Available online at www.sciencedirect.com
ScienceDirect
Comput. Methods Appl. Mech. Engrg. 397 (2022) 115120
Computer methods in applied mechanics and engineering
www.elsevier.com/locate/cma

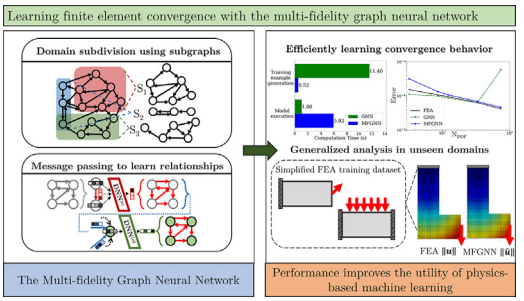
Learning finite element convergence with the Multi-fidelity Graph Neural Network☆

Nolan Black, Ahmad R. Najafi*

Department of Mechanical Engineering and Mechanics, Drexel University, Philadelphia, PA, 19104, USA

Received 8 November 2021; received in revised form 12 April 2022; accepted 10 May 2022
Available online xxxx

Graphical Abstract



Abstract

Machine learning techniques have emerged as potential alternatives to traditional physics-based modeling and partial differential equation solvers. Among these machine learning techniques, Graph Neural Networks (GNNs) simulate physics via graph models; GNNs embed relevant physical features into graph data structures, perform message passing within the graphs, and produce new attributes based on the system's relationships. Like many machine learning frameworks, GNNs are limited by excessive data generation costs and limited generalizability outside of a narrow training domain. To address these limitations, we introduce the Multi-Fidelity Graph Neural Network (MFGNN), a supervised machine learning framework that uses low-fidelity projections to inform high-fidelity modeling across arbitrary subdomains represented by subgraphs. We implement the MFGNN for two-dimensional elastostatic problems with finite element training data. The MFGNN is trained to produce accurate analysis given low-fidelity evaluations and emulate the convergence behavior of traditional finite element analysis (FEA). Through subdomain abstraction, we also extend the MFGNN as a general model for new boundary conditions and material domains outside of the training domain.

☆ Associated code and models used in this paper are available: github.com/MCMB-Lab/MFGNNs.

* Corresponding author.
E-mail address: arn55@drexel.edu (A.R. Najafi).

https://doi.org/10.1016/j.cma.2022.115120
0045-7825/

Keywords: Machine learning; Surrogate modeling; Multi-fidelity analysis; Graph Neural Networks

1. Introduction

Engineering models often represent physical systems with partial differential equations (PDEs) that are solved with numerical methods like finite element analysis (FEA). With increasing scale and complexity of physical models, traditional numerical methods become computationally prohibitive [1–3]. To overcome this computational burden, machine learning has been proposed as a new modeling paradigm to replace PDE solvers or to emulate FEA [4,5].

Machine learning models extrapolate a problem structure into a discrete framework by inferring underlying system behavior [4]. Recent applications of machine learning in computational mechanics have derived process–structure–property relations [6], designed advanced materials [7,8], and built optimized structures [9,10]. Neural Networks (NNs), trained through the iterative optimization of a loss function $\mathcal{L}(\boldsymbol{\theta})$ for some model parameters $\boldsymbol{\theta}$, often drive these machine learning applications. This optimization incurs significant computational cost [4,5], and the resulting model often fails to meet the stringent accuracy, reliability, and stability requirements of physics-based simulation [4]. Therefore, a successful application of machine learning for physics-based simulations must meet rigorous accuracy metrics with limited training resources.

To meet the demands of physics-based simulations, physics-constrained machine learning may supplement $\mathcal{L}(\boldsymbol{\theta})$ with additional objectives for PDE residual, boundary, or conservation terms [11–14]. Such soft constraints have been implemented to solve traditional diffusion and flow problems [5,15,16]; physics-informed machine learning models have also derived PDE forms from noisy data [5]. These applications indicate the NN's utility in supplementing or replacing numerical solvers, but physics-constrained NNs fail to generalize outside of a narrow problem domain [17].

General stability among a variety of PDE formulations or finite element domains is necessary for a generalized surrogate model [17,18]. Once trained, a generalized surrogate must accommodate a range of boundary conditions and material domains to approximate a system's physical response from a single execution of the model. Multi-fidelity modeling [19] has been proposed to address generalizability, and multi-fidelity physics-informed neural networks have been trained using low-fidelity data to evaluate high-fidelity physics [20–25]. Low-fidelity training data addresses the computational cost tradeoff of data pre-generation and model training. By applying multi-fidelity approaches to learning within a subdomain, some physics-based machine learning models have also generalized across multiple physical domains [26]. Training using multi-fidelity data can also improve the information gain of the machine learning model, allowing for increased accuracy and generality [21,27,28].

Graph data structures have been implemented in machine learning surrogates for PDE-driven physics models, incorporating domain knowledge with the graph's inherent structure. The Graph-informed Neural Network (GINN) encodes domain-specific information in graphs to generate physically sound distributions that drive physics-based surrogates [29]. Graphs can also encode unstructured data and when combined with PDE-informed objective functions, have modeled physics in irregular domains [30,31]. Parameterized graph calculus has also been used to generate low-fidelity surrogate models that strictly enforce physical behavior [32]. These approaches use graph structures to enrich training data and model architecture while directly incorporating knowledge of governing differential equations.

To further extend the generalizability of machine learning models, Graph Neural Networks (GNNs) use neural networks to learn relations between data in graph structures [33,34]. This relation-based reasoning has had success in physics-based rigid body dynamic modeling by representing mass objects as graph nodes and learning inter-mass relations [35–38]. GNNs have recently been implemented in the finite element domain, incorporating meshed domains into graph data structures [30,39,40] or performing element-wise operations [41]. GNNs have also been shown to generalize to conditions outside of the training domain [39–41]. The generalizability of GNNs is particularly appealing for learning physical simulation, as a single model may be trained in one domain then generalized to solve systems of varying size and complexity. This idea was explored in [42], where graph operators were used as mappings of a PDE input and their solutions at different resolutions.

This work seeks to expand the accuracy, reliability, and generalizability of machine learning models for physics simulation while exploring methods for increasing the utility of synthetic datasets. We introduce Multi-fidelity Graph Neural Networks (MFGNNs), a multi-fidelity approach to machine learning physics simulation. The MFGNN

uses a system of NNs to learn physics relations across a collection of subdomains. Without directly enforcing specific differential behavior, the MFGNN targets an underlying mapping from the pre-convergence solutions to the converged result. By training this model with the right selection of FEA examples, the MFGNN framework improves the accuracy of machine learning analysis. Additionally, MFGNN performance is shown to conform to traditional finite element convergence behavior. MFGNNs improve the scalability and generalizability of machine learning based physical simulation through a multi-fidelity subdivision of the problem domain, strict imposition of boundary conditions via graph-edge analysis, and implementation of low-fidelity projections to perform high-fidelity analysis.

The MFGNN framework is introduced in Section 2, illustrative examples are presented in Section 3 based on two dimensional (2D) solid mechanics problems, and MFGNNs are extended in Section 4 as a generalized surrogate model.

2. Multi-fidelity Graph Neural Networks (MFGNNs)

Graph Neural Networks (GNNs) operate over graph data structures to learn relationships between features [34]. In a domain Ω, relevant data are embedded in a graph G. For this work, G is defined as a set of N_n nodes $\{n_i\}_{i=1}^{N_n}$ connected by a set of N_e directed edges $\{e_j\}_{j=1}^{N_e}$. Each edge e_j connects a pair of nodes $(n_s^{(j)}, n_r^{(j)})$ defined as senders and receivers respectively. Graph nodes and edges are both assigned unique attribute vectors that represent features. In this work, a feature is defined as a specific quantity of interest (e.g., temperature at a point, displacement of a beam). When features are compiled to a graph node n_i or edge e_j, they are referred to as the graph's nodal attribute $\boldsymbol{n}_i$ or edge attribute $\boldsymbol{e}_j$. The graph G is fully defined by its set of attributes $(\{\boldsymbol{n}\}, \{\boldsymbol{e}\})$ and node–node connections. As illustrated in Fig. 1, G may include any number of node–node relationships defined by edge connections.

A given attribute neighborhood $\mathcal{N}$ in G is established by a sequence of message-passing steps in which a node receives information from its connected neighbors. Shown in Fig. 1, the message-passing step is defined by three stages: (*i*) edge attribute updates by a function $\boldsymbol{f}_e$, (*ii*) aggregation by a function $\boldsymbol{\rho}$, and (*iii*) node attribute updates by a function $\boldsymbol{f}_n$ [34]. The GNN performs attribute updates using Deep Neural Networks (DNNs). The message-passing step begins with an edge attribute update: given an edge attribute vector $\boldsymbol{e}_j$,

$$\boldsymbol{f}_e(\boldsymbol{e}_j, \boldsymbol{n}_s, \boldsymbol{n}_r) := DNN_{(e)}(\boldsymbol{e}_j, \boldsymbol{n}_s^{(j)}, \boldsymbol{n}_r^{(j)}) = \boldsymbol{e}'_j \tag{1}$$

where an edge attribute $\boldsymbol{e}_j$ is updated to $\boldsymbol{e}'_j$ based on its assigned edge attribute and associated nodal attributes $\boldsymbol{n}_s^{(j)}, \boldsymbol{n}_r^{(j)}$. Fig. 1, stage (*i*) shows the edge update process for a single edge in the input graph G. Edge updates are performed for each edge in the graph, producing an intermediate graph representation $G^* = (\{\boldsymbol{n}\}, \{\boldsymbol{e}'\})$. Next, the updated edge attribute is aggregated at each of its local nodes $\boldsymbol{n}_i$, so

$$\boldsymbol{\rho}^{(i)}(\boldsymbol{e}'_j) = \sum_{j|_{r_j=i}} \boldsymbol{e}'_j = \boldsymbol{e}^*_j, \tag{2}$$

where the aggregated edge attribute $\boldsymbol{e}^*_j$ represents the incoming edge information at each node. An aggregation step is illustrated in Fig. 1, stage (*ii*), where incoming edge information is limited to edges with receivers defined by n_i (i.e., $r_j = i$). Using the aggregation $\boldsymbol{e}^*_j$ and a given node attribute $\boldsymbol{n}_i$, nodal attributes are updated to $\boldsymbol{n}'_i$ with

$$\boldsymbol{f}_n(\boldsymbol{n}_i, \boldsymbol{e}^*_j) := DNN_{(n)}(\boldsymbol{n}_i, \boldsymbol{e}^*_j) = \boldsymbol{n}'_i. \tag{3}$$

Fig. 1, stage (*iii*) shows a single node attribute update which – along with an edge aggregation step – is repeated for each node in the graph to produce the message-passing step's output $G' = (\{\boldsymbol{n}'\}, \{\boldsymbol{e}'\})$.

A single message-passing step like that of Fig. 1 preserves the structure of the input graph while collecting relational information in each edge and node. Through sequential attribute updates and aggregations, $DNN^{(e)}$ and $DNN^{(n)}$ accrue new information based on the graph's connectivity. As the number of message-passing steps increases, so too does the size of the learned neighborhood $\mathcal{N}$.

