Computers and Structures 266 (2022) 106784

Contents lists available at ScienceDirect

Computers and Structures

journal homepage: www.elsevier.com/locate/compstruc

Two-dimensional finite element network analysis: Formulation and static analysis of structural assemblies

Mehdi Jokar, Fabio Semperlotti *

School of Mechanical Engineering, Ray W. Herrick Laboratories, Purdue University, West Lafayette, IN 47907, United States

ARTICLE INFO

Article history:
Received 1 July 2021
Accepted 14 March 2022

Keywords:
Deep learning
Bidirectional recurrent neural network
Computational structural mechanics
Reinforced panels

ABSTRACT

Finite element network analysis (FENA) is a physics-informed, deep-learning-based framework for the simulation of physical systems. FENA combines the conceptual flexibility of classical finite element methods with the computational power of pre-trained neural networks. A remarkable characteristic of FENA is the ability to simulate assemblies of physical elements by concatenating pre-trained networks serving as models of classes of physical systems. This characteristic places FENA in a new category of network-based computational platforms because, unlike other techniques, it does not require *ad hoc* training for problem-specific conditions.

The present study significantly expands the concept and functionalities of FENA by including 1D slender beams and 2D thin plates and by further extending its concatenation functionality. Concatenation, which is a key property to create multicomponent assemblies without requiring training, is reformulated following an energy-based variational approach that significantly enhances accuracy and speed of convergence. The approach is numerically validated against finite element solutions for different configurations of structural assemblies, loads, and boundary conditions. Although presented in the context of one- and two-dimensional structures, the present framework is extremely general and provides a foundation to potentially simulate a broad range of physical systems.

1. Introduction

The concept of FENA, recently introduced by Jokar and Semperlotti [1], refers to a computational framework powered by deep neural networks to achieve predictive simulations of the response of physical systems. From a high-level perspective, FENA combines the conceptual modularity and flexibility at the basis of the finite element (FE) method with the extreme computational speed of trained deep neural networks.

The most unique property of FENA stems from its ability to solve boundary value problems of multicomponent assemblies without requiring *ad hoc* training of the surrogate network models. Surrogate models of selected physical elements are stored in a library in the form of pre-trained deep neural networks. Note that these networks are pre-trained on general classes of problems (e.g. structural slender rod elements) independently of loading or boundary conditions that are instead specific to a prescribed analysis. It follows that a boundary value problem can be solved without the need to retrain the network for each specific condition. Even more importantly, models of multicomponent physical systems involving assemblies of multiple components can be built by interconnecting these pre-trained network models without the need for any further training. While FENA is not a finite element based approach, it is the underlying modular approach used to assemble models of physical systems (based on fundamental building blocks) that bears similarity with the general finite element method. Aside from this similarity, FENA does not employ any FE-based architecture, as in [2]. This difference is critical and it is at the foundation of the computational power of FENA. Indeed, the size of the model in FENA is not dictated by the discretization requirements (as in [2]), and the computationally expensive operation involving matrix assembly and inversion is entirely replaced by efficient surrogate neural network model predictions.

Abbreviations: BRNN, Bidirectional Recurrent Neural Network; DNN, Deep Neural Network; DOF, Degree of Freedom; FCE, Finite Concatenated Elements; FE, Finite Element; FEA, Finite Element Analysis; FNE, Finite Network Element; HPC, High Performance Computing; LE, Library of Elements; LHS, Latin Hypercube Sampling; LSTM, Long Short Term Memory; MA, Model Assessment; NS, Numerical Simulator; PDE, Partial Differential Equation; PPMCC, Pearson Product Moment Correlation Coefficient; RNN, Recurrent Neural Network; SLSQP, Sequential Least Squares Programming.

* Corresponding author.
E-mail addresses: mjokar@purdue.edu (M. Jokar), fsemperl@purdue.edu (F. Semperlotti).

https://doi.org/10.1016/j.compstruc.2022.106784
0045-7949/

1.1. Related work

To date, the most notable examples of deep learning techniques in physical sciences have been in the general areas of modeling complex nonlinear systems [3–7] as well as in the solution of differential boundary value problems [8–10]. The remarkable modeling power of deep learning methodologies builds on the universal approximation theorem [11] that guarantees a network can approximate any complex nonlinear model provided enough data, proper network architecture (number of layers and layer sizes), and enough network training iterations.

While finite element analysis (FEA) and other similar numerical methods based on discretization techniques (e.g. boundary element and spectral methods [12,13]) offer a very flexible and powerful route to the solution of initial and boundary value problems on complex geometries [14–18], their computational cost grows rapidly with the size of the model [19,20]. Takeuchi et al. [2] proposed one of the first studies aiming at overcoming this limitation by recasting the concept of finite element method in a deep neural network perspective; an approach that was dubbed finite element neural networks (FENN). In this initial attempt, the network layers and their size (i.e. the number of neurons per layer), the activation functions, and the connection between neurons were determined to mimic the node distribution and the element shape functions. This approach was further investigated in [21–24]. All the above studies employed a network architecture that closely resembled the internal structure of a traditional finite element model. While this strategy provided clear instructions to build the network's internal architecture, it also implicitly embedded in it the typical limitations and constraints characteristic of the finite element methodology. It follows that any modification of the physical system (such as material properties, loads, and boundary conditions) or even a simple re-meshing operation will require building an entirely new architecture as well as repeating the training process. In addition, building an architecture that mimics the data flow structure of the finite element process results in a network size that scales with the number of physical degrees of freedom, hence carrying over into the network model the same key limitations of the finite element method.

Other studies have focused on developing surrogate network models whose actual network architecture is independent of the underlying physical characteristics of the system. From a general perspective, these studies can be divided into two main categories, (1) purely data-driven [25–28], and (2) physics-informed approaches [29–34]. These two classes of methods differ particularly in the way training loss functions are defined. In data-driven approaches, training is typically based on a mean squared error cost function that is minimized according to numerical data contained in the training dataset [35]. Data-driven techniques generally require extensive datasets and are trained to model very specific problem conditions, which very often are not representative of more general cases. From a computational perspective, while the prediction of the output from a trained neural network is an extremely efficient process, the training phase is very expensive. Hence, in addition to data availability, another critical limitation is associated with the recurrent training which is needed every time the problem parameters (e.g. boundary conditions or input sequence size) change.

Physics-based techniques were conceived with the intent of alleviating some of the issues of data-driven methods by leveraging the knowledge provided by the governing equations, whenever available. The governing equations can be embedded in the network via the training cost function [29,36] following a residual minimization approach. Although this approach has shown significant potential to solve problems described by selected types of partial differential equations (PDEs) [37], to increase network model accuracy, and to reduce the size of the training dataset, it still suffers from the same critical limitation concerning the recurrent training needed following changes to the problem's conditions. Given the significant computational cost of training, the impact on the performance of this latter aspect cannot be understated. Recently, Wang et al. [38] introduced the idea of genomics flow network (GFNet) to address this issue. They presented an iterative method to assemble a set of trained networks, where each can simulate a boundary value problem with arbitrary boundary conditions on a smaller subdomain. While this approach eliminates the need for network retraining when the computational domain is larger than the fundamental domain used to train the GFNet, its application is still limited to the case of a uniform domain governed by a single type of PDE.

The concept of FENA [1] was proposed with the intent of overcoming the above limitations (particularly, the need for retraining the network for specific conditions) while still leveraging the outstanding computational efficiency of a trained network. In [1], the authors formulated the basic idea of FENA, developed the fundamental architecture, and performed a numerical investigation to validate the concept and assess the performance. While the general concept put forward was general, the application focused exclusively on a one-dimensional elasticity problem consisting in the axial static response of a thin rod. Additionally, the trained network concatenation strategy proposed in [1] was only applicable to the problem of a 1D rod. In contrast, the present study provides a significant expansion of both the formulation and the functionalities of FENA. More specifically, this paper makes the following key contributions. First, the formulation of surrogate network models is extended to include both one- and two-dimensional structural elements such as slender Euler–Bernoulli beams and thin Kirchhoff plates. Second, the concatenation strategy is modified and improved in order to be able to handle dissimilar elements; this is a key aspect to create multicomponent structural assemblies. This specific result is obtained by reformulating the concatenation algorithm based on the variational minimization of the total energy. This strategy is then numerically tested by simulating the static response of a stiffened panel, which is a prototypical example of a multicomponent model. Third, an extensive numerical study is performed to validate the formulation of the new surrogate models and of the concatenation strategy, and to assess the performance with respect to different types of structural systems. We highlight that the main emphasis of this paper is to extend FENA to the static simulation of 1D and 2D elements and to show the feasibility to model structural assemblies. The ability to connect networks in a form akin to finite elements (and without *ad hoc* training) is quite remarkable and, to the best of the authors' knowledge, never proved before. We believe that, with this methodology in place, many types of physical elements (not necessarily only structural) can be implemented in FENA. The procedure presented in the following is general and can be used to further develop FENA for different structural elements and applications.

The remainder of the paper is structured as follows. First, we briefly present the fundamental structure of FENA and describe the general procedure to perform simulations. Next, we describe the architecture of the newly introduced beam and plate finite network elements (FNE) followed by the variational formulation of the model assembly. Finally, we apply FENA to a set of different structural problems and discuss the overall performance.

2. A brief introduction to FENA

This section presents a brief review of the fundamental concepts at the basis of the *Finite Element Network analysis* (FENA) framework and of the key modules necessary to assemble models

and run computations. The fundamental building blocks that allow building models of physical systems in FENA consist of pre-trained surrogate neural networks based on a bidirectional recurrent architecture [39]. These elements are referred to as finite network elements and, conceptually, play the role of elements (like beams, plates, and solids) in FEA. The FNE surrogate models can represent potentially any physical system or component depending on the initial architecture and training. However, in the initial formulation of FENA [1], only FNEs representing the static axial response of one-dimensional slender rods were developed. The different FNEs are stored in a library of elements (LE) from which they can be selected to build a model. One of the distinguishing and most remarkable features of FENA is its ability to combine different FNEs in order to form the surrogate (assembly) model of a multicomponent physical system. Remarkably, the resulting model does not require any further training and can be used directly for computations. This ability to directly use pre-trained neural networks by simply interconnecting them (without retraining) is referred to as network concatenation, and it is implemented in a dedicated module named Finite Concatenated Elements (FCE). This module encompasses one of the most distinctive features of FENA and makes this methodology different from any other network-based computational tool currently available. While the LE and FCE modules are the key modules to build a physical model in the FENA environment, two other modules are needed to perform the actual computations, that are the Numerical Simulator (NS) and the Model Assessment (MA). The NS module can be regarded as the true computational engine of FENA and it is the component that receives and applies problem-specific inputs (e.g. material properties and applied loads), executes the network, and predicts the physical response. Once the numerical solution has been obtained, the MA module is applied to evaluate the accuracy of the numerical results. A comprehensive description of the framework and its modules can be found in [1].

Given the four modules described above, the main steps to build a model of a physical system in FENA are summarized schematically in Fig. 1 and described here below:

1. *FNE selection*: select the appropriate FNEs needed to simulate the different components of the physical system. This step is conceptually equivalent to selecting element types and shape functions in FEA.
2. *FCE algorithm selection*: based on considerations of the physical domain size and properties, an appropriate concatenation scheme is selected in order to form an assembly representative of the physical system. The stiffened plate is an example of a model built by assembling together surrogate networks representing beams and plates. This operation is reminiscent of the FEA meshing process, where the system is discretized and prepared for the numerical solution.
3. *System simulation*: Once the equivalent network model is assembled and the sequence of problem-specific loads and boundary conditions is provided, the NS module executes the network and computes the output. This operation is conceptually equivalent to the matrix assembly and numerical solution of the system of algebraic equations performed in FEA. However, in FENA, the matrix assembly and inversion, which are the most expensive FEA computational tasks, are replaced with efficient neural network predictions. This step is responsible for the most considerable differences in computational cost and efficiency observed between FEA and FENA.
4. *Model and results assessment*: the final step of the simulation process consists in analyzing the predicted system's response and assessing its reliability and accuracy. This is a critical step in FENA, given that the solution is obtained via surrogate models and not by performing a numerical solution of systems of differential equations.

3. Network element models of slender beams and thin plates

As previously highlighted, the ability of FENA to model physical systems rests on the availability of surrogate models (i.e. the FNEs) to simulate the response of the different physical components. The FNEs are pre-trained and stored in LE. Following from [1], the present study expands the modeling capabilities of FENA by formulating new surrogate models for 1D Euler–Bernoulli beams and 2D Kirchhoff plates [40]. These structural elements are the most common building blocks in lightweight structures and will allow building more elaborate structural assemblies in the FENA environment. To be able to interconnect these different elements, the FCE algorithm also requires further extension in order to manage the exchange of information at the interface between different types of surrogate models (or, equivalently, structural elements). We will specifically discuss this aspect in the context of the integration of beams and plates that are key components to build a stiffened plate.

This section presents the conceptual formulation of the problem and the development of neural networks serving as surrogate models for beam and plate elements. The numerical validation for either individual elements or structural assemblies will be provided in the next section.

