be put in place.

Thus, it looks like we should build a technique to broadly investigate the surroundings of a node. We believe that a **Breadth First Search (BFS)** algorithm is a good starting point to address the above-mentioned issues. The idea is that, given a random starting node, a full BFS is performed until a pre-specified number of nodes is reached. To achieve that, we must implement our own C++ backend sampler and then call it from the Python interface in PyTorch. One situation that needs to be addressed occurs when two independent BFS explorations overlap, in the sense that a subset of their explored nodes coincides. In this case, we decide to consider the subgraphs independently Fig. 4, i.e. we build a separate subgraph for each BFS by renaming the nodes as they were different from any other exploration.

Model architecture

The main purpose of the model is to extract insights out of wide neighboring areas within the graph: catching long-range relationships is remarkably relevant as minor changes in the geometry might trigger changes of the flowfield not only in the surrounding environment but also farther away. We would like to be able to predict, as best as possible, these causal effects. Moreover, we would like our model to be sufficiently light in terms of memory usage in order to handle big (sub)graphs in GPU.

General GNN model

Broadly speaking, working with *Graph Neural Network (GNN)* models implies dealing with graphs as underlying data structures and consequently taking advantage of them as heterogeneous sources of insight. The following notations are adopted throughout the article:

A graph $\mathcal{G}$ is represented by a tuple $\langle V, \mathcal{E} \rangle$, where $V = \{v_1, v_2, ..., v_N\}$ is an unordered set of nodes (vertices) and $\mathcal{E} \subseteq V \times V$ is a set of edges. Let N and M denote the number of nodes and edges, respectively. For convenience, a graph can be equivalently defined as an adjacency matrix $A \in \mathcal{A} \subset R^{N \times N}$ where a_{ij} denotes the link relation between node v_i and v_j. In our scenarios, nodes, and edges are associated with a node feature matrix $X \in \mathcal{X} \subset R^{N \times D}$ and an edge feature matrix $U \in \mathcal{U} \subset R^{M \times F}$, respectively. We use GNN operators that map the node feature matrix $\mathcal{X}$, the adjacency matrix A, and the edge feature matrix U to a transformed node feature matrix X':

$$f_w : \mathcal{X} \times \mathcal{A} \times \mathcal{U} \mapsto \mathcal{X}', \tag{2}$$

where $f_w(X, A, U)$ is parameterized by learnable parameters w.

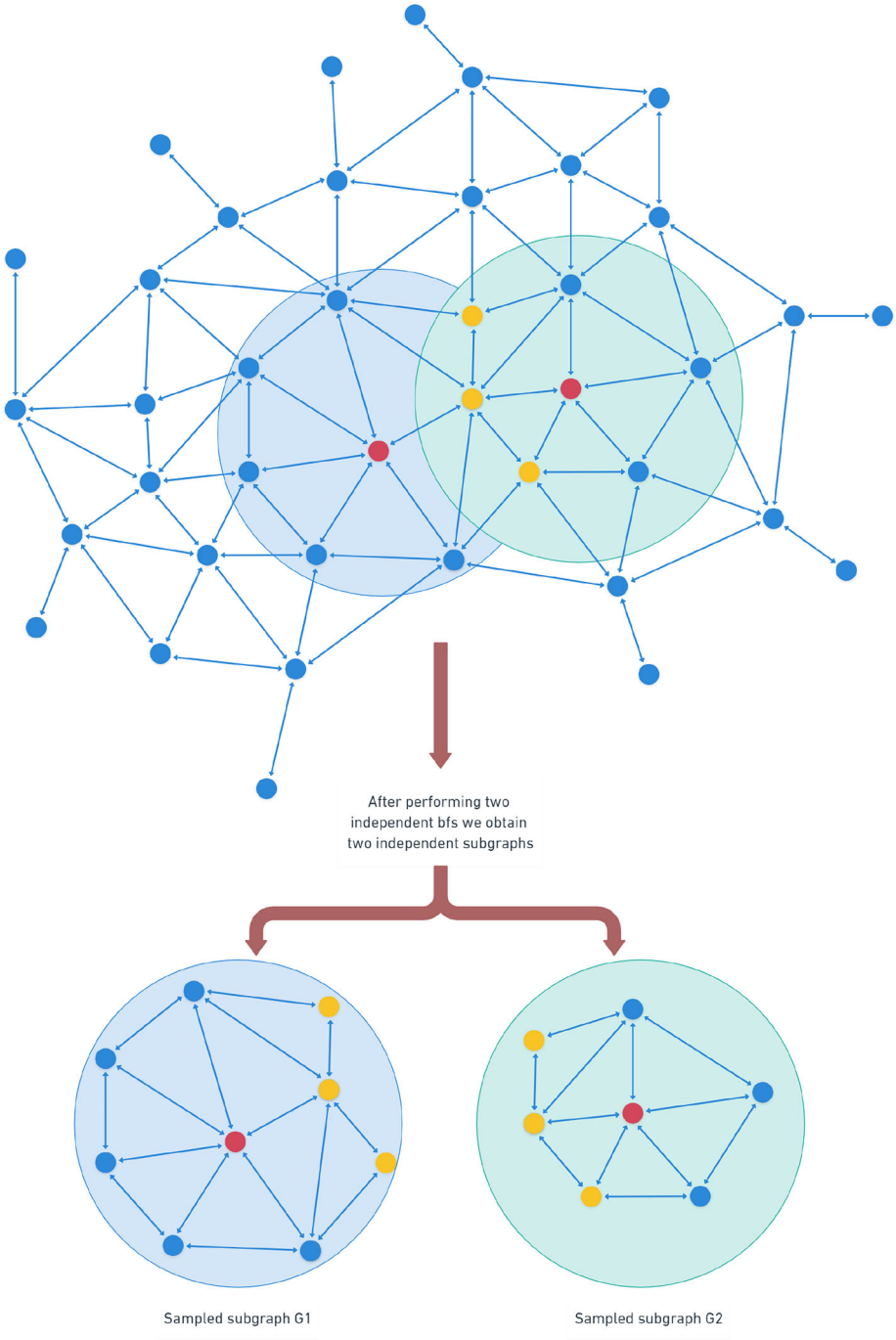


Fig. 4 Independent BFS explorations give birth to independent subgraphs, even if they overlap

REV-GNN model

Researching and designing a model that would fulfill our own requirements is not an easy task. In fact, most of the time, the research papers focus on maximizing certain accuracy metrics at the cost of training time and memory usage. Using resource-heavy models is not an issue in their cases as they do not hit environment constraints, i.e. they are far from running into out-of-memory (OOM) issues because of the tiny dataset they train on.