Multi-fidelity Graph Neural Networks (MFGNNs) represent the domain Ω with a low-fidelity graph $\tilde{G}$ and high-fidelity graph G. As shown in Fig. 2(*a*), $\tilde{G}$ is a condensed representation of G. Given a low-fidelity feature $\tilde{\boldsymbol{u}}$ assigned to attributes in $\tilde{G}$, a projection function $\boldsymbol{p}$ produces an approximation $\bar{\boldsymbol{u}}$ which may be applied to attributes in the high-fidelity graph G. The MFGNN uses the projected attributes to infer high-fidelity analysis $\boldsymbol{u}$, as in

$$\boldsymbol{p}(\tilde{\boldsymbol{u}}) = \bar{\boldsymbol{u}} \approx \boldsymbol{u}. \tag{4}$$

Algorithm 1 MFGNN

Input High-fidelity Graph $G = (\{\boldsymbol{n}\}, \{\boldsymbol{e}\})$, with unassigned attributes; Low-fidelity Graph $\tilde{G} = (\{\tilde{\boldsymbol{n}}\}, \{\tilde{\boldsymbol{e}}\})$ representing the feature $\tilde{\boldsymbol{u}}$

Project low-fidelity feature $\tilde{\boldsymbol{u}}$ into high-fidelity feature $\bar{u}$ (4).
Assign $\bar{u}$ to attributes in G
Perform subdivision of G to $\{S_s\}_{s=1}^{N_s}$ (cf. Fig. 2(b))
for each Subgraph $S_s := (\{\check{\boldsymbol{n}}\}, \{\check{\boldsymbol{e}}\}) \in \{S_s\}_{s=1}^{N_s}$,
  for m rounds of message-passing,
    for each edge $:= \check{\boldsymbol{e}}_j \in \{\check{\boldsymbol{e}}_j\}_{j=1}^{N_e}$,
      Update attribute, $\check{\boldsymbol{e}}'_j, = DNN^m_{(e)}(\check{\boldsymbol{n}}_r^{(j)}, \check{\boldsymbol{n}}_s^{(j)}, \check{\boldsymbol{e}}_j)$ (1)
    for each node $:= \check{\boldsymbol{n}}_i \in \{\check{\boldsymbol{n}}_i\}_{i=1}^{N_n}$,
      Aggregate local attribute, $\check{\boldsymbol{e}}^*_j = \sum_{j|_{r_j=i}} \check{\boldsymbol{e}}'_j$ (2)
      Update attribute, $\check{\boldsymbol{n}}'_i = DNN^m_{(n)}(\check{\boldsymbol{n}}_i, \check{\boldsymbol{e}}^*_j)$ (3)
  If applicable, perform edge recovery of features (6).
Recover G' from $\{S'_s\}_{s=1}^{N_s}$ (cf. Fig. 2(b))
Output Graph, $G' = (\{\boldsymbol{n}'\}, \{\boldsymbol{e}'\})$

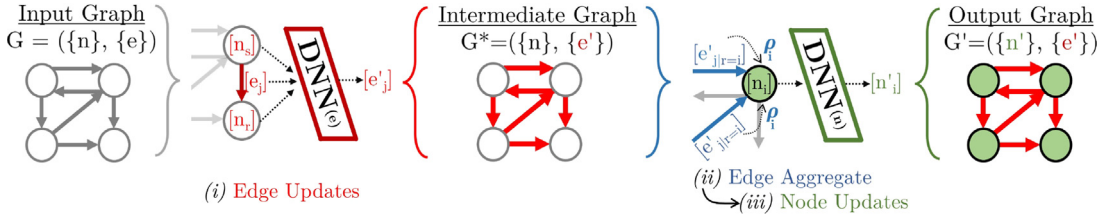


Fig. 1. The message-passing step drives relationship-based learning in three stages: (i) looping through edges to update each edge attribute through $DNN^{(e)}$, (ii) aggregating attributes, and (iii) looping through nodes to update each node through $DNN^{(n)}$.

The projection function (4) is widely interpretable. For example, projection may be performed through averaging of a nodal neighborhood $\mathcal{N}$ or distance-based projection based on the properties of Ω.

Following this projection, operations proceed exclusively in the high-fidelity Graph G. Subsequently G is subdivided into N_s subgraphs $\{S_s\}_{s=1}^{N_s} \in G$ as illustrated in Fig. 2(*b*). Attributes are copied as subsets $\{\check{\boldsymbol{n}}\}$ and $\{\check{\boldsymbol{e}}\}$ to a subgraph S_i based on the corresponding nodes or edges in G, so S_s is defined by nodes $\{\check{\boldsymbol{n}}\} \subset \{\boldsymbol{n}\}$ and edges $\{\check{\boldsymbol{e}}\} \subset \{\boldsymbol{e}\}$. Message-passing steps are performed as in Fig. 1, and each subgraph is independently updated using a single MFGNN such that $\text{MFGNN}(S_s) = S'_s$. For example, the three subgraphs $\{S_s\}_{s=1}^{3}$ illustrated in Fig. 2(*b*) are each passed to the same MFGNN model and evaluated independently for relevant attributes. Because the original domain Ω is represented by the graph G, the collection of output subgraphs $\{S'_s\}_{s=1}^{3}$ must be reconstructed as G' to recover interpretable metrics. Reconstruction follows the formation of subgraphs, so the attributes in $\{S'_s\}_{s=1}^{3}$ are copied back to their corresponding nodes and edges to form G'. If a node or edge is common to multiple subgraphs (as with S_2 and S_3), the common attributes are simply averaged.

Algorithm 1 summarizes the order of operations for the MFGNN framework. Within these operations, key design decisions influence the optimization of the MFGNN model, including the subgraph subdivision, normalization schemes, and feature recovery. Fig. 2(*b*) shows just three possible subdivisions of a graph G, but additional possibilities exist. With more unique subdivisions and shared attributes between subdivisions, the MFGNN will learn more localized analysis. Conversely, execution time over an entire domain is proportional to the number of subdivisions. As such, the arbitrary subdivision is a design decision that depends on the application's accuracy requirements and resource restrictions.

The MFGNN architecture allows for practical implementation of localized normalization (e.g., z-score standardization, min–max normalization, etc.) that normalizes features based on statistics of the projected feature $\bar{\boldsymbol{u}}$ within

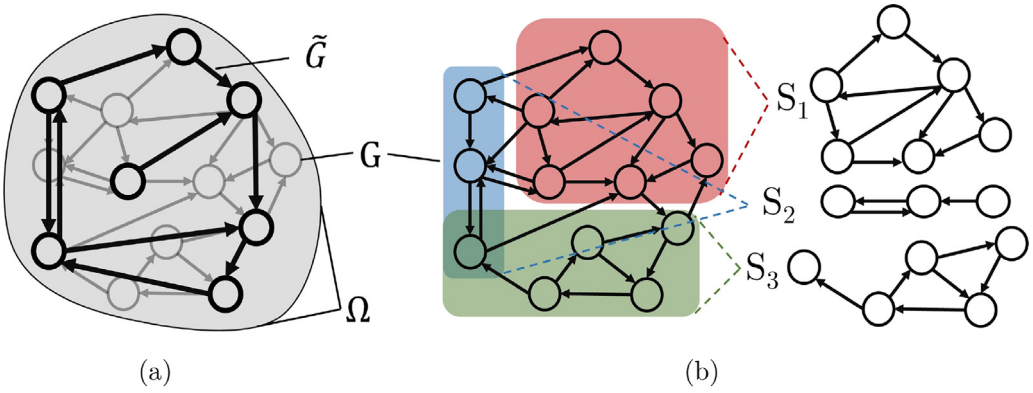


Fig. 2. (a) The domain Ω is modeled by a low-fidelity representation $\tilde{G}$ (shown in bold) and a high-fidelity representation G (shown in gray). The MFGNN is informed via a projection from $\tilde{G}$ to G. (b) Subdivision of G into a variety of subgraphs $\{S_s\}_{s=1}^{3} \in G$ ensures accuracy and scalability of the analysis.

each subgraph S_s. Because this standardization framework operates independently for any given S_s, the MFGNN may be reused for each S_s regardless of the feature's units or magnitude.

Graph-edge analysis (edge learning) assembles nodal features based on a relative feature embedded in graph edges. The generic relative feature is defined at edge j by

$$\Delta(\cdot)_j = (\cdot)_r^{(j)} - (\cdot)_s^{(j)} \tag{5}$$

for the edge's receiver node r and sender node s. As the MFGNN produces $\Delta(\cdot)'_j$ for each edge, a nodal feature $(\cdot)_r^{'(j)}$ for node n_r is recovered using

$$(\cdot)_r^{'(j)} = \frac{1}{N_r} \sum_{j|_{r_j=r}}^{N_r} \left(\Delta(\cdot)'_j + (\cdot)_s^{'(j)} \right), \tag{6}$$

where N_r is the number of incoming edges, $\Delta(\cdot)'_j$ is the relative feature as an edge attribute, and $(\cdot)_s^{'(j)}$ is the sender's associated feature for that edge. Edge-based recovery using (6) averages the edge-based approximation of a nodal feature $(\cdot)_r^{'(j)}$ for N_r incoming edges. Incoming edges are defined as the edges j with a receiver r_j defined at the node r, so (6) averages the nodal feature for all j where $j|_{r_j=r}$. Naturally, the sender features at $(\cdot)_s^{'(j)}$ must be known, so it is convenient to evaluate each nodal feature using the MFGNN prior to edge recovery. Alternatively, $(\cdot)_s^{'(j)}$ may be substituted with a prescribed condition. Because the MFGNN operates over subgraphs, the number of edge features – which may drastically outnumber nodal features – remains tenable. Therefore, observable nodal features in G' (reconstructed via the subgraphs $\{S'_s\}_{s=1}^{N_s}$) may be recovered directly from graph nodes, assembled from edge-based features, or a combination thereof.

Graph-based machine learning operates over sets of attributes, so a single model may accommodate graphs with different numbers of nodes and edges. Additionally, relational information is learned through iterative optimization of the DNN-driven attribute updates. Message passing establishes a neighborhood of information in which nodes and edges send, receive, and process their neighboring attributes. The MFGNN leverages these properties for physical simulation by (*i*) problem abstraction through an arbitrarily flexible domain subdivision, (*ii*) leveraging low-fidelity data to perform high-fidelity analysis, (*iii*) locally normalizing target data within each subgraph to improve relative accuracy, and (*iv*) enforcing boundary conditions through edge-based learning.

3. Applications of MFGNNs

This section applies the MFGNN framework to elastostatic solid mechanics problems. First, we briefly introduce the elastostatic formulation and associated finite element discretization. We then implement various training regimes to evaluate MFGNNs for structural analysis. Consider the domain Ω illustrated in Fig. 3 defined by the surface $\partial\Omega$

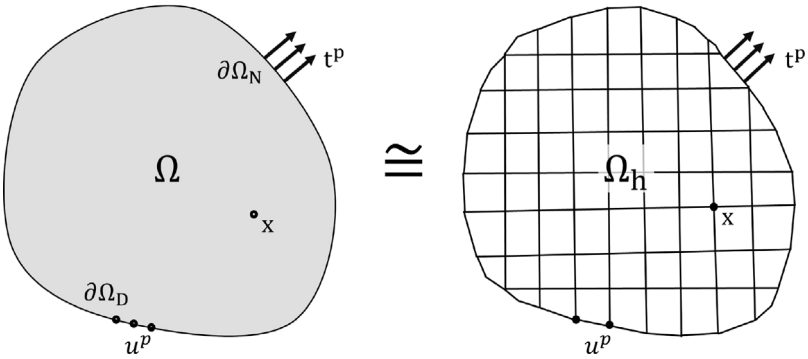


Fig. 3. The general elastostatic boundary value problem is defined in Ω and discretized to Ω_h for numerical analysis.

and subjected to the body force $\boldsymbol{b}_i$. The relevant elastostatic boundary value problem is stated as follows: for the displacement field $\boldsymbol{u}$,

$$\begin{aligned} \boldsymbol{\sigma}_{ij,j} - \boldsymbol{b}_i &= 0, \qquad \boldsymbol{x} \in \Omega \\ \boldsymbol{u}_i(\boldsymbol{x}) &= \boldsymbol{u}_i^p, \qquad \boldsymbol{x} \in \partial\Omega_D \\ \boldsymbol{\sigma}_{ij}\boldsymbol{n}_j &= \boldsymbol{t}_i^p, \qquad \boldsymbol{x} \in \partial\Omega_N, \end{aligned} \tag{7}$$

for the constitutive relation $\boldsymbol{\sigma}_{ij} = \boldsymbol{C}_{ijkl}\boldsymbol{\varepsilon}_{kl}$, kinematic condition $\boldsymbol{\varepsilon}_{kl} = \frac{1}{2}\left(\frac{\partial \boldsymbol{u}_k}{\partial \boldsymbol{x}_l} + \frac{\partial \boldsymbol{u}_l}{\partial \boldsymbol{x}_k}\right)$, Dirichlet boundary condition $\boldsymbol{u}_i^p$ on $\partial\Omega_D$, and Neumann boundary condition on $\partial\Omega_N$ characterized by the traction $\boldsymbol{t}_i^p$ and normal $\boldsymbol{n}_i$. With weights $\boldsymbol{w} \in \mathcal{W} = \{\boldsymbol{w} \in H^1(\Omega) | \boldsymbol{w} = 0 \text{ on } \partial\Omega_D\}$ and kinematically admissible values $\boldsymbol{u}_0 \in \mathcal{K} = \{\boldsymbol{u}_0 \in H^1(\Omega) | \boldsymbol{u}_0 = \boldsymbol{u}_i^p \text{ on } \partial\Omega_D\}$, the associated weak form may be expressed as: find $\boldsymbol{u} \in \mathcal{V} = \boldsymbol{u}_0 + \mathcal{W}$ such that

$$\int_\Omega \boldsymbol{w}_{i,j}\boldsymbol{\sigma}_{ij} dv = \int_\Omega \boldsymbol{w}_i \boldsymbol{b}_i dv + \int_{\partial\Omega_N} \boldsymbol{w}_i \boldsymbol{t}_i^p da, \quad \forall \boldsymbol{w} \in \mathcal{W}. \tag{8}$$

For a discretized domain $\Omega_h \cong \Omega$ (as in Fig. 3), the finite element solution $\boldsymbol{u}_h$ may be expressed as the combination of shape functions $N(x)$ and the field $\boldsymbol{u}$ such that

$$\boldsymbol{u}_h = \sum_{k=1}^{N_k} \boldsymbol{N}_k(x)\boldsymbol{u}_k \tag{9}$$

for N_k shape functions per element.

