POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Deep Learning-based surrogate models for parametrized PDEs: including geometrical features through graph neural networks

Laurea Magistrale in Mathematical Engineering - Ingegneria Matematica

Author: Filippo Tombari

Advisor: Prof. Andrea Manzoni

Co-advisors: Dr. Nicola Rares Franco, Dr. Stefania Fresca

Academic year: 2021-2022

1. Introduction

In many areas of Science and Engineering, models governed by Partial Differential Equations (PDEs) which may depend on one or more parameters are ubiquitous. These parameters may be related to either the physical properties or the geometry of the domain which we consider. Modeling this kind of problem typically requires the introduction of a suitable mesh as first basic ingredient, to discretize the computational domain and to define the set of approximating functions required to represent the problem solution, ultimately allowing the search for highly accurate approximations. In this framework, Full Order Models (FOMs), which rely on numerical schemes such as the Finite Element Method (FEM), Isogeometric Analysis, or the Spectral Element Method (SEM) usually offer high levels of accuracy, however featuring very high computational costs, that become infeasible in many applications where the solver has to be called multiple times. For this reason, there is a growing literature aimed at replacing these expensive models with suitable cheaper models, also known as surrogate or Reduced Order Models (ROMs), which usually offer a very good trade-off between the computational cost and the accuracy of the simulation. Deep learning methods, like Feed Forward Neural Networks (FFNNs) and Convolutional Neural Networks (CNNs), are widely used for building efficient surrogate models to solve mesh-based problems and they work well in modeling time-dependent PDEs. On the other hand, these architectures cannot infer anything about the geometry of the problem because their structure is strictly dependent on the employed mesh. Moreover, they do not leverage the features that characterize the mesh and its structure. For instance, when dealing with complex domains such as 3D domains and unstructured meshes, exploiting the mesh connectivity can provide a better understanding of the geometrical features of the problem. For this reason, we introduce a surrogate model based on Graph Neural Networks (GNNs), particular neural nets which do not require information about the number of nodes or the number of edges of the mesh, potentially leading to a much more flexible approach that can accommodate different geometries at the same time. This work explores the advantages of using GNN-like architecture to model physical problems involving time-dependent PDEs that also depend on physical and geometrical parameters. The model ex-

ploits the graph representation of the mesh and the GNN capability of inferring its geometrical structure. It is based on the ones implemented by Pfaff et Al. [4] and Hernández et Al. [3], but focuses on different examples, in which we can highlight the inductive capability of this approach. Moreover, we will see also how this approach significantly outperforms the models based on FFNNs and CNNs.

2. Methodology

For the sake of generality, despite the numerical experiments referring only to linear problems, we consider a generic nonlinear, time-dependent PDE, since the results can also be extended to more general problems. In particular, we consider a PDE depending on a set of input parameters $\boldsymbol{\mu} \in \mathcal{P}$, where the parameter space $\mathcal{P} \subset \mathbb{R}^{n_\mu}$ is given by a bounded and closed set; in our analysis input parameters may represent physical or geometrical properties of the system. In this work, we use a deep learning model based on graph neural networks. These deep neural networks allow us to find an approximate solution $\tilde{\mathbf{u}}(t, \boldsymbol{\mu})$ of a FOM solution of a PDE $\mathbf{u}_h(t, \boldsymbol{\mu}) \in \mathbb{R}^{N_h(\boldsymbol{\mu})}$, where $h > 0$ denotes a discretization parameter, such as the maximum diameter of elements in a computational mesh. The dimension $N_h(\boldsymbol{\mu})$ is related to the finite dimensional subspaces introduced for the sake of space discretization depends on $\boldsymbol{\mu}$ since changing some geometrical parameters may change the number of nodes in the computational mesh. Note that $\tilde{\mathbf{u}}$ is not strictly dependent on the space discretization parameter h of the computational mesh so our model is more flexible in accommodating different geometries at the same time.
More precisely, we aim at introducing a suitable deep neural network that can learn the map $\boldsymbol{\Phi}$ such that:

$$\begin{cases} \tilde{\mathbf{u}}^{n+1}(\boldsymbol{\mu}) = \boldsymbol{\Phi}(\mathbf{u}^n(\boldsymbol{\mu}); \boldsymbol{\mu}) \quad n = 0, \dots, N-1, \\ \tilde{\mathbf{u}}^0(\boldsymbol{\mu}) = \mathbf{u}_h^0(\boldsymbol{\mu}). \end{cases} \tag{1}$$

GNNs adopt a *graph-in, graph-out* architecture meaning that these model types accept a graph as input, with information loaded into its nodes and edges and progressively transform these embeddings, without changing the connectivity of the input graph.
The fundamental ingredients of these architectures are the so called *message passing* operations, which enable the aggregation of node information while leveraging the depth of the graph.
we can introduce a suitable graph representation of the mesh by considering an undirected graph $G = (V, E)$, where $V = (v_i)_{i=1}^N$ is the set of vertices of the mesh and $E = (e_{ij})_{i,j=1}^N$ is the set of edges. Unlike other types of graph data, such as social networks or citation graphs, computational meshes have a Euclidean structure, which means that every node i can be associated with its space coordinates $\mathbf{x}_i$. Often, every node is also associated with some features $\mathbf{v}_i$, which are also referred to as state variables. Since the solutions to PDEs, once an appropriate discretization is introduced, are effectively data defined on graphs (the computational mesh), we believe that these architectures can be useful in constructing surrogate models. The model proposed is the Encode-Process-Decode architecture shown in Figure 1, which first consists of the input encoding, then it performs M message passing steps through the Processor and ultimately combines with a call to the Decoder, which outputs the model prediction. In order to understand better the model, we introduce some notation:

- $\mathbf{v}_i \in \mathbb{R}^{d_{in}}$ are the features associated to each node i, such as the solution of the system $\tilde{u}_i$ evaluated at i;
- $\mathbf{e}_{ij} \in \mathbb{R}^{e_{in}}$ are the features associated to each edge (i, j);
- $E_v : \mathbb{R}^{d_{in}} \to \mathbb{R}^l$, where $l > d_{in}$ is the latent dimension of the network, is the nodes features encoder;
- $E_e : \mathbb{R}^{e_{in}} \to \mathbb{R}^l$, where $l > e_{in}$, is the edges features encoder;
- $\mathcal{E} : \mathbb{R}^{3l} \to \mathbb{R}^l$ is the edges features processor;
- $\mathcal{N} : \mathbb{R}^{2l} \to \mathbb{R}^l$ is the nodes features processor;
- $\mathcal{D} : \mathbb{R}^l \to \mathbb{R}^{d_{out}}$, where d_{out} is the dimension of the output, is the decoder map.

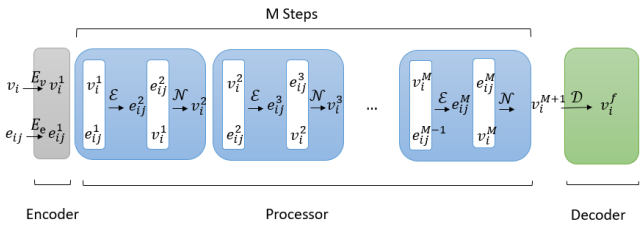

Figure 1: The Encode-Process-Decode model.

