scientific reports

Check for updates

OPEN

Mesh-based GNN surrogates for time-independent PDEs

Rini Jasmine Gladstone[1], Helia Rahmani[2], Vishvas Suryakumar[2], Hadi Meidani[1], Marta D'Elia[3]✉ & Ahmad Zareei[2]

Physics-based deep learning frameworks have shown to be effective in accurately modeling the dynamics of complex physical systems with generalization capability across problem inputs. However, time-independent problems pose the challenge of requiring long-range exchange of information across the computational domain for obtaining accurate predictions. In the context of graph neural networks (GNNs), this calls for deeper networks, which, in turn, may compromise or slow down the training process. In this work, we present two GNN architectures to overcome this challenge—the edge augmented GNN and the multi-GNN. We show that both these networks perform significantly better than baseline methods, such as MeshGraphNets, when applied to time-independent solid mechanics problems. Furthermore, the proposed architectures generalize well to unseen domains, boundary conditions, and materials. Here, the treatment of variable domains is facilitated by a novel coordinate transformation that enables rotation and translation invariance. By broadening the range of problems that neural operators based on graph neural networks can tackle, this paper provides the groundwork for their application to complex scientific and industrial settings.

Numerical models for solving partial differential equations (PDEs) are crucial to scientific modeling in a broad range of fields, including physics, biology, material science, and finance. While various techniques, such as finite element methods, finite difference methods, finite volume methods, etc., have been developed as high-fidelity solvers, accelerating and reducing their computational cost remains a challenge. Additionally, when the governing PDEs are unknown, predicting a physical system using classical techniques is not possible even when observations are available. Recent developments in deep learning have enabled faster and accurate algorithms to evaluate the response of a physical system by exploiting the governing equations, observational data, or both[1,2].

Neural networks trained by adding the governing PDEs and boundary conditions to the loss function, called physics informed neural networks (PINNs), are a prominent example of using deep-learning-based surrogate models to learn solutions of physical system[3–7]. Here, the neural network is used as an approximate representation of the solution of the PDE. While these surrogates are useful to find the specific solution of a PDE with a given set of parameters, a slight modification of such parameters, boundary conditions, or domain geometry requires re-training, making them less attractive in settings such as optimization, design, and uncertainty quantification.

Another class of neural network-based solvers is convolutional neural networks (CNNs) which uses snapshots of observed data over a discretized domain to predict the physical solution. While such data-driven methods do not require a priori knowledge of the governing PDE, they are often limited to the specific domain discretization and cannot be easily generalized to other domain geometries. CNN-based methods include PDE-inspired architectures[8], autoregressive dense encoder–decoder networks[9], and symbolic multi-layer neural networks[10].

In another class of data-driven, deep-learning-based surrogates, neural networks are used to learn a discretization that is then used in classical solvers[11–14]. Specifically, the neural network learns a field used for interpolation at coarse scales. Although these methods have been shown to improve the accuracy and further accelerate traditional solvers, it should be noted that they are still limited by the initial discretization.

With the purpose of obtaining resolution invariance, a class of neural network-based PDE solvers focuses on continuous graph representation of the computational domain[15–19]. In such representations, the continuous nature of the network update guarantees the invariance of the architecture with respect to the resolution of the data and the enhanced interaction among nodes, typical of graph-based networks, improves the accuracy of the architecture for complex physical systems. Related works have shown that the location of the graph nodes can be optimized to better learn the solution at different levels of precision[20]. Furthermore, some of these methods and their extensions[21,22], focus on learning a continuous mapping between input and output of a PDE and are referred to as *neural operators*. These and other neural operators, such as DeepONets[23], have shown significant success in accurately predicting the solution of several PDEs and they generalize relatively well with respect

[1]Civil and Environmental Engineering, University of Illinois Urbana-Champaign, Champaign, IL, USA. [2]Meta Reality Labs, Redmond, WA, USA. [3]Pasteur Labs, Brooklyn, NY, USA. ✉email: marta.delia@simulation.science

to the PDE input parameters. We mention that one advantage of using graph-based representations is that the system's dynamics can be recovered with sparse representation[18,21,24].

Another class of networks that exploits graph-based representation is given by MeshGraphNet and its extensions. Firstly introduced in[25] for time-dependent problems and then extended to a multi-scale setting in[26], these mesh-based GNNs encode, in a connected graph, the mesh information and the corresponding physical parameters such as loading and boundary conditions, and model parameters. In addition to mesh-based approaches, it is worth mentioning that particle-based methods and their graph representation have also been been successfully used in physics simulations[27]. Here, the physical system is described using particles in a reduced-order-model manner; the particles are then identified as nodes of a graph and a message passing neural network is used to learn and compute the dynamics of the particles. Both these graph-based approaches can accurately approximate physics simulation and generalize well to different resolutions and boundary conditions.

In this work we explore mesh-based GNN architectures; the latter exploit local (or short-range) interactions between neighboring nodes to estimate the response. When the nodes of a GNN are associated to a PDE computational domain, interactions between nodes can be interpreted as spatial interactions in the PDE domain. As most physical systems involve local interactions, GNN-based surrogates are among the best candidates to serve as effective simulator engines. However, when a physical system requires long-range interactions, such as in static solid mechanics problems, standard mesh-based, GNN approaches typically fail at capturing the physics accurately. This happens because in general the exchange of information between distant nodes requires a large number of message passing steps. This is unfavorable because of the poor scaling behaviors of GNNs with respect to the number of layers. In the context of graph-based neural operators, this problem has been addressed by using multi-resolution graphs to allow for faster propagation of information[16].

Here, we build on MeshgraphNets and propose two GNN architectures that overcome the challenge of long-range message passing without using a deep network. We introduce the Edge Augmented Graph Neural Network (EA-GNN) and the multi-graph neural network (M-GNN). In EA-GNN, we introduce "virtual" edges that yield faster information propagation resulting in better computational efficiency. In M-GNN we instead pursue a multi-resolution approach, inspired by the multi-grid method and based on the Graph U-Net architecture[28]. Contrary to the work proposed in[26] we obtain the low resolution graph by removing nodes from the original mesh and by adding new edges to second or third order neighbors. In this process, the nodes keep their original coordinate attributes and the interactions with higher order neighbors are gradually added to the graph. Furthermore, in order to make GNNs generalizable to different geometries, we introduce an invariant simulation coordinate space by moving the physical systems to a simulation space that is invariant to translation and rotation (see[29] for a technique that enables the same properties in the context of graph-based neural operators). This process helps faster training of GNNs and allows for generalization to new geometries.

Our major contributions include:

- A novel coordinate transformation method to make the proposed GNN invariant to rotation and translation, and generalizable to new domain geometries.
- An edge augmentation strategy that accelerates message passing across the graph, enabling more efficient training for problems involving long-range interactions.
- A multi-graph approach, where information is passed through different resolution graphs in a hierarchical manner, resulting in faster propagation of messages across domain.

With the purpose of stressing similarities and advantages with respect to the literature we next report on similar works and highlight substantial differences with the proposed approaches. First, the idea of augmenting a GNN with new edges is not new. MeshGraphNets[25], the architecture that inspires this work, introduces virtual edge based on spatial proximity; we instead add edges to increase connectivity between nodes that are far apart. A recent paper focused on cardiovascular applications[30] introduces virtual edges that connect all internal nodes with boundary nodes; although this modification is very powerful in one-dimensional settings, like the one presented in the paper, it could become prohibitive in case of three-dimensional geometries. More in general, we mention that edge augmentation is a well-established technique in the context of classification problems[31,32]. Second, in the context of both mechanics and fluid dynamics, several authors have considered ideas similar to ours to coarsen (and refine) the graph while training. For a mechanics application, papers[33,34] introduce a network model with a similar down- and up-sampling process, but a different aggregation step that does not necessarily guarantee generalization with respect to new geometries. Furthermore, these papers do not guarantee translation and rotation invariance. The idea of different resolution levels is also used in[35] that introduces a multifidelity GNN where the coarsening and refining operations happen before the graph is passed to the network. This work, while valuable, has different objectives from ours, i.e. addressing the data generation burden. For fluid mechanics applications, the papers[36–39] introduce a down-sampling/up-sampling technique that presents an intrinsic difference compared to ours, i.e., the down-sampling operation is established manually (based, for example, on standard mesh coarsening), whereas our down-sampling operation is *learned*. This guarantees that only nodes that carry relevant information are selected, making sure that no salient information is lost. In the context of fluids, but with data compression purposes, a technique similar to ours in encoder–decoder form is proposed in[40].