FEA yields an approximate solution to (8); through refinement of the finite element mesh Ω_h and polynomial approximation of $\boldsymbol{u}_h$, the FEA approximation converges to an analytical solution of (7) or (8) that satisfies measurable kinematic, static, and constitutive conditions of the system [3]. Following this convergence behavior, FEA approximations have inspired NN architecture and training datasets. Finite element polynomial approximations have been embedded in NN architecture, so the optimization of $\mathcal{L}(\boldsymbol{\theta})$ parallels the convergence of traditional FEA [43,44]. Element-wise stiffness matrices have also been approximated using NNs to increase computational efficiency of traditional FEA [45].

MFGNNs naturally accommodate mesh-based analysis, as mesh nodes may transfer information to graph nodes, and mesh connections may transfer to graph edges. Furthermore, as MFGNNs operate over sets of nodal and edge attributes, a single model can evaluate different mesh fidelities. Fig. 4 illustrates the translation of finite element meshes to graph data structures. An $L \times W$ domain is considered, which naturally produces an $L \times W$ graph representation G (Fig. 4(a)). Alternatively, multiple subgraphs S (Fig. 4(b)) can be defined by a coarse mesh and parent nodes (shown in red in Fig. 4) of the low-fidelity graph $\tilde{G}$. Each $l \times w$ subgraph is then evaluated by the MFGNN to recover relevant features. For fair comparison between GNN and MFGNN, we set the mesh resolution of each subgraph S identically to the high-fidelity mesh represented by G. Also note the local standardization used in the MFGNN representation (indicated by the red–blue gradient in Fig. 4) illustrates the abstraction of localized physics in each subdomain.

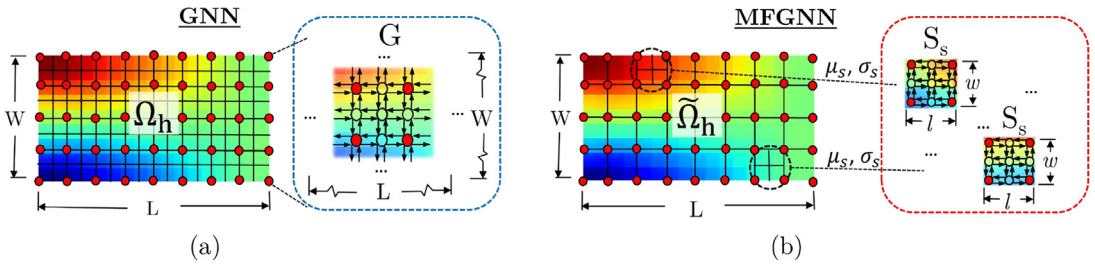


Fig. 4. (a) GNNs encode the entire finite-element representation Ω_h into one $L \times W$ graph G; (b) MFGNNs operate over smaller subdivisions where individual subdomains are standardized using μ_s and σ_s and analyzed independently. Parent nodes of the low-fidelity graph $\tilde{G}$, shown in red, are derived from low-fidelity FEA and used to project low-fidelity analysis onto a high-fidelity graph G that is then subdivided into the subgraphs S_s. (For interpretation of the references to color in this figure legend, the reader is referred to the web version of this article.)

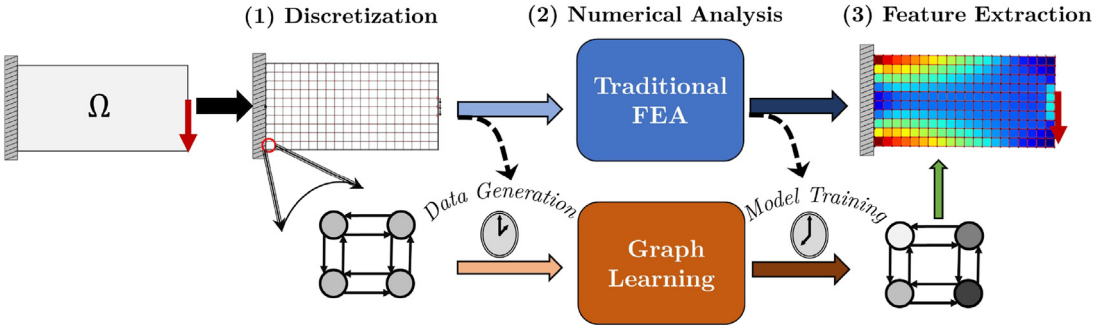


Fig. 5. Graph learning approaches like the Multi-fidelity Graph Neural Network operate on the same discretized domain as traditional finite element analysis but instead use pre-generated data to train a system of Deep Neural Networks. Once trained, these models can rapidly reproduce the finite element results.

Dirichlet boundary conditions are enforced using (6) where $(\cdot)'_s = \boldsymbol{u}^p$ on $\partial\Omega_D$. Low-fidelity finite element evaluations also inform high-fidelity graphs through simple distance-based projections as shown in Fig. 4(*b*). The MFGNN's parent nodes (shown in red) are directly transcribed from the coarse finite element evaluation $\tilde{\boldsymbol{u}}$. The distance from each parent node to a given child node (the transparent nodes in Fig. 4(*b*)) is represented by a 4×1 vector $\boldsymbol{d}$. The projected value at each node in Ω_h is then calculated as

$$p(\tilde{\boldsymbol{u}}) = \frac{\mathbf{1} - \hat{\boldsymbol{d}}}{\text{sum}\left(\mathbf{1} - \hat{\boldsymbol{d}}\right)} \cdot \tilde{\boldsymbol{u}} = \bar{\boldsymbol{u}}_i \in \Omega_h, \tag{10}$$

where $\hat{\boldsymbol{d}}$ represents the normalized distance vector $\boldsymbol{d}$. Eq. (10) is applied for each node in the subgraph, producing an approximation of the high-fidelity response based on the low-fidelity analysis. This weighted averaging projection produces similar approximations as (9) but may be applied to any graph attribute, so the projection is not necessarily confined to strict element representations. In other words, although Fig. 4(*b*) illustrates each $l \times w$ subgraph aligned directly with a coarse mesh, this may not always be the case; the MFGNN is designed to accommodate overlapping, irregular, or disjoint subgraphs (Fig. 2(*b*)).

The following applications use finite element mesh and analysis to inform MFGNNs. Training data is generated for a range of mesh fidelities to explore the necessary computational cost in training MFGNNs. Section 3.1 compares the MFGNN's subgraph analysis with its single graph GNN counterpart to evaluate the mesh-dependence of these FEA surrogates. Section 3.2 evaluates the convergence behavior of MFGNNs under different training regimes. The graph learning process, illustrated in Fig. 5, is evaluated based on data generation costs, model training requirements, and accuracy.

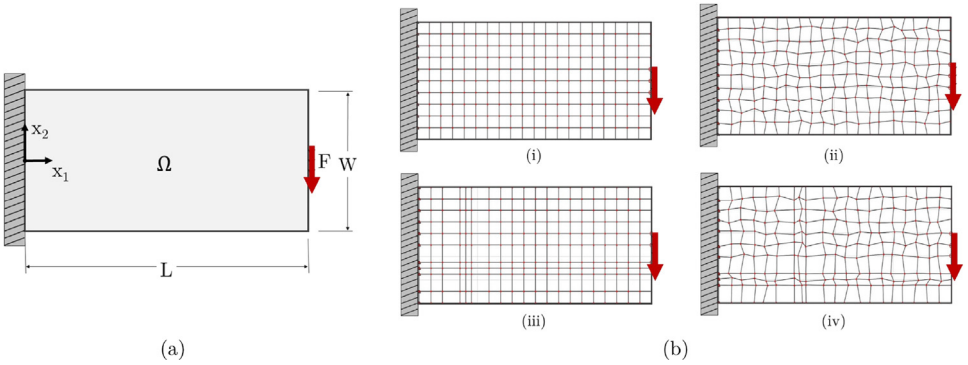


Fig. 6. (a) The cantilever beam problem is used to evaluate the mesh-dependence of MFGNNs for structural mechanics problems; (b) noise generation strategies are used to develop unique training datasets (shown with a 20 × 10 element mesh), including (*i*) noise-free, (*ii*) coordinate $\boldsymbol{x}$ noise, (*iii*) element size noise, and (*iv*) combination of (*ii*) and (*iii*).

3.1. MFGNNs as FEA surrogate models

In the application of MFGNNs as FEA surrogate models, various high-fidelity meshes Ω_h are generated; a selection of these meshes are illustrated in Fig. 6. A cantilever beam problem is considered, and various training strategies are developed to illustrate the MFGNN's accuracy as a FEA surrogate. The domain Ω, illustrated in Fig. 6(*a*), considers a single set of boundary conditions for an elastic material with Young's modulus of 200 GPa and Poisson's ratio of 0.3 under isotropic, 2D, linear-elastic plane-strain conditions. The beam ($L = 2$ m, $W = 1$ m) is subjected to a constant nodal force $F = 10$ MN distributed uniformly across 3 nodes symmetric to the $\boldsymbol{x}_2$ axis. FEA is performed with linear 4-node quadrilateral elements.

Each mesh discretization is characterized by a number of elements $m_1 \times m_2$ corresponding to the $\boldsymbol{x}_1$ and $\boldsymbol{x}_2$ directions. Unique training examples are generated using FEA mesh randomization techniques shown in Fig. 6(*b*). Random meshes are generated by selecting a variety of discretization fidelities (i.e., random m_1, m_2) and applying noise to the mesh discretization. Nodal position noise (Fig. 6(*b*)(*ii*)) is achieved through

$$\boldsymbol{x}_i^{(noise)} = \boldsymbol{x}_i + \boldsymbol{\lambda} h \tag{11}$$

where $\boldsymbol{\lambda}$ is random uniform noise limited by 0.2 and h is the element size. Additional noise is achieved through varying element size by shifting an entire group of nodes (Fig. 6(*b*)(*iii*)) or a combination of strategies (Fig. 6(*b*)(*iv*)). Training strategies are compared using a global mesh evaluated with a GNN and the same mesh subdivided and evaluated with a MFGNN. For these examples, subdivision is performed over a regular grid, so each subgraph mesh is discretized to the same fidelity as the global mesh (Fig. 4).

Table 1 summarizes the attributes embedded into graph nodes and edges. Nodal attributes are defined by $\bar{\boldsymbol{u}}_i$ evaluated at each node, and one-hot identifiers indicate a displacement (1_u) or forced (1_F) condition at the appropriate nodal locations. Edge attributes encode $\Delta\bar{\boldsymbol{u}}_j$, geometric information $\Delta\boldsymbol{x}_{1j}$, $\Delta\boldsymbol{x}_{2j}$, and edge length $\|\boldsymbol{x}_j\|$. The target graph's attributes are the high-fidelity finite element evaluation $\boldsymbol{u}_i$ at each node and relative evaluation $\Delta\boldsymbol{u}_j$ for each edge. As in [38], attributes exclude absolute nodal coordinates to improve generality. Because input geometric features are strictly relational, we impose a focus on node–node relationships.

To ensure equivalency in GNN and MFGNN models, geometric attributes are z-standardized using training dataset statistics, while both input and target displacement attributes are z-standardized using statistics collected from the projection $\bar{\boldsymbol{u}}$. Z-standardization uses the mean μ and standard deviation σ from a collection of data z to produce zero-mean, unit standard deviation data through

$$z_{\mu=0,\sigma=1} = \frac{z-\mu}{\sigma}. \tag{12}$$

Z-standardization via (12) is used in machine learning as data regularization to improve training convergence [46]. In the GNN approach, input attributes are standardized using statistics (μ and σ) collected from the entire training dataset. As an example, the feature $\Delta\boldsymbol{x}_1$ is compiled across all training data, its average and standard deviation are

Table 1
Attributes embedded into the graph data structure are used for learning FEA.

