mathematics

Article

Graph Neural Networks for Mesh Generation and Adaptation in Structural and Fluid Mechanics

Ugo Pelissier [1,2], Augustin Parret-Fréaud [2], Felipe Bordeu [2] and Youssef Mesri [1,*]

[1] Computing and Fluids Research Group (CFL), Centre for Material Forming (CEMEF), Mines Paris, PSL University, UMR7635 CNRS, 06904 Sophia Antipolis, France; ugo.pelissier@etu.minesparis.psl.eu
[2] Digital Sciences & Technologies Department, Safran Tech, Rue des Jeunes Bois, Châteaufort, 78114 Magny-Les-Hameaux, France; augustin.parret-freaud@safrangroup.com (A.P.-F.); felipe.bordeu@safrangroup.com (F.B.)
* Correspondence: youssef.mesri@minesparis.psl.eu

Abstract: The finite element discretization of computational physics problems frequently involves the manual generation of an initial mesh and the application of adaptive mesh refinement (AMR). This approach is employed to selectively enhance the accuracy of resolution in regions that encompass significant features throughout the simulation process. In this paper, we introduce Adaptnet, a Graph Neural Networks (GNNs) framework for learning mesh generation and adaptation. The model is composed of two GNNs: the first one, Meshnet, learns mesh parameters commonly used in open-source mesh generators, to generate an initial mesh from a Computer Aided Design (CAD) file; while the second one, Graphnet, learns mesh-based simulations to predict the components of an Hessian-based metric to perform anisotropic mesh adaptation. Our approach is tested on structural (Deforming plate–Linear elasticity) and fluid mechanics (Flow around cylinders–steady-state Stokes) problems. Our findings demonstrate the model's ability to precisely predict the dynamics of the system and adapt the mesh as needed. The adaptability of the model enables learning resolution-independent mesh-based simulations during training, allowing it to scale effectively to more intricate state spaces during inference.

Keywords: artificial intelligence; message passing graph neural networks; mesh adaptation; computational fluid dynamics

MSC: 65N50

Citation: Pelissier, U.; Parret-Fréaud, A.; Bordeu, F.; Mesri, Y. Graph Neural Networks for Mesh Generation and Adaptation in Structural and Fluid Mechanics. *Mathematics* **2024**, *12*, 2933. https://doi.org/10.3390/math12182933

Academic Editor: Xiangmin Jiao

Received: 22 August 2024
Revised: 12 September 2024
Accepted: 14 September 2024
Published: 20 September 2024

1. Introduction

The generation of meshes constitutes a crucial stage in numerically simulating diverse problems within the domain of computational science [1]. Unstructured meshes find prevalent application, notably in fields like computational fluid dynamics (CFD) [2,3] and computational solid mechanics (CSM) [4,5], wherein the finite element (FE) method is employed to approximate solutions for general partial differential equations (PDE), particularly in domains featuring challenging geometries [6,7]. The accuracy of the FE solution is intricately linked to the quality of the associated mesh.

Mesh generation depends on a number of parameters, such as the local metrics at given points, but also, in the general case, the maximum and minimum sizes of cell elements, the Hausdorff coefficient, the acceptable gradation of the metric, and so on [8]. A set of parameters whose choice is far from trivial and whose poor choice can lead to meshes of poor quality, a generation process that is too long (because it is over-constrained), or even a generation failure.

A mesh can be adapted and refined to suit the physical problem being solved. The adaptation of unstructured meshes has proven very effective for stationary calculations in Fluid Mechanics and Solid Mechanics, with the aim of capturing the behavior of physical phenomena while obtaining the desired accuracy for the numerical solution. In addition, this method considerably reduces the computational cost of the numerical solution by reducing the number of degrees of freedom. The a priori and a posteriori error estimators generally give an upper bound in the approximation error of the solved PDE. These estimates depend on the approximation space, the mesh size, the equation, the approximate solution in the case of a posteriori estimators and the exact solution in the case of a priori estimators.

Theoretical frameworks for anisotropic error estimation have been extensively developed, resulting in a degree of standardization in the adaptation process. Various works [9–11], have contributed to the estimation of approximation and interpolation errors. Recent analyses of interpolation errors [12–16] have refocused attention on metric-based mesh adaptation, particularly utilizing a metric derived from a recovered Hessian. Notably, Hessian-based metric mesh adaptation offers several advantages, including a general computational framework, a relatively straightforward implementation, and, most importantly, robustness.

Machine Learning (ML) and Deep Learning (DL) techniques in numerical simulations have primarily been applied to predicting stationary or time-dependent fields as solutions to specific PDEs. The goal has often been to reduce computation time by using neural networks (NNs) for predictions, instead of traditional FE solvers [17,18]. Convolutional neural networks (CNNs), originally designed to extract information from image-like data, have been widely employed in this domain [19]. In such cases, simulation domains are represented using Cartesian grids, where geometry and initial conditions are projected [20–25]. However, this approach can result in the loss of vital information contained in the original unstructured mesh and lead to inefficient meshes with higher computational costs.

To overcome these limitations, CNNs can be extended to operate on graph structures, which is critical since many solvers for CFD and CSM rely on unstructured meshes for FE discretization.

Message Passing Neural Networks (MPNNs), a subclass of Graph Neural Networks (GNNs) [26], have gained traction for mesh-based simulations due to their capacity to handle complex spatial data. Unlike conventional convolutional techniques, MPNNs offer greater flexibility in processing graph-structured data.

An exemplary application of MPNNs can be found in the work of Gilmer et al. (2017) [27], where a message-passing neural network was developed for quantum chemistry simulations. Their method splits the convolution process into two phases: first, aggregating information from neighboring nodes and edges into a hidden node state, and then using this state to update the node features.

Battaglia et al. (2018) [28] introduced the Graph Network (GN) framework, where messages are transmitted from nodes to edges using an edge convolution kernel, followed by aggregation of edge features at the nodes. The edge messages, formed through permutation-invariant operations, are then combined with original node features to produce updated node representations. This approach excels at capturing complex dependencies within graph-structured data.

Building on this, Pfaff et al. (2021) [29] applied GNs to mesh-based simulations, focusing on cases like incompressible flow around cylinders and compressible flow around airfoils. Their model acts as an accurate incremental simulator and adapts to the mesh structure. A key strength of this model is its ability to generalize to unseen mesh shapes and sizes, highlighting its robustness across diverse simulation scenarios.

Yet, this recent literature has not been applied to predict the complete process of AMR, from the initial geometry to the adapted mesh, in three dimensions. To this end, we propose Adaptnet, which is a framework for learning mesh parameters and tetrahedral mesh-based simulations using GNNs (Figure 1). This framework consists of two networks trained to generate unstructured meshes adapted to the physics under study:

- Meshnet is trained to predict mesh parameters of a given geometry (CAD file) later used to generate an initial mesh.

- Graphnet is trained to predict a metric field, either directly or indirectly by predicting the velocity field, later used to generate an anisotropic adapted mesh.

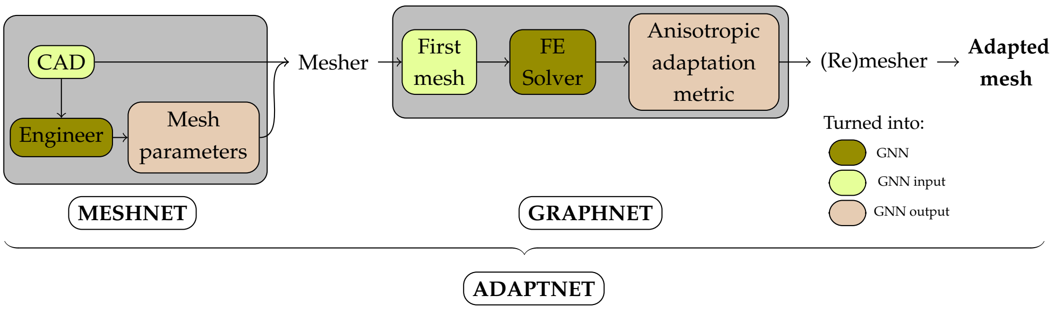


Figure 1. Diagram of Adaptnet framework to predict a mesh adapted to some physics, given an initial geometry.

Both models are message-passing graph neural networks that rely on the architecture of MeshGraphNets [29,30] that was further adapted to steady-state predictions. We will explain in detail the materials and methods and results for the case of the steady-state Stokes problem. The linear elasticity problem will be presented at the end to demonstrate the model's ability to switch from one physics to another. We made use of open-source libraries to direct our work towards open-source solutions and to promote reproducibility. The remainder of this paper is structured as follows:

- Section 2: Materials and Methods—This section presents the problem statement, describes the dataset used, elaborates on the model architecture, and details the training configuration.
- Section 3: Results—Here, we provide the results of our experiments, focusing on the performance of the proposed models Meshnet, Graphnet, and Adaptnet.
- Section 4: Discussion—This section discusses the implications of our findings, in particular the generalization of our approach to unseen data and configurations.
- Section 5: Conclusions—Finally, we summarize the key contributions of our work and suggest directions for future research.