Background

Graph neural networks

Graph neural networks (GNNs) are a class of deep learning methods that operate on graph structures consisting of nodes and edges[41]. GNNs use message passing between neighboring nodes in the graph to model the interdependence of features at various nodes. They have been successfully used for prediction and classification

tasks at the level of graphs, edges, or nodes. In recent years, variants of GNNs such as graph convolutional networks (GCNs)[42], graph attention networks (GATs)[43], and graph recurrent networks (GRNs)[44] have demonstrated ground-breaking performances on various tasks. The expressive nature and superior performance of GNNs have led to their application in a variety of domains where data is naturally represented with a graph structure, such as in particle physics[45], high energy physics detectors[46], power systems[47], etc. Additionally, GNN frameworks have been able to simulate complex physical domains involving fluids, and rigid solids, and deformable materials interacting with one another[27]. In MeshGraphNets, the mesh physical domain is represented using a mesh, which is basically a graph on which GNNs learn to predict physics (see e.g.,[25]).

Challenges in modeling deformation of elastic and hyper elastic materials with GNNs

In this work, we consider the stationary elasticity and hyper-elasticity problems, with a special focus on the Mooney–Rivlin hyper-elastic model[48], even though our proposed approaches have the potential of performing well for a broad class of elliptic partial differential equations. We seek to train GNN models that capture the constitutive behavior of a physical system, rather than approximate the response for a fixed problem setting. To this end, we expect the proposed GNN surrogates to perform well on varying domain geometries, boundary conditions, loadings, and material properties.

A challenging aspect of developing surrogates for stationary mechanics problems (and elliptic PDEs in general) is the fact that solution at any point depend on points that are farther away in the domain. In fact, the solution in these problems can be expressed as integrals over the entire domain. This means that the GNN must include connections between distant points within the network. In a way, the proposed architecture is designed to enable fast message propagation between any points in the domain.

MeshGraphNet for physics simulations

MeshGraphNets[25] take advantage of the mesh representations, which is extensively used for finite element simulations for structural mechanics, aerodynamic problems, etc. With the development of adaptive meshing, accurate, high-resolution simulations can be carried out for deformation of complex geometries with highly irregular meshes. MeshGraphNets predict the dynamics of the physical systems by encoding the simulation state in meshes into a graph structure using an encoder, approximating the differential operators that underpin the internal dynamics of the physical systems using message passing steps of the graph neural network and using a decoder to extract the node level dynamics information to update the meshes.

MeshGraphNet learns the forward model of the dynamic quantities of the mesh at time $t+1$ given the current mesh and meshes at previous time steps. The encoder encodes the current mesh into a graph, with mesh nodes becoming graph nodes and mesh edges becoming bi-directional edges in the graph. They also add extra edges, known as world edges, to learn external dynamics such as self-collision, contact etc., which are non-local to the meshes. World edges are created by spatial proximity, within a fixed radius. Once the graph is created, node and edge attributes are generated and encoded using a learnable encoder network. Next, the processor, consisting of L graph net (GN) blocks that can do message passing operations, update the world edges, mesh edges and the node features of the current graph. Finally, to predict the state at time $t+1$ from the state at time t, a decoder network, which is an MLP, is used to transform the updated node features to output features.

MeshGraphNets can accurately and efficiently model the dynamics of the physical systems. It also has good generalization capability, and can be scaled up at inference time. However, they have not been tested on time-independent problems such as static solid mechanics problems.

In[26], an improved version of MGNs, Multiscale MGN, was proposed with the purpose of recovering the concept of spatial convergence that characterizes mesh based discretizations and improving the training efficiency when refining the mesh. This work takes inspiration from multigrid approaches from mesh based discretizations, but its intrinsic architecture and scope are different from the one presented in this paper.

Methods

As explained in the previous section, solving time-independent solid mechanics problems requires fast propagation of information across all the degrees of freedom, since the deformation, or stress, of every point in space depends on all the other points. Guaranteeing such propagation becomes computationally challenging as the resolution of the computational domain increases. To address this challenge, when using GNNs as numerical solvers, a natural solution is to increase the depth of the network. However, this approach is computationally very expensive and may result in over-smoothing[49], making the model harder to train, or causing vanishing/exploding gradients[50]. To achieve fast propagation of information while avoiding these pitfalls, we propose two approaches: (a) edge-augmentated graph neural network , and (b) multi-graph neural network. In the following we describe each model separately.

(a) Edge augmentation: we refer to the first approach as "edge augmentation" of the existing graph. As the graph is generated from a mesh, nodes are only connected to their neighbors; if two nodes are n hops away, it requires n GN blocks for the message to be passed between these nodes[51]. Thus, as n increases, a deeper network is required to achieve broad message propagation. To avoid increasing the depth of the network, we increase the non-locality of the graph by introducing augmented edges between randomly selected nodes (see Fig. 1c). This edge augmentation significantly reduces the number of hops required for the nodes to gather messages from distant nodes, thus resulting in a smaller number of GN blocks for faster message propagation.

(b) Multi-graph: this approach draws inspiration from multigrid methods for PDEs, whose foundation is a hierarchy of discretizations (or meshes)[52]. By performing a series of graph down-sampling operations fol-

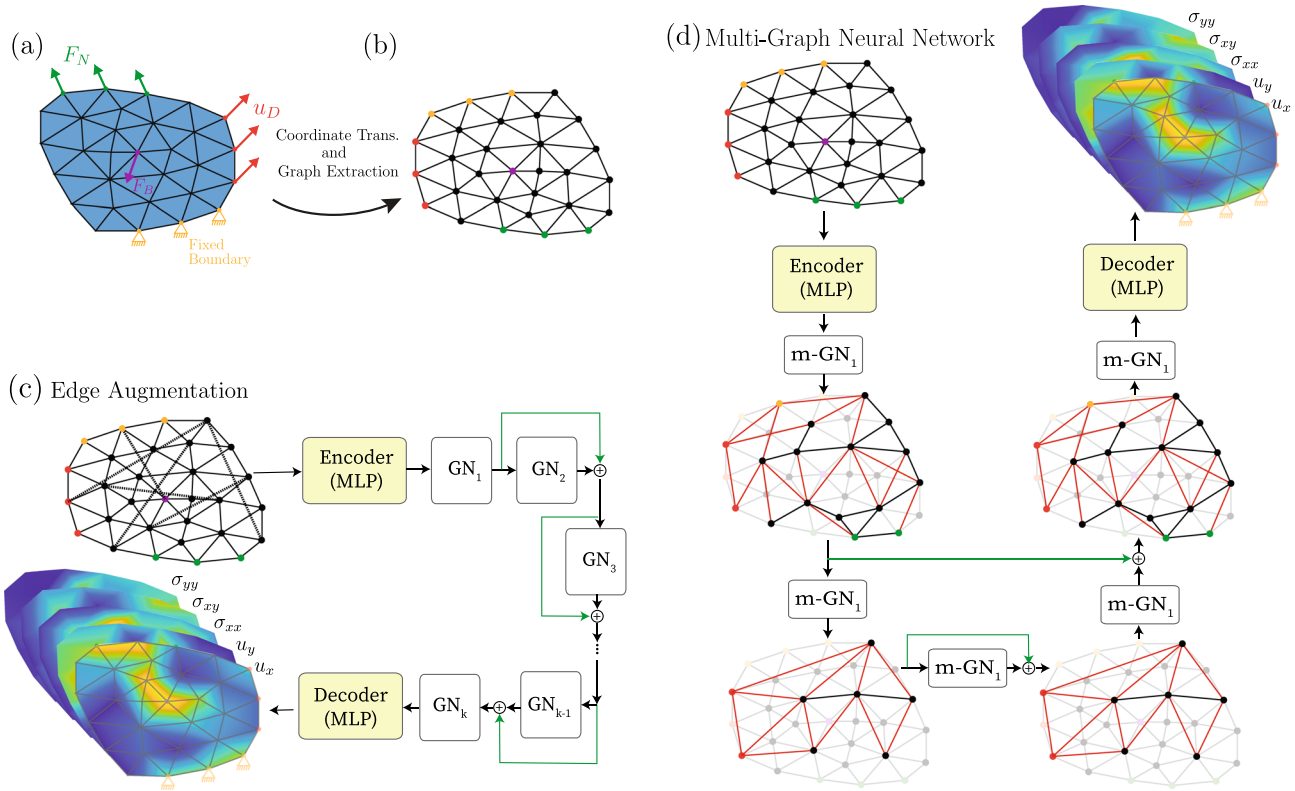


Figure 1. (**a**) Schematic of a 2D time-independent (static) solid mechanics problem. (**b**) Physical domains are transferred to a normalized coordinate. (**c**) Schematic of edge-augmented graph neural networks (agumented edges are marked as dashed line) where the normalized graph is passed to an MLP encoder and multiple passes of GN blocks and then a decoder to recover the displacement and stress vectors across the domain. (**d**) Schematic of multi-graph neural network, where features after encoding, pass through M-Net block with down-sampling/upsampling of the mesh.

lowed by up-sampling operations, we apply the multigrid concept to graphs and pass information across the computational domain in a hierarchical manner. In practice, the down-sampling operations correspond to mesh coarsening and result in smaller graphs connecting points that are far away in the domain. On the other hand, during up-sampling the information is redistributed across the entire graph. This approach resembles the graph U-Net architecture used in[28] for image processing tasks such as image classification and segmentation.