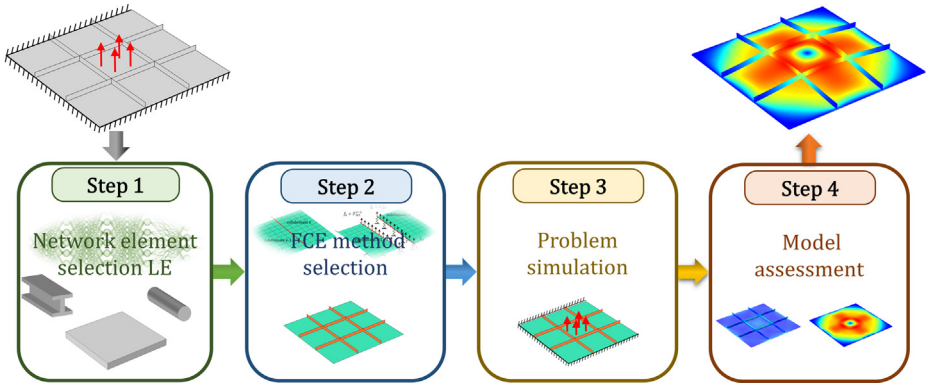

Fig. 1. Schematic showing the sequence of operations to build the model and calculate the physical response for a stiffened panel in FENA. In step 1, pre-trained FNEs of beams and plates are selected from the existing library of elements. In step 2, the elements are integrated (i.e. the FNEs are concatenated) to assemble the overall stiffened panel model. In step 3, the complete (network) model is executed to simulate the physical response. Finally, in step 4, the MA module is used to assess the accuracy of the calculated solution. The most remarkable aspect is that these steps do not require any additional training of the network; assembly, loads, and boundary conditions are handled by the trained FNEs available in the LE and the concatenation algorithms available in the FCE module.

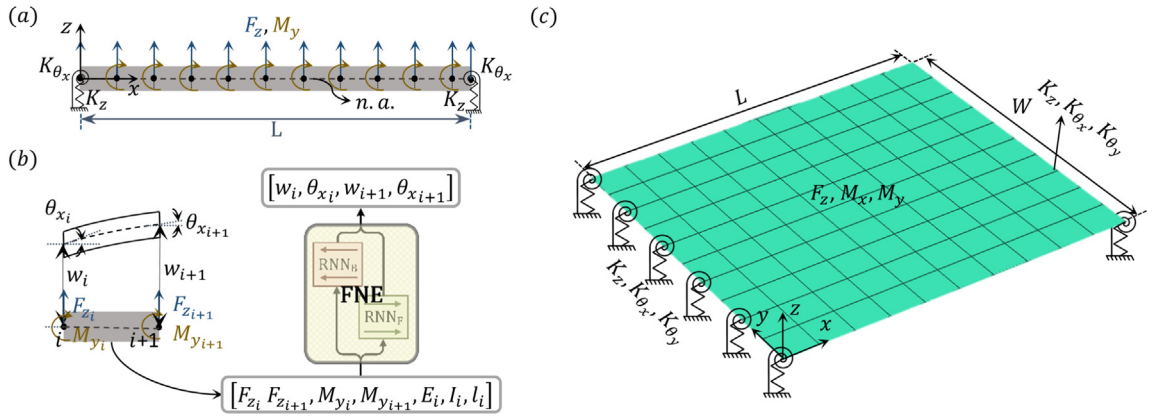

Fig. 2. Schematic showing the main degrees of freedom, external loads, and generalized stiffness boundary conditions for beam and plate elements. (a) Schematic of a slender Euler–Bernoulli beam subject to distributed transverse nodal forces F_z and bending moments along y-axis M_y. The ends of the beam at $x = 0$ and $x = L$ are connected to axial springs K_z (resisting the displacement w along the z-axis) and to torsional springs K_{θ_x} (resisting the rotation $\theta_x = \partial w/\partial x$). (b) Left: schematic view of a beam section between nodes i and $i+1$ subject to F_z and M_y resulting in the nodal generalized displacements w and θ_x. Right: High-level view of the BRNN-based FNE architecture for a beam element and the list of the corresponding input and output parameters. (c) The schematic of the plate problem. Plate is subject to the generic out-of-plane force nodal $F_z(x, y)$, and bending moments $M_x(x, y)$ and $M_y(x, y)$. The plate has free boundaries at $y = 0$ and $y = W$. Also, the boundaries $x = 0$ and $x = L$ have stiffness boundary condition. The stiffness K_z, K_{θ_x}, and K_{θ_y} resist $w, \theta_x = \partial w/\partial x$, and $\theta_y = \partial w/\partial y$ respectively.

Before we further proceed with the network details, it is worth clarifying our rationale for the selection of the aforementioned structural elements. While the systems being considered in this study are relatively simple from a structural mechanics perspective, in the context of neural-network-based analysis these systems are very challenging to simulate unless *ad hoc* training is performed. Most of the existing network-based techniques (regardless if data-driven or physics-informed) for the simulation of physical systems require training to be performed on the exact boundary conditions and loads. To date, this has been one of the major limitations in using neural networks for predictive forward simulations. This observation also explains why the main emphasis of this work is to show how a platform for neural network-based computations can be performed without continuously training the system for selected analysis conditions. Recall that, independently of the complexity of the physical system being analyzed, this property has not been achieved before in neural-network-based computations. Additionally, the FNEs of the fundamental structures (beam and plate FNEs) are also used to demonstrate the ability of FENA to deal with multicomponent (multi-element) physical systems by simply assembling multiple FNEs. We will showcase this remarkable capability by simulating a stiffened panel. Once again, a stiffened panel is certainly a relatively simple mechanical assembly from a structural mechanics perspective, but it is a quite challenging system for the state of the art of neural-network-based computation (if *ad hoc* training is not pursued).

Further, in this study FENA is presented and validated for analyses on linear systems. However, FENA's architecture is very general and does not prevent its application to different types of physical systems including, as an example, curved geometries and nonlinearities. Indeed, recall that the ability of a neural network to model any physical system derives from the universal approximation theorem [11] according to which networks can learn both linear and nonlinear input–output relationships. Hence, the proposed methodology is independent of the specific details of the analysis and could be readily extended to nonlinear systems (following the definition of proper nonlinear finite element networks to be included in LE). The concatenation also would require some adaptation; but again there are no conceptual limitations in FENA preventing this extension. On the other hand, contrary to FEA, the network elements size and calculations do not necessarily scale with the complexity and order of the elements because the neural network input-to-output mapping is intrinsically nonlinear. Therefore, it is foreseeable that extending FENA to more complex and nonlinear structures would further highlight the computational benefits over traditional techniques. Clearly, the above considerations are still theoretical, so that the actual performance of FENA for nonlinear analysis will have to be evaluated on specific problems and our current study does not allow us to provide any meaningful conceptual extrapolation. The concatenation algorithm also will need to be modified to account for the nonlinear nature of the system. However, given that the concatenation algorithm presented in the following is based on variational principles, we do expect to be able to extend it to nonlinear problems.

3.1. Expanding the library of elements: beam and plate finite network elements

As discussed in Section 2, the LE comprises various sets of pre-trained neural networks (i.e. the finite network elements) serving as surrogate models of classes of physical elements. Based on these fundamental classes of FNEs, a multicomponent model can be built by simply interconnecting different FNEs (via the concept of concatenation) as needed to achieve the final system configuration. It follows that the ability to build diverse models relies on the availability of a variety of surrogate elements in LE. For this reason, in this section we first expand the LE to include two new FNEs that simulate the response of two key structural elements: slender Euler–Bernoulli beams and thin Kirchhoff plates.

3.1.1. Euler–Bernoulli Beam: FNE architecture

The first FNE developed to expand LE represents a surrogate model that captures the static response of slender Euler Bernoulli beams. In the following, we will present the general approach to model the structural beam element via neural networks and describe the link between the structural mechanics elements and the beam FNE. Then, we provide specific details of the beam FNE architecture and its layers.

The schematic of the physical beam element is shown in Fig. 2a. The beam's neutral axis is aligned along the x-axis. A generic distributed nodal transverse force F_z is assumed to act in the z direction in addition to a distributed nodal bending moment M_y; both loads are applied at $n + 1$ nodes. The boundary points are assumed

connected to translational K_z and torsional K_{θ_x} stiffness elements (equivalent to linear and torsional springs) to allow for a generalized treatment of the boundary conditions. The beam FNE model is developed to emulate the static response of an Euler–Bernoulli beam element with two nodal variables, the transverse displacement in the z direction w and the rotation $\theta_x = \partial w/\partial x$[41]. In agreement with basic linear mechanics theory, small displacements and rotations are assumed. Note that these conditions will influence the FNE via the training data and do not affect directly the network architecture. The goal is to synthesize a FNE that can accept static loads (i.e. $F_z(x)$ and $M_y(x)$) and boundary conditions (i.e. K_z and K_{θ_x}) as input, and provide as output the traverse displacement w and rotation θ_x consistent with the static response of an Euler–Bernoulli beam element.

In FENA, FNEs are built by using deep bidirectional recurrent neural network (BRNN) architectures. A BRNN is composed of two sets of RNN cells each processing a sequence of input data (e.g. $\text{In}_1, \text{In}_2, \ldots, \text{In}_n$) in opposite directions ($1 \rightarrow n$ and $n \rightarrow 1$). The output of the RNN cells is then further processed to calculate the output sequence following each one of the n input states. Detailed discussions on the internal architecture of the BRNN, the rationale behind the selection of this type of network, and the role of bidirectionality in the simulation of physical systems can be found in [1]. The first step in the simulation of a problem via FNE is to form the input data sequence. This is done by discretizing the system domain into a set of n partitions (elements). We use the properties and applied input at the partitions to form the sequence of data used as input to the network (Fig. 2b). The size of these elements is determined by the problem domain size (e.g. length of the beam or size of the plate), the number of elements that the selected FNE is trained to simulate, and the point of application of the external input.

Once the partitioning of the physical system is determined, the input sequence ($\text{In}_i, i = 1, 2, \ldots, n$) can be built as (see also Fig. 2b):

$$\text{In}_i = [F_{z_i}, F_{z_{i+1}}, M_{y_i}, M_{y_{i+1}}, E_i, I_i, l_i], \ i = 1, 2, \ldots, n \tag{1}$$

where the subscript i is the node number, while E, I, and l are the Young's modulus, the second area moment of inertia, and the length of the element between the nodes i and $i+1$. Note that the element parameters E, I, and l can potentially be different for each element within the FNE. The left and right boundary stiffness values are also provided as input in order to initialize the hidden states of the BRNN core layers. Note that this will result in the propagation of the boundary condition to the entire data sequence, hence allowing for the generalization of the beam network element without any need for retraining. Clearly, the limit cases of the boundary stiffness, that is either $K_z \rightarrow \infty$ and $K_{\theta_x} \rightarrow \infty$ or $K_z \rightarrow 0$ and $K_{\theta_x} \rightarrow 0$, allow implementing clamped and free boundaries, respectively. Based on this input sequence, the network will return the output sequence (Out_i), that is:

$$\text{Out}_i = [w_i, w_{i+1}, \theta_{x_i}, \theta_{x_{i+1}}], \ i = 1, 2, \ldots, n \tag{2}$$

where w_i and $\theta_{x_i} = \partial w_i/\partial x$ are the transverse displacement and the rotation at the i-*th* node. The internal architecture of the beam FNE is presented in Fig. 3. The network consists of fully connected layers attached to a bidirectional core unit. The input sequence members are first processed by four fully connected layers (left side of Fig. 3) with size $[20, 20, 60, 60]$. The first layer employs $tanh(x)$ activation function while the remaining layers use E-swish activation function [42] defined by:

$$f(x) = \frac{\beta x}{1 + e^{-x}} \tag{3}$$

where $\beta = 1.5$. The E-swish activation function was selected since it provides faster learning and is able to train deeper networks than, as an example, the ReLU activation function [42]. However, our numerical simulations indicated that the use of the E-swish function for all of the four layers tended to produce numerically unstable training. Hence, we substituted the first layer activation function with *tanh*, which is limited to the range $[-1, 1]$, to avoid divergence of the loss function during training. Note that this combination of activation functions was ultimately determined by a trial and error process while monitoring the network prediction accuracy and the numerical stability of the training. The BRNN core layer uses 50 long short-term memory (LSTM) recurrent cells [43] in each of the forward and backward directions. These recurrent units are responsible for learning the mutual effect and the sequential logic behind the input sequence (Eq. (1)), as well as its relation with the output sequence (Eq. (2)). Despite the common practice of either zero or random initialization of the hidden states [35], we opted to use boundary stiffness values for initialization of the hidden states of the LSTMs. Due to the bidirectional nature of the BRNN, the effect of the boundary conditions propagates in both the forward and backward directions through the recurrent cells and affects the response of the FNE. To improve the learning capacity of the recurrent cells inside the BRNN core layer, we added two fully connected layers of size $[20, 60]$ connecting the boundary conditions to the core bidirectional layer hidden states, resulting in a unique initial hidden state for each recurrent cell in the BRNN layer. The BRNN core layer output is connected to 7 fully connected layers (right side of the BRNN in Fig. 3) with E-swish activation function and layer size of $[50, 60, 60, 60, 60, 60, 60]$. A linear activation function is used for the output layer.