The functions $E_v, E_e, \mathcal{E}, \mathcal{N}, \mathcal{D}$ are modelled with Multi-Layer Perceptron (MLP) architectures.
In our case, the goal is to model the function $\mathbf{\Phi}$ in Equation (1) through the outcome F of the decoder, in order to make predictions of the approximate solution of the system $\tilde{\mathbf{u}}$ at time t^{n+1} given the state of the system at time t^n. This means that for each $n = 0, \dots, N-1$, we compute:

$$\begin{aligned} \tilde{\mathbf{u}}^{n+1}(\boldsymbol{\mu}) &= \tilde{\mathbf{u}}^{n}(\boldsymbol{\mu}) + \Delta t F(\tilde{\mathbf{u}}^n; \boldsymbol{\mu}) \\ \tilde{\mathbf{u}}^{0}(\boldsymbol{\mu}) &= \mathbf{u}_h^0(\boldsymbol{\mu}) \end{aligned} \tag{2}$$

Notice that, following this notation, the outcome of the network F should approximate the temporal derivative of the solution at time t^n.
Despite the test predictions being done by exploiting the simulation rollout, model training is performed by one-step prediction, that is at each time step t^n we compute $F(\mathbf{u}_h^n)$ by passing to the network as input the FOM solution $\mathbf{u}_h^n$. This preserves memory usage from recursive training and it also allows parallelization through batches, exploiting the power of GPUs with tensor calculus.
We train the model by minimizing for each batch of time instants a loss computed as the weighted sum between two terms: (i) the MSE between the derivative of the FOM solutions $\dot{\mathbf{u}}_h^n$ estimated via finite differences and the model outcomes $F(\mathbf{u}_h^n)$, with $t^1, \dots, t^{N_{batch}}$; (ii) the MSE between the FOM solutions $\mathbf{u}_h^n$ and the Forward Euler predictions $\tilde{\mathbf{u}}^{n+1}(\boldsymbol{\mu}) = \mathbf{u}_h^n(\boldsymbol{\mu}) + \Delta t F(\mathbf{u}_h^n; \boldsymbol{\mu})$. Instead, the prediction error is computed as the relative MSE (RMSE) error between the network prediction and the ground truth solution:

$$RMSE(\tilde{\mathbf{u}}, \mathbf{u}_h) = \frac{1}{N} \sum_{n=0}^{N} \frac{\sum_{i=0}^{N_h-1} (\tilde{u}_i^n - u_{h,i}^n)^2}{\sum_{j=0}^{N_h-1} (u_{h,i}^n)^2}$$

3. Test Case 1: Advection-Diffusion problem

$$\begin{cases} \dfrac{\partial u}{\partial t} - D\Delta u + \mathbf{b} \cdot \nabla u = 0 \text{ in } \Omega \times (0, T] \\ u(x, y) = (x-1)^2 + (y-1)^2 \text{ on } \partial\Omega \times (0, T] \\ u_0(x, y) = (x-1)^2 + (y-1)^2 \text{ in } \Omega \end{cases} \tag{3}$$

where $\Omega = (0,1)^2 \setminus C$, with $C = \{(x, y) : (x - c_x)^2 + (y - c_y)^2 \leq (0.15)^2\}$. In our simulations, we parametrize the center of the circular obstacle as $\boldsymbol{\mu} = (c_x, c_y) \in \mathcal{P}$ where $\mathcal{P} = \{(x, y) : 0 < x < 1, \ y \geq 0.5\}$. We set also $T = 2$, $D = 0.1$ and $\mathbf{b} = [1 - t, 1 - t]$ for $t \in [0, 2]$. The generated dataset is composed of 100 simulations, each obtained from a different position of the center of the obsatcle so the number of mesh nodes varies from 770 to 790. The time step chosen is $\Delta t = 0.02$, resulting in 101 time snapshots for each simulation.
In this example, the node input is a $N_{batch} - 1 \times N_h \times 3$ tensor, where N_{batch} is a hyperparameter which is set equal to 25, N_h varies across the simulations and the 3 features for each node i are the ground truth solution $u_{h,i}^n$, the time instant t^n and a binary variable with value 1 if the corresponding node is on the boundary and 0 otherwise.
Another input information to provide is the edge attributes tensor. This is a $N_{batch} - 1 \times N_{edges} \times 3$ tensor containing for each edge, the coordinates of its nodes and its length calculated as the Euclidean norm of the distance of the coordinates.
The output, instead, is a $N_{batch} - 1 \times N_h \times 1$ tensor which represents the temporal derivative of the solution evaluated at time t^n.
The model has been trained on 80 simulations chosen randomly and tested on the remaining 20. The results of the rollout predictions of the test simulations are summarized in Table 1.
As we can see all the predictions RMSEs are of order 10^{-3} or 10^{-4}. Moreover, our model outperforms significantly the ground truth solver in the simulation time.