For both approaches, we assume that the computational domain is a triangular mesh which undergoes a transformation into a graph as detailed in "Data generation" and illustrated in Fig. 1.

Edge augmented graph neural network (EA-GNN)

With the purpose of achieving faster propagation of information between nodes we select a set of nodes from the graph and, if no edges exist between them, we add a bi-directional edge. We point out that, since this edge is not similar to other edges, we add an extra feature to the edge attributes to indicate that the edge belongs to the augmentation set. The number of additional edges in the augmented graph is a hyperparameter that is manually tuned during the training. Moreover, vertices of augmented edges are selected randomly, by following a uniform distribution. It is to be noted that edges have not been added based on spatial proximity[53] as that would not improve the message propagation to farther nodes. Instead, sparse and random edges have been added all over the domain, thus improving the long-range interactions between the nodes. As more sophisticated sampling methods, such as farthest point sampling[54], did not yield significant improvements in the performance of the model, we employed random sampling of nodes in all of our experiments.

The proposed EA-GNN has the same high-level network architecture as the MeshGraphNet[25] whose building blocks are: (1) an encoder, (2) *m* GN blocks, and (3) a decoder. As described in "Data generation", for the time-independent problems considered in this work, with the purpose of making the graph translation and rotation invariant, before the graph is created, the mesh is *transformed* to be in the principal axis coordinate system (referred to as simulation coordinate system). Then, the mesh is *converted* into a graph structure by identifying the vertices of the mesh as nodes and the connections between the vertices in the mesh as the edges (see Fig. 1a,b). The node attributes are nodal positions in simulation coordinates (x, y), nodal type (interior or boundary), type of the boundary condition applied (Dirichlet homogenous and non-homogenous, Neumann), direction and

magnitude of the boundary conditions, a flag for body forces and its magnitude and direction all in simulation coordinate system. The edge attributes are the Euclidean distance between the nodes, and positional difference in x and y directions, i.e., $\Delta x = x_2 - x_1$ and $\Delta y = y_2 - y_1$.

Once the graph is generated, the node attributes and edge attributes are encoded into a latent space through an encoder. The encoded nodes and edges attributes are then passed through m GN blocks and updated. Each GN block consists of an edge update module and a node update module. Let $\mathbf{u}'_i$ be the encoded node attribute vector of node i and $\mathbf{u}'_j$ be the encoded node attribute vector for node j, such that $j \in M(i)$, where $M(i)$ denotes the neighborhood of node i. Let $\mathbf{e}'_{ij}$ be the encoded edge attribute vector for the edge between i and j. The edge update module, χ, is a MLP that receives the attributes of the nodes connecting the edge along with the edge attributes and returns the updated attributes, i.e. $\mathbf{e}'_{ij} = \chi(\mathbf{e}'_{ij})$.

The node update module consists of three MLPs, ϕ, γ and β; the encoded node attributes of node i, $\mathbf{u}'_i$ are updated as follows:

$$\begin{aligned} \mathbf{m}_{ij} &= \phi(\mathbf{u}'_i, \mathbf{u}'_j, \mathbf{e}'_{ij}) \\ \mathbf{u}'_i &= \gamma(\mathbf{u}'_i, \frac{1}{N}\sum_{j \in M(i)} \mathbf{m}_{ij}) \\ \mathbf{u}'_i &= \mathbf{u}'_i + \beta(\mathbf{u}'_i). \end{aligned} \tag{1}$$

Between each GN block, a skip connection is added to avoid over-smoothing[50]. The following equation describes the update for two consecutive GN blocks of the network, GN_k and GN_{k+1}:

$$\begin{aligned} \mathbf{u}'^{(k)}, \mathbf{e}'^{(k)} &= GN_k(\mathbf{u}'^{(k-1)}, \mathbf{e}'^{(k-1)}) \\ \mathbf{u}'^{(k)} &= \mathbf{u}'^{(k)} + \mathbf{u}'^{(k-1)} \\ \mathbf{u}'^{(k+1)}, \mathbf{e}'^{(k+1)} &= GN_{k+1}(\mathbf{u}'^{(k)}, \mathbf{e}'^{(k)}) \\ \mathbf{u}'^{(k+1)} &= \mathbf{u}'^{(k+1)} + \mathbf{u}'^{(k)}. \end{aligned} \tag{2}$$

Finally, the updated node attributes, $\mathbf{u}'$ is passed through a decoder to return the nodal deformation or stress values. Both the encoder and decoder functions are MLP networks.

Multi-graph neural network (M-GNN)

This approach takes inspiration from multigrid solvers and relies on hierarchical learning. The architecture has three components: (1) an encoder, (2) an M-Net block, and (3) a decoder (see Fig. 1d). Encoder and decoder have the same network architecture and functionality as in EA-GNN; however, for reasons that are clarified later in this section, here we do not consider edge attributes.

The M-Net block starts with a single graph network block, m-GN, to update the encoded node attributes received from the encoder. The m-GN block uses GraphSAGE operator[55] for the node update. The encoded node attributes, $\mathbf{u}'$, are updated through m-GN block as follows.

$$\begin{aligned} \mathbf{u}'_i &= \text{GraphSAGE}(\mathbf{u}'_i, \mathbf{u}'_j), \forall j \in M(i) \\ \mathbf{u}'_i &= \mathbf{u}'_i + \beta(\mathbf{u}'_i), \end{aligned} \tag{3}$$

where β is a MLP. The updated node attributes, $\mathbf{u}'$, are then passed through a series of alternating layers of down-sampling operations and m-GN blocks. This is followed by a series of alternating up-sampling operations and m-GN blocks. The number of down-sampling and up-sampling layers is determined by the multi-graph depth hyperparameter, d. The output of the down-sampling layer is added to the output of the corresponding up-sampling layer leading to the same mesh refinement (or graph size). Note that all the m-GN block updates have the same structure reported in Eq. (3) and that the final up-sampling layer is such that the graph recovers its original size.

The down-sampling layer down-samples the data by adaptively selecting a subset of nodes corresponding to a coarser mesh; the number of nodes that are down-sampled is determined by the hyperparameter r, indicating the down-sampling ratio. For the adaptive selection of nodes, we use the "U-net sub-sampling" algorithm[28]. Specifically, the node attributes are projected onto the trainable vector $\mathbf{p}$ using the scalar projection $\mathbf{u}_i^T \mathbf{p}$ and top k nodes are selected based on the projected values. Since the scalar product measures the amount of information retained by node i when projected onto $\mathbf{p}$, sampling the top k nodes ensures that the smaller graph retains the maximum information. The up-sampling operation up-samples the graph by recording the locations of nodes selected in the corresponding down-sampling layer and uses this information to place the nodes back to their original positions in the graph.

In order to make sure that there are no disconnected nodes after down-sampling, as well as to improve the connectivity between the nodes, we compute the l^{th} graph power, similarly to[28], and use the resulting graph. This operation builds links between nodes which are at-most l hops away in the graph. This could be done by multiplying the adjacency matrix of the graph by itself l times. For the training, we choose $l = 3$, and use the augmented graph with better connectivity for every down-sampling layer. This step is particularly important in mesh-based physics simulations for uninterrupted propagation of information between nodes.

The GraphSAGE operator[55], used in the m-GN block, is a message aggregation algorithm which considers only the node attributes. In fact, as nodes are down-sampled, existing edges are lost and new edges are introduced to improve the connectivity. This makes updating edge attributes (for existing and newly established edges) a

nontrivial task. Since the nodal positions are considered as node attributes, the model can calculate the edge attributes such as Euclidean distance and positional difference indirectly from the node attributes. Thus no information is lost due to the removal of edge attributes.

Results and discussion

Data generation

The data generation process consists of three steps: (1) mesh generation, (2) finite element simulations, and (3) graph transformation. We generate two-dimensional random geometries using Bezier curves to ensure variability and nonlinearity in the domains. Few sampled geometries are shown in Fig. 2. Different boundary conditions (Dirichlet and Neumann) are assigned at randomly selected locations on the boundary. The length, location, magnitude and direction of the boundary conditions are randomly selected for each geometry. We also assign a body force at a randomly selected interior location of the domain. A uniform triangulated mesh is generated using the `Gmsh` package in python[56]. Using Abaqus[57], we carry out finite element simulations to obtain nodal deformation and stress values.