	Graph Node Attributes $\boldsymbol{n}_{i=1:N_n}$	Graph Edge Attributes $\boldsymbol{e}_{j=1:N_e}$
Input Graph	$[\bar{\boldsymbol{u}}, 1_u, 1_F]_i$	$[\Delta\bar{\boldsymbol{u}}, \Delta\boldsymbol{x}_1, \Delta\boldsymbol{x}_2, \|\boldsymbol{x}\|]_j$
Output Graph	$[\boldsymbol{u}]_i$	$[\Delta\boldsymbol{u}]_j$

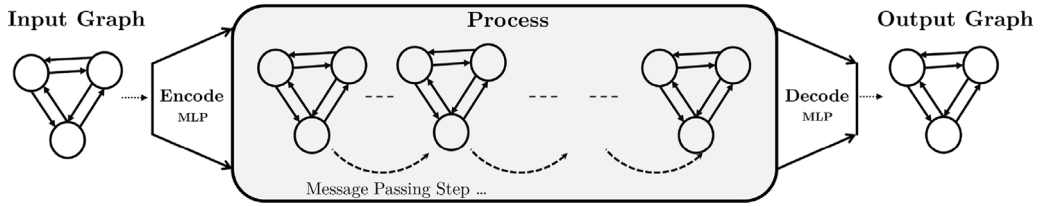


Fig. 7. The encode–process–decode modeling structure [34] uses an MLP to encode graph attributes to a high-dimensional latent space, a sequence of message-passing steps to learn relations within the graph, and a decoder to produce observable outputs.

calculated, and Z-standardization is applied to regularize the GNN input. The GNN then predicts a standardized output regularized using statistics collected from the pre-computed evaluations of $\boldsymbol{u}$. In contrast, the MFGNN collects statistics only within each subgraph and standardizes both inputs ($\bar{\boldsymbol{u}}$) and outputs ($\hat{\boldsymbol{u}}$) using statistics collected from the low-fidelity projection. Because the MFGNN standardization process is localized within each subgraph and entirely dependent on low-fidelity data, the MFGNN is tasked with advancing the model from low-fidelity to high-fidelity evaluation.

As indicated by the red–blue gradient of Fig. 4(a), statistics are collected in the graph G to standardize all attributes in the domain. Local standardization of the MFGNN attributes is achieved within each cell of an 8×4 grid illustrated by the red parent nodes in Fig. 4(b). The local mean μ_s and local standard deviation σ_s produce z-scored displacement attributes within each of these cells.

The message passing and graph analysis implemented here uses the GraphNets library [34] and the encode–process–decode structure presented by Sanchez-Gonzales et al. (Fig. 7) [38]. In this structure, an encoder builds a latent graph for node and edge attributes using a multilayer perceptron (MLP), a class of fully-connected feed-forward DNN. The processor updates the latent graph through a sequence of message-passing steps as in Fig. 1. The decoder reduces the latent space to the target space with another MLP. All MLPs are constructed with 2 hidden layers of size 128 activated by Rectified Linear Unit (ReLU) functions and followed by layer normalization [47]. The GNNs implemented here use 10 message-passing steps, while the MFGNNs use 5 message-passing steps. Because each message-passing step uses two DNNs to update edge and node attributes, the total number of model parameters $\boldsymbol{\theta}$ is a function of the weights and biases in each MLP and the number of message-passing steps.

Table 2 summarizes the models and training strategies used to emulate FEA with both GNNs and MFGNNs. The $\text{GNN}_{0.x}$ models were trained on graphs generated from noisy finite element nodes, as illustrated in Fig. 6(b)(ii). With increasing training data from $n = 500$, $n = 5,000$, $n = 50{,}000$ for $\text{GNN}_{0.1}$, $\text{GNN}_{0.2}$, $\text{GNN}_{0.3}$ respectively, each successive model must learn more general physical information. Generating 50,000 training examples represents a significant time investment ($\sim$ 1 day data generation time), so subsequent models aim to improve model performance with smaller training datasets.

The $\text{GNN}_{1.x}$ models expand the range of mesh fidelities used during training to evaluate the reliance on computationally expensive datasets (Table 2). We limit the training examples to $n = 5,000$ for $\text{GNN}_{1.0}$, which uses higher-fidelity training data than $\text{GNN}_{0.x}$. $\text{GNN}_{1.1}$ is limited further by just $n = 2,500$ training examples with mesh generated from randomized nodal coordinates and element sizes, as illustrated in Fig. 6(b)(iv). All GNNs are informed by a low-fidelity 8×4 finite element evaluation $\tilde{\boldsymbol{u}}$.

Each MFGNN was trained using 100 finite element evaluations performed on global meshes Ω_h divided to 8×4 regular subdomains (Fig. 4(b)). Each subgraph S, representing a mesh ranging from 8×8 to 16×16, is evaluated to infer the high-fidelity finite element solution from its low-fidelity counterpart. To that end, each MFGNN was trained using a variety of low-fidelity meshes and high-fidelity targets (Table 2). Despite the greater variety in training data, only 100 unique FEA evaluations were required to produce 3,200 training examples. $\text{MFGNN}_{1.0}$

Table 2
Models and their corresponding training datasets are presented.

Name	Training Examples, Iterations	Mesh Noise	Projection: Low-Fidelity Mesh	Subgraph Mesh	Target: High-Fidelity Mesh
$GNN_{0.1}$	500, 10^6	Nodal $\boldsymbol{x}$	8 × 4	–	32 × 16 - 48 × 24
$GNN_{0.2}$	5,000, 10^6	Nodal $\boldsymbol{x}$	8 × 4	–	32 × 16 - 48 × 24
$GNN_{0.3}$	50,000, $2 \cdot 10^6$	Nodal $\boldsymbol{x}$	8 × 4	–	32 × 16 - 48 × 24
$GNN_{1.0}$	5,000, 10^6	Nodal $\boldsymbol{x}$	8 × 4	–	32 × 16 - 64 × 32
$GNN_{1.1}$	2,500, 10^6	Nodal $\boldsymbol{x}$, element size	8 × 4	–	32 × 16 - 64 × 32
$MFGNN_{1.0}$	3,200, $2 \cdot 10^6$	None	16 × 8 - 30 × 16	8 × 8 - 18 × 18	64 × 32 - 180 × 90
$MFGNN_{2.0}$	3,200, $2 \cdot 10^6$	Nodal $\boldsymbol{x}$, element size	16 × 8 - 30 × 16	8 × 8 - 18 × 18	64 × 32 - 180 × 90

implements the MFGNN framework without any mesh noise (Fig. 6(*b*)(*i*)), while $MFGNN_{2.0}$ implements full noise in nodal position and element size (Fig. 6(*b*)(*iv*)).

Each model was trained using the Adam optimizer [48] with an exponentially decaying learning rate from 10^{-4} to 10^{-5} over 10^6 iterations on a single NVIDIA v100 GPU. Training was executed until overfitting was observed, resulting in models trained for 10^6 or $2 \cdot 10^6$ iterations. Models were trained using mean-squared-error (MSE) loss for randomized mini-batches of 2 training examples. MSE loss may be expressed as

$$\mathcal{L}_{\text{MSE}}(\boldsymbol{\theta}, \boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{1}{N}(\boldsymbol{y} - \hat{\boldsymbol{y}})^T(\boldsymbol{y} - \hat{\boldsymbol{y}}) \tag{13}$$

for model parameters $\boldsymbol{\theta}$, standardized target attributes $\boldsymbol{y}$, model output attributes $\hat{\boldsymbol{y}}$, and N total attributes. $\mathcal{L}_{\text{MSE}}$ is averaged for each target attribute across the output's nodes and edges.

We also experiment with physics-informed objective functions to improve model accuracy and stability. In the spirit of physics-informed neural networks [5] and other energy-based objectives [14,49,50], we introduce an additional objective term in the form of internal potential energy $\mathcal{U}$. Using the same model framework as $GNN_{1.1}$, model $GNN_{1.1}^{\mathcal{U}}$ utilizes this new objective term $\mathcal{U}$ evaluated as the internal potential energy of a linear elastic system,

$$\mathcal{U}(\boldsymbol{K}, \boldsymbol{u}) = \frac{1}{2}\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{u} \tag{14}$$

for a global finite element stiffness matrix $\boldsymbol{K}$ assembled from the high-fidelity mesh and a finite element nodal displacement $\boldsymbol{u}$. Because the total potential energy of the system $\mathcal{E}$ is determined from internal $\frac{1}{2}\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{u}$ and external $\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{f}$ contributions,

$$\mathcal{E} = \frac{1}{2}\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{u} - \boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{f} = -\frac{1}{2}\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{K}\boldsymbol{u} = -\mathcal{U}, \tag{15}$$

for the equilibrium condition $\boldsymbol{K}\boldsymbol{u} = \boldsymbol{f}$. The finite element solution's equilibrium is characterized by a minimization of the potential energy of the system [3], so the finite element solution $\boldsymbol{u}$ is a minimum of (15). We incorporate this physical characteristic through a new objective term. The new term uses the internal potential energy approximated by the GNN ($\mathcal{U}(\boldsymbol{K}, \hat{\boldsymbol{u}}_{\text{nodal}})$) compared to the minimum internal potential energy guaranteed by FEA ($\mathcal{U}(\boldsymbol{K}, \boldsymbol{u})$) in

$$\mathcal{L} = \mathcal{L}_{MSE} + \alpha\left(\frac{\mathcal{U} - \hat{\mathcal{U}}}{\mathcal{U}}\right)^2 \tag{16}$$

where α is an arbitrary scaling factor we set to $1/\mathcal{U}$. The additional energy-based objective term is designed to enforce an accurate prediction of the system's internal potential energy which is directly correlated to an accurate displacement analysis (cf. (15)).

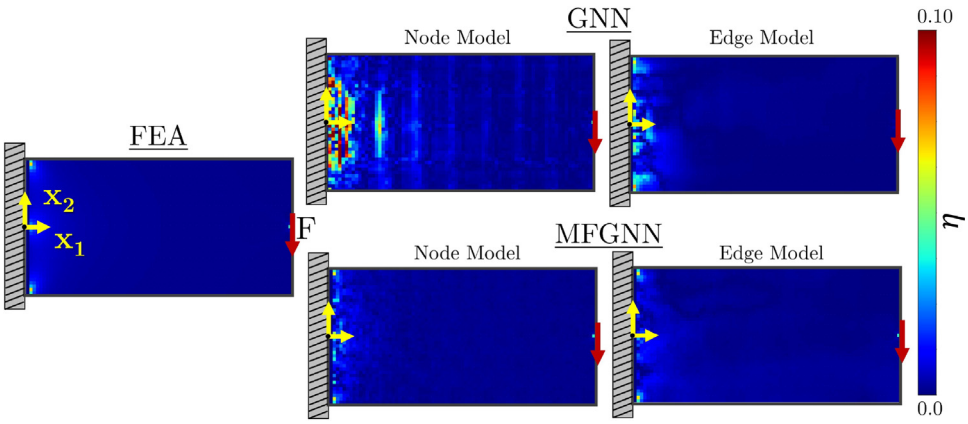


Fig. 8. Relative absolute error $\boldsymbol{\eta}$ in the predicted displacement is shown for a 80 × 40 evaluation mesh compared to the converged 960 × 480 result. Despite generating training data nearly 10 times more efficiently and training in half the time, MFGNNs improve performance for high-fidelity analysis.

Models were evaluated based on their conformity to (13). Additionally, the model output displacement $\hat{\boldsymbol{u}}$ was compared to the converged finite element evaluation $\boldsymbol{u}$ using relative absolute error at each finite element node,

$$\boldsymbol{\eta} = \left| \frac{\boldsymbol{u} - \hat{\boldsymbol{u}}}{\boldsymbol{u}} \right|, \tag{17}$$

where $\boldsymbol{u}$ was assumed from FEA of a super high-fidelity, uniform 960 × 480 mesh that is hereafter referred to as the *converged* solution.

Fig. 8 compares nodal error in the predicted displacement $\hat{\boldsymbol{u}}$ for an 80 × 40 regular mesh; $\hat{\boldsymbol{u}}$ is evaluated using FEA, $\text{GNN}_{1.1}$, and $\text{MFGNN}_{1.0}$. Fig. 8's node model extracts $\hat{\boldsymbol{u}}$ directly from graph nodes and achieved $\boldsymbol{\eta}_{avg}$ of 0.242%, 0.749%, and 0.251% for the FEA, GNN, and MFGNN respectively. When the nodal displacement is instead recovered from the model's edge evaluations $\Delta \boldsymbol{u}_j$ using (6) and the prescribed boundary condition $\hat{\boldsymbol{u}} = \boldsymbol{0}$ at $x_1 = 0$, the average error $\boldsymbol{\eta}_{avg}$ is 0.242%, 0.239%, and 0.236% for the FEA, GNN, and MFGNN respectively. The MFGNN achieves comparable performance to FEA despite having fewer model parameters than the GNN equivalent. As shown in Fig. 8 at $x_1 = 0$, the MFGNN's subgraph analysis has significantly improved relative accuracy near the boundary condition, and the model converges to a local error tolerance for each subdomain. Alternatively, the GNN's global graph analysis does not resolve the model's prediction accurately; Fig. 8's GNN Node Model shows striations of error that overlap with the low-fidelity mesh, indicating the global graph's inability to resolve local node analysis. Additionally, edge-based solutions via (6) were inherently more stable. As each nodal value is compiled as an average based on its incoming edges, the edge model solution becomes more error resistant.