Test case 1. Advection-Diffusion problem

	RMSE (mean)	RMSE (max)	RMSE (min)	t_{gt}(s)	t_{pred}(s)
AD 1	1.2×10^{-3}	6.1×10^{-3}	4×10^{-4}	≈ 159.8	≈ 9.83

Table 1: Test case 1. Results of the test set predictions.

In Figure 2 an example of test set prediction is shown together with the plot of its errors. The dynamic of the problem is well predicted and no propagation errors are spotted. Hence our model and the training strategy adopted seem to be in principle good at solving problems concerning evolutionary PDEs in which multiple rollout time steps need to be predicted.
The RMSE plot shows that, although the errors are larger on the boundary in the first rollout phase ($T < 1.00$), in the end they are higher around the obstacle, where the change of direction of the advection vector $\mathbf{b}$ makes the prediction of the solution more difficult. This happens because GNNs follow a *local-to-global* generalization approach, first processing information locally through the Encoder and then aggregating what has been processed from the neighborhood to connect the mesh nodes. However, this approach may be difficult for some trajectories and result in worse predictions at critical points of the mesh, such as along the boundary.

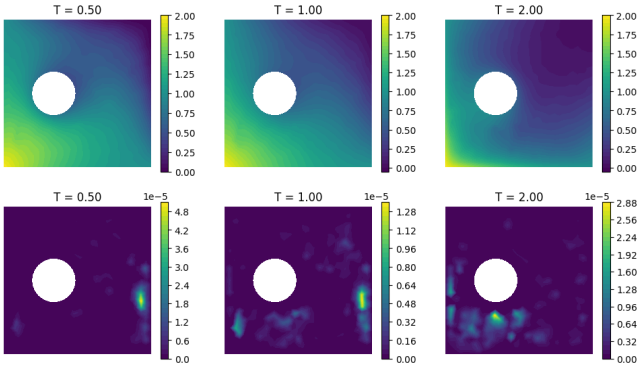

Figure 2: Test case 1, prediction obtained for $\boldsymbol{\mu} = (0.29, 0.5)$ with the obstacle close to the source. First row: rollout prediction. Second row: RMSE related to each time step between the prediction and the corresponding ground truth solution.

This advection-diffusion example can also be generalized to domains in which both the position and the dimension of the obstacle change. We slightly modify our training dataset by adding new simulations in which the obstacle has both a smaller and a higher radius than before. Hence, the geometrical parameter now is $\boldsymbol{\mu} = (c_x, c_y, r) \in \mathcal{P} = \{(x, y) : \ 0 < x < 1, \ y \geq 0.5\} \times \{0.1, 0.15, 0.2\}$. This will improve the generalization of the model on new simulations as we can see from Figure 3.

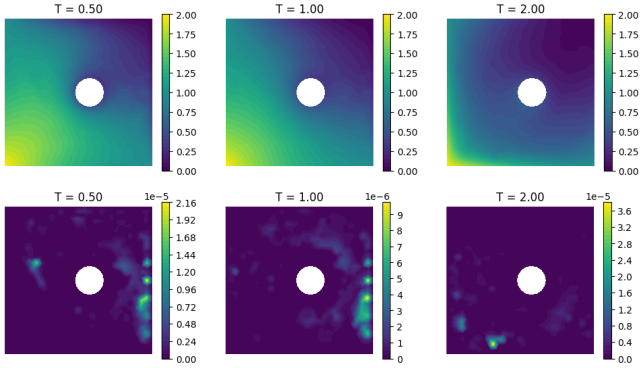

Figure 3: Test case 1, Prediction obtained for $\boldsymbol{\mu} = (0.6, 0.52, 0.1)$. First row: rollout prediction. Second row: ground truth solution.