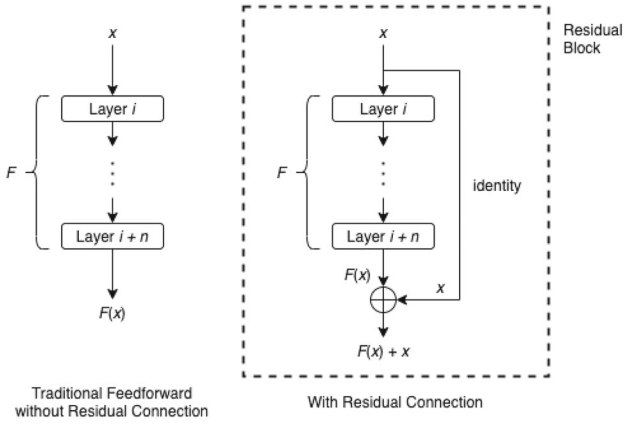


Fig. 5 Residual block featuring a skip connection

In our context, the choice of the model cannot be independent of the above-mentioned limitations. This implies, for example, that some specific message-passing algorithms are out-of-reach for us: therefore we must rely on graph-convolution techniques which are suitable for extracting relevant information while not requiring too heavy message propagations among nodes/edges.

We decided to adopt the implementation of the paper [30]. When coupled with the custom pipeline to handle our specific I/O, the model proposed by this paper addresses most of the challenges that we intend to face. Among several model categories experimented in the paper, our attention is drawn to the category that looks more aligned with our desire to explore the graph relationships in depth. Thus, we focus on a model called **REV-GNN**, which stands for reversible-connection graph neural networks [30]. The strong points of this model are here summarized:

1. presence of residual connections: it allows to extract deeper insights by propagating the information more effectively across the layers of the network;
2. reversibility of the architecture: since the amount of memory usage is not dependent on the number of layers anymore, we can again extract more information from wider neighborhood areas;
3. grouped connections: they help to reduce the parameter complexity.

The first mentioned point 1. is related to the concept of *skip connections* as shown in Fig. 5: besides feeding the information from layer i to layer $i+1$, the same (unchanged) information from layer i is also forwarded to layer $i+k$, where $k > 1$, bypassing a whole block of layers. This trick, which can be written as $X' = f_w(X, A, U) + X$, allows to train much deeper networks, by reducing the influence of issues that often arise such as exploding and vanishing gradients. Moreover, the whole training tends to reach convergence more easily. Fig. 5 displays the residual block, which can be successfully implemented in the context of graph convolutions by taking inspiration from the *ResNets* in the field of CNNs.

However, the memory complexity, which is $\mathcal{O}(LND)$, is still linear with respect to the number L of GNN layers. Since the memory footprint of the network parameters is usually negligible, we focus on memory consumption induced by the activations. The reversibility 2. of the architecture addresses this issue by making it feasible to add many layers without

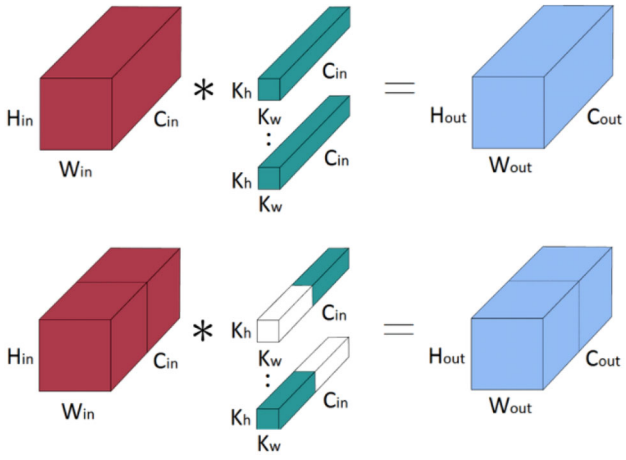


Fig. 6 The effect of grouped VS non-grouped connections across the channel dimension (image taken from [40])

falling into the risk of running into OOM issues. Grouped reversible GNNs only need to save the output node features of the last GNN block in GPU memory for backpropagation. This is because the full architecture is built in a standardized way which allows easy reconstruction. If we consider C groups across the channel dimension like in Fig. 6, we obtain $< X_1, X_2, ..., X_C >$ representing a partition of the feature matrix X, where $X_i \in \mathbb{R}^{N \times \frac{D}{C}}$. A fixed schema is used for the forward pass:

$$\textbf{FORWARD PASS}: \quad \begin{aligned} X'_0 &= \sum_{i=2}^{C} X_i \\ X'_i &= f_{w_i}(X'_{i-1}, A, U) + X_i \quad i \in \{1, 2, ..., C\} \end{aligned} \tag{3}$$

where we must remember that a REV-GNN block maps the inputs into the outputs as shown below:

$$< X_1, X_2, ..., X_C > \quad \mapsto \quad < X'_1, X'_2, ..., X'_C > . \tag{4}$$

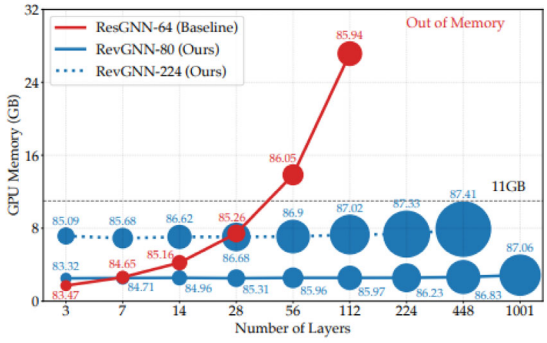
Similarly, the fixed schema for the backward pass is:

$$\textbf{BACKWARD PASS}: \quad \begin{aligned} X_i &= X'_i - f_{w_i}(X'_{i-1}, A, U) \quad i \in \{2, ..., C\} \\ X'_0 &= \sum_{i=2}^{C} X_i \\ X_1 &= X'_1 - f_{w_1}(X'_0, A, U) \end{aligned} \tag{5}$$

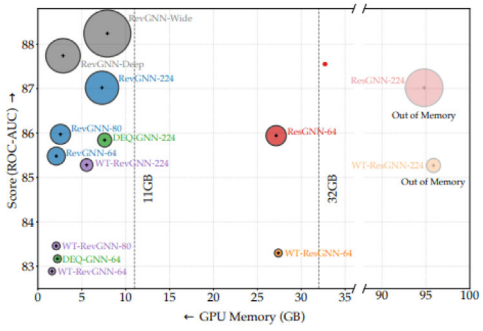
In practice, the REV-GNN architecture interchanges the following two steps:

$$\textbf{REV-GNN Architecture}: \quad \begin{aligned} &1.) \quad \hat{X}_i = Dropout(ReLU(Norm(X'_{i-1}))) \\ &2.) \quad \tilde{X}_i = GraphConv(\hat{X}_i, A, U) \end{aligned} \tag{6}$$

Specifically, it adopts a modified version of the dropout layer in which the dropout pattern is shared across layers in order to keep the memory consumption $\mathcal{O}(ND)$. In [30], the authors show the remarkable differences arising when comparing this architecture, whose



(a) Explosion of memory consumption when increasing the number of layers without reversibility.



(b) Reversible architectures on the left side; previous residual (but not reversible) architectures on the right side.