Table 3 presents error analysis of the GNN's edge model evaluation across a wide range of test cases. The improvement in $\mathcal{L}_{\text{MSE}}$ from $7.36 \cdot 10^{-6}$ with $\text{GNN}_{0.1}$ and 500 training examples to $1.93 \cdot 10^{-6}$ with $\text{GNN}_{0.3}$ and 50,000 training examples establishes a correlation between model performance and training dataset size. Relative error performance, however, is more correlated to variety – not quantity – of training data; $\text{GNN}_{1.x}$, trained on fewer than 5,000 training examples with more mesh noise and a broader range of mesh fidelities, achieved the best $\boldsymbol{\eta}_{Avg.}$ among all GNNs despite the relatively small amount of training data invested (∼ 2h data generation time for 2,500 examples) (Table 3). $\text{GNN}^{\mathcal{U}}_{1.1}$ implemented a physics-based objective term for potential energy in the GNN's optimization. In practice, this new objective did improve $\mathcal{L}_{\text{MSE}}$ from $19.51 \cdot 10^{-6}$ to $6.53 \cdot 10^{-6}$, however $\boldsymbol{\eta}_{Avg.}$ increased; the additional loss term induced some localized overfitting that deterred model generalizability.

The MFGNN's $\boldsymbol{\eta}$ performance within the training domain was consistently similar to FEA and improved over every iteration of GNN (cf. Tables 3 and 4). $\text{MFGNN}_{1.0}$ achieved the best results with $\boldsymbol{\eta}_{Avg.}$ within 1% of FEA's $\boldsymbol{\eta}_{Avg.}$ for the high-fidelity test cases. The magnitude of training loss $\mathcal{L}_{\text{MSE}}$ increased for MFGNNs compared to GNNs. This increase may be attributed to the local standardization within each subgraph and is not indicative of decreased accuracy in the predicted displacement, as supported by the MFGNN's improved $\boldsymbol{\eta}$ performance (Table 4). Mesh randomization of nodal position and element size improved performance in $\text{GNN}_{1.1}$, but decreased performance in

Table 3
GNN performance, including training loss, across different mesh fidelities compared to the converged (960 × 480 element) solution.

Model Name	Training Loss $\mathcal{L}_{\text{MSE}}(\boldsymbol{\theta}) \cdot 10^{-6}$	$\boldsymbol{\eta}_{Avg.}$, $\boldsymbol{\eta}_{Max.}(\cdot 10^3)$:					$r^2_{(h)}$
		24 × 12	40 × 20	48 × 24	80 × 40	120 × 60	
FEA	–	13.23, 76.39	6.49, 70.70	5.02, 69.64	2.42, 67.59	1.33, 65.99	1.00
$\text{GNN}_{0.1}$	7.36	20.38, 160.94	5.96, 71.33	4.31, 68.79	22.35, 1634.88	40.07, 2907.91	0.542
$\text{GNN}_{0.2}$	7.32	18.81, 108.64	6.56, 70.62	5.01, 69.39	16.89, 507.10	110.94, 2451.84	0.620
$\text{GNN}_{0.3}$	1.93	14.50, 90.50	6.76, 71.08	5.23, 69.89	13.08, 573.82	107.34, 1909.39	0.621
$\text{GNN}_{1.0}$	12.37	15.64, 97.27	5.79, 69.94	4.38, 68.66	4.20, 182.53	37.93, 619.07	0.396
$\text{GNN}_{1.1}$	19.51	10.32, 81.85	6.10, 71.46	4.56, 69.16	2.39, 74.28	12.57, 686.50	0.303
$\text{GNN}^{\mathcal{U}}_{1.1}$	6.53	19.35, 148.36	5.29, 69.38	1.87, 67.73	6.39, 67.99	17.42, 472.56	0.344

Table 4
MFGNN performance, including training loss, across different mesh fidelities compared to the converged (960 × 480 element) solution.

Model Name	Training Loss $\mathcal{L}_{\text{MSE}}(\boldsymbol{\theta}) \cdot 10^{-6}$	$\boldsymbol{\eta}_{Avg.}$, $\boldsymbol{\eta}_{Max.}(\cdot 10^3)$:					$r^2_{(h)}$
		24 × 12	40 × 20	48 × 24	80 × 40	120 × 60	
FEA	–	13.23, 76.39	6.49, 70.70	5.02, 69.64	2.42, 67.59	1.33, 65.99	1.00
$\text{MFGNN}_{1.0}$	55.62	43.22, 280.11	4.03, 65.87	2.93, 47.25	2.36, 65.62	1.23, 66.61	0.947
$\text{MFGNN}_{2.0}$	141.25	87.55, 227.37	3.17, 59.54	2.96, 57.45	2.81, 82.02	1.86, 70.04	0.896
$\text{MFGNN}^{(BC)}_{1.0}$[a]	2140.07	90.67, 876.54	61.76, 693.70	80.00, 873.42	119.43, 2613.59	45.45, 3113.11	0.403

[a] $\text{MFGNN}^{(BC)}_{1.0}$ is shown here for completeness but is not directly comparable to $\text{MFGNN}_{1.0}$ or $\text{MFGNN}_{2.0}$ (cf. Section 4.)

$\text{MFGNN}_{2.0}$. The global graph used in GNNs generally captures mesh randomization effectively, while the subgraph representations used in MFGNNs are more susceptible to local irregularities caused by the randomization process.

3.2. MFGNNs to learn finite element convergence

This section explores the performance of both GNNs and MFGNNs in learning to emulate finite element convergence using low-fidelity training data. Although the physics simulations evaluated here and shown in Fig. 6 are relatively simple, learning finite element convergence requires the model to generalize away from its training domain to solve systems with over 10 times more degrees of freedom. Section 3.1 indicates that the success of this generalization is largely dependent on the quality and diversity of training data. The subsequent application illustrates how training data may be manipulated to improve model performance and ultimately increase utility for engineering applications.

Error in the FEA approximation is largely induced by idealized material models, discretization of a continuous domain, and numerical integration in space [3,51]. This work considers *h-convergence*, which refers to the mesh refinement strategy for reducing error. Consider the nodal displacement's exact solution $\mathbf{u}$ and approximate solution $\hat{\mathbf{u}}$. The convergence of the finite element space may then be characterized using

$$\left\| \boldsymbol{u} - \hat{\boldsymbol{u}} \right\|_2 \leq ch^k = \frac{c}{N^k} \tag{18}$$

for the independent constant c, element size h, and degrees of freedom N [2,3].

To evaluate each model's performance in learning finite element convergence, (17) and (18) are evaluated for a range of target meshes. Conformity to (18) is evaluated using linear regression and measured using the r-squared linear correlation in log space $r^2_{(h)}$. Convergence behavior is evaluated using regular meshes across a range of fidelities. The results, summarized in Tables 3 and 4, indicate improved accuracy and stability of the MFGNN. The $r^2_{(h)}$ correlation generally improves with the MFGNNs, indicating convergence behavior similar to FEA.

Fig. 9 illustrates the GNN's clear dependence on training data. Despite training on 50,000 unique meshes with $\text{GNN}_{0.3}$, the mesh-dependent behavior of FEA was not accurately predicted beyond 2 to 3 times the mesh fidelities in the training range (Fig. 9(*a*)). Within this training range, however, the GNN learned the mesh dependence of

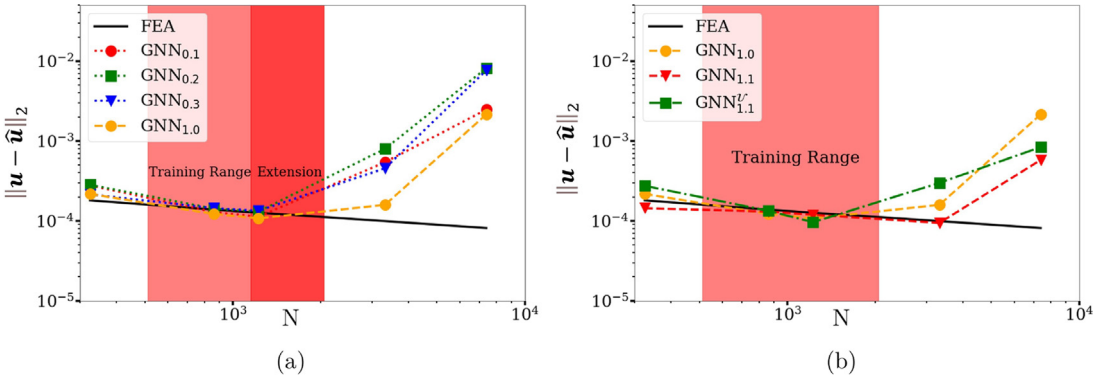


Fig. 9. Log-space h-convergence is shown for GNN models trained on different datasets. (a) The $\text{GNN}_{0.x}$ models, trained on varying amounts of training data, are compared to $\text{GNN}_{1.0}$, trained on less data but with an extended training range (cf. Table 2); (b) the $\text{GNN}_{1.x}$ models are shown, comparing the training strategies used to improve performance under training dataset restrictions (cf. Table 2).

FEA for this domain, producing stable and increasingly accurate displacement fields $\hat{\boldsymbol{u}}$. By extending the training range and applying a more varied mesh noise scheme in $\text{GNN}_{1.1}$, we achieved comparable convergence behavior to $\text{GNN}_{0.3}$ with 5% of the training examples (Fig. 9(*b*)).

MFGNNs expand the range of stable analysis with a more efficient training regime. Because multiple subgraphs can be derived from a single high-fidelity finite element evaluation, the MFGNN's training examples can be generated orders of magnitude faster. Fig. 10(*a*) demonstrates the MFGNN's improved data generation efficiency by comparing the computational performance for the test examples in Tables 3 and 4. Because MFGNNs require execution of each subdomain and post-processing to reassemble the global mesh, the MFGNN sacrifices execution efficiency. This compromise allows for the inclusion of a more varied, higher-fidelity, training dataset that was generated orders of magnitude faster than the equivalent dataset required to train GNNs. The higher-fidelity training dataset (Table 2) spans the range of test examples without directly including the test meshes. The MFGNN models trained using this dataset must abstract the analysis in each subdomain to emulate finite-element convergence behavior.

The resulting MFGNN model can emulate FEA convergence across a range of mesh fidelities (Fig. 10(*b*)). $\text{MFGNN}_{1.0}$, the best performing MFGNN, achieved an $r^2_{(h)}$ convergence correlation of 0.947. With just 100 high-fidelity training examples, the MFGNN accurately emulates the h-convergence behavior of FEA, producing stable analysis across domains spanning multiple orders of magnitude. The MFGNN's improved performance is due to the extended training dataset and accurate analysis in each subdomain. The comparison in Fig. 10(*b*) demonstrates the practicality of subdomain-level training, as the efficient MFGNN training strategy captured h-convergence behavior. Because both the GNN and MFGNN did not learn a governing convergence behavior beyond their respective training ranges, the MFGNN framework is necessary for practical implementation of FEA training data for generalized surrogate models.