The model can generalize well on this problem also if the geometries differ a lot from each other. Indeed, an important question that arises is whether our model can predict simulations where the obstacle has a different shape, without requiring retraining of the network. To investigate this, we present an example in Figure 4 of a prediction with a square obstacle located in the top right of the domain. Surprisingly, the errors, in this case, are of the same order of magnitude as those discussed earlier, and the prediction of the overall dynamics is remarkably accurate.

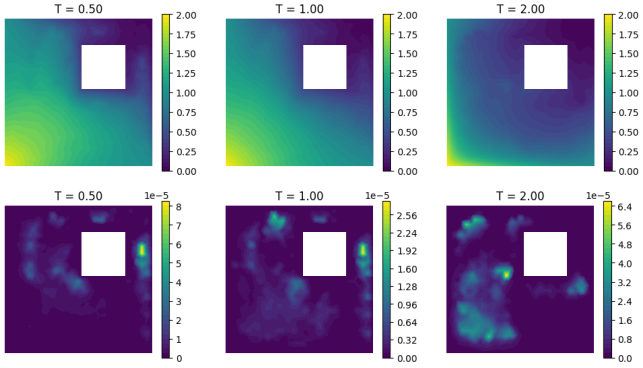

Figure 4: Test case 1, Prediction obtained for $\boldsymbol{\mu} = (0.7, 0.7, 0.3)$ with a square obstacle with edge $l = 0.3$. First row: rollout prediction. Second row: RMSE related to each timestep between the prediction and the corresponding ground truth solution.

4. Test case 2: Advection-Diffusion problem in a 2D Stokes flow

We want to solve an advection-diffusion problem like (3), but now the advection coefficient $\mathbf{b}$ is the velocity field obtained by the solution of a

Stokes problem.
The domain Ω is the rectangle $(0,1) \times (0,0.5)$ with a bump in the upper edge. Here $\Gamma_{in} = \{x = 0\}$, $\Gamma_D = \{y = 0\} \cup \{y = 0.5\}$ and $\Gamma_N = \{x = 1\}$. During our simulations, we shift the position of the bump in a way that its center c_x varies from 0.35 to 0.65. Hence, we have $\mu \in \mathcal{P} = [0.35, 0.65]$. At the inflow Γ_{in}, we impose the Dirichlet boundary condition $u_{in}(x,y) = \frac{4y(0.5-y)}{0.5^2}$, which is also the initial condition, on Γ_D we impose no-slip boundary condition and on Γ_N we set $\frac{\partial u}{\partial n} = 0$. The final simulation time is $T = 0.5$ and $D = 0.01$.
Our dataset is composed of 125 simulations, each obtained from a different position of the bump so the number of mesh nodes varies from 937 to 1042. The time step chosen is $\Delta t = 0.01$, resulting in 51 trajectories for each simulation. The training set is composed of 100 simulations while the test set has 25 simulations, both chosen randomly. We consider the same node and edge input features of the previous example.
The results of the rollout predictions of the test simulations are summarized in Table 2.
In this more complex problem the RMSEs are higher than the ones in the previous example, but we still outperform the ground truth solver in time complexity.

Test case 2. Advection-Diffusion problem in a 2D Stokes flow

	RMSE (mean)	RMSE (max)	RMSE (min)	t_{gt}(s)	t_{pred}(s)
AD 2	1.64×10^{-2}	7.35×10^{-2}	1.2×10^{-3}	≈ 115.65	≈ 7.51

Table 2: Test case 2. Results of the test set predictions.