Following from Eq. (2), the output layer size is 4, hence resulting in an output sequence of size $[4, n]$. We highlight that, in this work, we chose for the network elements to have the same input and output sequence sizes. In other terms, the network receives the properties and applied loads at the nodes of the partitioned system and returns the output only at those nodes. The reason for this decision was due to the fact that by using concatenation we can create an element with as many output nodes as desired, hence

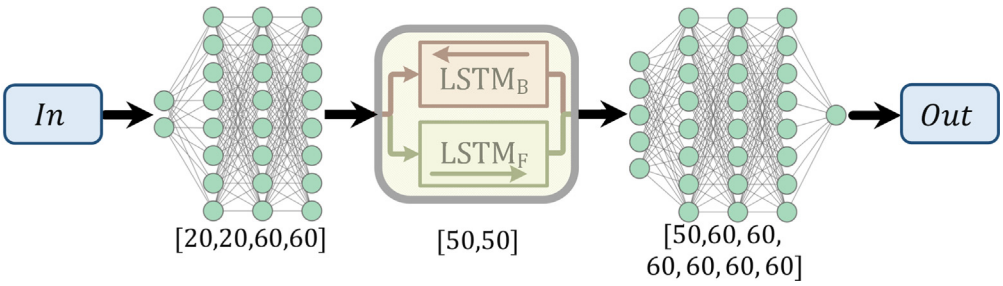

Fig. 3. The network contains a core bidirectional layer, with 50 LSTM cells in each direction. The BRNN architecture is used to learn sequential dependencies of the input sequence (In) and output sequence (Out). The vectors on the bottom indicate the layers size.

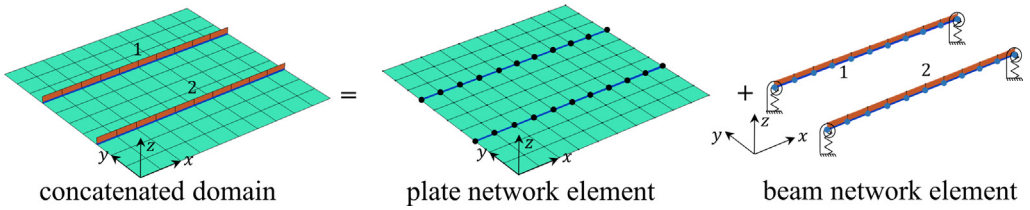

Fig. 4. Schematics illustrating how structural models can be assembled using concatenation of different (surrogate) elements. In this example, a stiffened panel is obtained by connecting two separate beam elements (labeled 1 and 2) to a plate element. While in this figure the concept of concatenation is shown via a physical illustration of the structural element, the actual process is performed via the FCE module applied to the FNEs, as explained in Section 3.2. The concatenation is performed by activating interface loads acting on all shared nodes at the common boundaries (see blue and black dot markers). (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

we still have the freedom to choose the level of discretization and where the output can be obtained. However, if one was interested in having different input and output sizes, we note that BRNNs can be trained to predict output sequences with a different size than the input. Such network elements would be able to predict the response at a finer (or coarser) mesh than the mesh used to define the input properties. The accuracy of the predictions in this case will have to be assessed but, based on our experience, we believe it should still provide high quality results. The details concerning the generation of the training dataset and the network training are presented in Appendix A.

3.1.2. Plate network element architecture

Consider a thin homogeneous Kirchhoff plate $(L \times W)[m^2]$ defined in the $x-y$ plane having thickness t. The plate is assumed subject to a distributed transverse force F_z in the z direction, and to bending moments M_x and M_y about the x and the y axes, respectively (see Fig. 2c). Note that we are neglecting in-plane forces (along x or y directions), hence the corresponding in-plane deformation of the plate's mid-surface is zero. The edges of the plate are assumed supported by a set of translational (K_z) and torsional ($K_{\theta_x}, K_{\theta_y}$) stiffness elements at $x = 0$ and $x = L$ to resist transverse deflection and rotation. The remaining two boundaries at $y = 0$and $y = W$ are free. We note that this choice of stiffness boundary does not provide the most general representation of the possible boundary conditions of a plate element; to achieve this generalization, stiffness elements should be included on all the four edges. However, while less general, this choice does not alter one of the main goals of this study that is to illustrate the procedure to develop 2D FENA elements and their concatenated form. Note that even in the case of the more general stiffness support at the four edges, the concatenation algorithm and the assessment strategy will remain unaltered while the plate FNE should be adapted to include the input stiffness values corresponding to the additional edges. The plate is uniformly partitioned (see the grid in Fig. 2c), resulting in a total of q nodes at which the external forces (F_z) and bending moments (M_x and M_y) can be applied.

As previously mentioned, the plate FNEs will be trained using data generated by FEA. In the present work, we use four-noded conforming rectangular elements [40] to discretize the plate. This element has four nodal variables $w, \theta_x = \partial w/\partial x, \theta_y = \partial w/\partial y$, and $\theta_{xy} = \partial^2 w/\partial x \partial y$, where w is the out-of-plane (z direction) nodal displacement. The input sequence of the plate FNE includes the three externally applied generalized forces (i.e. both forces and moments). More specifically, the network input sequence is set to $\text{In}_i = [F_{z_i}, M_{x_i}, M_{y_i}], i = 1, 2, \ldots, q$, where q is the total number of nodes. Following these assumptions, the input size is $[3, q]$. Except for the input and output layers, the same network architecture used for the beam FNE (see Fig. 3) was used for the plate. According to this definition, the plate FNE will be able to predict the value of the generalized nodal displacements $\text{Out}_i = [w_i, \theta_{x_i}, \theta_{y_i}, \theta_{xy_i}]$ for a given sequence of input loads. The details pertaining to the generation of the plate FNE training dataset and the training process are presented in Appendix B.

3.2. Assembly of dissimilar finite network elements

The previous section presented the general architecture of two new surrogate structural models that expand FENA's LE database to include the flexural static response of slender beams and thin plates. As recalled in Section 2 and discussed in detail in [1], FENA uses the concept of element concatenation to combine together pre-trained elements and form physical assemblies. The introductory study on FENA [1] dealt only with surrogate models of rod elements, hence the concatenation between different FNEs was more immediate because the elements shared the same type of degrees of freedom. In [1], the concatenation was used to assemble models of rods having an arbitrary length (i.e. beyond those available for the pre-trained elements in the database) and heterogeneous properties.

The addition of new element types to the LE opens the possibility to build more complex assemblies made of dissimilar structural elements. This latter concept is schematically exemplified in Fig. 4 for the case of a typical stiffened panel. In the remainder of this study, we choose the stiffened panel example to discuss how the concatenation methodology can be modified and applied to multiple element types (having dissimilar degrees of freedom). The stiffened panel will also be used as a benchmark model to numerically test FENA and to assess its performance.

From a practical perspective, the concatenation occurs at the level of the surrogate models and it is performed via the FCE module. A set of artificial loads (referred to as interface loads) is applied to the nodal points at the interface of the different elements to be connected. These interface loads bear a close similarity to the internal forces exchanged between structural elements through physical interfaces. Each node can have more than one degree of freedom, according to the element physical definition (see Section 3.1). Fig. 4 illustrates the interface nodes (highlighted by blue lines) selected to connect a plate element with beam elements. The black and blue dot markers on the plate and the beams, respectively, indicate the nodes that are associated with artificial interface loads. The concatenation strategy will be laid out in detail in the following sections and specialized for the different elements. From a general standpoint, the implementation of the concatenation condition leverages a few key common elements: (1) continuity of the generalized displacements at the interface nodes, and (2) satisfaction of the principle of virtual displacements. In the following, we detail the concatenation strategy for two different types of interfaces: (1) single-type element interface (e.g. beam-beam or plate-plate), and (2) multi-type element interface (e.g. beam-plate).

3.2.1. Single-type element interface

The case of single-element interface encompasses all the connections between elements of the same type. This case was initially considered in [1] and it is revisited and improved in this study in order to account for elements possessing a higher number of degrees of freedom per node. More specifically, while the initial FCE method [1] was designed for the problem of rod concatenation (having a single degree of freedom per node), we present a generalized concatenation algorithm applicable to any type of structural element and number of degrees of freedom per node. In the following, we will discuss the case of concatenation for both beam-beam and plate-plate interfaces.

Beam-beam interface: This type of interface emerges when two surrogate network models of beam elements are connected together. Consider a beam of length L, as shown in Fig. 5a. The beam is first divided into sections (referred to as subdomains). The number and length of these subdomains is an input parameter that is conceptually equivalent to the choices made to discretize a domain in mesh-based techniques (e.g. finite element method). Despite this analogy, it is important to understand that, in FENA, the solution accuracy and its stability are completely independent from the chosen spatial discretization. Indeed, high accuracy could be achieved even in the extreme case where only a single element was used for the calculation. This particular characteristic is due to the fact that the accuracy of the prediction is entirely controlled by the network architecture and by the training process, not by the number of elements and the corresponding degrees of freedom. On the other side, the choice of the subdomain does affect the number of points the response is available at.

In FENA, the two main parameters that determine the size of the subdomains are the range of the physical dimensions and the input sequence size (n). As a reminder, the first parameter refers to the range of physical dimensions (e.g. for a beam it consists in the length, while for a plate in both length and width) the network was trained for. The second parameter refers to the number of input loads and output points required by the problem. Recall also that each FNE surrogate network model should never be used to simulate systems exceeding the specified range of physical dimensions. When the dimensions of the physical system exceed this range, the concatenation strategy (i.e. the FCE module) should be used [1].

Consider two beam sections simulated via two independent FNEs (Fig. 5b). Each of these sections is modeled according to the general configuration in Fig. 5a. The two sections are subject to external distributed transverse loads $F_z(x)$ and bending moments $M_y(x)$. The translational and rotational end springs are initialized using two arbitrary values indicated by K_{z_0} and K_{θ_0} to avoid under-constrained conditions. The specific numerical value of these stiffness constants is immaterial since, as explained in the following section, the additional interface loads will converge to the correct value necessary to guarantee the satisfaction of the continuity and the virtual displacement conditions embedded in the cost function $\mathbf{J}$ (see Eq. (4)). The value of these springs will produce only an offset of the interface loads. Finally, interface loads (both translational and rotational) are added to interconnect the two beam elements.

Fig. 5b shows an example of beam-beam (single-type element) interface between two subdomains, labeled $s-1$ and s connected at node m. In addition to F_{z_m} and M_{y_m} (externally applied loads at node m), the interface loads $\mathcal{F}_{int}^{s-1}$ and $\mathcal{F}_{int}^{s}$ are also added to the interface. Recall that each interface load set includes an interface bending moment $M_{y_{int}}$ and an interface transverse force $F_{z_{int}}$ and are responsible for enforcing continuity. The four interface loads $[F_{z_{int}}^{s-1}, M_{y_{int}}^{s-1}, F_{z_{int}}^{s}, M_{y_{int}}^{s}]$ are added to the network input and tuned to guarantee continuity of the concatenated response at the interface node.

The value of the interface loads necessary to implement the interface is obtained via an optimization process that minimizes the following cost function $\mathbf{J}$:

$$\begin{aligned}
&\mathbf{J}(\mathbf{f}, \mathcal{N}_1, \ldots, \mathcal{N}_j; \mathcal{F}_{int}) = \mathbf{J}_C + \mathbf{J}_{\delta\mathcal{W}} \\
&\mathbf{J}_\mathbf{C}(\mathbf{f}, \mathcal{N}_1, \ldots, \mathcal{N}_j; \mathcal{F}_{int}) = a_1 \sum_{s=1}^{N_{int}} \left(w_{int}^{s-1} - w_{int}^{s}\right)^2 + a_2 \sum_{s=1}^{N_{int}} \left(\theta_{x_{int}}^{s-1} - \theta_{x_{int}}^{s}\right)^2 \\
&\mathbf{J}_{\delta\mathcal{W}}(\mathbf{f}, \mathcal{N}_1, \ldots, \mathcal{N}_j; \mathcal{F}_{int}) = b \sum_{i=1}^{n+1} \left(\frac{\partial \mathcal{W}}{\partial \mathcal{U}_i}\right)^2
\end{aligned} \tag{4}$$

where $\mathcal{F}_{int}$ is the vector containing all the interface loads, j is the total number of subdomains, $\mathcal{N}_j$ is the FNE selected to simulate the j^{th} subdomain response (w and θ_x of the nodes in the subdomain j), $\mathbf{f} = [F_{z_1}, M_{y_1}, \ldots, F_{z_{n+1}}, M_{y_{n+1}}]$ is the applied external load vector ($n+1$ is the total number of nodes in the structural assembly including all the FNEs), $\mathbf{J}_C$ is a term implementing the continuity condition for both the nodal displacement w and rotation θ_x, N_{int} is the total number of interface nodes, the superscripts refers to subdomain number, and the subscript *int* refers to interface nodes between the neighboring subdomains $s-1$ and s. a_1, a_2, and b are weight factors that are chosen using the initial guess for $\mathcal{F}_{int}$ (see

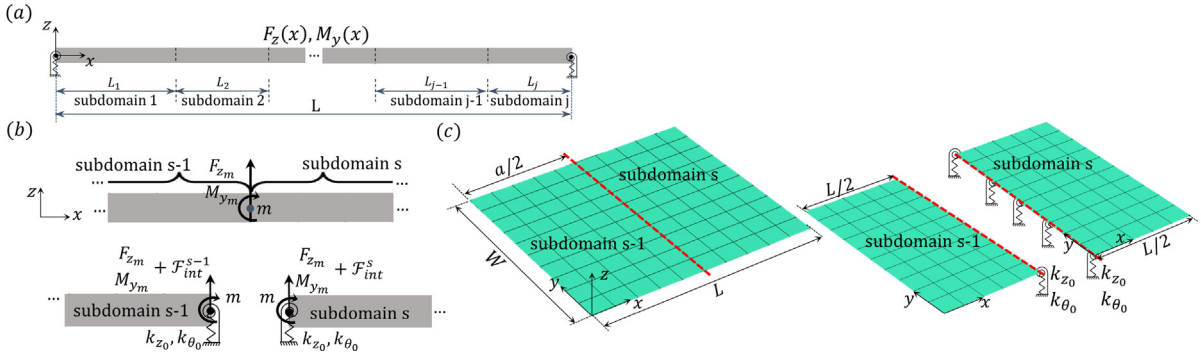

Fig. 5. Schematic of beam-beam and plate-plate (single-type element) concatenation. (a) A beam subjected to distributed loads $F_z(x)$ and $M_y(x)$ with total length L, divided into j subdomains to be solved with concatenation. (b) Schematic of beam-beam concatenation between section $s-1$ and s. A set of interface loads ($\mathcal{F}_{int}^{s-1}$ and $\mathcal{F}_{int}^{s}$) applied at each degree of freedom of the interface nodes are added to the original nodal load (F_{z_m} and M_{y_m}). (c) Schematic of plate-plate concatenation between section $s-1$ and s. In this case, the concatenation interface between the sections is a line that includes a set of nodes. Similar to the beam concatenation, the domain boundaries are also connected to virtual stiffness elements K_{z_0} and K_{θ_0} whose initial value is arbitrarily selected. In addition to the external applied load, an interface load $\mathcal{F}_{int}^{\square}$ is added to each degree of freedom of the interface nodes and tuned to achieve the continuous structural interface.