The convergence behavior of the FEA training data is intimately connected to the performance of both MFGNN and GNN models. Relative to the converged 960 × 480 FEA solution, the models $\text{GNN}_{1.0}$, $\text{GNN}_{1.1}$, and $\text{GNN}^{\mathcal{U}}_{1.1}$ all produced more accurate solutions compared to the equivalent FEA at 40 × 20 and 48 × 24 mesh fidelities (Fig. 9(*b*)). Because we also observed error increase at very low (24 × 12) and very high (120 × 60) fidelities, there is evidence that the GNNs converge to some general representation of the entire training range that is independent of the graph fidelity. In contrast, the MFGNN models $\text{MFGNN}_{1.0}$ and $\text{MFGNN}_{2.0}$ conform more precisely to the convergence behavior defined by (18). The localized analysis combined with efficient subgraph representation of the training data allowed the MFGNNs to match their predictions more precisely to each mesh fidelity, resulting in finite-element-like convergence behavior Fig. 10(*b*). The error metrics presented in Table 4 for $\text{MFGNN}_{1.0}$ also indicate that the MFGNN can extended the improved accuracy of the high-fidelity training data to lower fidelity meshes (e.g., $\boldsymbol{\eta}_{avg}$ improved at 40 × 20 and 48 × 24 mesh fidelities). Unlike the GNN, this improved accuracy continued throughout the training domain, emulating FEA convergence behavior as the mesh was refined. Because both the GNN and MFGNN models were so directly impacted by the training data formulation, other components

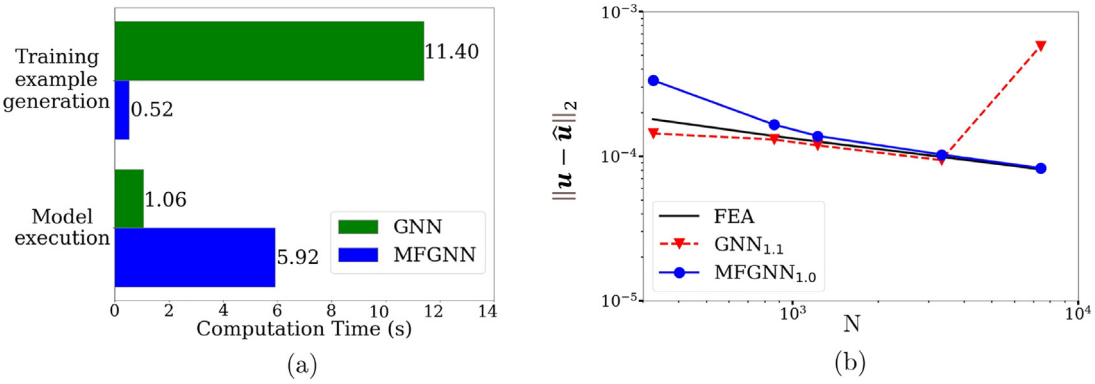


Fig. 10. Efficient training data generation and model training increase the MFGNN's utility for simulating physics. (a) Computational times are compared for the time to generate a single training example (i.e., mesh generation, finite element evaluation, and graph embeddings) and the model execution time for evaluating convergence behavior. (b) Convergence behavior is compared for GNN and MFGNN models.

of the FEA training data may also affect performance. These components, including element type, shape function formulation, and mesh quality, must be considered when generating a FEA training dataset.

4. Extending MFGNNs to generalized physics

The aforementioned applications of MFGNNs focused on the mesh-dependence of FEA training data. The MFGNN's subdomain analysis and local standardization improved training efficiency and model accuracy. An additional effect of this subdomain analysis is the abstraction of a specific boundary-value problem to a generalized representation. To examine the universality of this representation, an extended problem domain is considered. While still governed by (7), a variety of prescribed conditions are used to develop a diverse set of structural responses (Fig. 11(*a*)).

Training data for the generalized model consists of cantilever-style beams. The beam ($L = 2$ m, $W = 1$ m) may be constrained at one or both ends. Uniform loads of magnitude $F = 10$ MN are applied randomly at a free surface of the beam. Each load may be uniformly distributed across an entire surface or otherwise concentrated to 3 nodes at beginning, middle, or end of the surface. The angle of the load is randomized in 45° intervals. The material is kept constant and isotropic with $E = 200$ GPa and $\mu = 0.3$. Loads and boundary conditions are uniformly randomized within the training dataset to create sufficiently diverse training data; Fig. 11(*a*) illustrates three unique load cases within the training domain. With this new dataset, the model $\text{MFGNN}_{1.0}^{(BC)}$ was trained to examine the MFGNN's ability to transfer information from a few simple load cases to a general physics model.

Using the same MFGNN properties of $\text{MFGNN}_{1.0}$ as shown in Table 2, $\text{MFGNN}_{1.0}^{(BC)}$ was trained to learn generalized 2D solid elastomechanics using 100 finite element evaluations (resulting in 3,200 subgraphs for training data) from this new dataset with randomized boundary conditions (BCs). As expected, performance in both accuracy and convergence behavior decreases relative to $\text{MFGNN}_{1.0}$ when evaluated on a single set of BCs (Table 4). Due to the variable training dataset, however, $\text{MFGNN}_{1.0}^{(BC)}$ is stable across a variety of domains not present in the training data. This generalization is possible because of the subdomain abstraction and low-fidelity projections used to inform MFGNNs.

Fig. 11(*b*) demonstrates an example of this generalizability, in which a circular inclusion is introduced to the domain. This inclusion – marked by its conforming mesh – is defined by a soft material ($E = 200$ kPa, $\mu = 0.3$). The low-fidelity projection $\tilde{\boldsymbol{u}}$ is calculated using a 24 × 12 mesh, while the MFGNN evaluates a 48 × 24 mesh. The input projection generated from FEA on a 24 × 12 mesh has a relative accuracy of $\boldsymbol{\eta}_{Avg.} = 10.0\%$ compared to 48 × 24 FEA; with this subgraph projection as an input, the MFGNN improves the relative accuracy relative to $\boldsymbol{\eta}_{Avg.} = 6.7\%$ evaluated on the 48 × 24 mesh (Fig. 11(*b*)). Fig. 11(*c*) presents another generalization in which an L-bracket problem was evaluated. Using a 144 element low-fidelity analysis with $\boldsymbol{\eta}_{Avg.} = 6.3\%$ relative to 2304 element FEA, the MFGNN evaluated the 2304 element L-bracket with $\boldsymbol{\eta}_{Avg.} = 4.6\%$.

Just as convolutions work as feature extractors in convolutional neural networks [52], the subgraph representations of MFGNNs extract an abstract representation of the underlying physics. The abstraction has helped to improve

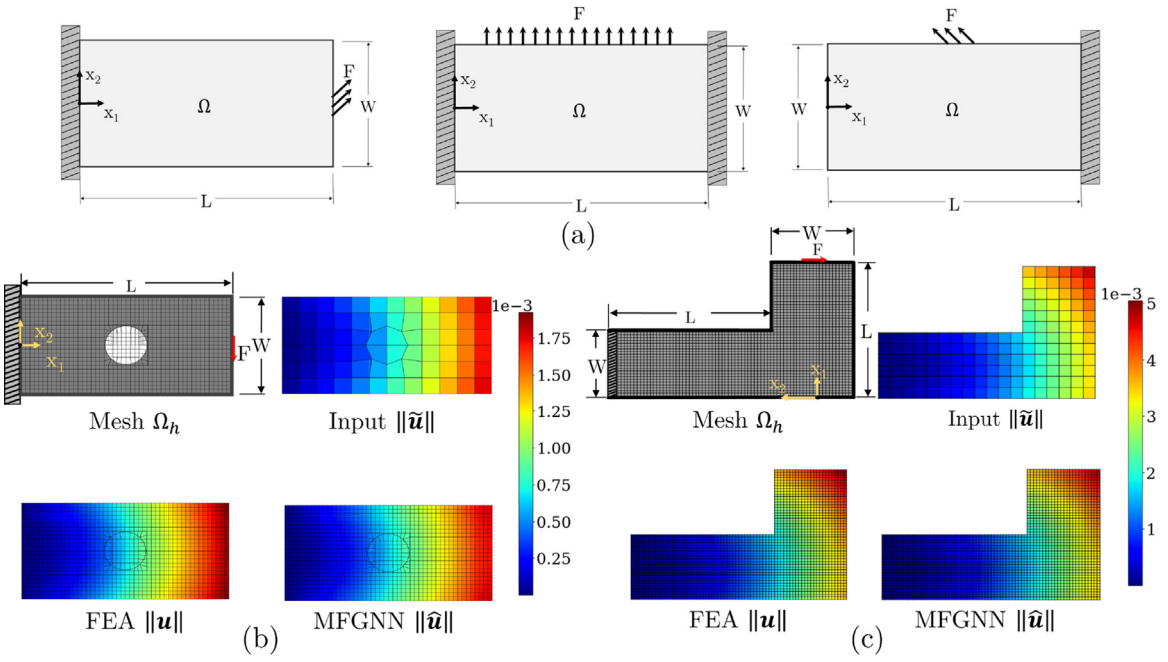


Fig. 11. Model generalizability is improved with MFGNNs, as the abstraction into subdomains allows for a more generalized representation of the underlying physics. (a) A few samples of the varied dataset used to train $\text{MFGNN}_{1.0}^{(BC)}$ are shown. (b) $\text{MFGNN}_{1.0}^{(BC)}$ is generalized to a conforming mesh with a circular inclusion defined by a weak ($E = 200$ kPa) material. (c) $\text{MFGNN}_{1.0}^{(BC)}$ is extended to an L-bracket geometry.

the generalizability of the MFGNN to new physics, but the MFGNN is still not an independent general solver. The input graph for the MFGNN relies on some low-fidelity FEA to remain stable. Convergence is not guaranteed with this model, only emulated. Finally, relative accuracy may suffer when the target feature spans multiple orders of magnitude, as $\mathcal{L}_{\text{MSE}}$ tends to produce a global tolerance.

Fig. 12 illustrates another pair of generalization tests. For a beam ($L = 4$, $W = 1$), a pair of prescribed displacements ($u_p = 0.2$) are applied to produce a shear load (Fig. 12(a)). Random uniform noise ($\boldsymbol{\lambda} \leq 0.2$) was also applied to nodal coordinates. Using an input projection evaluated on a 16×8 mesh, $\text{MFGNN}_{1.0}^{(BC)}$ was implemented to predict nodal displacements on a 80×40 mesh. The 16×8 input projection has an error $\boldsymbol{\eta}_{Avg.} = 36.5\%$ compared to the 80×40 FEA; $\text{MFGNN}_{1.0}^{(BC)}$ modeled a displacement with $\boldsymbol{\eta}_{Avg.} = 2.54\%$ compared to the 80×40 FEA. Fig. 12(b) presents another generalization case for a beam ($L = 2$, $W = 1$) subjected to prescribed displacement ($u_p = 0.2$) and nodal force ($F = 10$ MN). An input projection from a 16×8 mesh is used to inform $\text{MFGNN}_{1.0}^{(BC)}$ analysis of a 80×40 mesh. The low-fidelity input projection ($\boldsymbol{\eta}_{Avg.} = 6.29\%$ relative to 80×40 FEA), was used to produce a modeled displacement with $\boldsymbol{\eta}_{Avg.} = 2.43\%$ relative to 80×40 FEA.

These generalization tests introduced new conditions that differed significantly from the training domain, including void-like inclusions, geometric changes, modified boundary conditions, and mixed loading conditions. Where the GNN is less-suited towards general applications, the MFGNN's subgraph operations remain stable across the edge cases in Figs. 11 and 12. The simplified training strategy (Fig. 11(a)), unfortunately, is not enough for a complete surrogate model for 2D structural mechanics. As discussed in Appendix A.1, $\text{MFGNN}_{1.0}^{(BC)}$ is not yet an adequate replacement for more general interpolation schemes. Future work is necessary to develop the MFGNN as a general physics solver. Changes in the training dataset including irregular geometries, conforming mesh, and mixed loading will likely improve the MFGNN as a general solver. We view the MFGNN's emulation of FEA and efficient domain abstraction as necessary first steps towards a more general – yet still computationally viable – alternative to traditional physics-based approaches.