Fig. 7 Memory footprint: reversible VS non-reversible (images taken from [30])

memory footprint is independent of the number of layers L, with previous architectures which lack reversibility and grouping but are still endowed with residual connections. Figure 7 highlights this major divergent behavior among architectures.

Nevertheless, the largest drawback related to this architecture is due to the training time. The mentioned tweaks introduced to address memory consumption do not interfere with the time it takes to train the model in the context of multiple layers as well as channel dimensions. Therefore, the training itself tends to be quite slow. Though the overall analysis of the REV-GNN fits our goals, we will use it anyway. We will experiment with multiple settings in the choice of hyperparameters in order to prevent the training time from turning into an excessive burden for the overall process.

Training phase

The characteristic steps of the training procedure are described in this section, and summarized in the Algorithm 1. Related training parameters are elucidated in Table 2.

After loading some geometries (in the form of graphs) from the disk into CPU memory, our algorithm samples subgraphs and concatenates them into a single graph (this resulting graph is potentially much smaller than the original one, according to the user needs) and then trains the model on that. A training performed on subgraphs sampled from a big graph tends to converge to a training performed directly on the same big graph (i.e.

Algorithm 1 Simplified Pipeline for Training (Pseudocode)

```
epoch ← 0
max_epochs ← 900
max_steps ← 90
num_start_nodes ← 1
up_to_num_nodes ← 353996
while epoch < max_epochs do
    step ← 0
    while step < max_steps do
        if step is a multiple of a user-chosen number then
            flush the previous graph geometries contained in the CPU;
            load new geometries into the CPU based on available memory;
            initialize a new dataloader with the current loaded geometries;
        end if
        sample num_start_nodes source nodes;
        run a BFS from each source node, totaling up_to_num_nodes each;
        concatenate the edge lists related to each BFS-visited subgraph;
        concatenate the edge and node features as well;
        move the concatenated graph to GPU
        train the REV-GNN model on this concatenated graph;
        step++
    end while
    epoch++
end while
```

Table 2 Table of training parameters

Hyperparameters	Value	Description
max_epochs	900	Total number of epochs for the training phase
num_gpus	1	Number of gpus used
up_to_num_nodes	353996	Total number of nodes contained in the subgraph we sample (in this downsized situation, we decide to sample a subgraph which is almost as big as the full graph)
num_start_nodes	1	Number of randomly sampled source nodes for BFS explorations (subgraph sampling): if number == n, it means that n independent BFS will be performed and the resulting subgraphs aggregated in a sinle edge list
max_steps	90	Total number of training iterations per epoch
accum_steps	5	Number of gradient accumulation steps

full-batch mode with no sampling) only after a sufficiently high number of iterations [41]. Therefore, a subgraph-based REV-GNN model typically requires a lot of epochs to achieve good results. Since most real-world situations involve very demanding graphs, whose sizes often exceed the GPU size, the user would lean towards the subgraph-based training approach regardless of the increased required time. Our procedure has been built in a scalable way, in order to handle the whole training on large graphs by moving only the generated (and concatenated) subgraphs to GPU memory for training. In this context, more iterations will be required for convergence, but eventually, a similar level of accuracy can be achieved.

Given the fact that one motorbike geometry can fit into the only GPU at our disposal (*num_gpus*=1), for the purpose of this article we prefer to speed up the process

by training one full geometry per iteration. This translates into a single BFS (defined by *num_start_nodes=1*) which is performed until *up_to_num_nodes*=353,996 nodes are reached, which corresponds to the full size of each of our geometries. We specify that the approach is developed such that the workload can be distributed among several GPUs, making it possible to scale the algorithm to large, industrial problems: after the completion of the training step, the gradients are gathered (summed) in order to perform a step of the optimizer. We prefer to accumulate the gradient for *accum_steps* steps before performing an optimizer step, to better help convergence.

Concerning the computation of the loss function, we adopt an *l1* loss, which looks more stable (compared to a *l2* loss) in the prediction across the whole 3D case. To calculate it, we first compute the absolute value of the difference between the node prediction and the target values; then, we sum each component (rescaled by a normalization constant) in order to have an aggregate number for every single node in the graph. At this point, we average all these node values and obtain a measure of our loss. The above-mentioned normalization constant refers to previously computed statistics (gathered across all the training geometries) consisting of some numbers (one per variable to predict, i.e. 7 in our context) which, once multiplied by its own variable, yields the normalized version of the variable itself. A possible alternative to this might be applying a multi-task learning approach (aggregating loss computations coming separately from each variable), which might be explored in future works.

Inference phase

Algorithm 2 Simplified Pipeline for Inference (Pseudocode)

set_left_nodes ← *initialize_set* (*idx* for *idx* from 1 to *num_nodes*)
final_pred ← *init_zero_matrix* (size = *num_nodes* × *num_tasks*)
step ← 0
while *set_left_nodes* is not empty **do**
    sample a source node among the ones within *set_left_nodes*;
    perform a BFS until reaching a total of *up_to_num_nodes* nodes;
    infer on the BFS-visited graph via the trained REV-GNN model;
    sum the predictions to *final_pred*;
    for each node, update a counter if the node was part of the graph;
    remove the BFS-visited nodes from *set_left_nodes*;
end while
divide each row of *final_pred* by the times the node has been predicted;

After the model is fully trained, we might want to perform an inference on a new geometry. The Algorithm 2 describes how the inference works, while the Table 3 gives insights into the architecture settings, valid for both inference and training.

The main difference between training and inference is that here is that in the latter we want to have a procedure that is mathematically guaranteed to evaluate all the nodes in the graph geometry. In fact, while in the training phase, this is not that relevant, it must be ensured in the prediction. The key is creating a set, called *set_left_nodes*, containing all the indexes of the nodes that we still have not visited: as soon as we explore the graph via BFS, we can store the predictions and remove the visited nodes from the set, until all

Table 3 Table of model architecture parameters for training and inference

Hyperparameters	Value	Description
conv_encode_edge	True	Whether or not the edge information should be encoded in the convolution
Block	Res+	Graph backbone block to be chosen among the following types: {res+, res, dense, plain}
gcn_aggr	max	The aggregator of GENConv to be chosen among the following types: {mean, max, add, softmax, softmax_sg, power}
Norm	Batch	the type of normalization layer to be chosen among the following types: {batch, layer}
Backbone	rev	gcn backbone to be chosen among the following types: {deepergcn, weighttied, deq, rev}

of the nodes have been visited (and, thus, a prediction has been performed for each of them). Since some of the nodes might be visited more than once, we keep summing the just performed predictions to the previous ones, so that, in the end, we can divide each node prediction by the number of times a node has been predicted (average of predictions per node). To better clarify, at each inference step, the new predictions update (i.e. are summed to) the array containing the sum of the previous predictions, which is represented by *final_pred* in 2; after the last inference step, the *final_pred* vector contains the sum of all the predictions performed on each node. At this point, we just take the average prediction according to the number of times an inference has been performed on a specific node. This is also a way to make convergence smoother, as the inference on a node coming from different sampled subgraphs might be slightly different, due to this non-deterministic sampling procedure.