2. Materials and Methods

2.1. Problem Statement

Both the Stokes flow and linear elasticity problems will be examined using the same geometric configuration. Specifically, each problem will be analyzed within a domain featuring a rectangular box with circular obstacles at its center, allowing for a consistent comparison of the effects of fluid flow and structural deformation on a shared geometry.

2.1.1. Stokes Problem

We consider the steady-state Stokes problem of a fluid flow around a cylinder. The domain Ω is a box of size $(L \times h \times 1)$ with circular obstacles of radius r in the center. The boundary $\partial\Omega$ is divided into four parts: $\partial\Omega_{in}$, $\partial\Omega_{out}$, $\partial\Omega_{wall}$ and $\partial\Omega_{obs}$, corresponding to the boundaries of inlet, outlet, wall, and obstacles, respectively. The problem is defined as follows:

$$\begin{cases} -\mu\Delta\mathbf{u} = \nabla p - \rho\mathbf{f} & \text{in } \Omega \\ \nabla\cdot\mathbf{u} = 0 & \text{in } \Omega \\ \mathbf{u} = \mathbf{u}_{in} & \text{on } \partial\Omega_{in} \\ \mathbf{u} = \mathbf{0} & \text{on } \partial\Omega_{obs} \end{cases} \tag{1}$$

where $\mathbf{u}$ is the velocity field, p is the pressure field, $\mathbf{f}$ is the body force, ν is the dynamic viscosity, ρ is the density and $\mathbf{u}_{in}$ is the inlet velocity. The inlet velocity is set to $\mathbf{u}_{in} = (1,0,0)$ m·s^{-1}, the dynamic viscosity is set to $\mu = 0.1$ Pa·s and the density to $\rho = 1$ kg·m^{-3}. The body force is set to zero.

2.1.2. Linear Elasticity Problem

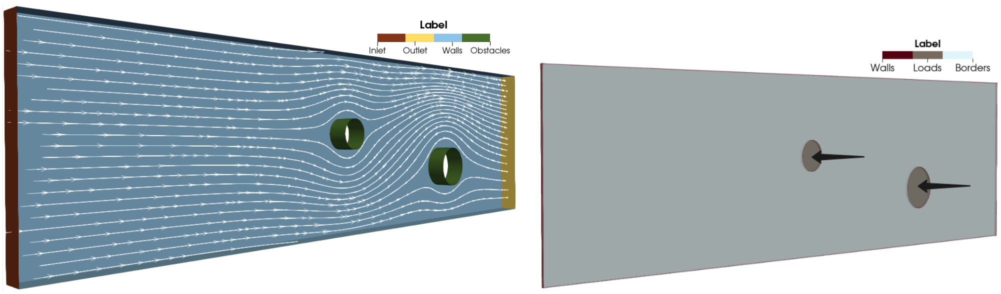
We consider the structural mechanics problem of an elastic plate, fixed on both sides, deformed by the action of two cylindrical actuators. The domain Ω is a plate of size $(L \times h \times 1)$. The boundary $\partial\Omega$ is divided into three parts: $\partial\Omega_{load}$, $\partial\Omega_{walls}$ and $\partial\Omega_{border}$, corresponding to the loaded cylinders, the fixed walls and the rest of the borders of the plate. The problem is defined as follows:

$$\begin{cases} -\nabla \cdot \sigma(\mathbf{u}) = \mathbf{f} & \text{in } \Omega \\ \sigma(\mathbf{u}) = \lambda \mathrm{tr}(\boldsymbol{\epsilon}(\mathbf{u}))\mathbf{I} + 2\mu\boldsymbol{\epsilon}(\mathbf{u}) & \text{in } \Omega \\ \boldsymbol{\epsilon}(\mathbf{u}) = \frac{1}{2}\left(\nabla\mathbf{u} + (\nabla\mathbf{u})^T\right) & \text{in } \Omega \\ \mathbf{u} = 0 & \text{on } \partial\Omega_{walls} \\ \sigma \cdot \mathbf{n} = \mathbf{T} & \text{on } \partial\Omega_{load} \end{cases} \tag{2}$$

where $\mathbf{u}$ is the displacement field, σ is the stress tensor, $\boldsymbol{\epsilon}$ is the strain tensor, $\mathbf{f}$ is the body force, λ and μ are the Lamé coefficients, $\mathbf{T}$ is the load and $\mathbf{n}$ is the normal vector. The load is set to $\mathbf{T} = (0, -10, 0)$ N, Lamé coefficients to $\lambda = 1.15 \times 10^5$ Pa and $\mu = 8.3 \times 10^4$ Pa and the body force is set to zero.

2.2. Dataset

The dataset is composed of 500 geometries of size $(L \times h \times 1)$ with two circular obstacles of radius (r_1, r_2) inside (Figure 2). The geometries are generated by randomly setting the box dimensions and the obstacle radius around control values.



(**a**) (**b**)

Figure 2. Labeled CAD from fluid and structural mechanics cases. (**a**) Labeled CAD file from sample 106 for Stokes problem. The boundary is divided into inlet, outlet, walls and obstacles. (**b**) Labeled CAD file from sample 106 for linear elasticity problem. The boundary is divided into walls, loads and borders.

The datasets are divided into 375 training samples, 75 validation samples, and 50 test samples. The training and validation samples are used to train the model, whereas the test samples are used to evaluate the model's performance. Each sample is numbered from 0 to 499. Most illustrative figures will use sample 106 from the test set.

2.3. Mesh Parameters

CAD files are automatically generated under *.geo_unrolled* extension using a Python script. In this work, the mesh parameters are limited to the mesh size defined at each point of the CAD file. Mesh size is set randomly around 0.1 at the circular obstacles while the one at the box extremities is set according to their proximity to an obstacle in order to test the network under real-case mesh parametrization, rather than a uniform one. Figure 3 highlights in red the 12 points representing each CAD file and labeled with a mesh size used to generate the initial mesh. Mesh generation is performed using *Gmsh* [31] software (4.8.4).

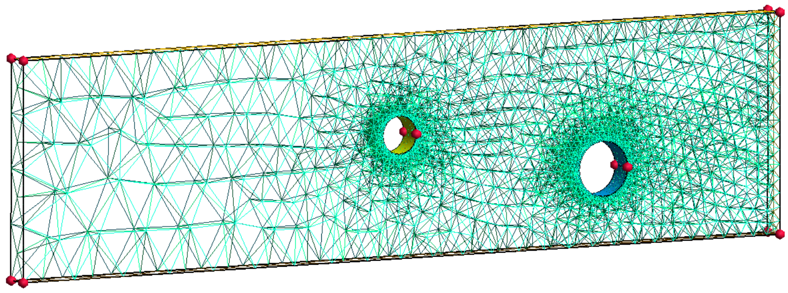


Figure 3. Generated 3D tetrahedral mesh for sample 106 using mesh size at geometry points highlighted in red. The mesh is generated using *Gmsh* [31].

2.4. Mesh-Based Simulations

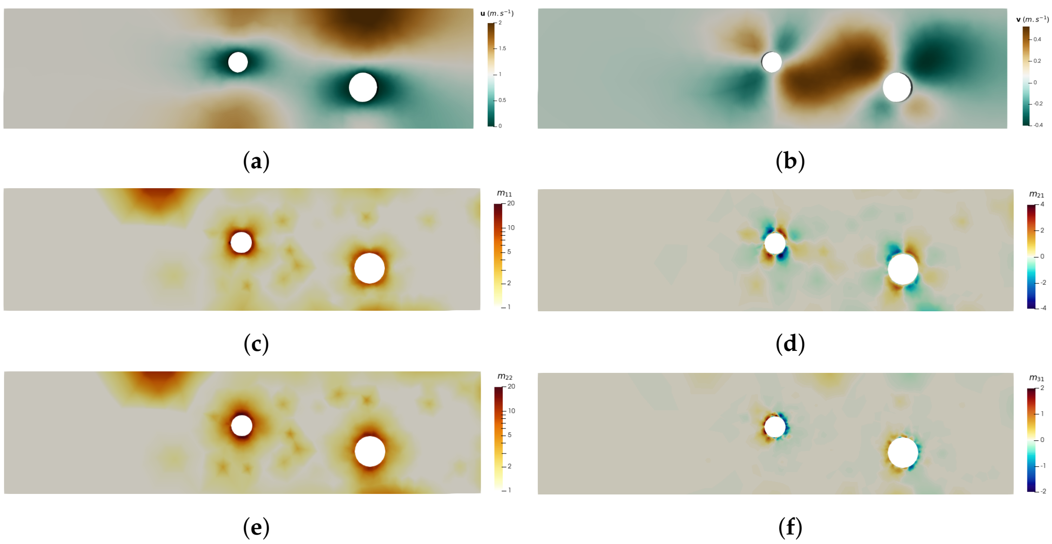
Mesh-based simulations are performed using the open-source FE library *FreeFem++* [32]. For the Stokes problem, it outputs the velocity field $\mathbf{u}$ and the pressure field p on the mesh (Figure 4a). This library uses a Hessian-based metric to perform anisotropic mesh adaptation. Given a triangulation Ω_h, one can derive an upper bound of the approximation error using an interpolation error analysis. This upper bound is expressed thanks to the recovered Hessian of the approximated solution u_h [9]. In fact, using $\mathcal{P}1$ linear elements, we usually cannot directly compute the Hessian of the solution. Instead, we compute an approximation called the recovered Hessian matrix $H_R(u_h(x))$ [10].