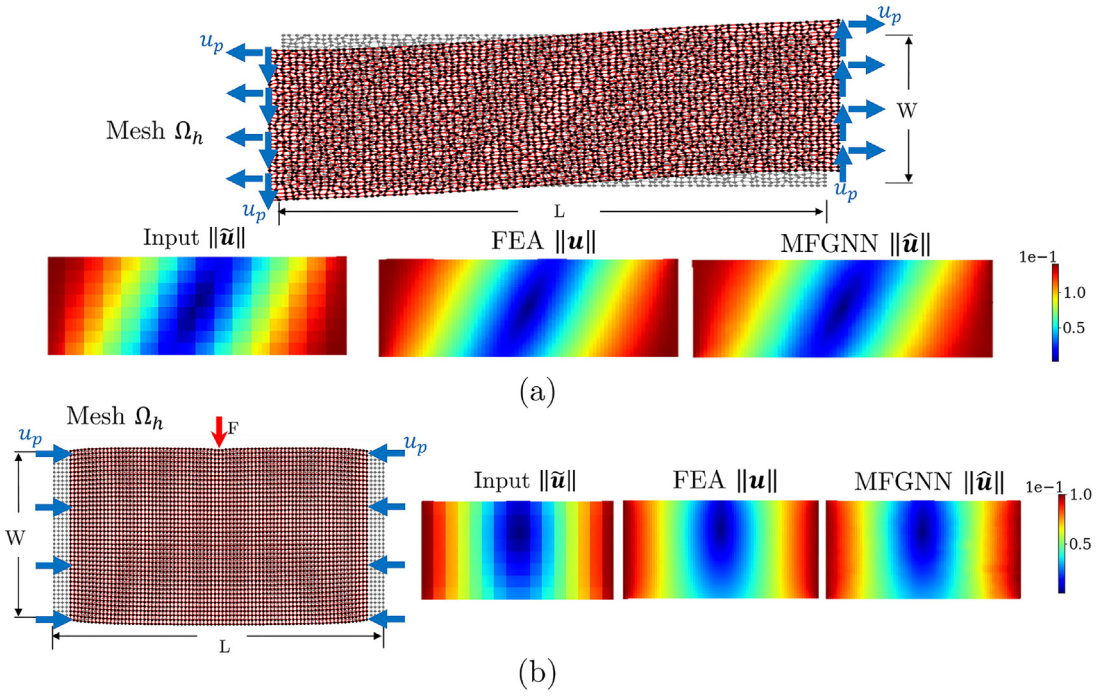


Fig. 12. The MFGNN was examined for loading conditions sufficiently different than the examples used during training. (a) A beam subjected to prescribed nodal displacements u_p is evaluated on the 80 × 40 mesh Ω_h using the low-fidelity input projection $\tilde{\boldsymbol{u}}$ to produce the accurate approximation $\hat{\boldsymbol{u}}$; similarly, (b) a beam subjected to nodal force F and prescribed displacements u_p is evaluated on an 80 × 40 mesh Ω_h.

5. Conclusions

Graph-based machine learning is particularly appealing for modeling physical systems, as the graph readily accommodates domain discretization at multiple fidelities. The ability to learn relationships between points in this discretization through message passing is a promising approach towards performing like traditional numerical solvers. In this paper, Graph Neural Network modeling techniques are employed to learn FEA convergence behavior. The MFGNN model was proposed as a new framework for graph-based machine learning. In this formulation, a low-fidelity projection informs the model's input, then the MFGNN operates across arbitrary subdomains in the form of subgraphs. Each subgraph is a local representation of the physical system, so the model is able to create an abstract representation of the modeled physics. This technique improved generalizability over traditional GNNs and was able to accurately analyze previously-unseen boundary conditions.

To study the relationship between MFGNN and FEA convergence, a combination of experimental sampling of the domain and randomized data generation was used to understand convergence behavior. Generalization from low-fidelity projections to high-fidelity analysis remained stable across a range of test examples. MFGNN models emulated FEA h-convergence within their training domains but do not offer guaranteed convergence behavior. We have found that generation of training data through sampling of high-fidelity subdomains is the most efficient means to develop a generalized solver.

Although the MFGNN has demonstrated progress towards a generalized physical solver, more work is necessary to compete with traditional numerical methods. To further evaluate the MFGNN, more diverse physics should be explored. Multiphysics and multiscale modeling are potential candidates for implementing the MFGNN and could be used to further evaluate the model's generalizability. Furthermore, model training objectives may need to be re-evaluated to emphasize criteria important for physical simulation such as relative error and smoothness. Finally, to improve the usefulness of machine learning for physical simulation, it is crucial to maximize the information gain from each training example. We have shown that creating arbitrary subdomains may improve this information gain, but this action should be investigated further.

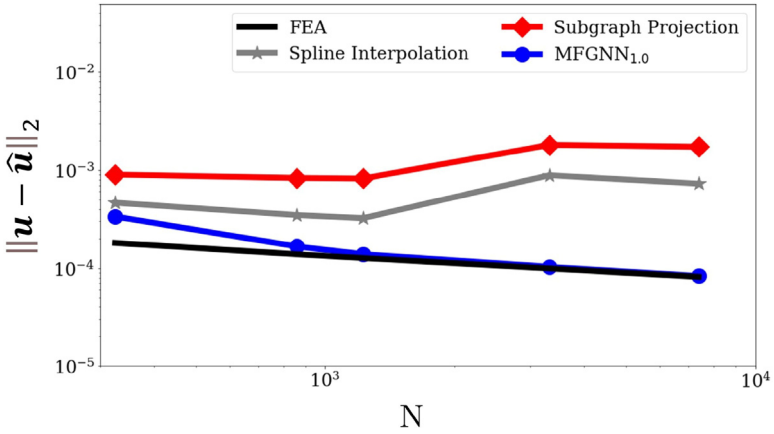


Fig. A.13. Convergence behavior is shown for the smooth-bivariate-spline interpolation, subgraph interpolation, and $\text{MFGNN}_{1.0}$ compared to FEA.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

The authors acknowledge support from Drexel University. N. Black is grateful for support from the GAANN Grant [grant number P200A190036]. The work is also supported by the NSF CAREER Award CMMI-2143422. N. Black would also like to acknowledge the fundamental contributions of Dr. Daniel A. Tortorelli to the development of computational algorithms for solid mechanics. The authors would also like to thank Reza Pejman for insightful discussions related to this work. The code to reproduce this work, as well as an illustrative tutorial, is available via github.com/MCMB-Lab/MFGNNs.

Appendix. Comparison with interpolation schemes

The MFGNN framework links an interpolation of some low-fidelity input (in our case, projected values of the displacement, cf. (10)) to a high-fidelity analysis. As a baseline for comparison, the MFGNN may be compared to traditional interpolation schemes. In this section, a nearest neighbor, linear, cubic, and smooth-bivariate-spline interpolation are compared to high-fidelity FEA, the MFGNN, and the subgraph interpolation (i.e., the MFGNN's input). The nearest neighbor, linear, cubic, and smooth-bivariate-spline interpolation were implemented using the SciPy interpolation API [53] and the appropriate grid coordinates for the low- and high-fidelity mesh.

Table A.5 shares the $\boldsymbol{\eta}$ performance for the interpolation schemes across a range of fidelities. Relative to the converged 960×480 solution, the $\text{MFGNN}_{1.0}$ is the only scheme to approach FEA-like accuracy. Furthermore, the convergence behavior illustrated in Fig. A.13 shows the MFGNN's conformance to FEA mesh-dependence.

A.1. Interpolation schemes across different boundary conditions

Table A.6 compares the interpolation schemes for the extension examples. For the circular inclusion, L-bracket, beam in shear, and mixed loading case, $\boldsymbol{\eta}_{\text{avg}} \cdot 10^3$ is compared. We also include the models $\text{MFGNN}_{1.0}$ and $\text{GNN}_{1.1}$ which were trained on a single set of boundary conditions. Although these models are not appropriate for general modeling, their comparison distinguishes the superiority of the MFGNN framework under general conditions. Unlike the GNN, the subgraph-informed analysis in the MFGNN allows for general application to a range of problems. While $\text{MFGNN}_{1.0}^{(\widetilde{BC})}$ often outperforms simpler interpolation schemes it is clear that the simple training approach shown in Fig. 11(a) cannot cover all linear elastic structural mechanics problems in 2D.

Table A.5
Performance for various interpolations schemes across different mesh fidelities is compared to the converged (960× 480 element) solution.

Model Name	$\eta_{Avg.}$, $\eta_{Max.}$($\cdot 10^3$):					$r^2_{(h)}$
	24 × 12	40 × 20	48 × 24	80 × 40	120 × 60	
FEA	13.23, 76.39	6.49, 70.70	5.02, 69.64	2.42, 67.59	1.33, 65.99	1.00
$MFGNN_{1.0}$	43.22, 280.11	4.03, 65.87	2.93, 47.25	2.36, 65.62	1.23, 66.61	0.947
Nearest Neighbor	139.63, 1000.00	92.31, 1510.28	82.37, 2345.05	101.97, 1472.61	73.45, 1423.14	0.709
Linear	38.04, 519.43	19.84, 703.69	15.00, 502.53	29.91, 1695.83	16.49, 1448.02	0.358
Cubic	35.25, 148.64	17.90, 186.96	14.03, 211.34	26.86, 780.06	15.12, 840.27	0.359
Spline	32.35, 215.16	21.01, 969.76	18.10, 538.12	28.97, 2015.43	29.94, 8049.48	0.429
Subgraph Projection	73.12, 797.66	50.20, 814.35	43.52, 809.87	73.92, 4691.58	50.51, 4061.27	0.679

Table A.6
Relative error, reported as $\eta_{\text{avg}} \cdot 10^3$ for the target finite element mesh fidelity, is shown for various interpolation schemes; the mean error relative error per DOF is then reported for the given examples.

	Example $\eta_{\text{avg}} \cdot 10^3$ (DOF)				Mean per DOF
	Circular inclusion (2450)	L-bracket (4900)	Beam in Shear (6642)	Mixed Loading (6642)	
$MFGNN^{(BC)}_{1.0}$	67.4	45.8	**25.4**	24.3	**0.0111**
$MFGNN_{1.0}$	**50.0**	42.1	40.7	279	0.0193
$GNN_{1.1}$	67.2	428	1014	992	0.104
Subgraph Projection	100	63.3	365	62.9	0.0295
Nearest Neighbor	138	95.8	95.1	79.7	0.0255
Linear	55.7	31.2	138	10.3	0.0129
Cubic	53.1	**27.9**	139	**9.90**	0.0124
Spline	63.0	58.9	145	41.0	0.0164

References

[1] I. Smith, D. Griffiths, L. Margetts, Programming the Finite Element Method, in: Wiley Series in Computational Mechanics Series, Wiley, 2013, URL https://books.google.com/books?id=ZbtiAAAAQBAJ.

[2] O. Zienkiewicz, R. Taylor, J. Zhu, Chapter 15 - errors, recovery processes, and error estimates, in: O. Zienkiewicz, R. Taylor, J. Zhu (Eds.), The Finite Element Method: Its Basis and Fundamentals, seventh ed., Butterworth-Heinemann, Oxford, 2013, pp. 493–543, http://dx.doi.org/10.1016/B978-1-85617-633-0.00015-0, URL https://www.sciencedirect.com/science/article/pii/B9781856176330000150.

[3] K.-J. Bathe, Finite Element Procedures, second ed., Bathe, Klaus-Jurgen, Watertown, MA, 2014, p. 1043.

[4] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, K. Willcox, S. Lee, Report (2019-02-10 2019), 2019, http://dx.doi.org/10.2172/1478744, https://www.osti.gov/biblio/1478744.

[5] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, J. Comput. Phys. 378 (2019) 686–707, http://dx.doi.org/10.1016/j.jcp.2018.10.045, URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

[6] W. Yan, S. Lin, O.L. Kafka, Y. Lian, C. Yu, Z. Liu, J. Yan, S. Wolff, H. Wu, E. Ndip-Agbor, M. Mozaffar, K. Ehmann, J. Cao, G.J. Wagner, W.K. Liu, Data-driven multi-scale multi-physics models to derive process–structure–property relationships for additive manufacturing, Comput. Mech. 61 (5) (2018) 521–541, http://dx.doi.org/10.1007/s00466-018-1539-z.

[7] G.X. Gu, C.-T. Chen, D.J. Richmond, M.J. Buehler, Bioinspired hierarchical composite design using machine learning: simulation, additive manufacturing, and experiment, Mater. Horiz. 5 (5) (2018) 939–945, http://dx.doi.org/10.1039/C8MH00653A, URL http://dx.doi.org/10.1039/C8MH00653A.

[8] L. Wang, Y.-C. Chan, F. Ahmed, Z. Liu, P. Zhu, W. Chen, Deep generative modeling for mechanistic-based learning and design of metamaterial systems, Comput. Methods Appl. Mech. Engrg. 372 (2020) 113377, http://dx.doi.org/10.1016/j.cma.2020.113377, URL https://www.sciencedirect.com/science/article/pii/S0045782520305624.

[9] S. Bonfanti, R. Guerra, F. Font-Clos, D. Rayneau-Kirkhope, S. Zapperi, Automatic design of mechanical metamaterial actuators, Nature Commun. 11 (1) (2020) 4162, http://dx.doi.org/10.1038/s41467-020-17947-2.

[10] K.O. Lye, S. Mishra, D. Ray, P. Chandrashekar, Iterative surrogate model optimization (ISMO): An active learning algorithm for PDE constrained optimization with deep neural networks, Comput. Methods Appl. Mech. Engrg. 374 (2021) 113575, http://dx.doi.org/10.1016/j.cma.2020.113575, URL https://www.sciencedirect.com/science/article/pii/S004578252030760X.