Coordinate transformation

As the output values (deformation and stress) are coordinate dependent and every sample in the training set is characterized by a different geometry, it is necessary to assign the nodal coordinates as node attributes. However, this makes the graph translation and rotation variant, i.e. if the mesh is translated or rotated, the network would consider it as a different geometry and outputs different deformation and stress values. This makes the training process difficult as it requires redundant data in training to make the model invariant to rotation and translation. To resolve this issue, we use group equivariance as our inductive bias where we ensure the graph is invariant to translation and rotation by transforming the geometry into the principal axis coordinate system. As such, the nodal coordinates stay the same when the geometry is rotated or translated, and as a result, the transformed domain is invariant to the rotation and translation. From now on, we refer to these coordinates as "simulation coordinates" (SC).

Let $\mathbf{X} \in \mathbb{R}^{N*2}$ be the original nodal coordinates of the graph and $\mathbf{X_{SC}} \in \mathbb{R}^{N*2}$ be the corresponding simulation coordinates, where N is the number of vertices in one mesh. First, the coordinates are made translation invariant by moving the center of the original coordinate system to the centroid of the graph, i.e, $\mathbf{X}_c = \mathbf{X} - \overline{\mathbf{X}}$. Here, $\overline{\mathbf{X}}$ is the centroid of the graph calculated as $\overline{\mathbf{X}} = \sum_{n=1}^{N} \mathbf{X_n}/N$, where N is the total number of nodes in the graph. Further, the coordinates are made rotation invariant by rotating the coordinate system to principal axes. This is done by calculating eigenvalues and eigenvectors of the matrix $\mathbf{X}_c^T\mathbf{X}_c$, denoted by $\lambda_1, \lambda_2, \lambda_3$ and $\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3$ respectively, where $\lambda_1 \geq \lambda_2 \geq \lambda_3$. Let $\mathbf{B}$ be a matrix composed of the eigenvectors, $\mathbf{B} = [\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3]$. The simulation coordinates, $\mathbf{X}_{SC}$ can be calculated as $\mathbf{X}_{SC} = \mathbf{X}_c\mathbf{B}$.

Data augmentation

To improve the generalization capability of the model with respect to unseen geometries, we introduce two data-augmentation strategies. First, we use both the Delaunay and the "Packing of Parallelograms" algorithms for mesh generation. Second, we add noise to nodal coordinates by shifting the nodal coordinates by adding to

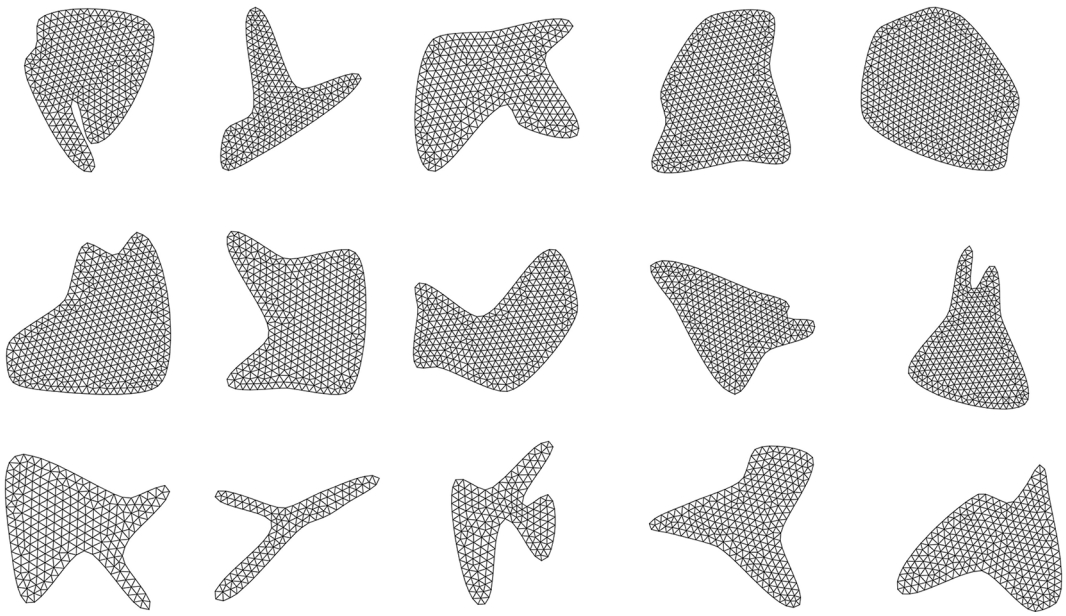


Figure 2. A few samples of the random geometry generated using Bezier curves for training and validation of the models.

each node a random value sampled from the normal distribution $\mathcal{N}(0, 0.01)$ which is equivalent to 10% of edge distance on average.

Data conversion

Nodal deformations and stress values are assigned to the node and edge attributes for the mesh-to-graph conversion outlined in "Edge augmented graph neural network (EA-GNN)". For GNN with edge augmentation, augmented edges are added to the existing edge connections together with an augmentation flag (1 for augmented edges and 0 otherwise) as edge attribute. The node and edge attributes along with the edge connections are used to generate the graph objects using PyTorch Geometric.

Model architectures

EA-GNN

During graph creation, edge augmentation is done by sampling nodes from the graph and adding bi-directional edges between them, if there are no existing edges. The percentage of augmented edges is determined by the hyper parameter, augmented-edge-percentage, A_{perc} . It is to be noted that A_{perc} is the percentage of the existing edges. Therefore, the total number of edges after augmentation is $N_e + A_{perc} \times N_e$, where N_e is the total number of existing edges. We use $A_{perc} = 20\%$ for our experiments.

The encoders for the node and edge attributes of the augmented graph are MLPs with a single hidden layer and a ReLU activation function. The MLP for node attributes has a network architecture $14 - 64 - 128$ (i.e., 14 input features, 64 nodes in the hidden layer and 128 output features) and that for edge attributes has network architecture $4 - 64 - 128$. The encoded graph is then passed through a series of alternating layers of Graph Net (GN) blocks and MLP layers. A total of 6 GN blocks are added in the network. Each GN block consists of (a) *Edge update module*—the edge update function, χ, is a MLP, with a single hidden layer architecture $3 \times 128 - 128 - 128$, and ReLU activation function. The size of the input is due to concatenating the edge attributes (128 features) and the attributes of the two nodes connected by the edge (2×128 features). The output consists of the updated edge attributes of size $n_e \times 128$, where n_e is the total number of edges. (b) *Node update module*—this module consists of three functions, ϕ for message passing, γ for message aggregation and node update and β for the final node update. All the three functions are MLPs with a single hidden layer and ReLU activation functions. The function ϕ has network architecture $2 \times 128 - 128 - 128$, where the input features are the concatenation of the attributes of the neighboring node and the corresponding edge attributes. The messages from all the neighboring nodes are added and concatenated to the node attributes of the selected node and the result is given as the input to γ to calculate the node attributes of the corresponding nodes. Thus, the network architecture of γ is $2 \times 128 - 128 - 128$. The output from γ is passed to β to get the updated node attributes as shown in Eq. (1). The MLP representing β has network architecture $128 - 128 - 128$.

The parameters of all the four functions ($\chi, \phi, \gamma, \beta$) are shared across all the GN blocks. After each GN block, a skip connection is added as shown in Eq. (2). The updated node attribute from the final GN block is the input to the decoder, a MLP with a single hidden layer, ReLU activation function, and architecture $128 - 64 - 2$ for the displacement (u_x and u_y) and $128 - 64 - 3$ for the stress (σ_{xx}, σ_{xy} and σ_{yy}). We use a dropout layer after the encoder and between each GN block with a dropout percentage of 0.1 to reduce overfitting.

M-GNN

M-GNN has an encoder architecture similar to that of EA-GNN. However, since edge attributes are not used, the encoder is required only for the node attributes. The encoder is a MLP with architecture $14 - 64 - 128$. The encoded graph is passed through a message aggregation block followed by a ReLU activation layer. The message aggregation block used here is the GraphSAGE operator[55], which updates the node attributes for the full graph.

This is followed by a series of down-sampling modules consisting of the following operations: (a) *connectivity enhancement*—the connectivity of the graph is enhanced by connecting the nodes that are 3 hops away. This is done by multiplying the adjacency matrix, A by itself twice, i.e. $A_{upd} = A \times A \times A$. This ensures that we do not have any disconnected nodes i.e., all the nodes are connected to at least one neighbor, allowing for effective message exchange. (b) *Down-sampling*—a layer that projects the node attributes onto a one-dimensional, trainable projection vector $\mathbf{p}$, and samples k nodes based on the projected values. The value of k is determined by the down-sampling ratio, r. For the experiments, we have set $r = 0.6$. (c) *Node update*—a block consisting of the GraphSAGE operator and the function β, as described in Eq. (3).