Numerical results and discussion

After training, we test the model on 10 different geometries not seen during training. We investigate the forecasted flowfield variables, in particular velocity and pressure. In fact, our analysis focuses on the Total pressure Coefficient (CpT), which is a function of velocity and pressure defined by the following general equation:

$$CpT = \frac{p - p_\infty + \frac{1}{2}\rho U^2}{\frac{1}{2}\rho_\infty U_\infty^2}, \tag{7}$$

where p is the fluid static pressure, ρ is the fluid density and U is the 3-dimensional velocity vector. Moreover, p_∞ is the freestream static pressure, ρ_∞ is the freestream fluid density and U_∞ is the freestream flow velocity (in our case inlet velocity). Since we simulate an incompressible fluid, then our 7 can be simplified. In particular, since ρ is constant, instead of p we use $\frac{p}{\rho}$. Given the incompressible fluid assumption, we can conventionally fix $p = 0$ (outlet pressure value). As a consequence, Eq. 7 becomes:

$$CpT = \frac{p + \frac{1}{2}U^2}{\frac{1}{2}U_\infty^2}, \tag{8}$$

Table 4 For each model, we display a metric that is based on the L^1-norm of the CpT residuals, accounting for all the test geometries

	8 hidden channels	16 hidden channels	32 hidden channels
8 layers	0.124881 ± 0.001269	0.088454 ± 0.001328	0.055246 ± 0.001795
16 layers	0.112793 ± 0.001493	0.071492 ± 0.001587	0.048296 ± 0.002411
32 layers	0.105534 ± 0.001447	0.062148 ± 0.001642	0.041300 ± 0.003315

The metric appears in the form $\mu \pm \sigma$. In particular, $\mu = \frac{1}{10}\sum_{j=1}^{10}\frac{1}{n_j}\|\varepsilon\|_{1j}$ and $\sigma = \sqrt{\frac{1}{10}\sum_{j=1}^{10}\left(\frac{1}{n_j}\|\varepsilon\|_{1j} - \mu\right)^2}$, where $\|\varepsilon\|_{1j} = \sum_{i=1}^{n_j}|\varepsilon_{ij}|$ is the L^1-norm of the CpT residuals for the j-th geometry. ε_{ij} represents the difference between the CpT of the model and the CpT of the CFD simulation, respectively, for the i-th node of the j-th geometry of the test set; $i = 1, ..., n_j$ where n_j represents the total number of nodes in the j-th geometry, while $j = 1, ..., 10$

Table 5 For each model, we display a metric that is based on the L^2-norm of the CpT residuals, accounting for all the test geometries

	8 hidden channels	16 hidden channels	32 hidden channels
8 layers	0.045317 ± 0.001580	0.025068 ± 0.001421	0.010147 ± 0.001498
16 layers	0.037635 ± 0.001770	0.016534 ± 0.001557	0.008142 ± 0.001728
32 layers	0.033002 ± 0.001690	0.012337 ± 0.001524	0.006335 ± 0.001969

The metric appears in the form $\mu \pm \sigma$. In particular, $\mu = \frac{1}{10}\sum_{j=1}^{10}\frac{1}{n_j}\|\varepsilon\|_{2j}^2$ and $\sigma = \sqrt{\frac{1}{10}\sum_{j=1}^{10}\left(\frac{1}{n_j}\|\varepsilon\|_{2j}^2 - \mu\right)^2}$, where $\|\varepsilon\|_{2j} = \sqrt{\sum_{i=1}^{n_j}|\varepsilon_{ij}|^2}$ is the L^2-norm of the CpT residuals for the j-th geometry. ε_{ij} represents the difference between the CpT of the model and the CpT of the CFD simulation, respectively, for the i-th node of the j-th geometry of the test set; $i = 1, ..., n_j$ where n_j represents the total number of nodes in the j-th geometry, while $j = 1, ..., 10$

Test error

For the testing phase, every single model variation (determined by hidden channels and layers) goes through the same evaluation procedure in order to obtain a pair of values (i.e. CpT mean and standard deviation) which stand for the final metric for that specific model.

The model performs an inference on one geometry at a time and, for each of the 10 test geometries, the CpT residuals are computed for each node and averaged across all nodes within the geometry. Afterward, the mean and standard deviation of the above 10 values are calculated: thus, we end up with $mean \pm std$. The results for this metric are gathered in Table 4, while in Table 5 we display an estimate, across the test geometries, of the MSE (mean square error).

We can notice that the residuals get smaller if we increase the number of layers. However, this is not as big as the increasing accuracy by varying the number of hidden channels, which significantly boosts the performance.

However, it is worth drawing attention to the 32 hidden channels column: it looks like the variance of the model increases a bit more rapidly, especially for the {16, 32} layer models. This is to be expected, as the model seems to be starting to experience some overfitting phenomenon. This is a signal that, in our context, a model with a number of layers or hidden channels greater than 32 is probably to be avoided given the likelihood of lower generalization capabilities.

A collage of slices (at $x = 0.5$) of one of the test geometries 8 further highlights what Table 4, 5 have already emphasized: the depicted CpT errors show a generally stronger performance for the models characterized by higher expressivity (i.e. more hidden channels) and a more marginal benefit by varying network layers.

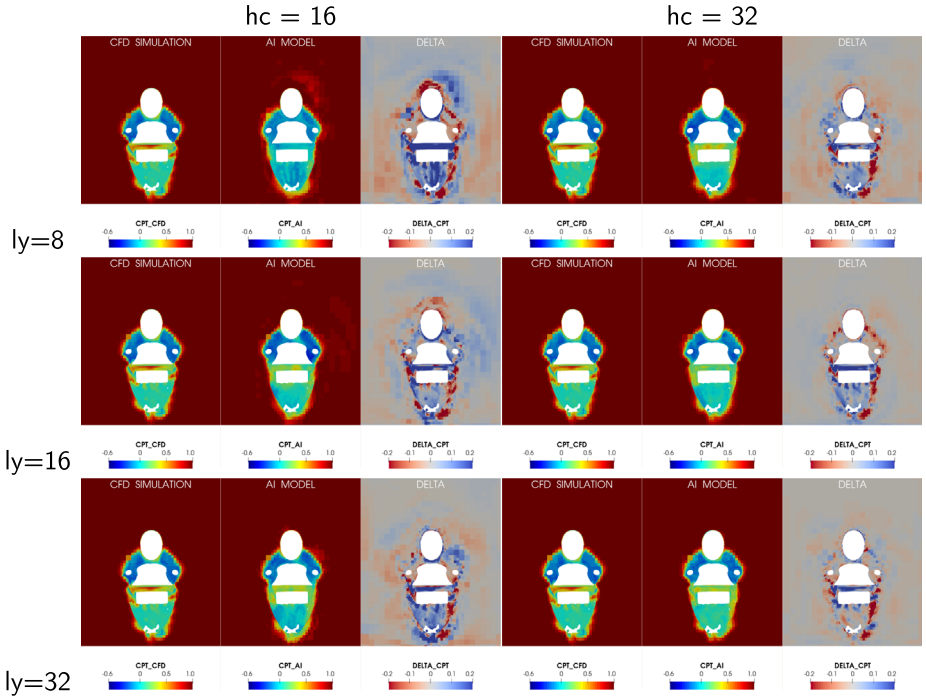


Fig. 8 Image collage of slices with $x = 0.5$, depicting the CpT. Comparison of all models defined by {8, 16, 32} hidden channels (hc) and {16, 32} layers (ly). Each model prediction (central) is juxtaposed with the ground truth on the left (assumed to be the RANS simulation) and the delta on the right