Eq. (6)). Also, $\mathbf{J}_{\delta\mathcal{W}}$ is an energy-based functional that follows from the principle of virtual displacements [40], $\mathcal{W}$ is the total energy and $\mathcal{U}_i = [w_i, \theta_{x_i}]$ is the vector of the generalized nodal displacements at node i. The total energy $\mathcal{W}$ for the concatenated beam is given by:

$$\begin{aligned} \mathcal{W} &= \mathcal{W}_I + \mathcal{W}_E \\ \mathcal{W}_I &= \int_\Omega \sigma_{xx}\epsilon_{xx} dv \\ \mathcal{W}_E &= -\mathbf{f}\cdot\mathcal{U} \end{aligned} \tag{5}$$

where $\mathcal{W}_I$ is the total strain energy of the beam, $\mathcal{W}_E$ is the work done by the external loads, σ_{xx} is the normal stress in the x direction, ϵ_{xx} is the normal strain in the x direction, Ω is the beam domain, and $\mathcal{U} = [w_1, \theta_{x_1}, \ldots, w_{n+1}, \theta_{x_{n+1}}]$ is the beam displacement vector.

The cost function (Eq. (4)) is made up of two fundamental components. The first component, indicated by $\mathbf{J}_C$, implements a continuity condition of the generalized displacements (i.e. translations and rotations). This condition ensures that the interface nodes occupy the same physical location (i.e. remain superimposed) at all times, hence guaranteeing a first order continuity. However, this condition is not sufficient to guarantee a well-posed inverse problem. Indeed, many different sets of interface loads can be found to satisfy the same continuity condition. In order to improve the formulation of the inverse problem, we complement the cost function with an additional energy-based term $J_{\delta\mathcal{W}}$ that is essentially a statement of the principle of virtual displacements. According to this latter principle, if a continuous body is in equilibrium, the virtual work of all forces in moving through a virtual displacement is zero [40]. The additional energy-based term guarantees the convergence to a set of nodal interface loads consistent with the equilibrium of the system under the applied external loads.

Implementing the FCE algorithm for a beam interface starts with the identification of the subdomains and the definition of the corresponding interface loads. Then, we form the cost function J using Eqs. 4 and 5. Note that the component $\mathcal{W}_I$ can be calculated by numerical integration (we use a Gauss integration method leveraging the Gaussian points of the beam element [41]). Also, the gradient of $\mathcal{W}$ with respect to the nodal displacements is analytically calculated by reverse mode automatic differentiation method by means of Python Autograd package. The initial guess for the interface loads $\mathcal{F}_{int}$ is obtained randomly within the range $[0, 0.1]$. Using this initial guess the weight factors a_1, a_2, and b are given by:

$$\begin{aligned} a_1 &= \left.\frac{1}{\sum_{s=1}^{N_{int}} \left(w_{int}^{s-1} - w_{int}^{s}\right)^2}\right|_{\mathcal{F}_{int}=\mathcal{F}_{int_0}} \\ a_2 &= \left.\frac{1}{\sum_{s=1}^{N_{int}} \left(\theta_{x_{int}}^{s-1} - \theta_{x_{int}}^{s}\right)^2}\right|_{\mathcal{F}_{int}=\mathcal{F}_{int_0}} \\ b &= \frac{1}{\mathcal{W}(\mathcal{F}_{int_0})} \end{aligned} \tag{6}$$

where $\mathcal{F}_{int_0}$ is the initial guess for $\mathcal{F}_{int}$. Note that $w_{int}^{\square}$ and $\theta_{x_{int}}^{\square}$ (the superscript $\square$ indicates the section number) are calculated using the selected FNEs $(\mathcal{N}_1, \ldots, \mathcal{N}_j)$ for the applied external load $\mathbf{f} + \mathcal{F}_{int_0}$. The optimal values of interface loads $(\mathcal{F}_{int}^*)$ are obtained by:

$$\mathcal{F}_{int}^* = \arg\min_{\mathcal{F}_{int}} \mathbf{J}(\mathcal{F}_{int}) \tag{7}$$

The cost function $\mathbf{J}(\mathcal{F}_{int})$ is then optimized to determine the interface loads. Once convergence is reached, the response of each subdomain is calculated (by executing the network) in the presence of both external and interface loads $\mathcal{F}_{int}^*$. For the numerical samples presented in Section 4, we utilized the Sequential Least Squares Programming (SLSQP) method which is well suited for multi-variable optimization. In addition, for the practical numerical implementation of the FCE algorithm, we used Python, NumPy, and Autograd packages.

Plate-plate interface. This type of interface occurs when two surrogate network models of plate elements are connected together. The simulation strategy for the concatenation of this element type follows along the same lines described for the beam-beam interface. Consider a plate occupying the domain $\Omega = [L \times W]$ (Fig. 5c). Assume the plate is simulated by using two subdomains $\bar{\Omega} = [L/2, W]$ concatenated together along $x = W/2$ (so that $\Omega = \bar{\Omega} \cup \bar{\Omega}$), as shown in Fig. 5c. The most significant difference for this type of interface is due to the different nodal degrees of freedom and, consequently, to the number of interface loads per node. Considering only the transverse deflection of the plate, the nodal interface loads can be defined in vector form as $\mathcal{F}_{int}^{\square} = [F_{z_{int}}^{\square}, M_{x_{int}}^{\square}, M_{y_{int}}^{\square}, M_{xy_{int}}^{\square}]$, where the superscript $\square$ indicates the section number, and $F_{z_{int}}, M_{x_{int}}, M_{x_{int}}$, and $M_{xy_{int}}$ are the interface loads corresponding to the nodal parameters w, θ_x, θ_y, and θ_{xy}.

The same cost function defined in Eq. (4) can be applied to the plate-plate interface by simply updating the corresponding terms. The continuity condition is expressed as:

$$\begin{aligned} \mathbf{J_C} = a_1 \sum_{s=1}^{N_{int}} \left(w_{int}^{s-1} - w_{int}^{s}\right)^2 + a_2 \sum_{s=1}^{N_{int}} \left(\theta_{x_{int}}^{s-1} - \theta_{x_{int}}^{s}\right)^2 \\ + a_3 \sum_{s=1}^{N_{int}} \left(\theta_{y_{int}}^{s-1} - \theta_{y_{int}}^{s}\right)^2 + a_4 \sum_{s=1}^{N_{int}} \left(\theta_{xy_{int}}^{s-1} - \theta_{xy_{int}}^{s}\right)^2 \end{aligned} \tag{8}$$

where a_i with $i = 1, \ldots, 4$ are weight factors. The energy term $\mathbf{J}_{\delta\mathcal{W}}$ is calculated according to Eq. (5). Note that, in this case, the displacement vector is defined as $\mathcal{U} = [w_1, \theta_{x_1}, \theta_{y_1}, \ldots, w_q, \theta_{x_q}, \theta_{y_q}]$ and the external force vector is given by $\mathbf{f} = [F_{z_1}, M_{x_1}, M_{y_1}, \ldots, F_{z_q}, M_{x_q}, M_{y_q}]$ where q is the total number of nodes in the plate assembly. Also, in this case $\mathcal{W}_I$ is given by:

$$\mathcal{W}_I = \int_\Omega \sigma_{xx}\epsilon_{xx} + \sigma_{yy}\epsilon_{yy} + 2\sigma_{xy}\epsilon_{xy} dv \tag{9}$$

Similarly to the case of beam-beam concatenation, to calculate the internal elastic energy $\mathcal{W}_I$ of the concatenated plate, we use a numerical integration leveraging the Gaussian points of the plate element [41]. The continuity and the energy conditions are used to find the interface loads, analogously to what described for the beam-beam interface.

3.2.2. Multi-type element interface

This kind of interface addresses the situation in which dissimilar structural elements (or, equivalently, dissimilar FNEs) are connected together to form a multicomponent assembly. As mentioned above, this study uses as a benchmark model the case of a stiffened panel.

From a general perspective, the same concatenation strategy presented in the previous section does apply to this type of interface. The approach starts with the identification of the fundamental subdomains forming the complete assembly. For example, consider the stiffened plate shown in Fig. 4. In this specific case, the system is composed of a plate and a set of two stiffeners (labeled as 1 and 2). For each nodal point at the interface (see the black and blue dots in Fig. 4) between the plate and the stiffeners, a set of interface loads must be applied to enforce continuity. The load takes the general form $[F_{z_{int}}^{\square,p}, M_{x_{int}}^{\square,p}, M_{y_{int}}^{\square,p}, M_{xy_{int}}^{\square,p}]$ for a node belonging to the plate (the black dot markers in Fig. 4), and

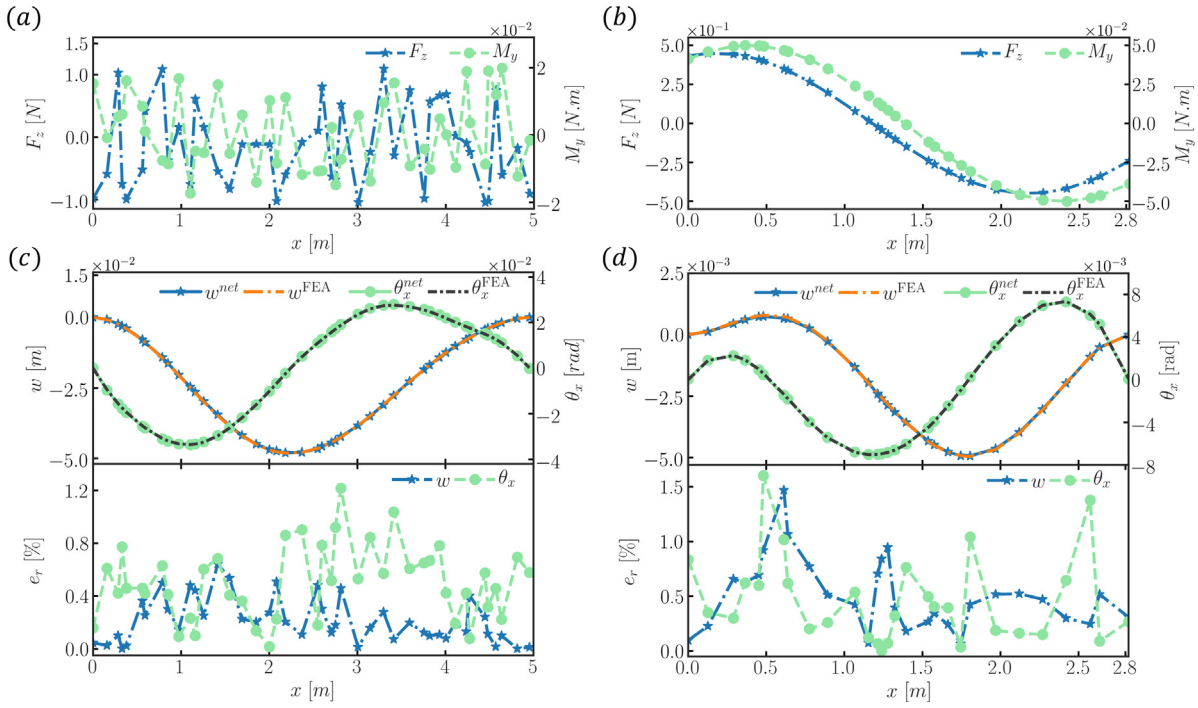

Fig. 6. Numerical predictions produced by FENA for the static response of an Euler–Bernoulli beam under generalized loads. Nodal locations are identified by markers. (a) The external applied transverse force F_z and bending moment M_y used for *Beam 1*. (b) The external applied transverse force F_z and bending moment M_y used for *Beam 2*. (c) The static response of *Beam 1* obtained by averaging predictions from network models (superscript *net*) B_1, B_2, and B_3. The result is compared with the finite element model solution (superscript FEA). (d) The static response of *Beam 2* obtained by averaging predictions from network models (superscript *net*) B_1, B_2, and B_3. The result is compared with the finite element model solution (superscript FEA). In both (c) and (d), the top figure shows the static response expressed in terms of the transverse displacement w and the rotation θ_x. The bottom figure shows the percentage relative nodal error e_r calculated with respect to the FEA solution.