The number of down-sampling modules is determined by the multi-graph depth hyperparameter, d. After tuning of the hyperparameters, we have set this value to be $d = 3$. The depth and pool ratio for the network are chosen such that our A-3 connected graphs are not overpopulated. These are followed by same number of up-sampling modules, consisting of the following operations: (a) *skip connection*—the node attributes from the corresponding down-sampling layer are added to the updated node attributes. (b) *Up-sampling*—a layer that adds the previously removed nodes back to the graph, with their updated node attributes. (c) *Node update*—a block consisting of the GraphSAGE operator and the function β, as described in Equation 3.

The final pooling module restores the full graph with all the nodes with updated node attributes. This is decoded to return the nodal displacement or stress using the Decoder, a network with the same architecture used for EA-GNN. Similar to EA-GNN, we use a dropout layer after the encoder and between each GN block with a dropout percentage of 0.1 to reduce overfitting.

Experimental set up

By using the data generation process detailed in section "Data generation", we generate the following three datasets and carry out data augmentation as explained in "Data generation". (a) We generate 3000 random geometries

and run 10 variations of boundary conditions per geometry assuming a homogeneous and linear elastic material with constant Young's modulus $E = 100.0$ and Poisson's ratio $\nu = 0.3$. This dataset is used to evaluate the ability of the model to capture geometrical nonlinearities. (b) We generate 5000 random geometries and assume a nonlinear hyper-elastic material. We again consider 10 different boundary conditions per geometry. The nonlinear material model of choice is the Mooney–Rivlin model[58], with material properties $C_{01} = 0.3$ and $C_{10} = 0.03$[59]. Training with this dataset enables us to test the network performance for system nonlinearities. (c) We generate 10,000 random geometries with varying hyper-elastic material properties and again consider 10 different boundary conditions per geometry. The material properties for each geometry are randomly selected from a uniform distribution, $C_{01} \sim \mathscr{U}(0.034, 0.34)$ and $C_{10} \sim \mathscr{U}(0.005, 0.05)$. We use this dataset to train and test our architecture on generalization with respect to unseen materials. For all the datasets, there are, on average, 1100 nodes per graph/mesh. The mesh resolution length (distance between two nodes in the graph) is approximately 0.1. The message passing distance which is calculated by the number of hops needed to connect all the nodes is greater than 10 for graphs of this size. This implies at least 10 message-passing blocks are needed for complete passing of information across the graph for the baseline GNN with no edge augmentation or multi-scale architecture.

For all the experiments, 70%, 10%, 20% of the data is used for training, validation, and testing, respectively. For each dataset we train four networks to predict the nodal deformations, u_x and u_y. As a baseline model (B) we consider MeshGraphNet. Since our simulation coordinate transformation plays an important role in the generalization properties of our model, we further consider a version of MeshGraphNet trained after such a transformation and refer to it as B + SC. We additionally provide results for our edge augmented graph neural network using simulation coordinates (EA-GNN + SC), and multi-graph neural network using simulation coordinates (M-GNN + SC). Similarly, the four networks are trained to predict nodal stress values, σ_{xx}, σ_{xy} and σ_{yy}.

While the baseline model is trained using mean squared error loss function as used in[25], we train the networks B+SC, EA-GNN+SC and M-GNN+SC with a scaled mean absolute error loss function L. The scaling depends on a combination of the boundary conditions associated with each sample; formally,

$$L(\theta) = \frac{1}{n}\sum_{n=1}^{N}(\|\mathbf{d}_n\|_{\ell^1} + \|\mathbf{n}_n\|_{\ell^1})\|\mathbf{y}_n - \widehat{\mathbf{y}}_n\|_{\ell^1}, \tag{4}$$

where $\mathbf{y}$ and $\hat{\mathbf{y}}$ are respectively the ground-truth and the predicted outputs and $\mathbf{d}_n$ and $\mathbf{n}_n$ are the Dirichlet loading and Neumann displacement vectors of the n-th sample.

For all networks we use the Adam optimizer and we train all the networks for 1500 epochs on a Tesla V100-SXM2. For B, B+SC and EA-GNN+SC, we prescribe a learning rate ranging from 1E−4 to 1.5E−4 and a weight decay of 1E−5. The learning rate is decreased using the cosine annealing scheduler with warm restart[60], which avoids stagnation in local minima. For M-GNN+SC, we use a learning rate ranging from 2E−3 to 3E−3 with a weight decay of 1E−6.

Numerical results

We test our proposed models on three datasets detailed in "Experimental set up" and compare them with the baseline models. In order to compare the prediction power of our models, we define relative error as

$$e(f) = \frac{\|\widehat{f} - f\|_{\ell^1}}{\|f\|_{\ell^1}}, \tag{5}$$

where f is the variable of interest (either displacement values u_x, u_y or stress values $\sigma_{xx}, \sigma_{yy}, \sigma_{xy}$) and $\widehat{f}$ is the predicted value of f using our network.

Table 1 shows the relative errors on the test dataset for the predicted nodal displacement components (u_x and u_y) and nodal stress components (σ_{xx}, σ_{yy} and σ_{xy}). We have also included the relative errors in the predicted displacement magnitude (u) and von Mises stress (σ_v). Here, E stands for linear elastic model and HE for hyper-elastic (Mooney–Rivilin) material model. Additionally, S stands for single material and it means that a single material is used across the entire dataset, whereas V stands for varying material and it means that varying materials are used across the data set.

As shown in Table 1, both EA-GNN and M-GNN perform significantly better than the baseline models either with or without coordinate transformation, i.e., B and B+SC respectively. This shows that both the edge augmentation and the multi-graph modifications have the ability to further improve message passing. We further observe that by adding the coordinate transformation, the performance of the baseline model improves significantly, however, it is still very large compared to our proposed approaches. This comparison allows us to distinguish the improvement brought by the coordinate transformation and the graph modifications. It is interesting to see that the edge augmentation technique consistently performs better than all other methods, as highlighted by the bold values used for the best performing models. Comparing the rows of single linear elastic material (E,S) with single nonlinear hyper-elastic materials (HE,S), we find that the performance of all the models slightly decrease and the relative absolute errors increases as expected. The non-linearity introduces new complexities into the system which requires more data for the learning. Note that adding further complexity by varying the nonlinear material among test cases (i.e., HE, V), we find that the performance stays the same and the proposed model can generalize to unseen material properties. The similar test performance of our models on HE, V test-dataset, as opposed to the performance of the baseline methods, shows the generalization capabilities of our model, which is indeed able to learn the true physics and predict accurate results for unseen nonlinear material properties. We also observe that the errors of displacement magnitude and von Mises stress remain closely aligned with the component-wise errors for displacement and stress across all the datasets and models. Additionally, in Table 2

Mat.	Model	Param.	$e(u_x) \downarrow$	$e(u_y) \downarrow$	$e(u) \downarrow$	$e(\sigma_{xx}) \downarrow$	$e(\sigma_{yy}) \downarrow$	$e(\sigma_{xy}) \downarrow$	$e(\sigma_v) \downarrow$
E,S	B	7.2e5	0.64	0.63	0.60	0.78	0.79	0.49	0.57
E,S	B+SC	7.2e5	0.25	0.26	0.23	0.30	0.30	0.19	0.22
E,S	EA-GNN+SC	7.2e5	**0.05**	**0.05**	**0.04**	**0.09**	**0.09**	**0.05**	**0.05**
E,S	M-GNN+SC	2.8e5	0.13	0.13	0.12	0.17	0.18	0.11	0.10
HE,S	B	7.2e5	0.81	0.81	0.81	0.88	0.88	0.64	0.62
HE,S	B+SC	7.2e5	0.33	0.34	0.32	0.30	0.32	0.29	0.28
HE,S	EA-GNN+SC	7.2e5	**0.08**	**0.09**	**0.07**	**0.11**	**0.13**	**0.09**	**0.10**
HE,S	M-GNN+SC	2.8e5	0.16	0.16	0.16	0.18	0.19	0.15	0.15
HE,V	B	7.2e5	0.73	0.74	0.72	0.92	0.91	0.85	0.86
HE,V	B+SC	7.2e5	0.28	0.29	0.28	0.39	0.39	0.29	0.30
HE,V	EA-GNN+SC	7.2e5	**0.09**	**0.09**	**0.08**	**0.12**	**0.12**	**0.09**	**0.10**
HE,V	M-GNN+SC	2.8e5	0.16	0.15	0.14	0.18	0.20	0.16	0.15

Table 1. Relative error in predicting nodal displacement components, u_x, u_y , and displacement magnitude, u, as well as in nodal stress components, $\sigma_{xx}, \sigma_{xy}, \sigma_{yy}$ and von Mises stress, σ_v, given different linear and nonlinear material selection and models.