The recovered Hessian matrix is not a metric because it is not positive definite. Therefore, we define the following metric tensor:

$$\mathcal{M} = \mathcal{R}\Lambda\mathcal{R}^T \tag{3}$$

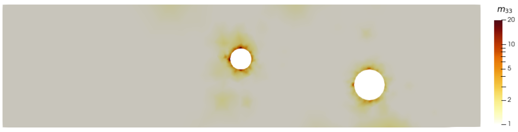
where $\mathcal{R}$ is the orthogonal matrix built with the eigenvectors (e_1, e_2, e_3) of $H_R(u_h(x))$ and $\Lambda = diag(|\lambda_1|, |\lambda_2|, |\lambda_3|)$ is the diagonal matrix of the absolute value of the eigenvalues of $H_R(u_h(x))$. By construction in *FreeFem++* and *Mmg* [33], the prescribed size is the inverse of the square root of the metric eigenvalues, so $\frac{1}{\lambda_1^2}$ is the size imposed in the first direction, $\frac{1}{\lambda_2^2}$, in the second direction, and $\frac{1}{\lambda_3^2}$ in the third direction. Ultimately, $\mathcal{M}$ is symmetric positive definite and is characterized by only six components: $(m_{11}, m_{21}, m_{22}, m_{31}, m_{32}, m_{33})$.

A similar set of figures can be found in Appendix A.1 for the linear elasticity problem.

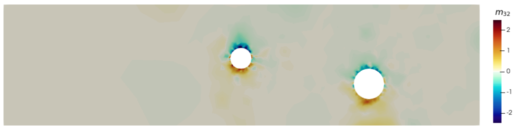


(a) **(b)** **(c)** **(d)** **(e)** **(f)**

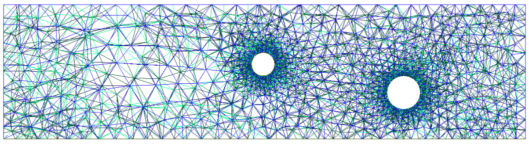
Figure 4. *Cont.*



(**g**)



(**h**)



(**i**)

Figure 4. Finite Element solving of Stokes problem and anisotropic metric computation. (**a**) Velocity field for sample 106 solved by *FreeFem++* along $x-$ axis. (**b**) Velocity field for sample 106 solved by *FreeFem++* along $y-$ axis. (**c**) First diagonal component (m_{11}) of the metric tensor for sample 106 computed by *FreeFem++*. (**d**) First non-diagonal component (m_{21}) of the metric tensor for sample 106. (**e**) Second diagonal component (m_{22}) of the metric tensor for sample 106. (**f**) Second non-diagonal component (m_{31}) of the metric tensor for sample 106. (**g**) Third diagonal component (m_{33}) of the metric tensor for sample 106. (**h**) Third non-diagonal component (m_{32}) of the metric tensor for sample 106. (**i**) Adapted mesh to Stokes problem for sample 106.

2.5. Graph Representation

CAD files and unstructured meshes are essentially a list of points and edges. We can take advantage of this structure to encode these input data into graphs. Each node and edge of the graph will carry initial information (based on coordinates, boundary conditions, and initial conditions) and the target to predict (either mesh size or physical field). For each data, we can access:

- Geometry: 3D coordinates
- Topology: node connections
- Boundary Conditions (BC): inlet, outlet, walls, obstacles or fluid
- Initial Conditions (IC): velocity and pressure field (mesh-based simulations only)

A graph is denoted by $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ where $\mathcal{V}$ and $\mathcal{E}$ are vertex and edge sets, respectively. Two nodes graph, i and j, connected are represented by two graph edges, one pointing to node j and another pointing to node i.

For both networks, node features include the physical surface information (BC) and, in the case of Graphnet, the velocity field. The edge features are enhanced by incorporating both the signed distance between nodes, $\mathbf{u}_j - \mathbf{u}_i \in \mathbb{R}^3$, and its absolute value. Figures 5 and 6 illustrate this for Meshnet and Graphnet, respectively.

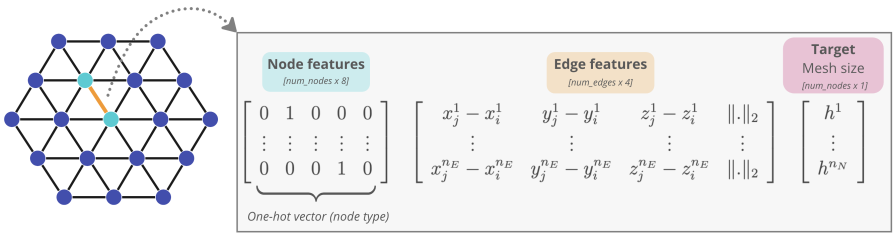


Figure 5. Meshnet encoding illustration. The encoder encodes the current mesh into a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$. Mesh nodes become graph vertices $\mathcal{V}$, and mesh edges become bidirectional mesh-edges in the graph $\mathcal{E}$.

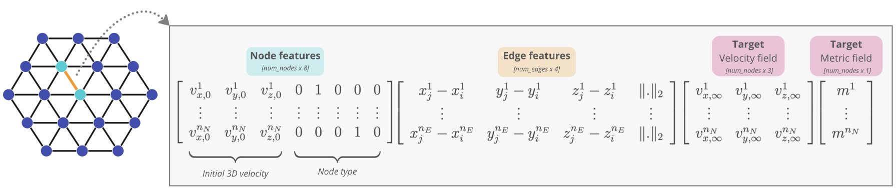


Figure 6. Graphnet encoding illustration.

2.6. Model

We use a graph neural network model with an Encoder–Processer–Decoder architecture [29,30,34] (Figure 7).

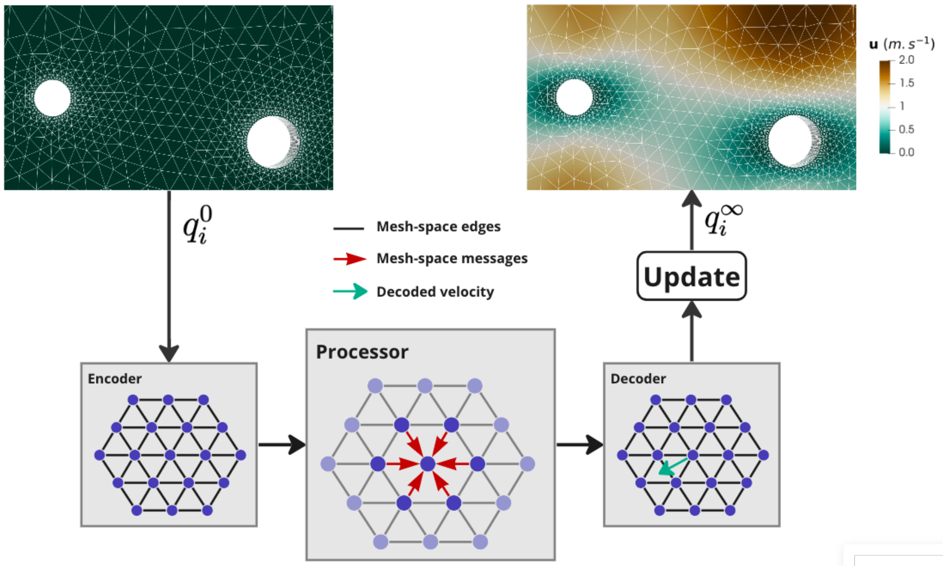


Figure 7. Diagram of Graphnet operating on Stokes problem. The model uses an Encoder–Processer–Decoder architecture. The encoder transforms the input mesh into a graph. The processor performs several rounds of message passing along mesh edges, updating all node and edge embeddings. The decoder returns the velocity or metric field for each node, which is then used to compute the Hessian and the metric field in order to trigger remeshing.

The architecture proposed by Pfaff et al. [29] is essentially designed to predict the dynamic states of a system over time. Since our test cases are steady-state predictions, we adapt this architecture following the work of Harsch et al. [35] to better capture local and global features.

2.6.1. Encoder

The Encoder is responsible for transforming the input mesh into a graph. The encoder consists of multilayer perceptrons (MLPs) that map the physical features of the nodes and edges to latent feature vectors (Table A1).

2.6.2. Processor

The processor consists of sequential message-passing steps. At each step, the feature of an edge is updated based on its value from the previous step, along with the features of the adjacent nodes. Similarly, each node is updated by incorporating its previous value and an aggregation of its incident edge features. This process iteratively computes an updated set of node features.

2.6.3. Decoder

The decoder is responsible for predicting the target field. The decoder consists of an MLP that maps the node features to the target field.