$[F_{z_{int}}^{\square,b}, M_{x_{int}}^{\square,b}]$ for a node belonging to the stiffener (the blue dot markers in Fig. 4). In the previous notation, the superscript $\square$ indicates the interface node number, while the superscripts b and p identify the interface nodes on either the plate or the beam stiffener, respectively. In a similar way, a revised form of the continuity term is needed:

$$\mathbf{J_c} = a_1 \sum_{s=1}^{N_{int}} \left(w_{int}^{s,p} - w_{int}^{s,b} \right)^2 + a_2 \sum_{s=1}^{N_{int}} \left(\theta_{x_{int}}^{s,p} - \theta_{x_{int}}^{s,b} \right)^2 \tag{10}$$

where N_{int} is the total number of interface nodes, and all other terms follow the previous definition. Note that in the continuity term, we only apply this condition to the nodal degrees of freedom in common between a plate and a beam element, that is to w and θ_x. To calculate $\mathbf{J}_{\delta\mathcal{W}}$, we calculate the total energy of the entire structure using Eq. (5) and (9). The derivatives (Eq. (4)) are calculated using reverse mode differentiation method that is implemented via the Python Autograd package.

The rest of the concatenation process follows exactly the same steps as for the previous type of interfaces. We highlight that the overall procedure is very flexible and can be extended to any other structural component by a proper consideration of the associated degrees of freedom. Algorithm 1 summarizes the general procedure to solve a multi-element structural problem with the presented concatenation strategy.

Algorithm 1. Finite Concatenated Element (FCE) for structural simulations

identify fundamental subdomains
from LE, select $\mathcal{N}_1, ..., \mathcal{N}_j$ (FNE models representing subdomains $1, ..., j$)
form $\boldsymbol{\mathcal{F}}_{int}$:
  for *nodal DOF in* N_{int} **do**
    add an (unknown) interface load
form the cost function $\mathbf{J}$:
minimize $\mathbf{J}$ to calculate $\boldsymbol{\mathcal{F}}_{int}^*$
  $\boldsymbol{\mathcal{F}}_{int}^* = \arg\min_{\boldsymbol{\mathcal{F}}_{int}} \mathbf{J}\left(\mathbf{f}, \mathcal{N}_1, ..., \mathcal{N}_j; \boldsymbol{\mathcal{F}}_{int}\right)$
use $\mathcal{N}_1, ..., \mathcal{N}_j$ to calculate the response of the subdomains $1, .., j$ subject to $\mathbf{f} + \boldsymbol{\mathcal{F}}_{int}^*$

3.3. Accuracy assessment

As discussed in Section 2 and more in detail in [1], FENA utilizes network elements to calculate the system static response to a given input load. From a general perspective, neural networks are trained to model a specific type of problem (e.g. a specific physical response) based on a set of training samples. After the training, the network performance is typically evaluated by using sample test data [35]. However, when the networks are used as surrogate models of classes of elements (as opposed to modeling specific systems), such as in the present study, dedicated tools are needed to assess the prediction accuracy. In other terms, for any specific physical configuration that does not belong to either the training or testing dataset, the accuracy of the prediction should be estimated based on predefined metrics. Different types of metrics were introduced and discussed in [1]. In the present study, every FENA model will be evaluated using the model ensemble (similarity) index [1] due to its ability to provide an overall level of confidence in the accuracy of the results.

Model ensemble index derives from the observation that the use of a group of networks can improve the overall quality of the prediction [44] compared to the case of a single network. The underlying idea is that if an ensemble of trained models predicts a set of highly correlated results, it is highly unlikely that these results are inaccurate. In order to implement the model ensemble index, a group of trained FNEs representative of the given system properties and analysis conditions is selected from the LE. Then, the predicted output of individual networks is inspected based on the Pearson product-moment correlation coefficient (PPMCC). PPMCC is a scale-free metric that allows assessing the correlation level (i.e. the similarity) of different data sequences. In the present case, it allows assessing the level of correlation between the predicted output of different networks. The PPMCC provides a symmetric matrix whose elements represent the scaled correlation between the predictions of each pair of models. For a set of accurate predictions, the datasets should be highly correlated, and hence all the elements of the PPMCC matrix should tend to unity. The closer this index is to unity (for multiple FNE predictions), the more reliable and accurate the predictions are. Based on this matrix, highly correlated models are picked and their average provides the problem solution. Further details on the assessment of FENA's performance by means of the PPMCC index will be discussed in Section 4.

4. Static analysis of structural elements and assemblies

This section presents practical examples and numerical results obtained by applying FENA to a set of different structural elements. In particular, these validation problems will consist in the static analysis of beams, plates, and stiffened panels under the effect of distributed loads. The first two classes of problems intend to demonstrate the performance of FENA for the two newly introduced surrogate models: the slender Euler Bernoulli beam and the thin Kirchhoff plate. The third and last class of problems will instead demonstrate the ability to assemble models of structural assemblies by combining together FNEs of different elements. FENA's predictions for all these case studies will be compared directly with those obtained via traditional FEA so to assess the overall accuracy of the method.

4.1. Static analysis of slender Euler–Bernoulli beams

We consider the static analysis of homogeneous elastic slender beams with finite stiffness boundary conditions applied at both ends and subject to a combination of a distributed transverse load and bending moment (Fig. 2a). Two different cases, labeled *Beam 1* and *Beam 2*, are considered. The two cases differ in the type of applied external loads as well as in the geometric and material parameters. Concerning the external loads, *Beam 1* is subject to a random (both in amplitude and location) load, while *Beam 2* is subject to a sinusoidal spatially distributed load. Both the transverse force F_z and the bending moment M_y are assumed to follow the same type of distribution. The applied external loads for both cases are plotted in Fig. 6a-b. Table 1 provides the specific geometric, material, and boundary parameters for the two case studies. Note that the properties were randomly selected from a uniform distribution within the training dataset parameters ranges (see Table A.1).

To simulate the static response of a beam under general transverse force F_z and bending moment M_y, we built and trained three networks (indicated, in the following, by the label $B_{\#}$). Recall that multiple FNEs are needed to implement the model ensemble index and assess the accuracy of the FNEs predictions. (Section 3.3). The architecture of each network follows the discussion in Section 3.1. The details of the training strategy as well as the performance of the trained network follow from the established methodologies and are discussed in the Appendix A.

Numerical results: Recall that the first step to simulate a structure in FENA consists in selecting the FNEs representative of the physical system. For both sample cases, *Beam 1* and *Beam 2*, the entire beam section can be represented by a single network since the length of both beams lies within the training (length) range of the available network models (B_{1-3}), hence not requiring any concatenation. The parameters for both beams were provided in Table 1. In both configurations, the beam is subject to the distributed transverse force and bending moment shown in Fig. 6a-b. The loads are applied at the nodal points that, following the discretization indicated in Table 1, amount at 29 and 45 points for *Beam 1* and *Beam 2*, respectively. In the case of *Beam 1*, the amplitude of the applied loads was randomly generated from a uniform distribution within the ranges of the network training dataset (see Table A.1). The external loads are only applied at nodal points, hence the discretization of the loads should be accomplished consistently with the FNE sectioning process.

Once the FNEs are selected and their input sequence is formed, the models are ready to simulate the response via the NS module. We simulated the two beam cases using the three available network models and then used the ensemble index (Section 3.3) to assess the accuracy of the predictions. The PPMCC matrices for the predicted responses are:

Table 1
Material, geometric, and boundary condition parameters selected for the analysis of *Beam 1* and *Beam 2*. The format $(\square,\square)$ used for K_z and K_{θ_x} indicates the boundary stiffness value at $x = 0$ and $x = L$, respectively.

	$E[GPa]$	$I[m^4]$	ν	$L[m]$	n	$K_z[kN/mm]$	$K_{\theta_x}[kN.m/rad]$
Beam 1	3.1	1.09×10^{-8}	0.3	4.95	44	$(703, 887)$	$(944.6, 10.1)$
Beam 2	11.7	2.27×10^{-9}	0.3	2.82	28	$(966, 1002)$	$(234.5, 10.2)$

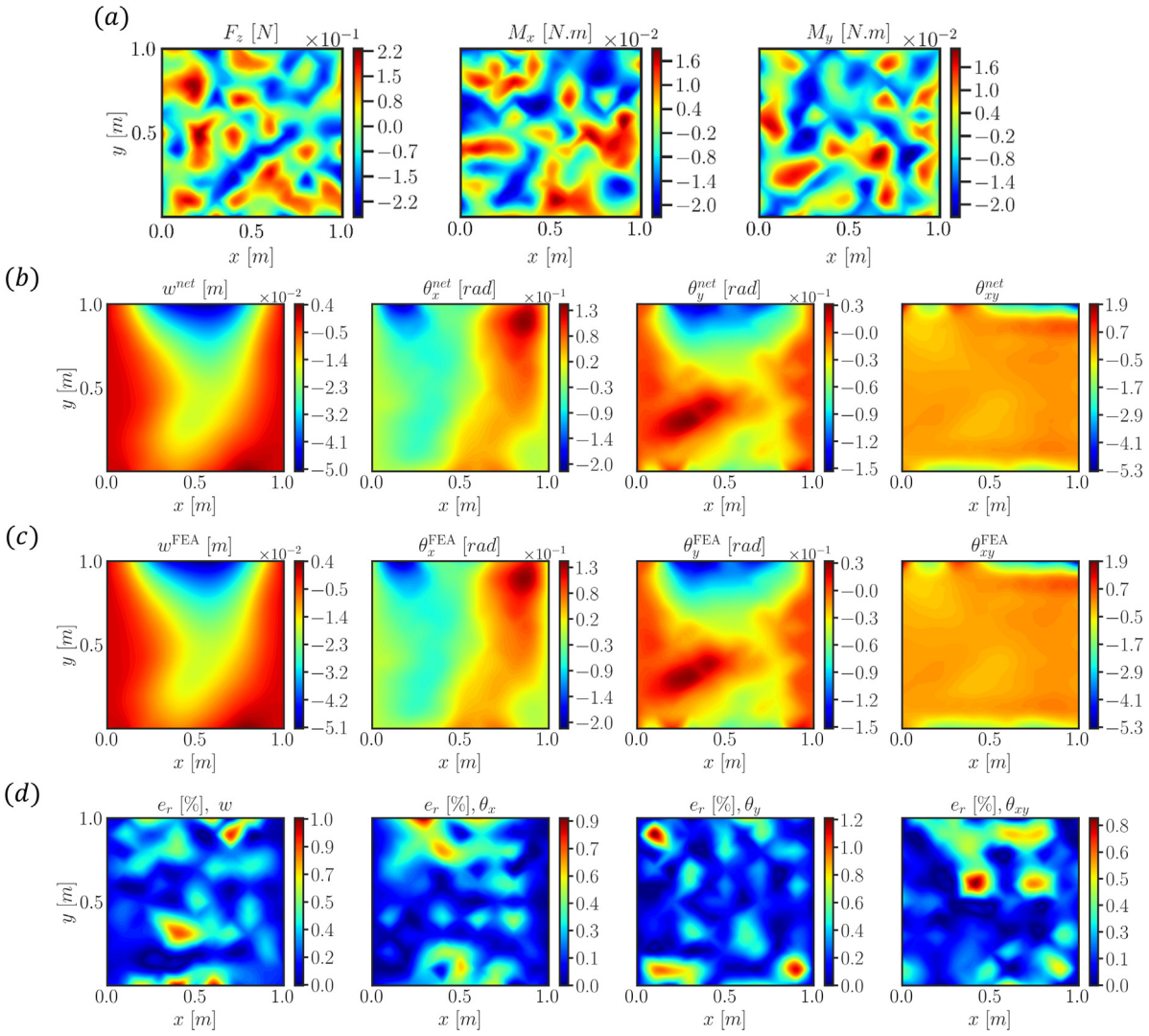

Fig. 7. Static response of *Plate 1* subject to a spatially distributed load with random amplitude. (a) 2D profile of the applied distributed force F_z and bending moments M_x and M_y. (b) Results for *Plate 1*. The predicted fields w, θ_x, θ_y, and θ_{xy} obtained by averaging predictions from models P_1, P_2, and P_3. (c) Reference solution of each nodal degree of freedom calculated via FE approach. (d) Distribution of the percentage relative error of the nodal degrees of freedom. The errors are calculated with respect to the FE results.

$$
\textit{Beam 1:}\begin{array}{c} \\ B_1 \\ B_2 \\ B_3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} B_1 & B_2 & B_3 \end{array} \\ \begin{bmatrix} 1 & 0.9999 & 0.9999 \\ & 1 & 0.9999 \\ \text{sym.} & & 1 \end{bmatrix} \end{array}\!\!\begin{array}{c} \\ B_1 \\ B_2 \\ B_3 \end{array}
$$

$$
\textit{Beam 2:}\begin{array}{c} \\ B_1 \\ B_2 \\ B_3 \end{array}\!\!\begin{array}{c} \begin{array}{ccc} B_1 & B_2 & B_3 \end{array} \\ \begin{bmatrix} 1 & 0.9997 & 0.9998 \\ & 1 & 0.9997 \\ \text{sym.} & & 1 \end{bmatrix} \end{array}\!\!\begin{array}{c} \\ B_1 \\ B_2 \\ B_3 \end{array}
\tag{11}
$$

The column and row labels describe the corresponding network model, while the numerical value indicates the normalized correlation between each pair of network models. The matrices are obtained via an element-wise average of the PPMCC matrix of w and θ_x for the case study. The matrices obtained above indicate that, for both problems, the network models predict a response that correlates very well with each other, within a maximum error of 0.03%. Note that highly correlated network element predictions indicate that the responses are likely to be accurate and reliable.