Mat.	Model	$\lvert e(u_x)\rvert_{max} \downarrow$	$\lvert e(u_y)\rvert_{max} \downarrow$	$\lvert e(\sigma_{xx})\rvert_{max} \downarrow$	$\lvert e(\sigma_{yy})\rvert_{max} \downarrow$	$\lvert e(\sigma_{xy})\rvert_{max} \downarrow$
E,S	B	0.84	0.86	0.90	0.91	0.84
E,S	B+SC	0.53	0.59	0.56	0.55	0.49
E,S	EA-GNN+SC	**0.15**	**0.17**	**0.19**	**0.19**	**0.17**
E,S	M-GNN+SC	0.30	0.31	0.35	0.34	0.31
HE,S	B	1.10	1.12	1.20	1.20	0.99
HE,S	B+SC	0.69	0.70	0.66	0.68	0.63
HE,S	EA-GNN+SC	**0.21**	**0.24**	**0.22**	**0.22**	**0.17**
HE,S	M-GNN+SC	0.35	0.36	0.38	0.37	0.34
HE,V	B	0.87	0.90	1.25	1.24	1.03
HE,V	B+SC	0.61	0.63	0.70	0.75	0.65
HE,V	EA-GNN+SC	**0.20**	**0.23**	**0.21**	**0.19**	**0.17**
HE,V	M-GNN+SC	0.33	0.34	0.37	0.37	0.32

Table 2. Maximum relative error in predicting nodal displacement components u_x and u_y and nodal stress components $\sigma_{xx}, \sigma_{xy}, \sigma_{yy}$ over the testing dataset given different linear and nonlinear material selection and models.

we report the maximum relative errors e across the testing dataset and we notice that even the worst-case errors are low.

In Tables 3 and 4, we report the relative ℓ_∞ error and the corresponding 90th percentile values (p_{90}) over all degrees of freedom for the predicted displacement and stress respectively. Although the ℓ_∞ norms are significant across all the models, it is to be noted that these high errors correspond to areas where the displacement and stress values are at least 2 orders of magnitude lower than that of the average values across the dataset. The 90th percentile error values reported in Tables 3 and 4 also confirm that the majority of the DOFs have lower error values. Moreover, we point out the maximum absolute error for the prediction of the von Mises stress for our models is very low. For reference, the absolute errors corresponding to the maximum relative errors for the σ_v prediction for the three datasets using EA-GNN are 0.003, 0.09, 0.08 and using M-GNN are 0.001, 0.07, 0.09. Therefore, we argue that these deviations are of less critical concern from an engineering-application perspective as they are associated with areas in the domain that undergo very small displacements and stresses.

In addition to error values, we show the loss values for the test dataset in correspondence of all the models for both the displacement, u_x, u_y and the stress values, $\sigma_{xx}, \sigma_{yy}, \sigma_{xy}$ in Fig. 3a,b respectively. The lines represent the moving average, while the shadow corresponds to the variance. The sudden jumps in the loss values observed in M-GNN and EA-GNN correspond to the annealing process. Overall, we find that the coordinate transformation, improves the training of the baseline model, however, great boost in performance is only achieved when considering Edge Augmentation or Multi-Graph approach.

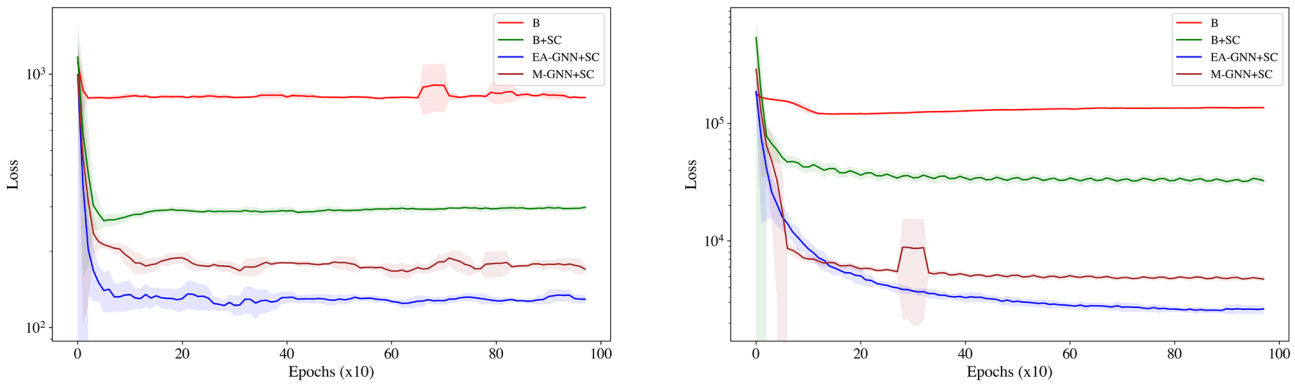
Next, in order to evaluate the generalization capability of our models and confirm that the models have indeed learned the true physics, we test the trained networks on out-of-distribution datasets, i.e. datasets that are not from the distribution used in training. To this end, we consider three out-of-sample distributions: (a) smaller/larger physical domains characterized by half or double the domain size with the same mesh characteristic length. The results for scale = 0.5 and 2.0 are shown in Table 5. (b) New Dirichlet and Neumann boundary conditions applied at different sections of the boundary. Specifically, Dirichlet and Neumann boundary conditions are

Mat.	Model	$\|e(u_x)\|_\infty \downarrow$	$\|e(u_y)\|_\infty \downarrow$	$p_{90}(e(u_x)) \downarrow$	$p_{90}(e(u_x)) \downarrow$
E,S	B	92.14	90.78	0.87	0.88
E,S	B+SC	65.22	64.29	0.61	0.63
E,S	EA-GNN+SC	**23.28**	**25.17**	**0.21**	**0.23**
E,S	M-GNN+SC	40.22	41.57	0.39	0.38
HE,S	B	103.45	101.21	1.06	1.12
HE,S	B+SC	70.11	69.12	0.72	0.73
HE,S	EA-GNN+SC	**30.10**	**31.21**	**0.26**	**0.27**
HE,S	M-GNN+SC	43.44	45.76	0.43	0.45
HE,V	B	93.11	92.32	0.88	0.90
HE,V	B+SC	67.87	66.39	0.65	0.67
HE,V	EA-GNN+SC	**32.29**	**31.65**	**0.28**	**0.27**
HE,V	M-GNN+SC	42.22	40.98	0.40	0.38

Table 3. Maximum and 90th quartile values of relative error, across all degrees of freedom, in the testing dataset, for predicting nodal displacements, u_x, u_y, given different linear and nonlinear material selection and models.

Mat.	Model	$\|e(\sigma_{xx})\|_\infty \downarrow$	$\|e(\sigma_{yy})\|_\infty \downarrow$	$\|e(\sigma_{xy})\|_\infty \downarrow$	$p_{90}(e(\sigma_{xx})) \downarrow$	$p_{90}(e(\sigma_{yy})) \downarrow$	$p_{90}(e(\sigma_{xy})) \downarrow$
E,S	B	110.32	111.97	105.44	0.98	0.98	0.92
E,S	B+SC	70.31	68.82	61.55	0.63	0.64	0.60
E,S	EA-GNN+SC	**29.73**	**31.85**	**24.76**	**0.26**	**0.26**	**0.22**
E,S	M-GNN+SC	45.51	46.11	40.98	0.42	0.44	0.38
HE,S	B	120.40	119.71	109.26	1.14	1.15	1.05
HE,S	B+SC	68.88	69.12	64.71	0.73	0.73	0.69
HE,S	EA-GNN+SC	**29.85**	**30.56**	**27.90**	**0.25**	**0.25**	**0.23**
HE,S	M-GNN+SC	46.39	44.11	40.87	0.47	0.47	0.42
HE,V	B	123.32	123.89	111.63	1.17	1.17	1.06
HE,V	B+SC	70.47	72.39	65.24	0.79	0.84	0.72
HE,V	EA-GNN+SC	**30.21**	**29.40**	**28.10**	**0.25**	**0.26**	**0.24**
HE,V	M-GNN+SC	45.22	43.98	38.36	0.47	0.45	0.40

Table 4. Maximum and 90th quartile values of relative error, across all degrees of freedom, in the testing dataset, for predicting nodal stress components, σ_{xx}, σ_{yy} and σ_{xy} , given different linear and nonlinear material selection and models.