2.6.4. Local and Global Features

To increase the ability to extract local and global properties, we adopt the structure proposed by Wang et al. [36] for Dynamic Graph CNN and adapted by Harsch et al. [35]. This adaptation aggregates the information given by each message passing block to calculate a global feature vector. In that sense, the number of layers is reduced to five, instead of 15 previously. To avoid potential overfitting, a pooling operation is added, and the result is concatenated with the previous local features. This adapted version is called Graphnet (Pool) (Figure 8).

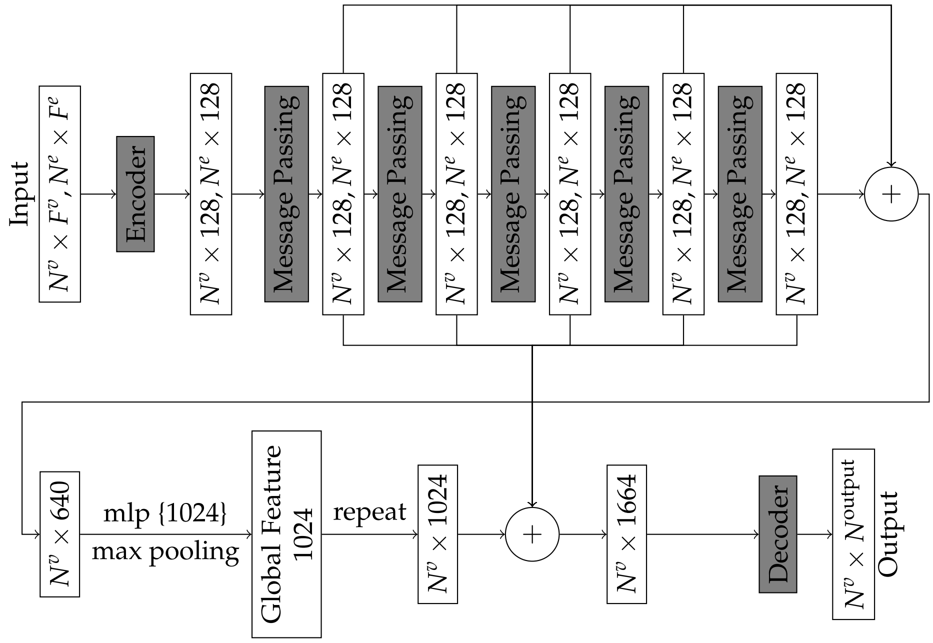


Figure 8. Adapted model architecture using message passing block with local and global descriptor [35] (operator $\oplus$ denotes the concatenation).

2.7. Loss Function and Optimizer

We train both networks by minimizing the error between the true node label and the predicted node label. We use the per-node root mean square error (RMSE) loss to quantify the data error for each simulation. The loss function reads:

$$\mathcal{L} = \sqrt{\frac{1}{n_B}\sum_{i=1}^{n_B} \|y_i - \hat{y}_i\|_2^2} \tag{4}$$

where n_B denotes the number of nodes in a batch of training meshes, y_i denotes the true output in the dataset, $\hat{y}_i$ is the output predicted by the network.

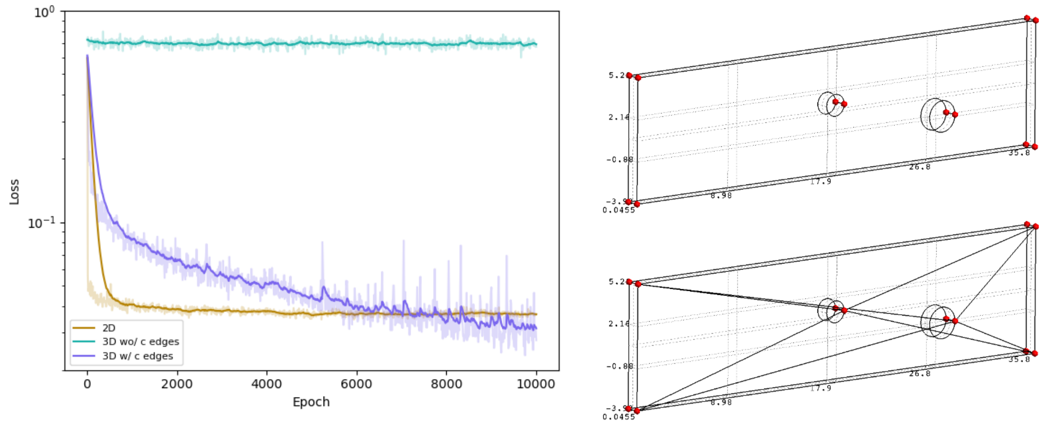
ADAM (https://geoprograms.com/software/) optimizer is used and the learning rate is constant, equal to 0.001. With the dataset and a fixed set of hyper-parameters, each epoch takes 30 s on two NVIDIA A100 GPUs. The value of specific training hyperparameters is given in Table A2.

3. Results

3.1. Meshnet

3.1.1. Fine Tuning Input Data for Message-Passing GNN

Meshnet was trained to learn to predict the mesh size at each point of a given geometry. These mesh sizes are then used to generate a first mesh using *Gmsh* [31]. The loss curve highlights the message-passing nature of this GNN. However, when we first trained the network, the case did not converge, giving the blue cyan loss curve in Figure 9a. During the graph-encoding step (Figure 5), we pass to the network the distance of each edge of the geometry in the edge feature matrix. However, in our geometrical representation, the points of the box are not linked to the obstacles (Figure 9b). Thus, the network is blind and cannot learn this specific mapping. As a workaround, we manually added controlled edges so that the new edge feature would keep the distance between the points of the box and the obstacles (Figure 9b).

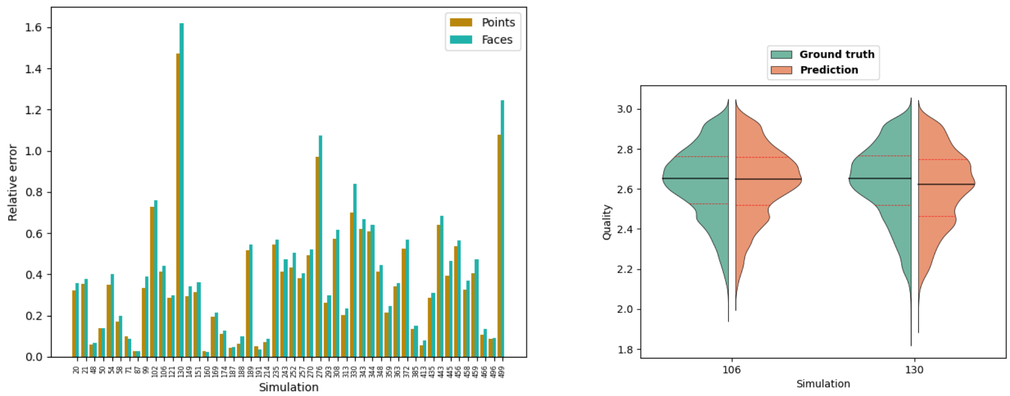


(**a**) (**b**)

Figure 9. Influence of the controlled edges on Meshnet learning curve through message-passing. (**a**) Meshnet training in 2D and 3D for 10,000 epochs, without and with controlled edges. (**b**) Controlled edges addition (**bottom**) to initial geometry (**top**) on simulation 106.

3.1.2. Evaluating Meshes Similarity and Quality

To evaluate the network, we are also interested in the mesh similarities between the mesh generated with the ground truth parameters, and the one generated by the predictions. The comparison of the number of points and tetrahedra between the ground truth and the prediction can only give us a small overview (Figure 10a).



(**a**) (**b**)

Figure 10. Quality of meshes predicted by Meshnet on simulations from the test set. (**a**) Points and tetrahedra relative error on test set simulations. (**b**) "Glassmaier" parameter distribution for simulations 106 and 130 (black line: median–red dotted lines: quartiles).

We can take, for instance, the cases of simulations 106 and 130. The former shows a good similarity in terms of points and tetrahedra, while the latter has a prediction with twice as many points and tetrahedra as the ground truth. To better assess the quality of each mesh, we use "Glassmaier" parameter [37,38] defined as

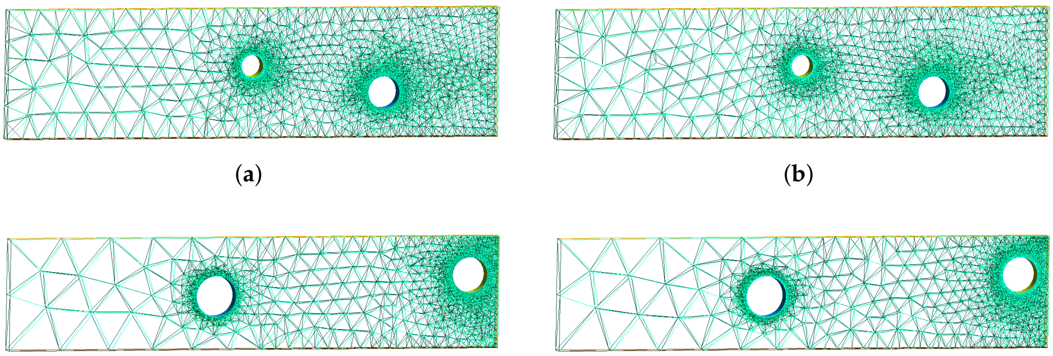
$$Q_G = 1 + \frac{\text{True Surf.}}{\text{Ideal Surf.}} + \frac{\text{True Vol.}}{\text{Ideal Vol.}} \tag{5}$$

and takes on values between 1 and 3. It tends to describe the dimensionality of the figure, as listed in Table 1. The ideal volume and surface are calculated for a regular tetrahedron with a side length equal to the average of the six distances between the 4 points.