Since the results of all three models are well correlated, the average of their predictions is taken as the actual beam response. The average response from the three networks (B_1, B_2, B_3) is shown in Fig. 6c-d for the two beams, respectively. As expected based on the PPMCC results, a highly accurate response is predicted for both beam cases. The maximum relative percentage error (e_r) is less than 1.2% for *Beam 1*, and 1.5% for *Beam 2*. The error e_r is calculated as:

$$
e_{r_\square} = \frac{|u_\square^{net} - u_\square^{FEA}|}{max(|u^{FEA}|)} \times 100 \tag{12}
$$

In the above expression, $|(\cdot)|$ denotes absolute value, $u_\square^{FEA}$ is the reference solution at a nodal degree of freedom (either in terms of w or θ_x) and $u_\square^{net}$ is its corresponding network model prediction, where the subscript $\square$ indicates the node number. The reference solution is obtained by solving the same physical model via a finite element approach implemented in an in-house code. The excellent agreement between the two predictions shows that the beam FNEs are well capable of simulating the static response of a beam subject to generic loading and boundary conditions.

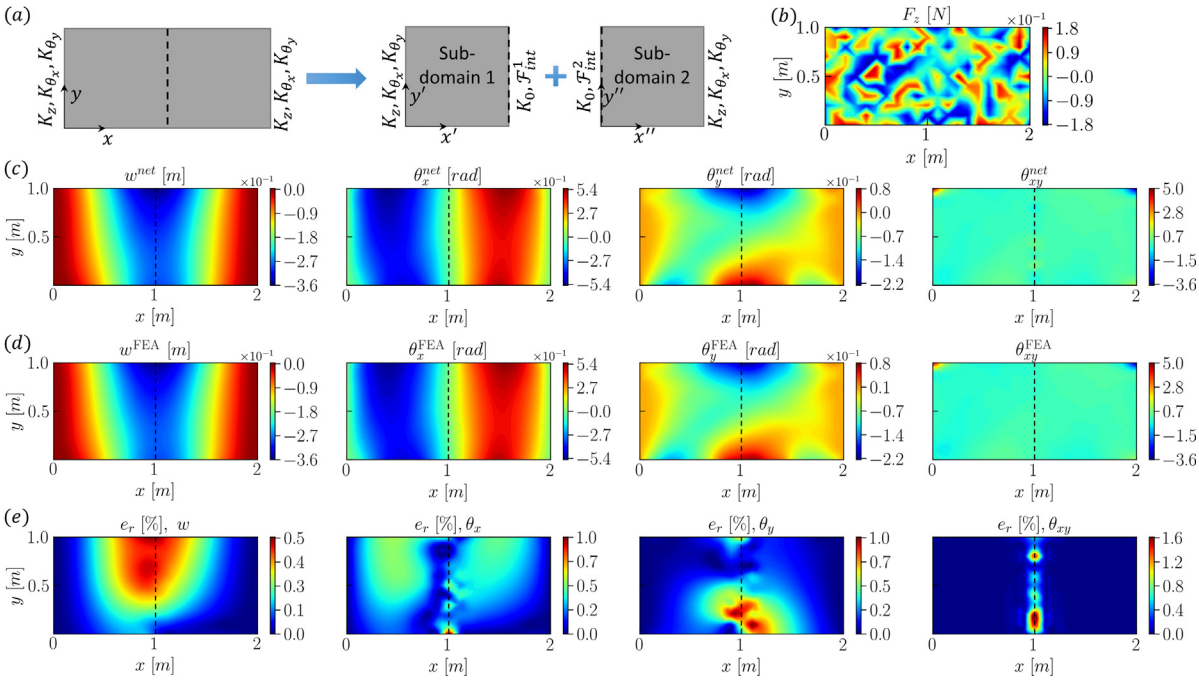

Fig. 8. Numerical response of *Plate 2* subject to a randomly valued and randomly distributed out-of-plane force F_z calculated using plate-plate FCE. (a) Schematic of *Plate 2* which consists of two plates of smaller size (subdomain 1 ($0 \leqslant x \leqslant 1[m]$) and subdomain 2 ($1 \leqslant x \leqslant 2[m]$)) connected along black dashed line. As in previous cases, the edges of the subdomains are connected to stiffness elements $K_z, K_{\theta_x}, K_{\theta_y}$. The interface loads $\mathcal{F}_{int}^{1,2}$ and fictitious stiffness $K_0 = \{K_{z_0}, K_{\theta_{x_0}}, K_{\theta_{y_0}}\}$ are added to the interface boundary of each subdomain. The values of the interface loads are determined by optimizing of the cost function $\mathbf{J}(\mathcal{F}_{int})$. (b) 2D profile of applied distributed force F_z for *Plate 2*. (c) The response of *Plate 2* to the applied force obtained by concatenation of subdomains 1 and 2. The black dashed line represents the interface between the concatenated sections. (d) Reference solution of *Plate 2* obtained from its FE model. (e) Percentage relative nodal error profile of each degree of freedom calculated with respect to the FE model of the plate.

4.2. Static analysis of thin Kirchhoff plates

Following the discussion and approach presented above to perform the static response of an elastic beam, we follow an equivalent path to present the static analysis of thin plates. As for the beam, two sample problems, labeled as *Plate 1* and *Plate 2*, are considered. The main difference between the two plate samples is their domain size. More specifically, *Plate 1* has dimensions $(L \times W) = (1 \times 1)[m^2]$ that fall within the training range of an individual plate FNE, hence allowing the simulation to be performed using a single FNE. In the case of *Plate 2*, the dimensions are $(L \times W) = (2 \times 1)[m^2]$ which requires the use of concatenation. The case of *Plate 1* practically represents the situation illustrated in Fig. 2c, while *Plate 2* corresponds to the case depicted in Fig. 8a. A summary of the material and geometric parameters for the two plates is provided in Table 2. The specific parameters were randomly selected within the ranges reported in Table B.1). In both cases, the boundaries at $y = 0$ and $y = 1$ were left free and those at $x = 0$ and $x = 1$ in *Plate 1* (or, equivalently, at $x = 0$ and $x = 2$ in *Plate 2*) were connected to elastic elements (i.e. a stiffness boundary condition) acting both in the direction of the translation w and of the rotation θ_x and θ_y. The corresponding values of the stiffness elements are $K_z = 15 \times 10^6[N/m], K_{\theta_x} = K_{\theta_y} = 17.17[N.m/rad]$. The specific applied loads are shown in Fig. 7a and 8b for *Plate 1* and *2*, respectively.

To simulate the response of a thin plate subject to generalized distributed loads (including a transverse force F_z, and the bending moments M_x, and M_y), we trained three FNEs; they will be indicated in the following by the label $P_{\#}$. The general architecture of the plate FNEs was described in Section 3.1. Also, we highlight again that we will use the predicted response of the three plate FNEs to implement the model assessment strategy described in Section 3.3 and to evaluate the accuracy and reliability of the predictions. Details on the training process of the plate FNEs can be found in Appendix B.

Numerical results: In this section, we present the prediction performance provided by FENA for the two different plate samples.

For *Plate 1*, the computational domain size is consistent with the ranges used for the training of the plate network elements in the LE, hence concatenation is not needed and a single FNE can be used to simulate the whole plate. The external loads were applied on a grid of 11×11 evenly spaced nodes (corresponding to a spatial discretization of $0.1[m]$ in both the x and y directions). This grid was consistent with the one used during the training phase. The response of the plate was simulated using the three networks and their prediction accuracy was analyzed via the model ensemble index. The PPMCC matrix for the predicted response of *Plate 1* is equal to:

$$\textit{Plate 1:} \begin{array}{c} \\ P_1 \\ P_2 \\ P_3 \end{array} \begin{array}{c} \begin{array}{ccc} P_1 & P_2 & P_3 \end{array} \\ \begin{bmatrix} 1 & 0.9999 & 0.9999 \\ & 1 & 0.9999 \\ \text{sym.} & & 1 \end{bmatrix} \end{array} \tag{13}$$

Table 2
Material, geometric, and boundary condition parameters selected for the analysis of *Plate 1* and *2*.

	$E[MPa]$	ν	$L[m]$	$W[m]$	$t[m]$	K_z $[kN/m]$	K_{θ_x} $[N.m/rad]$	K_{θ_y} $[N.m/rad]$
Plate 1	30	0.3	1.0	1	0.005	15×10^3	17.17	17.17
Plate 2	30	0.3	2.0	1	0.005	15×10^3	17.17	17.17

The above matrix was obtained first by calculating the four PPMCC matrices corresponding to $w, \theta_x, \theta_y, \theta_{xy}$ and then by averaging the correlation of each pair of models over the four degrees of freedom. It is seen that the three models provide highly correlated predictions with less than 0.01 % error, which ultimately indicates the accuracy of the predicted responses. The average of the P_{1-3} predictions is used as the plate response to the applied external loads. The numerical prediction of the static response of the plate is presented in Fig. 7b. The reference solution is, once again, generated via an in-house finite element code and plotted in Fig. 7c. The results show that the network model accurately predicts the response of *Plate 1*, within 1.21% margin of error (see θ_y relative error map in Fig. 7d) compared to the FE model solution. The distributions of the relative percentage error between the prediction and the reference solution are also plotted in Fig. 7d. It is worth highlighting that, in most of the domain, the prediction error falls below 0.5 %.

In order to further illustrate the capabilities and performance of FENA, we also produced an independent set of results based on other possible deep-learning-based methodologies. More specifically, we developed a surrogate model of a plate based on traditional architecture consisting in a deep fully connected feedforward neural network (DNN). The results, presented in Appendix Section B.3, show that FENA's predictions are significantly more accurate than DNN-based networks.

For *Plate 2*, the dimensions exceed the range used in the training phase, hence requiring the use of concatenation to assemble the full domain and to perform the simulation. Without loss of generality, we assumed that the external moments M_x and M_y were equal to zero while the external transverse force F_z was random in amplitude and applied over a grid of 21×11 evenly spaced nodes (q = 231).

The full plate model was built by concatenating two plate FNEs having dimensions $(L \times W) = (1 \times 1)[m]$ and containing 121 nodes (grid of 11×11 nodes) each. The elements were concatenated along the interface at $x = 1$ (dashed black line in Fig. 8a) into a plate of dimensions $(L \times W) = (2 \times 1)[m]$. As described in Section 3.2, the concatenation involves the introduction of a set of interface loads to all the interface nodes (Fig. 5c). The interface between the left and right sections located at $x = 1$ contains 11 nodes (according to the selected discretization), hence $N_{int} = 11$. Four interface loads are assigned to each node of each section, hence the size of the vector $\mathcal{F}_{int}$ is equal to 88 (total number of loads). Following the formulation presented in Eqs. (4) and (8), (9), a cost function was assembled. The optimization at the basis of the concatenation process was initiated using the weights obtained by randomly initializing the values of the interface loads, as described in Section 3.2. For this specific problem, the optimization was performed using the SLSQP method in Python (SciPy package) and converged in 214 steps.

The response of the full plate is presented in Fig. 8c and compared with the reference solution produced by the finite element method, as shown in Fig. 8d. The average relative percentage error among the four nodal degrees of freedom yields a maximum of 1.6% in the prediction of θ_{xy} as seen in Fig. 8e, hence indicating that the concatenation algorithm accurately captures the behavior of the interface. It is important to recall that at no time during the solution process, the networks were retrained. The simulations only used the general pre-trained network models available in LE to predict the response of the composite plate.

4.2.1. Post-Processing: Calculation of derived quantities

It is well known that the FE solution of a structural model produces the nodal quantities, such as nodal displacements. Other key quantities, such as stress and strain fields, are calculated as a secondary step by post-processing the displacement data. Similarly, in FENA, the FNEs provide the displacement field and other depen-

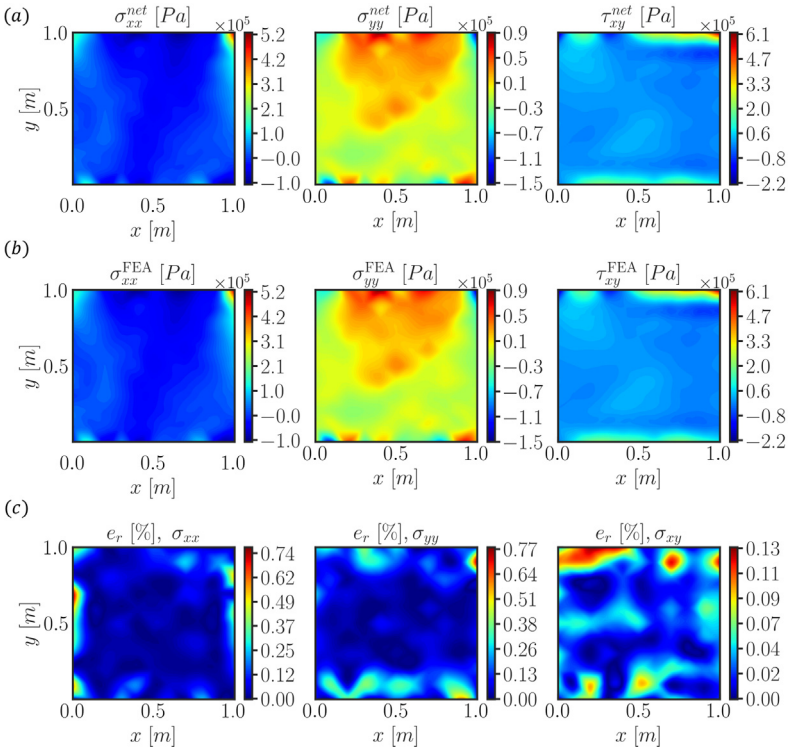

Fig. 9. Components of the stress field for *Plate 1*. (a) Stress field predicted by FENA. The stress fields σ_{xx}, and σ_{yy}, τ_{xy} are obtained by averaging the predictions from the three models $P_{S_{21}}, P_{S_{22}}$, and $P_{S_{23}}$. (b) Reference stress fields calculated via FEA. (c) Percentage relative error between the predictions of FENA and FEA.

dent quantities should be calculated in the post-processing step. In this section, we present an example of how stresses and strains of a plate element can be calculated in FENA.