(a) Deformation **(b)** Stress

Figure 3. Loss on testing data for predicting (**a**) deformation and (**b**) stress for elastic model with single material for four models—B, B+SC, EA-GNN+SC and M-GNN+SC.

Model	B	B+SC	EA-GNN+SC	M-GNN+SC
Scale = 0.5				
$e(u_x)$	0.87 ± 0.56	0.47 ± 0.29	**0.09 ± 0.07**	0.20 ± 0.11
$e(u_y)$	0.96 ± 0.64	0.47 ± 0.33	**0.09 ± 0.07**	0.20 ± 0.11
Scale = 2				
$e(u_x)$	1.09 ± 0.67	0.85 ± 0.41	**0.22 ± 0.12**	0.29 ± 0.11
$e(u_y)$	0.95 ± 0.51	0.76 ± 0.33	**0.21 ± 0.10**	0.28 ± 0.09
BC				
$e(u_x)$	0.87 ± 0.55	0.57 ± 0.42	**0.11 ± 0.08**	0.22 ± 0.10
$e(u_y)$	0.95 ± 0.65	0.60 ± 0.43	**0.13 ± 0.09**	0.24 ± 0.10
Rot + Tra				
$e(u_x)$	0.91 ± 0.69	0.57 ± 0.42	**0.11 ± 0.08**	0.22 ± 0.10
$e(u_y)$	1.04 ± 0.78	0.60 ± 0.43	**0.13 ± 0.09**	0.24 ± 0.10

Table 5. Relative error in predicting nodal deformation for various out of sample distributions of data.

applied at disconnected sets of boundary nodes as opposed to connected sets. The results for out-of-distribution boundary conditions are shown Table 5 with the label BC. (c) A combination of all the above along with random rotation and translation, i.e. the graphs from the above data are translated and rotated to a different coordinate system to evaluate if the models are rotation and translation invariant. The results for rotation and translation are also presented in Table 5 with label Rot + Tra.

The first observation is that the training based on simulation coordinates helps the generalization on all fronts. Next, we observe that EA-GNN consistently outperforms other architectures, followed by M-GNN, both evaluated in simulation coordinate system. Both EA-GNN and M-GNN have relative errors close to the in-distribution testing errors from Table 1 for half scaling. The models perform worse for the double scaling case, as almost half of the graph corresponds to coordinate positions the model has not seen before. The errors for the rotation and translation do not change for B+SC, EA-GNN+SC and M-GNN+SC as the graphs are transformed to the simulation coordinate system before they are fed into the model and as expected the models perform well, as long as they are trained using simulation coordinates. Across all the cases, the baseline model B consistently performs the worst, with errors 1.5–2 times bigger than B+SC. This is the only model which is not invariant to translation and rotation, as can be observed from the higher errors for rotation and translation case for B.

Our results indicate that M-GNN consistently under performs compared to EA-GNN. To address this concern, one should consider both the choice of the m-GN block and the pooling/unpooling algorithm. We address the former first. We carry out an ablation study to verify if EA-GNN is consistently performing better than M-GNN because EA-GNN uses edge attributes in the GNN layer while M-GNN uses a GraphSAGE layer that does not involve edge attributes. For this, we trained EA-GNN to predict displacement and stress for elastic dataset with GraphSAGE as the message propagation GN block. Since GraphSAGE does not take edge attributes, the network does not differentiate between real and augmented edges. The results of this experiment can be found in Table 6. The results are similar to the reported results for EA-GNN in Table 1. This is an indication that the better performance of EA-GNN is due to the addition of the augmented edges, and it is not impacted by the selection of the message propagation algorithm.

Regarding the choice of the pooling/unpooling algorithm, the method used in this paper has been successfully used for coarsening GNNs' graphs for fluid dynamics simulations[40]. While this suggests that this choice is very promising for PDE regression problems, one alternative to a learned pooling selection is to consider the relative spatial distance between vertices in the parent graph. This approach has been considered in[33,38,61]. We stress that a complete comparison with state-of-the-art methods is fundamental to fully understand our model's capabilities; given this limitation in the presented comparison studies, the integration of distance-based approaches is the subject of our current studies.

From all these experiments, we can infer that both EA-GNN and M-GNN are able to successfully learn the underlying physics of the data and generalize well to unseen domains, geometries, and boundary conditions, which the baseline MeshGraphNet model fails at. Training and evaluating the models in simulation coordinate system enables them to be invariant to rotation and translation, which makes these models an effective tool for faster and accurate physics simulations specially for time-independent physics simulations that require long-range interaction between different parts of the domain. In terms of computational time, Abaqus takes on an

Model	$e(u_x)$	$e(u_y)$	$e(\sigma_{xx})$	$e(\sigma_{yy})$	$e(\sigma_{xy})$
EA-GNN	0.05	0.05	0.09	0.09	0.05
EA-GNN with graph-SAGE	0.06	0.06	0.10	0.09	0.06

Table 6. Ablation study using GraphSAGE as the GN block for EA-GNN in predicting nodal deformation and stress.

average 20 seconds in a single CPU machine with 64-bit Intel® Xeon® Processor, 2.50 GHz and 256 GB Memory and GNN models take on an average 0.10 s on Tesla V100 GPU with 5120 cores and 16 GB HBM 2 for inference.

Conclusion

In this paper, we propose improved GNN architectures for modeling static mechanics behavior by efficiently capturing long-range interactions. We show that Edge augmented GNNs and Multi-GNNs can capture accurately and efficiently the constitutive behaviour of static elastic and hyper elastic materials thanks to their enhanced connectivity. Furthermore, by learning the physics in a reference coordinate system, our models are automatically rotation and translation invariant. With several numerical tests, we show that both our proposed architectures learn time-independent solid mechanics efficiently and generalize well to unseen materials, boundary conditions and domains. We note that the proposed approach for coordinate transformation can be easily used in any graph-based architecture, as we demostrate for MeshGraphNets. Our approach represents an easily implementable solution for learning challenging time-independent physical systems using deep learning and, as such, it is a good candidate for simulating complex static systems in science and engineering.

Data availability

The simulation data generated and utilized in this manuscript is not readily available for distribution. However, the methods and the procedures for data generation are disclosed in the manuscript. The authors are willing to assist and provide support to generate data from the disclosed methods/procedures.

Received: 9 May 2023; Accepted: 29 January 2024
Published online: 09 February 2024

References

1. Thuerey, N. *et al.Physics-based Deep Learning* (WWW, 2021).
2. Karniadakis, G. E. *et al.* Physics-informed machine learning. *Nat. Rev. Phys.* **3**, 422–440 (2021).
3. Lagaris, I. E., Likas, A. & Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Trans. Neural Netw.* **9**, 987–1000 (1998).
4. 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.* **378**, 686–707 (2019).
5. Sirignano, J. & Spiliopoulos, K. Dgm: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* **375**, 1339–1364 (2018).
6. Yu, B. *et al.* The deep ritz method: A deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.* **6**, 1–12 (2018).
7. Khoo, Y., Lu, J. & Ying, L. Solving parametric pde problems with artificial neural networks. *Eur. J. Appl. Math.* **32**, 421–435 (2021).
8. Ruthotto, L. & Haber, E. Deep neural networks motivated by partial differential equations. *J. Math. Imaging Vis.* **62**, 352–364 (2020).
9. Geneva, N. & Zabaras, N. Modeling the dynamics of pde systems with physics-constrained deep auto-regressive networks. *J. Comput. Phys.* **403**, 109056 (2020).
10. Long, Z., Lu, Y. & Dong, B. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *J. Comput. Phys.* **399**, 108925 (2019).
11. Bar-Sinai, Y., Hoyer, S., Hickey, J. & Brenner, M. P. Learning data-driven discretizations for partial differential equations. *Proc. Natl. Acad. Sci.* **116**, 15344–15349 (2019).
12. Kochkov, D. *et al.* Machine learning-accelerated computational fluid dynamics. *Proc. Natl. Acad. Sci.* **118**, e2101784118 (2021).
13. Zhuang, J., Kochkov, D., Bar-Sinai, Y., Brenner, M. P. & Hoyer, S. Learned discretizations for passive scalar advection in a two-dimensional turbulent flow. *Phys. Rev. Fluids* **6**, 064605 (2021).
14. Han, J., Jentzen, A. & Weinan, E. Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci.* **115**, 8505–8510 (2018).
15. Li, Z. *et al.* Neural operator: Graph kernel network for partial differential equations. arXiv:2003.03485 (arXiv preprint) (2020).
16. Li, Z. *et al.* Multipole graph neural operator for parametric partial differential equations. *Adv. Neural. Inf. Process. Syst.* **33**, 6755–6766 (2020).
17. You, H., Yu, Y., D'Elia, M., Gao, T. & Silling, S. Nonlocal kernel network (nkn): A stable and resolution-independent deep neural network. *J. Comput. Phys.* **469**, 111536 (2022).
18. Iakovlev, V., Heinonen, M. & Lähdesmäki, H. Learning continuous-time pdes from sparse data with graph neural networks. arXiv:2006.08956 (arXiv preprint) (2020).
19. Belbute-Peres, F. D. A., Economon, T. & Kolter, Z. Combining differentiable pde solvers and graph neural networks for fluid flow prediction. In *International Conference on Machine Learning*, 2402–2411 (PMLR, 2020).
20. Alet, F. *et al.* Graph element networks: Adaptive, structured computation and memory. In *International Conference on Machine Learning*, 212–222 (PMLR, 2019).
21. Li, Z. *et al.* Fourier neural operator for parametric partial differential equations. arXiv:2010.08895 (arXiv preprint) (2020).
22. Li, Z., Huang, D. Z., Liu, B. & Anandkumar, A. Fourier neural operator with learned deformations for pdes on general geometries. arXiv:2207.05209 (arXiv preprint) (2022).
23. Lu, L., Jin, P., Pang, G., Zhang, Z. & Karniadakis, G. E. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nat. Mach. Intell.* **3**, 218–229 (2021).
24. Poli, M. *et al.* Graph neural ordinary differential equations. arXiv:1911.07532 (2019) (arXiv preprint).
25. Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A. & Battaglia, P. W. Learning mesh-based simulation with graph networks. *CoR-Rabs/2010.03409* (2020). arXiv:2010.03409.
26. Fortunato, M., Pfaff, T., Wirnsberger, P., Pritzel, A. & Battaglia, P. Multiscale meshgraphnets. arXiv:2210.00612 (arXiv preprint) (2022).
27. Sanchez-Gonzalez, A. *et al.* Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning, Vol. 119 of Proceedings of Machine Learning Research* (III, H. D. & Singh, A., eds), 8459–8468 (PMLR, 2020).
28. Gao, H. & Ji, S. Graph u-nets. In *International Conference on Machine Learning*, 2083–2092 (PMLR, 2019).
29. Liu, N., Yu, Y., You, H. & Tatikola, N. Ino: Invariant neural operators for learning complex physical systems with momentum conservation. arXiv:2212.14365 (arXiv preprint) (2022).
30. Pegolotti, L. *et al.* Learning reduced-order models for cardiovascular simulations with graph neural networks. arXiv:2303.07310 (arXiv preprint) (2023).