Table 1. Special values of the Glassmeier parameter.

Q_G	Meaning
1.0	The four points are colinear.
2.0	The four points all lie in a plane.
3.0	A regular tetrahedron is formed.

The distribution of this quality estimator among the ground truth and predicted meshes are very similar for both cases (Figure 10b), and the visual observation confirms this conclusion (Figure 11). In fact, the number of points and tetrahedra is greatly sensitive to the mesh size defined on the obstacles, since it is the smallest one, thus giving the greatest number of points of tetrahedra. Thus, in the case of simulation 130, the ground truth and the prediction have a similar pattern of mesh, but one is finer at the obstacles.



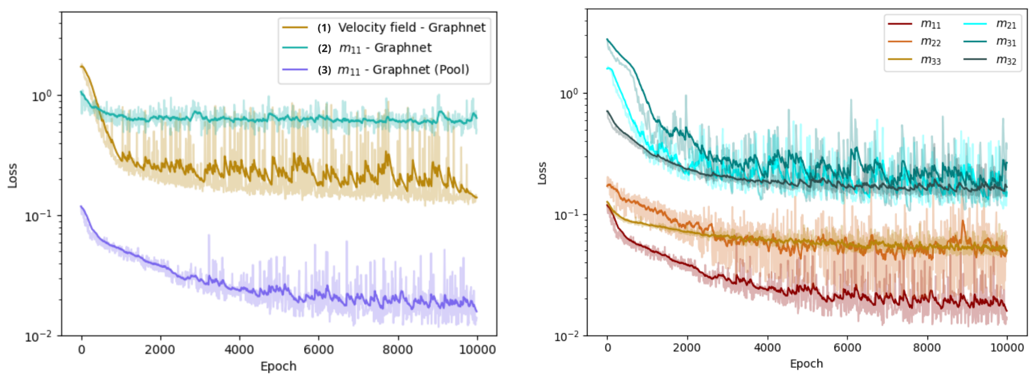
(**a**) (**b**) (**c**) (**d**)

Figure 11. Mesh visualization of simulations 106 and 130 using Gmsh [31]. (**a**) Simulation 106—Ground truth mesh. (**b**) Simulation 106—Predicted mesh. (**c**) Simulation 130—Ground truth mesh. (**d**) Simulation 130—Predicted mesh.

3.2. Graphnet

3.2.1. Direct Predictions: Velocity and Anisotropic Metric Fields

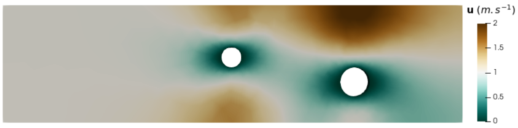
Graphnet was trained to learn mesh-based simulations. First, we trained one network to predict the three components of the velocity field, in order to use it to compute an anisotropic metric from the Hessian of the velocity field (Figure 12a (1)). Then, we decided to predict directly the metric field by training one network for each of the six components of the metric tensor (Figure 12a (2)). Finally, we use the adapted architecture Graphnet (Section 2.6.4) to improve the training (Figure 12a (3)).



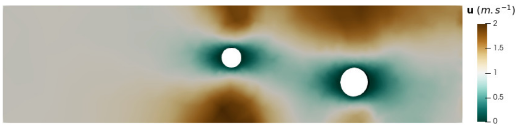
(**a**) (**b**)

Figure 12. Graphnet training for 10,000 epochs. (**a**) Comparison of training loss as a function of the number of predicted components ((1)–(2)) and network architecture ((2)–(3)). (**b**) Training loss for the six components of the metric tensor. Each component was learned by one specific network.

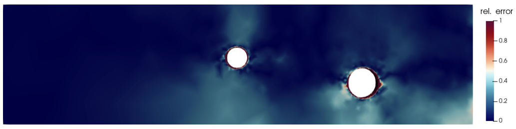
The case of simulation 106 highlights that the velocity field, in addition to being a three-component field, is a more complex learning task than the metric field. As expected, the error is concentrated around the obstacles, where the gradients are higher and the boundary conditions change (Figure 13). The metric field prediction still shows some error, but catches precisely the evolution of this variable across the simulation domain (Figure 14).



(**a**)

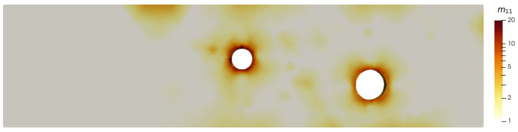


(**b**)

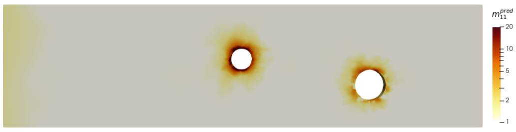


(**c**)

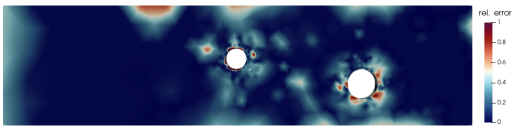
Figure 13. Graphnet prediction of the velocity field on simulation 106 from the test set—x component. (**a**) Ground truth velocity field—x component. (**b**) Predicted velocity field—x component. (**c**) Velocity field relative error map—x component.



(**a**)



(**b**)

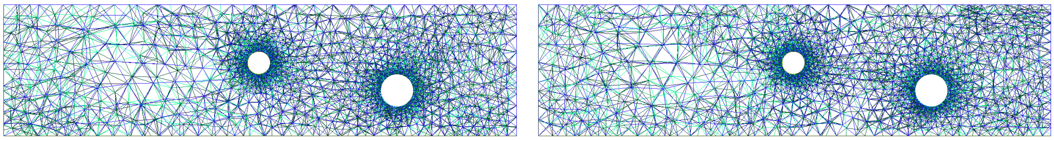


(**c**)

Figure 14. Graphnet prediction of the metric field on simulation 106 from the test set. (**a**) Ground truth metric field—m_{11} component. (**b**) Predicted metric field—m_{11} component. (**c**) Metric relative error map—m_{11} component.

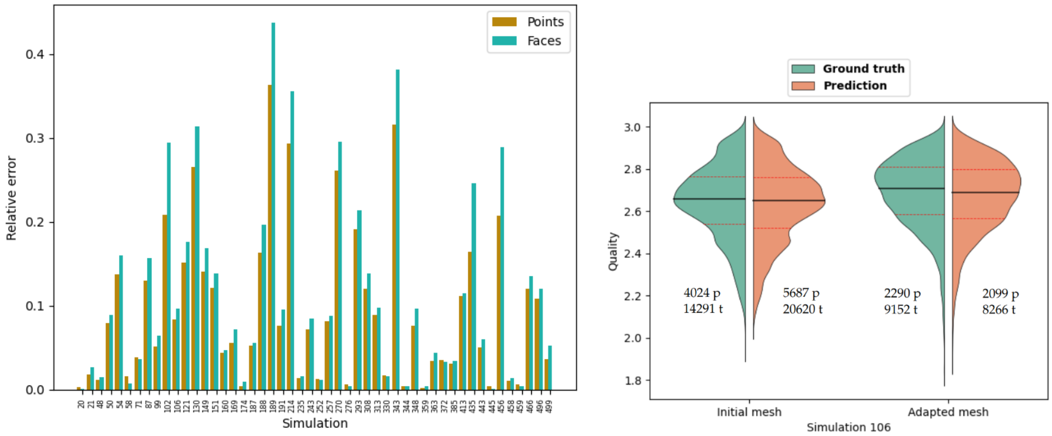
3.2.2. Mesh Adaptation

Ultimately, we are looking for the resulting adapted mesh generated through this metric field. We use *Mmg* software (5.7.0) [33] to realize such an adaptation in 3D. Again, generated meshes show great visual similarity (Figure 15), confirmed by quality analysis (Figure 16b). As expected, the number of points and tetrahedra has been divided by two through the AMR process, both for ground truth and prediction in the case of simulation 106.



(**a**) (**b**)

Figure 15. 3D mesh adaptation using *Mmg* for the ground truth metric field and the one predicted by Graphnet on simulation 106. (**a**) Ground truth mesh adaptation. (**b**) Predicted mesh adaptation.



(**a**) (**b**)

Figure 16. Quality of meshes predicted by Graphnet on simulations from the test set. (**a**) Points and tetrahedra relative error on test set simulations. (**b**) "Glassmaier" parameter distribution for simulations 106 on Meshnet and Graphnet ground truths and predictions (black line: median–red dotted lines: quartiles and p: points–t: tetrahedra).