Table 6 Training time (in hours) on an Nvidia Quadro P5000 16 Gb, at varying network specifications

	8 hidden channels	**16 hidden channels**	**32 hidden channels**
8 layers	51.6	74.1	126.3
16 layers	91.0	132.3	233.7
32 layers	165.8	256.5	465.3

While Fig. 8 draws our attention to analyzing the sensitivity in behavior and performance for each variant of the model with regard to layers and hidden channels, the Figs. 9, 10, 11, 12, 13 focus on the same model while varying the slice cut.
In particular, all the mentioned visual comparisons (slice images) are taken by slicing the final-CpT-prediction 3D case on the x axis: therefore, the interpretation is that the reader is looking at the motorbike frontally as if it was coming straight against him/her. The leftmost slice image represents the CpT output coming out of the CFD simulations, taken as ground truth; the image in the middle shows the CpT computed after the inference performed by our graph machine learning model; finally, the rightmost image outlines the node-wise difference between the CFD and ML model.

Comparison of the network dimension

In the previous sections, we explained that 9 models in total were trained and tested, each of them coming from all the possible combinations of {8, 16, 32} hidden channels and {8, 16, 32} layers. Table 6 displays the amount of training time in hours on the GPU at our disposal, for each single trained model.

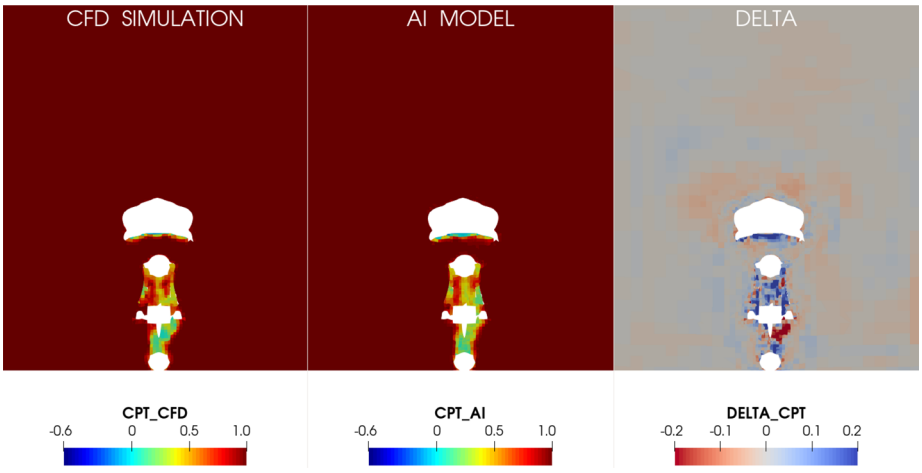


Fig. 9 Test Image: slice $x = 0.0$ of model with 32 layers and 32 hidden channels

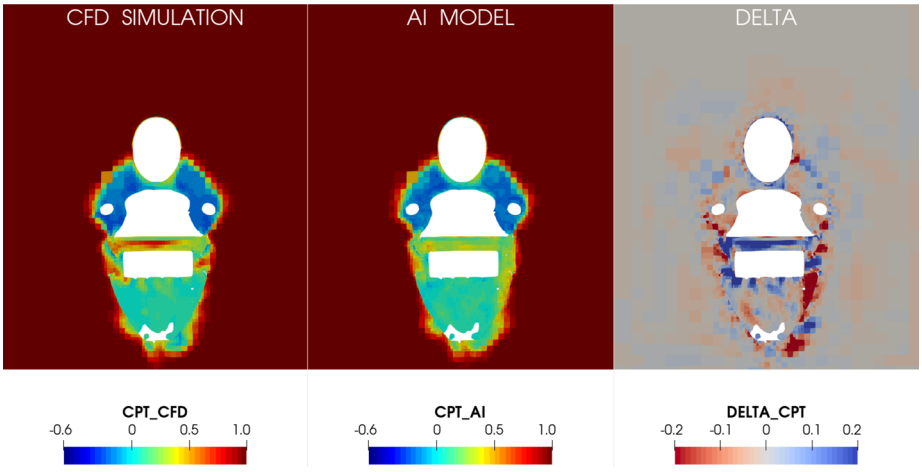


Fig. 10 Test Image: slice $x = 0.5$ of model with 32 layers and 32 hidden channels

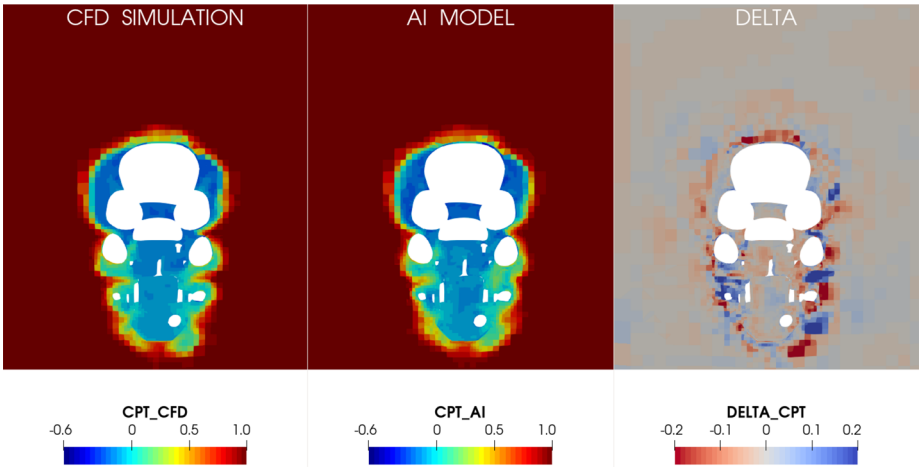


Fig. 11 Test Image: slice $x = 1.0$ of model with 32 layers and 32 hidden channels