The plate FNEs developed in this study are specifically trained to predict (solely) nodal displacements in response to input loads. In order to calculate the stress fields in FENA, we can envision two possible approaches: 1) numerical stress–strain calculation, and 2) neural network-based stress–strain calculations. The first approach could somewhat be considered as analogous to the one used in FEA, that is, strain fields are calculated from the numerical derivative of the displacements and the stresses are computed by constitutive arguments. However, it is known that this approach may result in large errors associated with the implementation of the numerical derivatives. In addition, this approach might be potentially inferior to its counterpart in FEA, which instead leverages the derivatives of the shape functions to calculate the strains. While the above is certainly a possible approach, we believe that the network-based approach is superior and is the one to be followed.

In the second approach, we can leverage neural networks to perform the stress–strain calculations. In this case, the network elements could be developed and trained to predict also stress and strain fields, in addition to generalized displacements. For example, in the case of plate FNEs, in addition to the four output channels that provide the nodal generalized displacement $[w, \theta_x, \theta_y, \theta_{xy}]$, six output channels should be added to the FNEs to predict the stress and strain fields. It follows that the network output vector is represented by $[w, \theta_x, \theta_y, \theta_{xy}, \sigma_{xx}, \sigma_{yy}, \tau_{xy}, \epsilon_{xx}, \epsilon_{yy}, \epsilon_{xy}]$, where σ is normal stress, τ is shear stress, and ϵ is strain. Another possibility is to train a separate network for the prediction of stress and strain fields based on the external loads applied to the structure. In this case, we would obtain a network with the output channel $[\sigma_{xx}, \sigma_{yy}, \tau_{xy}, \epsilon_{xx}, \epsilon_{yy}, \epsilon_{xy}]$ that maps the plate applied external loads to stress and strain fields. Along this similar line of thought, a separate network could also be designed and trained to map the displacement field generated by FENA to the stress and strain fields. In the case of network-based calculations, the possibility of large errors would be minimal, since no numerical calculations are performed and the FNEs are specifically trained to simulate resultant stress fields. Certainly, the quality of these predictions will be dominated by classical considerations on the quality of training data and the overall accuracy of a neural-network-based prediction.

To assess the prediction performance and demonstrate the feasibility of the network-based approach, we developed two groups of networks dedicated to stress and stress fields calculations. The first group directly maps the applied external loads to stress and strain fields. The second group receives the displacement field predicted by FENA and maps it to the stress and strain fields. We note that this second group is the most realistic because it takes the actual (displacement) output from FENA and post-process it to obtain additional quantities. The networks in the first group are labeled $P_{S_{11}}, P_{S_{12}}$, and $P_{S_{13}}$ while those in the second group are labeled $P_{S_{21}}, P_{S_{22}}$, and $P_{S_{23}}$. The average of the predictions of the three networks will be used to define the stress or strain fields for a given problem (analogously to the approach already used for displacements). The technical details of the network development and training are presented in Appendix Section B.4, while here we focus on the results.

To showcase the performance of the network-based approach, we focus directly on the second group of networks (that is displacements to stresses and strains) applied to *Plate 1* which we already defined above as the most suitable and logical for FENA.

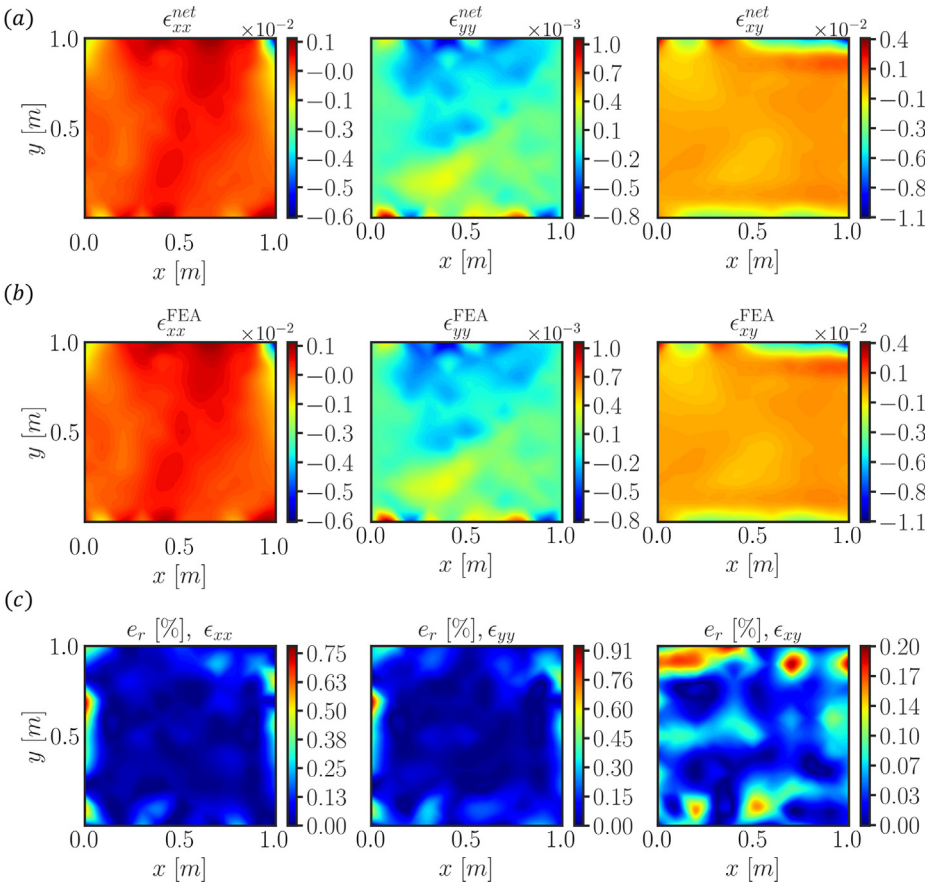

Fig. 10. Components of the strain field for *Plate 1*. (a) Strain field predicted by FENA. The strain fields ϵ_{xx}, and $\epsilon_{yy}, \epsilon_{xy}$ are obtained by averaging the predictions from the three models $P_{S_{21}}, P_{S_{22}}$, and $P_{S_{23}}$. (b) Reference strain fields calculated via FEA. (c) Percentage relative error between the predictions of FENA and FEA.

Figs. 9a and 10a present the stress and strain fields predicted by processing the displacement field calculated by FENA through the stress–strain networks for the sample problem *Plate 1*. Recall that these fields are the result of the average of the prediction of the three networks $P_{S_{21}}, P_{S_{22}}$, and $P_{S_{23}}$. The reference solution, which is generated via FEA, is plotted in Figs. 9b and 10b. The distributions of the relative percentage prediction error are plotted in Figs. 9c and 10c. The results show a remarkable accuracy of the network ensemble predictions with maximum relative error always below 1% (absolute maximum across all results is.91%) compared to the FEA solution. In conclusion, the networks are able to accurately map the displacement field predicted by FENA to the corresponding stress and strain fields.

For completeness, we note that the prediction results of the first group of networks showed relatively lower accuracy. Specifically, the average maximum prediction error was within the range $[\%2, \%5]$ (see discussion in Appendix Section B.4).

4.3. Static analysis of structural assemblies: stiffened panel

The introduction of two new network elements (i.e beam and plate) and the extended concatenation capabilities, we can address the more general problem of building the model and simulating the static response of structural assemblies made of multiple dissimilar elements. Given the availability of both beam and plate elements, it is quite natural to choose a stiffened plate as a representative example of a structural assembly.

In this case, the problem domain is composed of a $(L \times W) = (1 \times 1)[m]$ plate and two stiffeners (i.e. beam elements) placed orthogonal to each other (Fig. 11a). The material and geometric properties of the plate as well as the boundary conditions are consistent with those used in *Plate 1* (see Table 2). The two identical stiffeners connected to the plate have length equal to $1[m]$, Young's modulus $E = 5[GPa]$, and area moment of inertia $I = 5.73 \times 10^{-9}[m^4]$ (stiffeners are assumed to have a rectangular cross section with the dimension $width \times height = 5.7 \times 22.9[mm]$). The plate was loaded by an external transverse load F_z acting in the out-of-plane direction while the bending moments were set to zero. The F_z load was made of nine identical concentrated loads applied at the grid points $(x, y) = [.4, .5, .6] \times [.4, .5, .6][m]$. Each load had an amplitude of $0.5[N]$. The resulting load is plotted in Fig. 11a. Fig. 11b compares the transverse displacement profile, obtained via FEA, for the plate with and without stiffeners. These plots highlight the effect of the stiffeners on the static response of the plate and are provided as a reference solution to better interpret the prediction obtained from FENA.

In FENA, the stiffened plate model is assembled using the multi-element concatenation method described in Section 3.2. The structure is composed of one plate and two beam stiffeners; each subdomain is simulated via a single FNE directly extracted from LE. The plate is discretized using a grid of 11×11 evenly spaced nodes analogous to that used in the finite element solution, as shown in Fig. 11a. Each beam is also divided into ten sections (a total of 11 nodes per beam) so to match the plate discretization and easily allow interfacing the elements at the 11 interface nodes. As previously explained, the implementation of the interface requires the addition of a set of interface loads per interface node; these interface loads must be added on the nodes belonging to both the plate and the beam stiffeners. Following the procedure outlined in Section 3.2.2, the chosen interface requires 132 interface loads. The specific load values are determined via an optimization process that minimizes the cost function $\mathbf{J}(\mathcal{F}_{int})$ (Eqs. (4) and (10)). The same optimization strategy followed for the simulation of *Plate 2* was also used in this case.

Fig. 11c shows the response predicted by FENA for the fully assembled system. red with the reference results obtained by solving the same structural assembly via an in-house FE code. The relative percentage error (Eq. (12)) is also estimated for each degree of freedom by using the FEA solution as reference. Across the four nodal degrees of freedom, the maximum error never exceeds 1.0%. This maximum value is observed on the θ_{xy} degree of freedom, as seen in Fig. 11e. As seen from previous analysis conducted on individual element types, also for structural assemblies FENA is capable of providing a highly accurate prediction without requiring any training after model assembly. Recall that all the FNEs represent surrogate models of generic classes of elements. When building a model, they are pulled directly from LE and never trained on the specific configuration.

4.4. Remarks on the computational performance of FENA

At this point, it is important to provide a few remarks on the computational aspects of FENA. There are different contributions to the overall computational cost associated with the development and application of FENA.

The first contribution is related to the development of FENA's elements database (LE), that is the cost of generating the training data samples and of training the FNEs. To this point, it is important to recall that both the training data generation and the network training in FENA are operations that are performed only once when building the simulation capabilities. This operation can be conceptually compared to building a finite element package; once the package is built in all its components, it can be used for any future analysis with no further computational burden. It is from this perspective that the training cost in FENA should not be counted as part of the total simulation time because it would not be required for a simulation if the FENA package is already available. Despite these considerations, we still provide here below an analysis of the computational cost for both data generation and network training in addition to the most critical component that is the cost of the actual predictions.

(1) Generation of training datasets: With reference to the specific structural elements considered in this study, data generation and network training are not very computationally expensive steps since the simulated domains do not involve a large number of degrees of freedom. Specifically, generating the training datasets for both beams and plates required less than 3600 [s]. The data were generated on a HPC cluster node equipped with a Xeon Gold 5218R CPU with 40 cores at 2.10 GHz base frequency and 192 GB of memory. Nevertheless, it can be envisioned that the data generation step can increase with the increasing complexity of either the models or the analyses; as an example, consider the cases of either largescale systems or nonlinear analyses.