This result can be extended to the remaining simulations of the test sets. The relative errors in the number of points and tetrahedra (Figure 16a) show great improvement in comparison to Meshnet (Figure 10a), meaning that the gap in points and tetrahedra created by the prediction of the initial mesh has very little influence on the ultimate adapted mesh. Mesh quality is more important, as it is almost conserved through the AMR process (Figure 16b).

3.3. Adaptnet

Adaptnet is the framework combining Meshnet and Graphnet (Figure 1). Given an initial geometry, it will predict initial mesh sizes (Meshnet) to produce an initial mesh and predict on this mesh a metric field to adapt it to the Stokes problem. This process has already been illustrated through the example of the three-hole geometry. First, we showed how Meshnet produces an initial mesh (Figure 17), and then, we use this mesh to predict the anisotropic metric tensor at each point of the simulation domain and adapt the mesh (Figure 18). By predicting directly the metric tensor, we are able to skip the calculation of the velocity field, unlike an FE solver, speeding up calculation time accordingly (Table 2). For Graphnet, we compare the prediction time of the six models (one for each component of the metric tensor) with the computation time of *FreeFem++*, on the three-hole test case, using a single CPU. For Meshnet, the prediction time should be compared with the engineering time needed to choose the mesh parameters.

Table 2. CPU time comparison for computation and prediction of each phase

Model	Meshnet	Graphnet
Computation	Engineer time	50.06 s
Prediction	0.68 s	0.79 s
Speed-up	ND	63.4

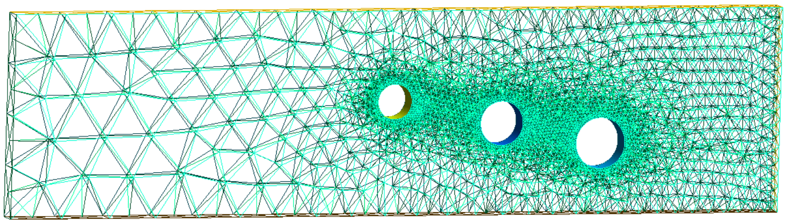
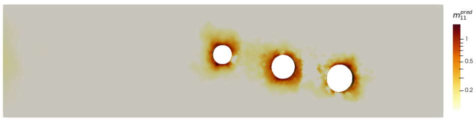
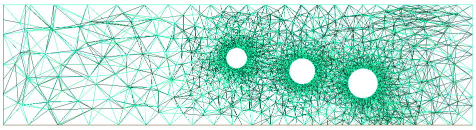


Figure 17. Generalization of the Meshnet prediction for three-hole geometry.



(**a**)



(**b**)

Figure 18. Generalization of the Graphnet prediction for three-hole geometry. (**a**) Metric field prediction. (**b**) Adapted mesh using *Mmg*.

4. Discussion

4.1. Meshnet Generalization

The architectural choice of using relative encoding on graphs has been shown to be very conducive to generalization [34]. Our model generalizes well, as can be seen when we want it to predict the mesh size at each point on a three-hole geometry (Figure 17).

4.2. Graphnet Generalization

The same observation can be made when we want to predict the adaptation metric in a three-hole geometry (Figures 18 and 19).

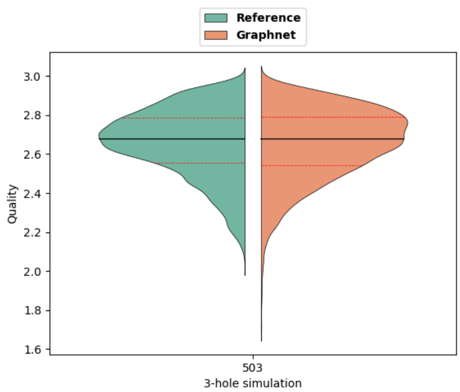
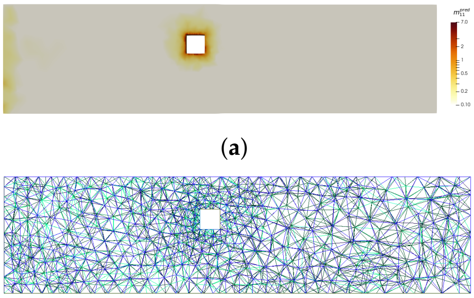


Figure 19. "Glassmaier" parameter distribution (black line: median–red dotted lines: quartiles) for Meshnet and Graphnet predictions for three-hole geometry.

Finally, we tested our model with a different geometry from those seen previously. This time, we chose a square (Figure 20). The point here is not to test our model with every possible shape but simply to highlight its ability to interpolate in a wide spectrum of graph structures (Figure 21).



(**a**)

(**b**)

Figure 20. Generalization of the Graphnet prediction for square geometry. (**a**) Metric field prediction. (**b**) Adapted mesh using *Mmg*.

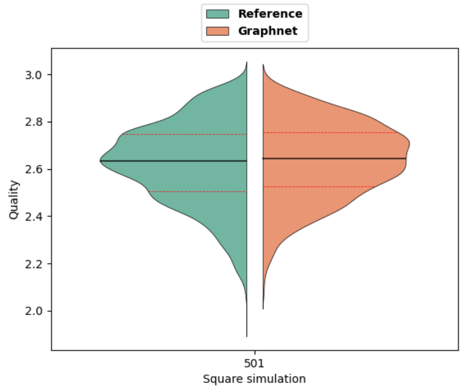


Figure 21. "Glassmaier" parameter distribution (black line: median–red dotted lines: quartiles) for Meshnet and Graphnet predictions for square geometry.

4.3. Linear Elasticity Problem

The attempt to train the graph neural network (GNN) on the linear elasticity problem proved to be less successful than expected (Figure 22), highlighting the challenges inherent in this complex task. Despite meticulous parameter tuning and a comprehensive dataset, the network struggled to effectively capture and generalize the underlying patterns within the graph structures. The intricate interplay of nodes and edges posed a formidable challenge, and the model exhibited difficulties in discerning meaningful relationships and dependencies.

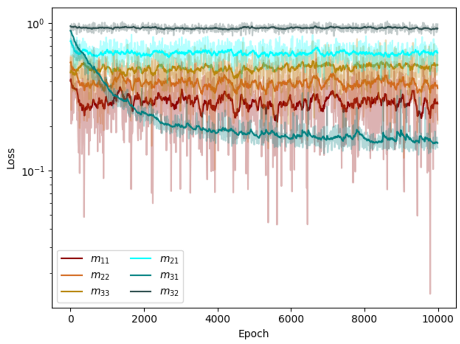
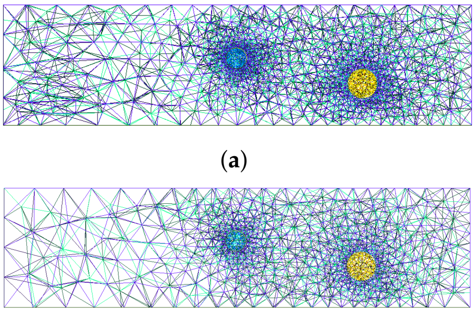


Figure 22. Training loss for the six components of the metric tensor. Each component was learned independently by one specific network.

These results underscore the need for further investigation into refining GNN architectures and training methodologies, emphasizing the intricate nature of graph-based data and the nuances associated with their representation in neural networks.

At first glance, the quality of the mesh produced may seem even better than the reference (Figure 23). However, quantitative analysis highlights the poorer quality of prediction in this case (Figure 24). Still, it can be noticed that the plate is very thin along the z-axis. As a consequence, the tetrahedra generated will necessarily be stretched in this direction, resulting in lower quality. But the overall quality distribution is more satisfactory, as the first visual impression suggested.



(**a**)

(**b**)

Figure 23. Graphnet prediction for three-hole geometry. (**a**) Reference mesh for sample 106. (**b**) Adapted mesh using *Mmg* for sample 106.

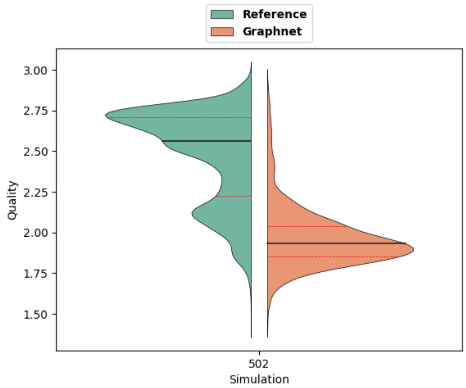


Figure 24. "Glassmaier" parameter distribution (black line: median–red dotted lines: quartiles) for Reference and Graphnet predictions.

5. Conclusions

We propose a Graph Neural Network mesh-based method to model fluid dynamics and solid mechanics problems for an accurate and efficient prediction of anisotropic adaptation metric fields. We extended the approach of Pfaff et al. [29] to tetrahedral meshes in three dimensions. Following the work of Harsch et al. [35], we adapted the GNN architecture to better suit the prediction of steady states. Our method may allow for more efficient simulations than traditional simulators, and because it is differentiable, it could be used to retrieve the Hessian matrix directly, when traditional solvers have to compute a recovered Hessian matrix.