31. Wang, B. & Gong, N. Z. Attacking graph-based classification via manipulating the graph structure. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2023–2040 (2019).
32. Zhao, T. *et al.* Data augmentation for graph neural networks. *In Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 35, 11015–11023 (2021).
33. Deshpande, S., Lengiewicz, J. & Bordas, S. Magnet: A graph u-net architecture for mesh-based simulations. arXiv:2211.00713 (arXiv preprint) (2022).
34. Deshpande, S., Sosa, R. I., Bordas, S. & Lengiewicz, J. Convolution, aggregation and attention based deep neural networks for accelerating simulations in mechanics. *Front. Mater.* **10**, 1128954 (2023).
35. Black, N. & Najafi, A. R. Learning finite element convergence with the multi-fidelity graph neural network. *Comput. Methods Appl. Mech. Eng.* **397**, 115120 (2022).
36. Lino, M., Cantwell, C., Bharath, A. A. & Fotiadis, S. Simulating continuum mechanics with multi-scale graph neural networks. arXiv:2106.04900 (arXiv preprint) (2021).
37. Lino, M., Fotiadis, S., Bharath, A. A. & Cantwell, C. Towards fast simulation of environmental fluid mechanics with multi-scale graph neural networks. arXiv:2205.02637 (arXiv preprint) (2022).
38. Lino, M., Fotiadis, S., Bharath, A. A. & Cantwell, C. D. Multi-scale rotation-equivariant graph neural networks for unsteady eulerian fluid dynamics. *Phys. Fluids* **34**, 25 (2022).
39. Yang, Z., Dong, Y., Deng, X. & Zhang, L. Amgnet: Multi-scale graph neural networks for flow field prediction. *Connect. Sci.* **34**, 2500–2519 (2022).
40. Barwey, S., Shankar, V. & Maulik, R. Multiscale graph neural network autoencoders for interpretable scientific machine learning. arXiv:2302.06186 (arXiv preprint) (2023).
41. Zhou, J. *et al.* Graph neural networks: A review of methods and applications. *AI Open* **1**, 57–81 (2020).
42. Yao, L., Mao, C. & Luo, Y. Graph convolutional networks for text classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 33, 7370–7377 (2019).
43. Veličković, P. *et al.* Graph attention networks. arXiv:1710.10903 (arXiv preprint) (2017).
44. Seo, Y., Defferrard, M., Vandergheynst, P. & Bresson, X. Structured sequence modeling with graph convolutional recurrent networks. In *International Conference on Neural Information Processing*, 362–373 (Springer, 2018).
45. Shlomi, J., Battaglia, P. & Vlimant, J.-R. Graph neural networks in particle physics. *Mach. Learn. Sci. Technol.* **2**, 021001 (2020).
46. Ju, X. *et al.* Graph neural networks for particle reconstruction in high energy physics detectors. arXiv:2003.11603 (arXiv preprint) (2020).
47. Donon, B., Donnot, B., Guyon, I. & Marot, A. Graph neural solver for power systems. In *2019 International Joint Conference on Neural Networks (Ijcnn)*, 1–8 (IEEE, 2019).
48. Bergström, J. 5—elasticity/hyperelasticity. In *Mechanics of Solid Polymers* (ed. Bergström, J.) 209–307 (William Andrew Publishing, 2015). https://doi.org/10.1016/B978-0-323-31150-2.00005-4.
49. Zhao, L. & Akoglu, L. Pairnorm: Tackling oversmoothing in gnns. arXiv:1909.12223 (arXiv preprint) (2019).
50. Wu, Z. *et al.* Representing long-range context for graph neural networks with global attention. *Adv. Neural. Inf. Process. Syst.* **34**, 13266–13279 (2021).
51. Hamilton, W. Synthesis lectures on artificial intelligence and machine learning. *Graph Represent. Learn.* **20**, 20 (2020).
52. Briggs, W., Henson, V. & McCormick, S. *A Multigrid Tutorial*, 2nd Edition (2000).
53. Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A. & Battaglia, P. W. Learning mesh-based simulation with graph networks. arXiv:2010.03409 (arXiv preprint) (2020).
54. Eldar, Y., Lindenbaum, M., Porat, M. & Zeevi, Y. The farthest point strategy for progressive image sampling. *IEEE Trans. Image Process. Publ. IEEE Signal Process. Soc.* **6**, 1305–15. https://doi.org/10.1109/83.623193 (1997).
55. Hamilton, W., Ying, Z. & Leskovec, J. Inductive representation learning on large graphs. In *Advances in Neural Information Processing Systems* Vol. 30 (eds Guyon, I. *et al.*) (Curran Associates Inc., 2017).
56. 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.* **79**, 1309–1331 (2009).
57. Manual, A. U. Abaqus user manual. *Abacus* (2020).
58. Liu, I.-S. *et al.* A note on the Mooney–Rivlin material model. *Contin. Mech. Thermodyn.* **24**, 583–590 (2012).
59. Kumar, N. & Rao, V. V. Hyperelastic Mooney–Rivlin model: Determination and physical interpretation of material constants. *Parameters* **2**, 01 (2016).
60. Loshchilov, I. & Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. arXiv:1608.03983 (arXiv preprint) (2016).
61. Suk, J., Haan, P. d., Lippe, P., Brune, C. & Wolterink, J. M. Mesh convolutional neural networks for wall shear stress estimation in 3d artery models. In *International Workshop on Statistical Atlases and Computational Models of the Heart*, 93–102 (Springer, 2021).

Acknowledgements

R.G., M.D., and A.Z. would like to thank Prajjwal Jamdagni for the great and detailed discussions on FEM numerical simulations. A.Z. and V.S. extend their gratitude to Abhishek Sharma who provided encouragement and support in the initiation of this project, and whose guidance and mentorship will always be appreciated. All authors would like to sincerely thank the reviewers whose contributions made this paper more complete and rigorous.

Author contributions

A.Z. and V.S. initiated the concept for this study. R.G. and H.R. were responsible for conducting the finite element method simulations necessary for training the AI models. R.G., M.D., and A.Z. devised the AI architectures and R.G. implemented and carried out the AI model training and testing. All authors contributed to analysis of the results, writing of the manuscript, and reviewing the final version.

Competing interests

The authors declare no competing interests.

Additional information

Correspondence and requests for materials should be addressed to M.D.