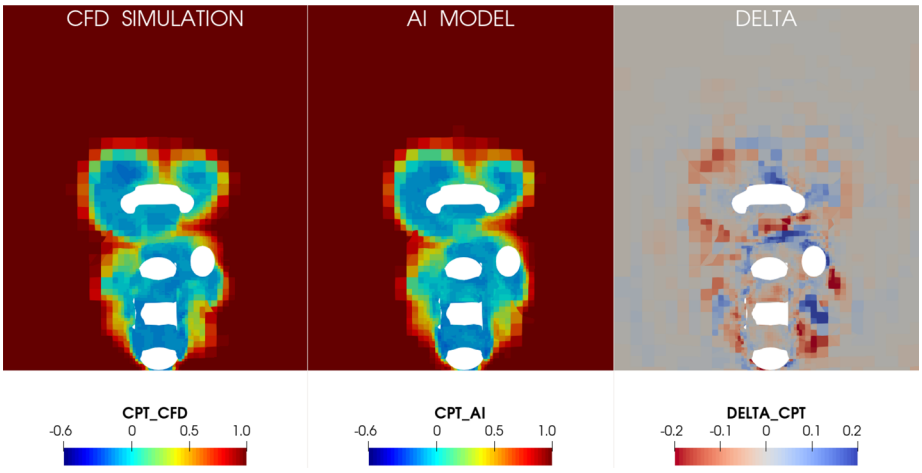


Fig. 12 Test Image: slice $x = 1.5$ of model with 32 layers and 32 hidden channels

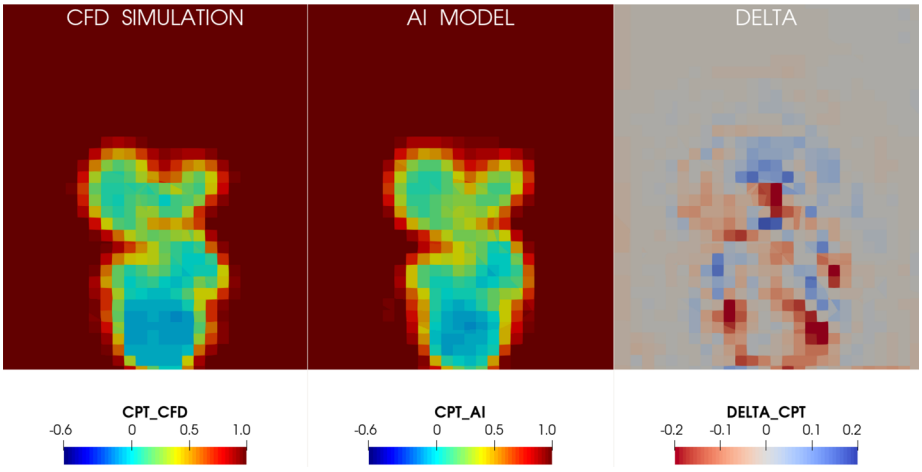
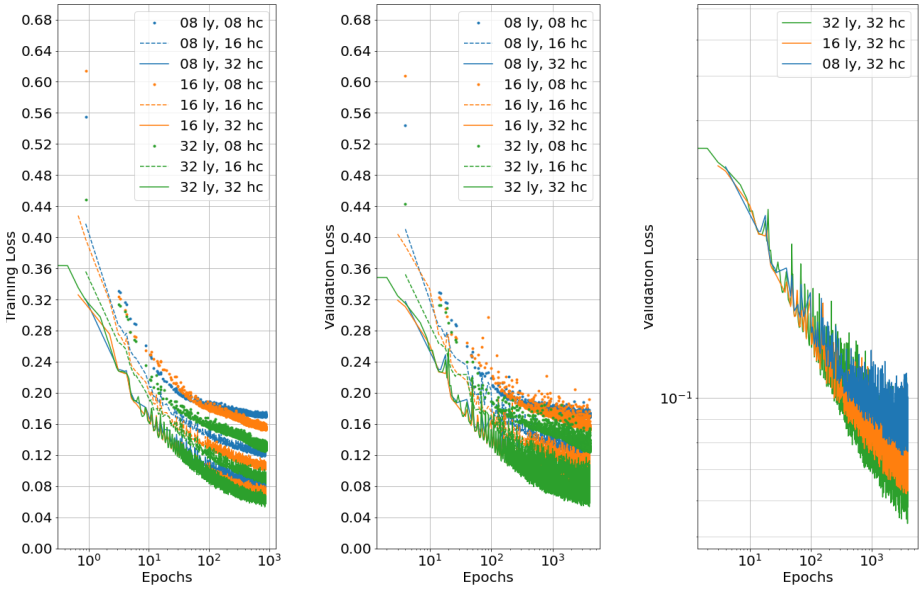


Fig. 13 Test Image: slice $x = 2.0$ of model with 32 layers and 32 hidden channels

What we can notice in Table 6 is that increasing the number of hidden channels and layers is computationally expensive. The training is indeed expected to take quite some time: as a reward, we can have a much faster inference compared to running full CFD simulations (typically less than 1 min using the same architecture to infer a new geometry). It is worth paying attention to the fact that increasing the number of hidden channels, as said in the previous section, is more beneficial performance-wise compared to increasing the number of hidden channels. Moreover, it appears to be even less computationally expensive as a marginal surge in the number of hidden channels takes less to train than the same surge in number of layers. This suggests that increasing the number of hidden channels should probably be the first move, as long as we do not experience overfitting effects.

Figure 14 shows the loss curves for all the 9 models, within a single plot. Models with the same number of hidden channels share the line shape (dotted, dashed, or continuous), while models with the same number of layers share the color.

The leftmost plot represents the training loss while the one in the middle stands for the validation loss, computed epoch by epoch while training. The validation loss is only for



(a) Training Loss (train set made of 90 geometries)

(b) Validation Loss (validation set made of 1 single geometry)

(c) Validation Loss: zoom in on the 32-hidden channels.

Fig. 14 Training and validation losses for several models with different configurations of layers and hidden channels. As far as the legends are concerned, "ly" stands for layers while "hc" stands for hidden channels. For test values, please refer to Tables 4, 5

monitoring and has been computed on a single geometry different from the ones used for training. This is also clear by looking at the increased variance of the models characterized by a higher number of hidden channels, a behavior that we already discussed in Tables 4, 5. Finally, the rightmost plot is nothing more than a *zoom-in* on the validation loss of the 32 hidden channel models, to better appreciate the loss function directions, as we propose an opposite overlapping of the colored lines.

Conclusions and outlooks

In this article, we applied some graph machine-learning surrogate models capable of handling distributed and large geometries (potentially millions of nodes) while providing a sufficiently accurate inference of the 3D flowfield variables for a new motorbike case. In order to achieve that, we first researched and found a suitable model whose capabilities aligned with our needs; then, we further adapted it to our own necessities and created a proper pipeline to efficiently deal with I/O operations. Once everything was correctly integrated, we set the hyperparameters and trained a different model for any combination of the {8, 16, 32} hidden channels and {8, 16, 32} layers. We plotted the training and validation loss, together with a direct visual comparison of 2D slices extracted from the 3D flowfield.

The best results in terms of validation loss and, visually, in terms of CpT delta (node by node CpT difference between each model and the CFD ground truth) tend to be towards the models made of 32 hidden channels, which are the most expressive. The training time, as we can see in Table 6, is higher as well for such models. The variance of the validation loss is larger as well: thus, further increasing ($>$ 32) the number of hidden channels

would probably make the model predictions a bit too fluctuating and very expensive from the training time perspective. Once trained, all of these models perform a prediction in between 20 s and 1 min (varying according to the number of hidden channels and layers), definitely less than RANS simulations.