The experiments demonstrate the model's strong understanding of the geometric structure. The method is independent of the structure of the simulation domain, being able to capture highly irregular meshes. We show that the model does not require any *a priori* domain information, e.g., inflow velocity or material parameters. Thus, the model can be used for any other systems represented as field data.

The training phase underlined the model's strong understanding of the geometric structure of the unstructured meshes. It was shown that it can achieve effective prediction with the sole knowledge of connectivity and belonging to a physical surface. It does not rely on any prior information such as the inflow velocity or the material parameters. This can be easily adapted and scaled to other physical problems governed by PDEs. However, our test shows that some specific tuning might be necessary to achieve correct predictions when adapting from a fluid problem to a solid one for instance.

The strength of this work lies in the tetrahedral mesh generation and adaptation pipeline. The emphasis was placed on producing work based on open-source tools [31–33] to promote reproducibility and enable others to build on it. Future work in this area should address several topics. It could explore more complex geometries to prove its applicability to industrial problems. Alternative architectures could also be explored. Adding concatenation and pooling has been shown to improve performance, but other options could show even better improvements. Finally, this framework could be further improved by adding specific physical constraints to the loss function, similar to Physics-Informed Neural Networks (PINNs) [39], which could enable learning on sparse datasets, reducing the need for generating hundreds of simulations.

Author Contributions: Conceptualization, Y.M. and A.P.-F.; methodology, U.P. and Y.M.; software, U.P.; validation, Y.M., A.P.-F. and F.B.; formal analysis, U.P.; investigation, U.P.; resources, Y.M., A.P.-F. and F.B.; data curation, U.P.; writing—original draft preparation, U.P.; writing—review and editing, U.P., Y.M., A.P.-F. and F.B.; visualization, U.P.; supervision, Y.M., A.P.-F. and F.B.; project administration, Y.M. and A.P.-F.; funding acquisition, Y.M. and A.P.-F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data is publicly available at: https://github.com/UgoPelissier/adaptnet accessed on 21 July 2023.

Acknowledgments: This work has been supported by SafranTech.

Conflicts of Interest: The authors declare no conflicts of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AMR	Adaptive Mesh Refinement
ADAM	Adaptive Moment Estimation
BC	Boundary Conditions
CAD	Computer Aid Design
CFD	Computational Fluid Dynamics
CSM	Computational Solid Mechanics
CNN	Convolutional neural network
DL	Deep Learning
FE	Finite Element
GN	Graph Network
IC	Initial Conditions
ML	Machine Learning
NN	Neural Network
PINN	Physics-Informed Neural Networks
RMSE	Root Mean Square Error

Appendix A

Appendix A.1. Linear Elasticity Problem

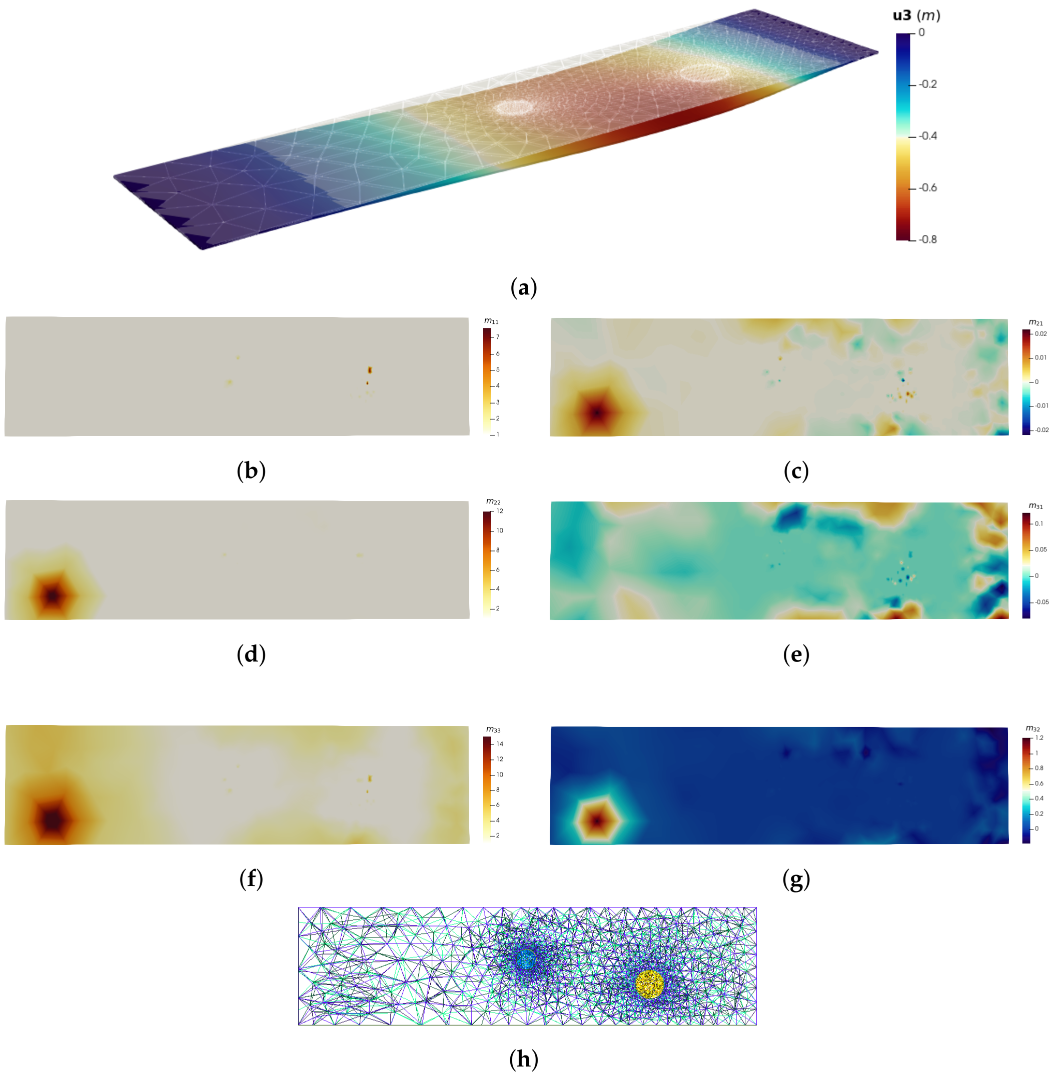


Figure A1. Finite Element solving of Elasticity problem and anisotropic metric computation for sample 106. (**a**) Displacement field solved by *FreeFem++* along $z-$ axis. (**b**) First diagonal component (m_{11}) of the metric tensor computed by *FreeFem++*. (**c**) First non-diagonal component (m_{21}) of the metric tensor. (**d**) Second diagonal component (m_{22}) of the metric tensor. (**e**) Second non-diagonal component (m_{31}) of the metric tensor. (**f**) Third diagonal component (m_{33}) of the metric tensor. (**g**) Third non-diagonal component (m_{32}) of the metric tensor. (**h**) Adapted mesh to Stokes problem.

Appendix A.2. Model

The MLPs of the Encoder, Processor, and Decoder are ReLUactivated two-hidden-layer MLPs with layer and output size of 128, except for the Decoder MLP whose output size matches the prediction. All MLPs outputs except the Decoder one are normalized by a LayerNorm. All input and target features are normalized to zero mean unit variance, using dataset statistics.

Table A1. Details of node and edge encoders.

Node MLP ($\mathbf{mlp}_v$)	Edge MLP ($\mathbf{mlp}_e$)
Input: x	Input: x
$x = \text{Linear}(5/8, 128)(x)$	$x = \text{Linear}(3, 128)(x)$
$x = \text{RELU}(x)$	$x = \text{RELU}(x)$
$x = \text{Linear}(128, 128)(x)$	$x = \text{Linear}(128, 128)(x)$
$x = \text{RELU}(x)$	$x = \text{RELU}(x)$
$x = \text{Linear}(128, 128)(x)$	$x = \text{Linear}(128, 128)(x)$
$x = \text{RELU}(x)$	$x = \text{RELU}(x)$
$x = \text{Linear}(128, 128)(x)$	$x = \text{Linear}(128, 128)(x)$
$x = \text{LayerNorm}(128)(x)$	$x = \text{LayerNorm}(128)(x)$

Table A2. Hyperparameters used for processor.