[11] I. Lagaris, A. Likas, D. Fotiadis, Artificial neural networks for solving ordinary and partial differential equations, IEEE Trans. Neural Netw. 9 (5) (1998) 987–1000, http://dx.doi.org/10.1109/72.712178, URL http://dx.doi.org/10.1109/72.712178.

[12] R. Ranade, C. Hill, J. Pathak, DiscretizationNet: A machine-learning based solver for Navier–Stokes equations using finite volume discretization, Comput. Methods Appl. Mech. Engrg. 378 (2021) 113722, http://dx.doi.org/10.1016/j.cma.2021.113722, URL https://www.sciencedirect.com/science/article/pii/S004578252100058X.

[13] J. Han, A. Jentzen, W. E, Solving high-dimensional partial differential equations using deep learning, Proc. Natl. Acad. Sci. 115 (34) (2018) 8505–8510, http://dx.doi.org/10.1073/pnas.1718942115, URL https://www.pnas.org/content/pnas/115/34/8505.full.pdf.

[14] E. Samaniego, C. Anitescu, S. Goswami, V.M. Nguyen-Thanh, H. Guo, K. Hamdia, X. Zhuang, T. Rabczuk, An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications, Comput. Methods Appl. Mech. Engrg. 362 (2020) 112790, http://dx.doi.org/10.1016/j.cma.2019.112790, URL https://www.sciencedirect.com/science/article/pii/S0045782519306826.

[15] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Comput. Methods Appl. Mech. Engrg. 361 (2020) 112732, http://dx.doi.org/10.1016/j.cma.2019.112732, URL https://www.sciencedirect.com/science/article/pii/S004578251930622X.

[16] E. de Bezenac, A. Pajot, P. Gallinari, Deep learning for physical processes: Incorporating prior scientific knowledge, 2018, http://arxiv.org/abs/1711.07970 [arXiv:1711.07970].

[17] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, P. Perdikaris, Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data, J. Comput. Phys. 394 (2019) 56–81, http://dx.doi.org/10.1016/j.jcp.2019.05.024, URL https://www.sciencedirect.com/science/article/pii/S0021999119303559.

[18] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020, http://arxiv.org/abs/2001.04536 [arXiv:2001.04536].

[19] M. Giselle Fernández-Godino, C. Park, N.H. Kim, R.T. Haftka, Issues in deciding whether to use multifidelity surrogates, AIAA J. 57 (5) (2019) 2039–2054, http://dx.doi.org/10.2514/1.j057750, URL http://dx.doi.org/10.2514/1.J057750.

[20] H. Chi, Y. Zhang, T.L.E. Tang, L. Mirabella, L. Dalloro, L. Song, G.H. Paulino, Universal machine learning for topology optimization, Comput. Methods Appl. Mech. Engrg. 375 (2021) 112739, http://dx.doi.org/10.1016/j.cma.2019.112739, URL https://www.sciencedirect.com/science/article/pii/S0045782519306292.

[21] M. Raissi, P. Perdikaris, G.E. Karniadakis, Inferring solutions of differential equations using noisy multi-fidelity data, J. Comput. Phys. 335 (2017) 736–746, http://dx.doi.org/10.1016/j.jcp.2017.01.060, URL https://www.sciencedirect.com/science/article/pii/S0021999117300761.

[22] S. Chakraborty, Transfer learning based multi-fidelity physics informed deep neural network, J. Comput. Phys. 426 (2021) 109942, http://dx.doi.org/10.1016/j.jcp.2020.109942, URL http://dx.doi.org/10.1016/j.jcp.2020.109942.

[23] D. Liu, Y. Wang, Multi-fidelity physics-constrained neural network and its application in materials modeling, J. Mech. Des. 141 (12) (2019) http://dx.doi.org/10.1115/1.4044400.

[24] X. Meng, G.E. Karniadakis, A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems, J. Comput. Phys. 401 (2020) 109020, http://dx.doi.org/10.1016/j.jcp.2019.109020, URL https://www.sciencedirect.com/science/article/pii/S0021999119307260.

[25] A.A. Gorodetsky, J.D. Jakeman, G. Geraci, MFNets: data efficient all-at-once learning of multifidelity surrogates as directed networks of information sources, Comput. Mech. 68 (4) (2021) 741–758, http://dx.doi.org/10.1007/s00466-021-02042-0.

[26] H. Wang, R. Planas, A. Chandramowlishwaran, R. Bostanabad, Train once and use forever: Solving boundary value problems in unseen domains with pre-trained deep learning models, 2021, http://arxiv.org/abs/2104.10873 [arXiv:2104.10873].

[27] C. Chen, Y. Zuo, W. Ye, X. Li, S.P. Ong, Learning properties of ordered and disordered materials from multi-fidelity data, Nat. Comput. Sci. 1 (1) (2021) 46–53, http://dx.doi.org/10.1038/s43588-020-00002-x, URL https://www.nature.com/articles/s43588-020-00002-x, Bandiera_abtest: a Cg_type: Nature Research Journals Number: 1 Primary_atype: Research Publisher: Nature Publishing Group Subject_term: Atomistic models;Computational methods Subject_term_id: atomistic-models;computational-methods.

[28] X. Zhang, K. Garikipati, Machine learning materials physics: Multi-resolution neural networks learn the free energy and nonlinear elastic response of evolving microstructures, Comput. Methods Appl. Mech. Engrg. 372 (2020) 113362, http://dx.doi.org/10.1016/j.cma.2020.113362, URL https://www.sciencedirect.com/science/article/pii/S0045782520305478.

[29] E.J. Hall, S. Taverniers, M.A. Katsoulakis, D.M. Tartakovsky, GINNS: Graph-informed neural networks for multiscale physics, J. Comput. Phys. 433 (2021) 110192, http://dx.doi.org/10.1016/j.jcp.2021.110192, URL http://arxiv.org/abs/2006.14807, arXiv:2006.14807.

[30] H. Gao, M.J. Zahr, J.-X. Wang, Physics-informed graph neural Galerkin networks: A unified framework for solving PDE-governed forward and inverse problems, 2021, arXiv:2107.12146 [Cs].

[31] Y. Kumar, S. Chakraborty, GrADE: A graph based data-driven solver for time-dependent nonlinear partial differential equations, 2021, arXiv:2108.10639 [Physics, Stat].

[32] N. Trask, A. Huang, X. Hu, Enforcing exact physics in scientific machine learning: A data-driven exterior calculus on graphs, J. Comput. Phys. 456 (2022) 110969, http://dx.doi.org/10.1016/j.jcp.2022.110969, URL https://www.sciencedirect.com/science/article/pii/S0021999122000316.

[33] F. Scarselli, M. Gori, A.C. Tsoi, M. Hagenbuchner, G. Monfardini, The graph neural network model, IEEE Trans. Neural Netw. 20 (1) (2009) 61–80, http://dx.doi.org/10.1109/TNN.2008.2005605.

[34] P.W. Battaglia, J.B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, C. Gulcehre, F. Song, A. Ballard, J. Gilmer, G. Dahl, A. Vaswani, K. Allen, C. Nash, V. Langston, C. Dyer, N. Heess, D. Wierstra, P. Kohli, M. Botvinick, O. Vinyals, Y. Li, R. Pascanu, Relational inductive biases, deep learning, and graph networks, 2018, arXiv:1806.01261.

[35] J. Shlomi, P. Battaglia, J.-R. Vlimant, Graph neural networks in particle physics, 2021, 021001, http://dx.doi.org/10.1088/2632-2153/abbf9a, 2 (2) Publisher: IOP Publishing.
[36] A. Sanchez-Gonzalez, N. Heess, J.T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, P. Battaglia, Graph networks as learnable physics engines for inference and control, 2018, http://arxiv.org/abs/1806.01242 [arXiv:1806.01242].
[37] H. Shao, T. Kugelstadt, T. Hädrich, W. Pałubicki, J. Bender, S. Pirk, D.L. Michels, Accurately solving physical systems with graph learning, 2021, http://arxiv.org/abs/2006.03897 [arXiv:2006.03897].
[38] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P.W. Battaglia, Learning to simulate complex physics with graph networks, 2020, http://arxiv.org/abs/2002.09405 [arXiv:2002.09405].
[39] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, P.W. Battaglia, Learning mesh-based simulation with graph networks, 2021, http://arxiv.org/abs/2010.03409 [arXiv:2010.03409].
[40] F.d.A. Belbute-Peres, T. Economon, Z. Kolter, Combining differentiable PDE solvers and graph neural networks for fluid flow prediction, in: International Conference on Machine Learning, PMLR, pp. 2402–2411.
[41] F. Alet, A.K. Jeewajee, M. Bauza, A. Rodriguez, T. Lozano-Perez, L.P. Kaelbling, Graph element networks: adaptive, structured computation and memory, 2019, http://arxiv.org/abs/1904.09019 [arXiv:1904.09019].
[42] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Neural operator: Graph kernel network for partial differential equations, 2020, http://arxiv.org/abs/2003.03485 [Cs, Math, Stat].
[43] S. Saha, Z. Gan, L. Cheng, J. Gao, O.L. Kafka, X. Xie, H. Li, M. Tajdari, H.A. Kim, W.K. Liu, Hierarchical deep learning neural network (HiDeNN): An artificial intelligence (AI) framework for computational science and engineering, Comput. Methods Appl. Mech. Engrg. 373 (2021) 113452, http://dx.doi.org/10.1016/j.cma.2020.113452, URL https://www.sciencedirect.com/science/article/pii/S004578252030637X.
[44] L. Zhang, L. Cheng, H. Li, J. Gao, C. Yu, R. Domel, Y. Yang, S. Tang, W.K. Liu, Hierarchical deep-learning neural networks: Finite elements and beyond, Comput. Mech. 67 (1) (2021) 207–230, http://dx.doi.org/10.1007/s00466-020-01928-9.
[45] J. Jung, K. Yoon, P.-S. Lee, Deep learned finite elements, Comput. Methods Appl. Mech. Engrg. 372 (2020) 113401, http://dx.doi.org/10.1016/j.cma.2020.113401, URL https://www.sciencedirect.com/science/article/pii/S0045782520305867.
[46] Y. Lecun, L. Bottou, G. Orr, K.-R. Müller, Efficient BackProp, 2000.
[47] J.L. Ba, J.R. Kiros, G.E. Hinton, Layer normalization, 2016, http://arxiv.org/abs/1607.06450 [arXiv:1607.06450].
[48] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, 2017, http://arxiv.org/abs/1412.6980 [arXiv:1412.6980].
[49] G.H. Teichert, A.R. Natarajan, A. Van der Ven, K. Garikipati, Machine learning materials physics: Integrable deep neural networks enable scale bridging by learning free energy functions, Comput. Methods Appl. Mech. Engrg. 353 (2019) 201–216, http://dx.doi.org/10.1016/j.cma.2019.05.019, URL https://www.sciencedirect.com/science/article/pii/S0045782519302889.
[50] N.N. Vlassis, R. Ma, W. Sun, Geometric deep learning for computational mechanics Part I: anisotropic hyperelasticity, Comput. Methods Appl. Mech. Engrg. 371 (2020) 113299, http://dx.doi.org/10.1016/j.cma.2020.113299, URL https://www.sciencedirect.com/science/article/pii/S0045782520304849.
[51] I. Babuska, B. Szabo, On the rates of convergence of the finite element method, Internat. J. Numer. Methods Engrg. 18 (3) (1982) 323–341, http://dx.doi.org/10.1002/nme.1620180302, URL https://onlinelibrary.wiley.com/doi/abs/10.1002/nme.1620180302.
[52] W. Rawat, Z. Wang, Deep convolutional neural networks for image classification: A comprehensive review, Neural Comput. 29 (9) (2017) 2352–2449, http://dx.doi.org/10.1162/neco_a_00990.
[53] P. Virtanen, R. Gommers, T.E. Oliphant, M. Haberland, T. Reddy, D. Cournapeau, E. Burovski, P. Peterson, W. Weckesser, J. Bright, S.J. van der Walt, M. Brett, J. Wilson, K.J. Millman, N. Mayorov, A.R.J. Nelson, E. Jones, R. Kern, E. Larson, C.J. Carey, I. Polat, Y. Feng, E.W. Moore, J. VanderPlas, D. Laxalde, J. Perktold, R. Cimrman, I. Henriksen, E.A. Quintero, C.R. Harris, A.M. Archibald, A.H. Ribeiro, F. Pedregosa, P. van Mulbregt, SciPy 1.0 Contributors, SciPy 1.0: Fundamental algorithms for scientific computing in python, Nature Methods 17 (2020) 261–272, http://dx.doi.org/10.1038/s41592-019-0686-2.