The model performs well in predicting simulations in which the bump has different positions, as shown in Figure 5. However, some numerical errors can be observed in nodes close to the inflow, which are generally kept under control and remain of the order 10^{-5} as shown in the plot. The behavior of the nodes around the inflow has a significant impact on the accuracy of the simulation, as any errors that arise in this region can propagate throughout the domain, particularly in correspondence with the bump.

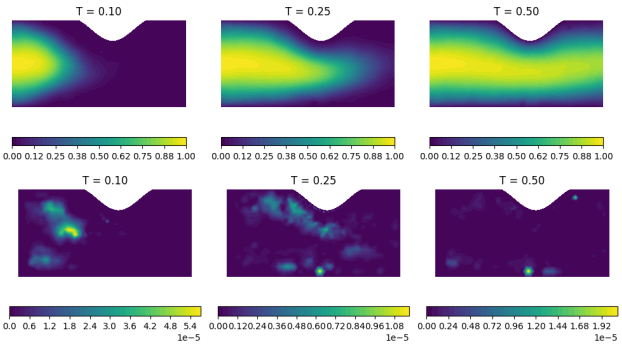

Figure 5: Test case 2, prediction obtained for $\mu = 0.58$ with the bump on the right part of the upper edge. First row: rollout prediction. Second row: RMSE related to each timestep between the prediction and the corresponding ground truth solution

This example can be generalized by letting the bump vary also along the lower edge and also letting his dimension change. Hence we consider a new dataset composed of 185 simulations in which the height of the bump might be in the set $h = \{0.08, 0.12, 0.175\}$ and its center can vary along both the upper and lower edge in the interval $[0.4, 0.6]$. In this way, we keep it sufficiently far from both the inflow and the outflow since we have previously seen that this may imply numerical errors in the GNN prediction. Hence, the geometrical parameter is $\boldsymbol{\mu} = (c_x, c_y, h) \in \mathcal{P} = [0.4, 0.6] \times \{0., 0.5\} \times \{0.08, 0.12, 0.175\}$.
In Figure 6 The height of the bump influences a lot the dynamic of the system, however, the network correctly infers the behavior of the flow around the obstacle.

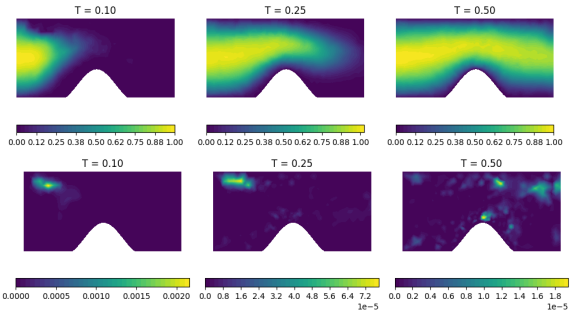

Figure 6: Test case 2, prediction obtained for $\boldsymbol{\mu} = (0.453, 0, 0.175)$ with the bump in the lower edge with center at $x = 0.453$ and height $h = 0.175$. First row: rollout prediction. Second row: RMSE related to each timestep between the prediction and the corresponding ground truth solution.

5. Comparison with FFNN

Feed Forward Neural Networks are usually used for building reduced-order models because they have the capability to capture strong nonlinearity through their fully connected structure.
Hence, we compare the performance of a common FFNN to our GNN on the two examples discussed in the previous sections. In particular, to make things simpler for FFNN we consider for each problem the following datasets:

- for the first Advection-Diffusion problem we let the obstacle vary only in its position, resulting in 100 simulations, among which we choose randomly 80 simulations for the training set and 20 for the test set;
- For the Advection-Diffusion problem in a 2D Stokes flow we let the bump vary in its position along the upper and lower edges but not in its height, resulting in 125 simulations, among which we choose randomly 100 simulations for the training set and 25 for the test set.