Parameter Name	Value
Number of GNN layer for the processor	15
Latent size for the processor	128
Activation	ReLu
Type of normalization	Layer normalization
Input feature size of Node MLP	256
Input feature size of Edge MLP	384

References

1. Zhang, Z.; Wang, Y.; Jimack, P.K.; Wang, H. MeshingNet: A New Mesh Generation Method Based on Deep Learning. In Proceedings of the Computational Science—ICCS 2020, Amsterdam, The Netherlands, 3–5 June 2020; Krzhizhanovskaya, V.V., Závodszky, G., Lees, M.H., Dongarra, J.J., Sloot, P.M.A., Brissos, S., Teixeira, J., Eds.; Springer International Publishing: Cham, Switzerland, 2020; pp. 186–198.
2. Economon, T.D.; Palacios, F.; Copeland, S.R.; Lukaczyk, T.W.; Alonso, J.J. SU2: An Open-Source Suite for Multiphysics Simulation and Design. *AIAA J.* **2016**, *54*, 828–846. [CrossRef]
3. Ramamurti, R.; Sandberg, W. Simulation of Flow about Flapping Airfoils Using Finite Element Incompressible Flow Solver. *AIAA J.* **2001**, *39*, 253–260. [CrossRef]
4. Panthi, S.; Ramakrishnan, N.; Pathak, K.; Chouhan, J. An analysis of springback in sheet metal bending using finite element method (FEM). *J. Mater. Process. Technol.* **2007**, *186*, 120–124. [CrossRef]
5. Yazid, A.; Abdelkader, N.; Abdelmadjid, H. A state-of-the-art review of the X-FEM for computational fracture mechanics. *Appl. Math. Model.* **2009**, *33*, 4269–4282. [CrossRef]
6. Shewchuk, J. Delaunay refinement algorithms for triangular mesh generation. *Comput. Geom.* **2002**, *22*, 21–74. [CrossRef]
7. Si, H.; Sadrehaghighi, I. Tetgen—A Delaunay-Based Quality Tetrahedral Mesh Generator. *ACM Trans. Math. Softw.* **2022**, *41*, 11. [CrossRef]
8. Lei, N.; Li, Z.; Xu, Z.; Li, Y.; Gu, X. What's the Situation with Intelligent Mesh Generation: A Survey and Perspectives. *IEEE Trans. Vis. Comput. Graph.* **2024**, *30*, 4997–5017. [CrossRef]
9. Almeida, R.C.; Feijóo, R.A.; Galeao, A.C.; Padra, C.; Silva, R.S. Adaptive finite element computational fluid dynamics using an anisotropic error estimator. *Comput. Methods Appl. Mech. Eng.* **2000**, *182*, 379–400. [CrossRef]
10. Mesri, Y.; Zerguine, W.; Digonnet, H.; Silva, L.; Coupez, T. Dynamic Parallel Adaption for Three Dimensional Unstructured Meshes: Application to Interface Tracking. In Proceedings of the 17th International Meshing Roundtable, Pittsburgh, PA, USA, October 12–15 2008; pp. 195–212. [CrossRef]
11. Mesri, Y.; Khalloufi, M.; Hachem, E. On optimal simplicial 3D meshes for minimizing the Hessian-based errors. *Appl. Numer. Math.* **2016**, *109*, 235–249. [CrossRef]
12. Loseille, A.; Alauzet, F. Continuous mesh framework part I: Well-posed continuous interpolation error. *SIAM J. Numer. Anal.* **2011**, *49*, 38–60. [CrossRef]
13. Agouzal, A.; Vassilevski, Y.V. Minimization of gradient errors of piecewise linear interpolation on simplicial meshes. *Comput. Methods Appl. Mech. Eng.* **2010**, *199*, 2195–2203. [CrossRef]
14. Labbé, P.; Dompierre, J.; Vallet, M.G.; Guibault, F. Verification of three-dimensional anisotropic adaptive processes. *Int. J. Numer. Methods Eng.* **2011**, *88*, 350–369. [CrossRef]
15. Mesri, Y.; Guillard, H.; Coupez, T. Automatic coarsening of three dimensional anisotropic unstructured meshes for multigrid applications. *Appl. Math. Comput.* **2012**, *218*, 10500–10519. [CrossRef]
16. Kamenski, L.; Huang, W. How a nonconvergent recovered Hessian works in mesh adaptation. *SIAM J. Numer. Anal.* **2014**, *52*, 1692–1708. [CrossRef]

17. Ladický, L.; Jeong, S.; Solenthaler, B.; Pollefeys, M.; Gross, M. Data-driven fluid simulations using regression forests. *ACM Trans. Graph.* **2015**, *34*, 1–9. [CrossRef]
18. Wiewel, S.; Becher, M.; Thuerey, N. Latent Space Physics: Towards Learning the Temporal Evolution of Fluid Flow. *Comput. Graph. Forum* **2019**, *38*, 71–82. [CrossRef]
19. Lecun, Y.; Bengio, Y. Convolutional Networks for Images, Speech, and Time-Series. *Handb. Brain Theory Neural Netw.* **1995**, *3361*, 1995.
20. Guo, X.; Li, W.; Iorio, F. Convolutional Neural Networks for Steady Flow Approximation. In Proceedings of the KDD'16: 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, USA, 13–17 August 2016; pp. 481–490. [CrossRef]
21. Zhu, Y.; Zabaras, N. Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification. *J. Comput. Phys.* **2018**, *366*, 415–447. [CrossRef]
22. Jin, X.; Cheng, P.; Chen, W.L.; Li, H. Prediction model of velocity field around circular cylinder over various Reynolds numbers by fusion convolutional neural networks based on pressure on the cylinder. *Phys. Fluids* **2018**, *30*, 047105. [CrossRef]
23. Lee, S.; You, D. Data-driven prediction of unsteady flow over a circular cylinder using deep learning. *J. Fluid Mech.* **2019**, *879*, 217–254. [CrossRef]
24. Thuerey, N.; Weißenow, K.; Prantl, L.; Hu, X. Deep Learning Methods for Reynolds-Averaged Navier–Stokes Simulations of Airfoil Flows. *AIAA J.* **2020**, *58*, 25–36. [CrossRef]
25. Patil, A.; Viquerat, J.; Larcher, A.; El Haber, G.; Hachem, E. Robust deep learning for emulating turbulent viscosities. *Phys. Fluids* **2021**, *33*. [CrossRef]
26. Scarselli, F.; Gori, M.; Tsoi, A.C.; Hagenbuchner, M.; Monfardini, G. The Graph Neural Network Model. *IEEE Trans. Neural Netw.* **2009**, *20*, 61–80. [CrossRef]
27. Gilmer, J.; Schoenholz, S.S.; Riley, P.F.; Vinyals, O.; Dahl, G.E. Neural message passing for Quantum chemistry. In Proceedings of the 34th International Conference on Machine Learning (ICML'17), Sydney, NSW, Australia, 6–11 August 2017; pp. 1263–1272.
28. Battaglia, P.W.; Hamrick, J.B.; Bapst, V.; Sanchez-Gonzalez, A.; Zambaldi, V.; Malinowski, M.; Tacchetti, A.; Raposo, D.; Santoro, A.; Faulkner, R.; et al. Relational inductive biases, deep learning, and graph networks. *arXiv* **2018**, arXiv:cs.LG/1806.01261.
29. Pfaff, T.; Fortunato, M.; Sanchez-Gonzalez, A.; Battaglia, P.W. Learning Mesh-Based Simulation with Graph Networks. In Proceedings of the International Conference on Learning Representations, Vienna, Austria, 3–7 May 2021.
30. Ju, X.; Hamon, F.P.; Wen, G.; Kanfar, R.; Araya-Polo, M.; Tchelepi, H.A. Learning CO_2 plume migration in faulted reservoirs with Graph Neural Networks. *arXiv* **2023**, arXiv:cs.LG/2306.09648. [CrossRef]
31. Geuzaine, C.; Remacle, J.F. Gmsh: A 3-D finite element mesh generator with built-in pre- and post-processing facilities. *Int. J. Numer. Methods Eng.* **2009**, *79*, 1309–1331. [CrossRef]
32. Hecht, F. New development in FreeFem++. *J. Numer. Math.* **2012**, *20*, 251–265. [CrossRef]
33. Dobrzynski, C. *MMG3D: User Guide*; Technical Report RT-0422; INRIA: Paris, France, 2012.
34. Sanchez-Gonzalez, A.; Godwin, J.; Pfaff, T.; Ying, R.; Leskovec, J.; Battaglia, P.W. Learning to simulate complex physics with graph networks. In Proceedings of the 37th International Conference on Machine Learning (ICML'20), Virtual Event, 13–18 July 2020.
35. Harsch, L.; Riedelbauch, S. Direct Prediction of Steady-State Flow Fields in Meshed Domain with Graph Networks. *arXiv* **2021**, arXiv:physics.flu-dyn/2105.02575.
36. Wang, Y.; Sun, Y.; Liu, Z.; Sarma, S.E.; Bronstein, M.M.; Solomon, J.M. Dynamic Graph CNN for Learning on Point Clouds. *ACM Trans. Graph.* **2019**, *38*, 1–12. [CrossRef]
37. vom Stein, R.; Glassmeier, K.H.; Dunlop, M. *A Configuration Parameter for the Cluster Satellites*; Institut für Geophysik und Meteologie der Technischen Universität Braunschweig: Braunschweig, Germany, 1992.
38. Daly, P. Tetrahedron Quality Factors. Available online: https://www2.mps.mpg.de/homes/daly/CSDS/t4h/tetra.htm (accessed on 5 September 2023).
39. Raissi, M.; Perdikaris, P.; Karniadakis, G.E. 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. [CrossRef]