Overall, the work done marks a promising beginning for further graph machine learning research and applications in the context of 3D flowfield modelling in aerodynamics. Strong attention should be drawn to controlling memory, aiming at shorter training and inference times, and, in particular, to building more scalable models. In fact, our models work well with geometries of medium to large size (up to a few million nodes), but struggle when dealing with extra-large cases, which are prevalent in the F1 contexts.

At the same time, finding and adding more features might be beneficial to reduce the degree of overfitting.

Additionally, it would be valuable to explore various research domains within the context of graph machine learning, in order to improve the generalization capabilities of the aerodynamics models. Some possible ideas would revolve around injecting some *physics awareness* into the models themselves (i.e. PINN), which might help in achieving more stable training and inference. Even though estimating the impact of any action is difficult without first experimenting, some approaches might involve tweaking the loss function by embedding some physical constraints, which might be based upon some approximated form of the Navier–Stokes equations.

An alternative solution might be to directly act on the message-passing algorithms which allow for information aggregation among the nodes of the graph in a GNN model: enabling some kind of physical enforcement might teach the model not to brutally learn vertex embeddings by replicating the output of a CFD simulation, but rather learn to replicate the behaviour leading to that output.

The intuition that we would like to convey here is that, even though we cannot fully reproduce physics, we would like our models to be at least "gently" guided by the actual underlying physical phenomena. If the model is guided in the correct direction, there might be a chance that it manages to reconstruct the meaning of the original equations via model weights rather than via actual equations.

Furthermore, we are confident our work is well-positioned to become a helpful methodology especially in repetitive frameworks, like shape optimization and optimal control, allowing for a faster procedure thanks to the quick inference time. Speeding up such processes would be of immense value. In spite of not having proper time and conditions to present the details in the current work, we believe the reader is deserving of a high-level idea.

In a high-performing environment such as motorsport, the geometry optimization process is multi-step and entails an elevated level of domain expertise and specialized roles. Currently, the optimization flow in an Motorsport Team comprises the following steps, within an iterative process:

1. Aerodynamicists have intuitions about a specific geometry that should be better than the current baseline geometry. They ask CAD designers to implement the update as a new 3D CAD geometry.
2. After the change is implemented, A CFD simulation is performed on top of the 3D model of the F1 car geometry which is under evaluation

3. Simulated variables (velocity, pressure, . . .) are visualized on single 2D-slices of the 3D domain (often in the form of an aggregate metric, like the CpT)
4. The 2D slices of the current geometry are compared to the ones of the previous geometry: this allows aerodynamicists to validate whether the change to the car structure is actually worth it. They use their domain expertise to keep the car well attached to the ground, by inspecting vortex formation and several small details. If monitored metrics and aerodynamicists' evaluations agree on the improvement, then the new geometry becomes the current baseline

Within this process, our model would support the aerodynamicists' decision making, allowing them to test several geometries quickly without running full and lengthy CFD simulations continuously.

Elucidating the process is useful to prove that each single environment has its own set of constraints. For example, in the motorsport industry, the level of detail might be overwhelming, thus it requires specific professionals to do relevant 'manual' inspection and assessment. In other fluid dynamics domains, there might be more automatized processes for shape optimization given fewer concerns about regulation, security or safety. In such sectors, the optimization process can become much more automatized if the right tools are put in place for the specific needs.

If you intend to optimize upon a specific metric, you might employ a suitable remeshing algorithm, which might make use of generative AI solutions. This algorithm might produce interpolation-based mesh variations of input geometries, aiming to produce more optimal shapes. Each geometry can be quickly inferred via our model; the reference metric can be computed, hoping that a better and better value is obtained; improvements can keep being produced until a certain threshold is met.

As we believe this process is heavily dependent on the needs of the specific company or institute, we will develop this in-depth analysis in a future work of ours.

Ultimately, we are convinced that everything we have done up to now is well posed for an extension toward dynamic integration, in order to compute the transient response. If we were to put the derivative in time, then there would be the possibility of extending the current framework with temporal autoregression.

We acknowledge the fact that static modeling assumes a steady-state condition, neglecting the effects of temporal changes and leading to potentially lower accuracy. Moreover, they may not provide a realistic representation of experimental observations when the system behavior is inherently dynamic. That is why transient modeling would allow better validation against experimental data.

More and more research articles are heading in this direction and our future work will also be contributing to such a dynamic integration.

Acknowledgements
The authors acknowledge the support of Sauber Motorsport for hosting DR during his internship, and for making this work possible.

Author contributions
DR wrote most of the text, prepared the results, and wrote the code to perform the machine-learning simulations. GS prepared the setup of the full-order problem, collected the data running the simulations, wrote part of the text, reviewed, and edited the contents. ND wrote part of the text and reviewed and edited the contents. DF contributed to the development of the code, ran the machine learning simulations, and reviewed and edited the contents. GR acquired funding and reviewed the work and the contents. All authors discussed the results and revised the draft.

Funding
This work was partially funded by European Union Funding for Research and Innovation in the framework of European Research Council Executive Agency: H2020 ERC CoG 2015 AROMA-CFD project 681447 "Advanced Reduced Order Methods with Applications in Computational Fluid Dynamics" and in the framework of the ERC POC 2022 ARGOS project 101069319 "Advanced Reduced order modellinG: Online computational web server for complex parametric Systems" P.I. Professor Gianluigi Rozza.

Availability of data and materials
Data is available upon reasonable request.

Declarations

Competing interests
There are no competing interests.

Received: 17 October 2023 Accepted: 30 January 2024
Published online: 23 March 2024