In order to train an FFNN, we need to bring all the simulations to the same degrees of freedom, so we interpolate the region of interest a modeling 128×128 vertices grid so that the order of the nodes in the mesh is preserved.
The prediction is done by exploiting the rollout of the simulation as shown in (2) in order to compare the robustness to propagation errors of the two models. In Figure 7 we can clearly see that the predictions done with GNN have less variance than the ones done with FFNN. Even if the range of the errors is similar, the GNN errors are overall better despite some anomalies.

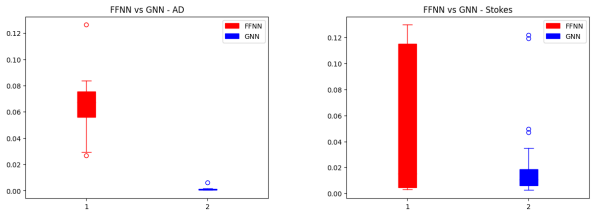

Figure 7: Boxplots of the RMSEs of the two predictions.

Another key aspect to analyze is the number of parameters of the model. The parameters of an FFNN may increase fast due to its fully connected structure. This implies a higher tendency to overfit, as we can see from Figure 8, and moreover, it gets the model less scalable as the complexity of the problem increases.

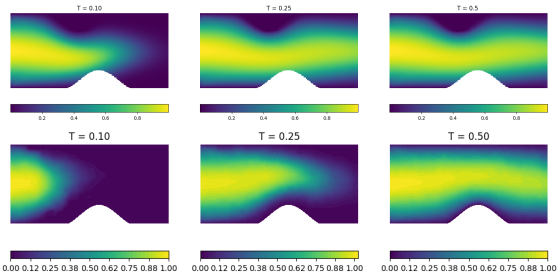

Figure 8: Test case 2, Comparison between FFNN and GNN prediction for $\boldsymbol{\mu} = (c_x, c_y) = (0.6, 0)$. First row: FFNN prediction. Second row: GNN prediction.

6. Conclusions

Graph Neural Networks have been designed to perform *inductive* inference of the geometric structure of a given graph-structured problem, such as a mesh-based simulation of a problem governed by PDEs.
In contrast to Feedforward Neural Networks (FFNNs) and Convolutional Neural Networks (CNNs), GNNs can naturally handle problems with varying geometrical parameters since they intrinsically incorporate mesh features. Moreover, since they are independent of the input degrees of freedom (dofs) of the mesh, they can generalize to different mesh structures.
GNNs exhibit a lower tendency to overfit than FFNNs because the learned map $\boldsymbol{\Phi}$ is the same for each mesh node by definition. Additionally, the robustness of this model to propagation errors during rollout prediction is another major advantage.
A limitation of using Graph Neural Networks is the computational complexity associated with simulating over fine meshes. This can lead to a higher number of message-passing steps, resulting in an increased computational cost and reduced accuracy due to inefficient propagation.
Hence, in future research, it may be advantageous to explore the potential of combining well-established deep learning-based reduced order models, such as the ones introduced by Franco et Al. [1] and Fresca et Al. [2], with graph representations. This hybrid approach could potentially lead to improved accuracy in

predicting simulations, even on more refined meshes, while simultaneously reducing computational complexity.

References

[1] Nicola Franco, Andrea Manzoni, and Paolo Zunino. A deep learning approach to reduced order modelling of parameter dependent partial differential equations. *Mathematics of Computation*, 92(340):483–524, 2023.

[2] Stefania Fresca, Luca Dede', and Andrea Manzoni. A comprehensive deep learning-based approach to reduced order modeling of nonlinear time-dependent parametrized pdes. *Journal of Scientific Computing*, 87:1–36, 2021.

[3] Quercus Hernández, Alberto Badías, Francisco Chinesta, and Elías Cueto. Thermodynamics-informed graph neural networks. *arXiv preprint arXiv:2203.01874*, 2022.

[4] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.