References

1. Rozza G, Stabile G, Ballarin F, et al. Advanced reduced order methods and applications in computational fluid dynamics. Soc Ind Appl Math. 2022. https://doi.org/10.1137/1.9781611977257.
2. Georgaka S, Stabile G, Star K, Rozza G, Bluck MJ. A hybrid reduced order method for modelling turbulent heat transfer problems. Comput Fluids. 2020;208: 104615. https://doi.org/10.1016/j.compfluid.2020.104615.
3. Stabile G, Rozza G. Finite volume POD-Galerkin stabilised reduced order methods for the parametrised incompressible Navier-Stokes equations. Comput Fluids. 2018;173:273–84. https://doi.org/10.1016/j.compfluid.2018.01.035.
4. Hesthaven JS, Rozza G, Stamm B. Certified reduced basis methods for parametrized partial differential equations. Springer; 2015.
5. Kim Y, Choi Y, Widemann D, Zohdi T. A fast and accurate physics-informed neural network reduced order model with shallow masked autoencoder. J Comput Phys. 2022;451: 110841. https://doi.org/10.1016/j.jcp.2021.110841.
6. Lee K, Carlberg KT. Model reduction of dynamical systems on nonlinear manifolds using deep convolutional autoencoders. J Comput Phys. 2020;404: 108973. https://doi.org/10.1016/j.jcp.2019.108973.
7. Romor F, Stabile G, Rozza G. Non-linear manifold ROM with convolutional autoencoders and reduced over-collocation method. J Sci Comput. 2023. https://doi.org/10.1007/s10915-023-02128-2.
8. Stabile G, Zancanaro M, Rozza G. Efficient Geometrical parametrization for finite-volume based reduced order methods. Int J Numer Methods Eng. 2020;121(12):2655–82. https://doi.org/10.1002/nme.6324.
9. Hijazi S, Stabile G, Mola A, Rozza G. Data-Driven POD-Galerkin reduced order model for turbulent flows. J Comput Phys. 2020;416: 109513. https://doi.org/10.1016/j.jcp.2020.109513.
10. Tezzele M, Demo N, Stabile G, Mola A, Rozza G. Enhancing CFD predictions in shape design problems by model and parameter space reduction. Adv Model Simul Eng Sci. 2020. https://doi.org/10.1186/s40323-020-00177-y.
11. Zancanaro M, Mrosek M, Stabile G, Othmer C, Rozza G. Hybrid neural network reduced order modelling for turbulent flows with geometric parameters. Fluids. 2021;6(8):296. https://doi.org/10.3390/fluids6080296.
12. Hesthaven JS, Ubbiali S. Non-intrusive reduced order modeling of nonlinear problems using neural networks. J Comput Phys. 2018;363:55–78. https://doi.org/10.1016/j.jcp.2018.02.037.
13. Umetani N, Bickel B. Learning three-dimensional flow for interactive aerodynamic design. ACM Trans Graphics (TOG). 2018;37(4):1–10.
14. Scillitoe AD, Seshadri P, Wong CY. Instantaneous flowfield estimation with Gaussian ridges. In: AIAA Scitech 2021 Forum; 2021. p. 1138.
15. Zhao Y, Yin F, Gunnarsson F, Hultkratz F, Fagerlind J, Gaussian processes for flow modeling and prediction of positioned trajectories evaluated with sports data. In,. 19th international conference on information fusion (FUSION). New York: IEEE. 2016;2016:1461–8.
16. Guo X, Li W, Iorio F. Convolutional neural networks for steady flow approximation. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining; 2016. p. 481–90.
17. Tangsali K, Krishnamurthy VR, Hasnain Z. Generalizability of convolutional encoder-decoder networks for aerodynamic flow-field prediction across geometric and physical-fluidic variations. J Mech Des. 2021;143(5): 051704.
18. Eivazi H, Veisi H, Naderi MH, Esfahanian V. Deep neural networks for nonlinear model order reduction of unsteady flows. Phys Fluids. 2020;32(10): 105104.
19. Wang J, He C, Li R, Chen H, Zhai C, Zhang M. Flow field prediction of supercritical airfoils via variational autoencoder based deep learning framework. Phys Fluids. 2021;33(8): 086108.
20. Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys. 2019;378:686–707.
21. Cheng C, Zhang G-T. Deep learning method based on physics informed neural network with resnet block for solving fluid flow problems. Water. 2021;13(4):423.
22. Ogoke F, Meidani K, Hashemi A, Farimani AB. Graph convolutional neural networks for body force prediction; 2020. arXiv preprint arXiv:2012.02232.
23. Yang Z, Dong Y, Deng X, Zhang L. AMGNET: multi-scale graph neural networks for flow field prediction. Connect Sci. 2022;34(1):2500–19.
24. Kipf TN, Welling M. Semi-supervised classification with graph convolutional networks. In: 5th international conference on learning representations (ICLR-17); 2016.
25. Sanchez-Gonzalez A, Godwin J, Pfaff T, Ying R, Leskovec J, Battaglia P. Learning to simulate complex physics with graph networks. In: International conference on machine learning; 2020.

26. Pfaff T, Fortunato M, Sanchez-Gonzalez A, W Battaglia P. Learning mesh-based simulation with graph networks. In: International conference on learning representations (ICLR 2021); 2021.
27. Hamilton WL, Ying R, Leskovec J. Inductive representation learning on large graphs. Neural Information Processing Systems (NeurIPS); 2017.
28. Zeng H, Zhou H, Srivastava A, Kannan R, Prasanna V. GraphSAINT: graph sampling based inductive learning method. In: International conference on learning representations (ICLR); 2020.
29. Li G, Muller M, Thabet A, Ghanem B. DeeperGCN: All You Need to Train Deeper GCNs. In: Proceedings of the IEEE international conference on computer vision (ICCV); 2019.
30. Li G, Müller M, Ghanem B, Koltun V. Training graph neural networks with 1000 layers. In: ICML; 2021.
31. Veličković P, Cucurull G, Casanova A, Romero A, Liò P, Bengio Y. Graph attention networks. In: 6th international conference on learning representations (ICLR 2018); 2018.
32. Yun S, Jeong M, Kim R, Kang J, Kim HJ. Graph transformer networks. Advances in neural information processing systems; 2019. 32.
33. Ying R, He R, Chen K, Eksombatchai P, Hamilton WL, Leskovec J. Graph convolutional neural networks for web-scale recommender systems. In: Proceedings of the 24th ACM SIGKDD international conference on knowledge discovery & data mining; 2018. p. 974–83.
34. You J, Ying R, Leskovec J. Position-aware graph neural networks. In: International conference on machine learning; 2019. p. 7134–43. PMLR.
35. Chiang W-L, Liu X, Si S, Li Y, Bengio S, Hsieh C-J. Cluster-gcn: An efficient algorithm for training deep and large graph convolutional networks. In: Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining; 2019. p. 257–66.
36. He X, Hooi B, Laurent T, Perold A, LeCun Y, Bresson X. A generalization of vit/mlp-mixer to graphs. In: International conference on machine learning; 2023. p. 12724–5. PMLR.
37. Sederberg TW, Parry SR. Free-form deformation of solid geometric models. ACM SIGGRAPH Comput Graph. 1986;20(4):151–60. https://doi.org/10.1145/325165.325247.
38. OpenFOAM Foundation https://www.openfoam.com. OpenFOAM is an open-source software package for computational fluid dynamics (CFD) simulations developed by the OpenFOAM Foundation; 2023.
39. OpenFOAM motorbike tutrorial; 2023. https://develop.openfoam.com/Development/openfoam/-/tree/master/tutorials/incompressible/simpleFoam/motorBike. Accessed 30 Aug 2023.
40. Gibson P, Cano J, Turner J, Crowley E, O'Boyle M, Storkey A. Optimizing grouped convolutions on edge devices. 2020. https://doi.org/10.1109/ASAP49362.2020.00039.
41. Zeng H, Zhou H, Srivastava A, Kannan R, Prasanna V. Graphsaint: graph sampling based inductive learning method; 2019. arXiv preprint arXiv:1907.04931.

Publisher's Note