From a more general perspective, the data generation for problems involving largescale systems can be approached in different ways. In its current formulation, FENA leverages the powerful concept of concatenation to assemble arbitrary sized models from the basic classes of elements available in LE. This concept was extensively explained in the manuscript and allows modeling domains with any size starting from pre-trained FNEs. The data generation in this approach is neither computationally expensive nor depends on the availability of powerful computational resources since only small domains need to be simulated and trained. Another possible deep-learning-based approach for problems with a limited number of available training data (for example, in the case of largescale and nonlinear systems), the concept of transfer learning [45–48] could be implemented. In this method, trainable parameters of a trained neural network are fine-tuned to solve a similar physical problem with a slightly different configuration. Transfer learning has been

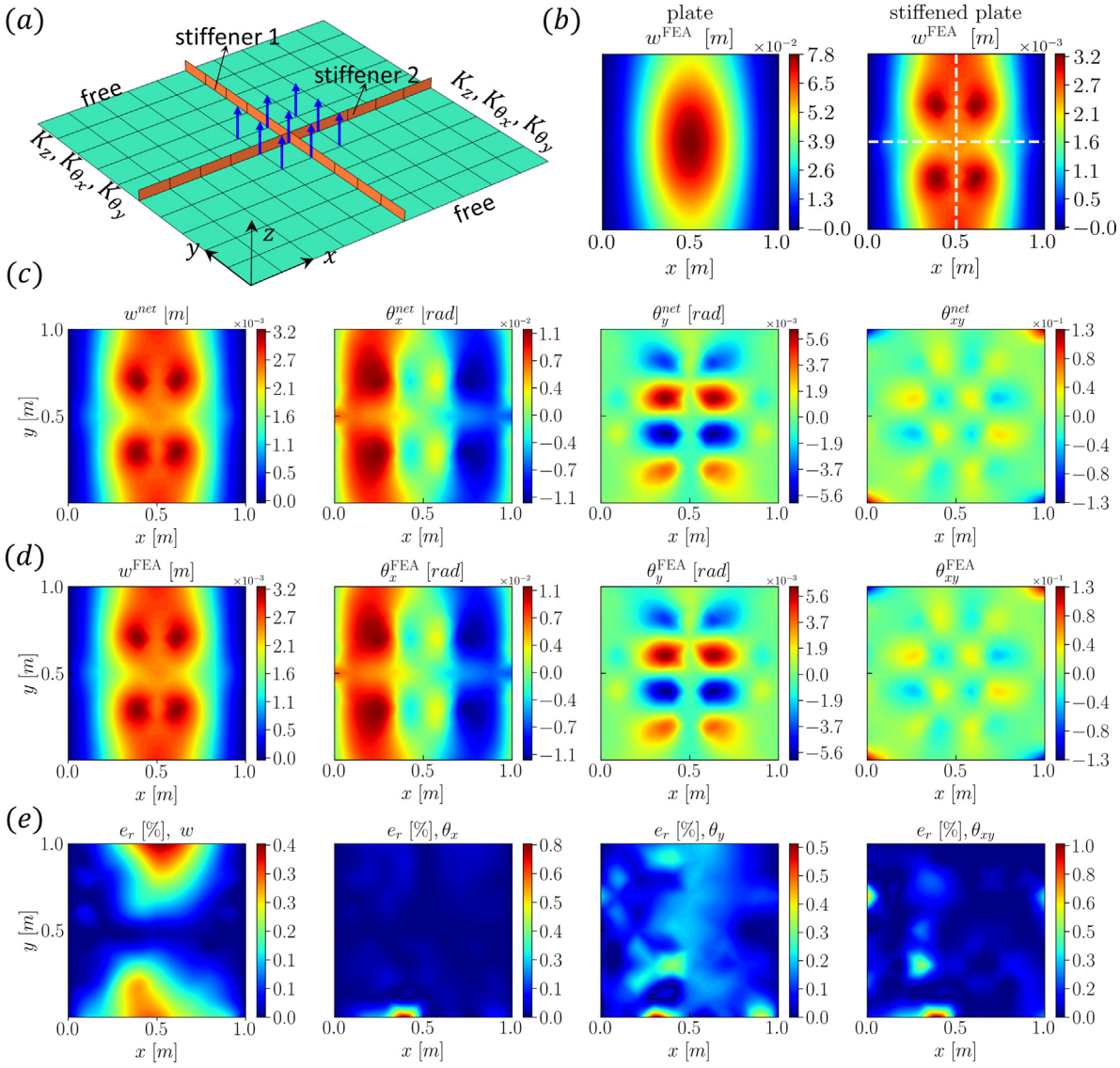

Fig. 11. Static response of a stiffened plate to a transverse distributed load. (a) Schematic of the stiffened plate. The plate is reinforced with two stiffeners (labeled 1 and 2) and subject to a transverse load applied to nine nodes, as shown by the blue arrows. While the figure shows the assembled plate, recall that the three structural elements are connected only in terms of FNEs via concatenation. The black grid shows the uniform mesh used in both models (i.e. FE and FENA). The plate has free boundaries at $y = 0$ and $y = 1$, and stiffness boundary conditions at $x = 0$ and $x = 1[m]$. The stiffness values of $K_z, K_{\theta_x}, K_{\theta_y}$ are presented in Table 2. (b) The static deflection of the plate calculated by finite element method (left) without and (right) with stiffeners. The stiffeners location is indicated by the dashed white lines. (c) Static response of the stiffened plate calculated by FENA and presented in terms of each individual degree of freedom w, θ_x, θ_y, and θ_{xy}. (d) Reference solution for the static response of the stiffened plate obtained via FEA. (e) Percentage relative error distribution for each degree of freedom (calculated with respect to the reference FEA solution).

widely used in image processing and natural language processing. In these applications, the networks are pre-trained with very large datasets [45]. These networks are then fine-tuned to perform a similar task using a smaller dataset. The transfer learning concept is based on the fact that the network parameters ideally converge to slightly different values for similar tasks. Hence, a neural network can be trained with an available dataset (with a large number of training data samples) and then fine-tuned to perform a similar task using a smaller number of data that are available for this new similar task.

(2) Network training and predictions: The HPC node used for data generation was also equipped with an NVIDIA Tesla V100 GPU, which was used to train the networks. The average training time for the beam and plate FNEs was 8.5 and 22$[h]$, respectively. We highlight that, once the networks are trained, the computational cost for network predictions is negligible (given that the prediction phase does not involve solving the governing equations of the problem). As discussed in Q1, a trained network can be interpreted as a transfer function that relates the input to the output via the system parameters. Hence, a minimal computational effort is needed to predict the network output once the transfer function is determined (i.e. the network trainable parameters are calculated). As an example, the FE model takes 1.052$[s]$ to simulate *Plate 1* and calculate the deformation of the plate. However, the plate network elements predict the response in 0.033$[s]$. The difference in the computational time is mainly due to the stiffness matrix assembly and inversion that are performed in the FE analysis. When using neural networks, these steps are completely eliminated, hence providing a remarkable computational advantage.

Continuing with the analysis of the computational performance, in the sample problem *Plate 2* we looked at the analysis of multi-component models based on interconnected pre-trained networks

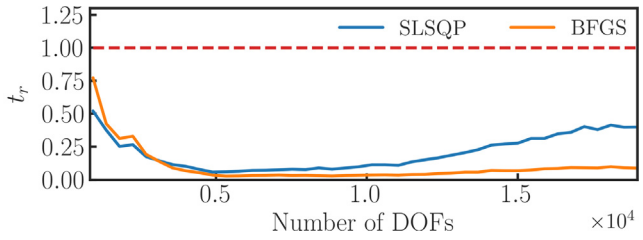

Fig. 12. Ratio t_r of the computational time of FENA and FEA as a function of the increasing number of degrees of freedom for the case of *Plate 2*.

via FCE. The computational time for FEA and FENA is 9.680[*s*] and 5.88[*s*], respectively. Hence, FENA is 39.3% faster than FEA. The simulation time of FEA and FENA for the stiffened plate problem is 6.733[*s*] and 1.905[*s*], respectively. This is a 72% decrease in the computational time.

We further illustrate the performance of FENA and its concatenation approach (FCE) via an example that uses a structural system similar to *Plate 2*, but it is simulated using an increasing number of elements. More specifically, we simulated plates with an increasing number of DOFs ranging from 924 to 19404 in increments of 440 (by adding a subsection with 10×10 elements to the domain at each increment). The properties and boundary conditions of the sample problems were the same as *Plate 2*. The applied external loads of each problem were selected randomly from a uniform distribution. The structural response was obtained via both FEA and FENA (using FCE). In these simulations, we calculated the interface load with BFGS and SLSQP optimization methods. All the simulations were performed on the same HPC node. The performance of FENA are expressed in terms of a ratio between the simulation time of FENA and FEA ($t_r = t_{\text{FENA}}/t_{\text{FEA}}$). Fig. 12 presents t_r vs DOF for the two optimization methods. The results show that for all the simulated sample problems, FENA simulation time is lower than FEA. It is also seen that the concatenation simulation time (and consequently t_r) depends on the optimization algorithm. More specifically, for lower DOF, SLSQP method can determine the interface loads faster than BFGS, while for larger DOF BFGS optimization method requires less time to calculate the interface loads. Note that the proposed concatenation methodology has not been optimized for performance yet. Therefore, we envision that further computational gains can be obtained by further refining this step. Nonetheless, FENA consistently outperforms FEA. Indeed, an optimized concatenation strategy that does not rely on a numerical optimization method will further enhance the computational efficiency of FENA. An additional supporting example case can be found in [1].

5. Conclusions

The concept of Finite Element Network Analysis (FENA), a neural network-based computational framework for the simulation of physical systems, was significantly extended in two key areas: (1) the database of elements was expanded to include additional structural element types, and (2) to demonstrate the ability of FENA to simulate structural assemblies made of dissimilar elements. FENA uses a library of network elements trained to simulate general classes of structural elements, spanning a variety of geometric and material properties, loads, and boundary conditions. These elements can be specialized for a given type of analysis provided detailed properties, external loads, and boundary conditions without requiring any additional training. This is a remarkable feature compared to other network-based simulation approaches that require *ad hoc* training for any minor change in the problem conditions.

This unique behavior is made possible thanks to a combination of innovative factors. The surrogate models are synthesized based on bidirectional recurrent neural networks (BRNN) architectures. BRNNs allow a bidirectional flow of information that enables the network to be affected not only by prior but also by future states. When the states represent the spatial response of the systems, this property allows a unique capability to deal with different boundary conditions without any need for retraining the network. In addition, the basic RNN architecture allows dealing with input sequences having different sizes, which is a property uniquely suited to account for problem-specific input (e.g. external loads).

The BRNN architecture also enables implementing the fundamental concept of concatenation, which allows assembling different pre-trained network elements (available from the library) to form multicomponent models and without requiring any further training. While this characteristic was shown in a previous study for one-dimensional elements, this study extended it for one- and two-dimensional elements further generalizing the concept by means of a variational formulation. Following the introduction of the new elements formulation and of the concatenation approach, several numerical test cases were presented in order to test FENA's performance. The comparison with traditional finite element solutions, retained as baseline results in this study, highlighted the excellent predictive capabilities of FENA and, even more important, validated the ability to build network-based models without any need for problem-specific training. Several additional result sets were also provided to further elucidate the computational cost of FENA and the ability to obtain post-processed quantities such as stress and strain fields.

It could be envisioned that, by further expanding FENA's functionalities and analysis capabilities, the framework could enable the accurate and rapid simulation of extremely large physical systems, such as those characterized by multiscale features in either space or time.

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 gratefully acknowledge the financial support of the National Science Foundation (NSF) under grant CAREER #1621909. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Appendix A. Beam FNE training

In the following, we describe the dataset used to build FNEs representative of Euler–Bernoulli beams. We also discuss the training details of the beam FNEs.

A.1. Training dataset

The dataset used to train the beam FNE was generated via an in-house finite element code formulated according to [41]. The developed FEA code was validated by comparing a few sample cases with the solution obtained via the commercial FEA software COMSOL. The decision to use an in-house FEA code was made because it allows for a more efficient dataset generation when considering a variety of applied loads and boundary conditions. The ranges of the input parameters used for the data generation are reported

Table A.1
Parameters used to generate the training dataset for the beam FNEs.

parameter	value	parameter	value
$E[GPa]$	$[2.5, 20]$	$F_z[N]$	$[-1, 1]$
$I[m^4]$	$[5.21 \times 10^{-11}, 1.33 \times 10^{-8}]$	$M_y[N/m]$	$[-0.2, .2]$
$l[m]$	$[2.5 \times 10^{-2}, 2.0 \times 10^{-1}]$	$K_z[N/m]$	$[2.56 \times 10^{6}, 2.05 \times 10^{8}]$
n	$[10, 60]$	$K_{\theta_x}[N.m/rad]$	$[1.95, 19.53]$

in Table A.1. Note that the assumption of linearity for the beam behavior allows for a simple and intuitive scaling process of the range of parameters and of the output of the trained network in order to simulate problems outside the range indicated in Table A.1. More specifically, since the problem is linear, the ranges used for training can be easily scaled. We also assumed that each sample problem had a uniform area moment of inertia I and Young's modulus E. The values of E and I were randomly selected from a uniform distribution within the ranges presented in Table A.1 in order to obtain general networks that are not biased towards a specific region of the training parameters. Further, for each input sequence size n (total number of elements in the beam domain), we sampled the values of the length of each element and the applied loads (F_z and M_y) via Latin Hypercube Sampling (LHS) method to ensure that all portions of the parameters space were appropriately sampled [49].

For each n within the range $[10, 60]$, we generated 3200 sample problems (25 batches of 128 sample problems). Hence, the entire training dataset includes 163,200 ($25 \times 128 \times 51$) sample problems. We also generated three validation datasets to assess any eventual overfitting during the training for each beam FNE. The validation datasets included 24480 samples, that is equal to 15% of the training dataset size.

We generated three datasets with the above mentioned conditions and used them to train the network elements B_1, B_2, and B_3, respectively. Individual randomly generated datasets were used to guarantee statistical independence of the networks B_{1-3}. Recall that independent networks are needed to apply the model ensemble index of the MA module(Section 3.3).

We note that the solutions based on FEM is only one of the possible options to construct training datasets. In fact, any available solution of a system (e.g. solution of an analytical model or even experimentally obtained data) can be leveraged to train network elements representing the system. Even a hybrid training approach based on data from different sources (analytical, numerical, and experimental) can be envisioned. Once the training is completed, it is expected that the network element predicts a solution close to (hence an approximation of) the original solution used to generate training dataset samples.

A.2. Network training

In this section we discuss the training of the beam FNE. As mentioned in Section A.1, to simulate the static response of a beam under general transverse force F_z and bending moment M_y, we built and trained three networks B_1, B_2, and B_3 via Python, Keras, and Tensorflow packages. The architecture of each network follows the discussion in Section 3.1. The described architecture of the beam FNE results in a total of 81,394 trainable parameters.

From a general perspective, a neural network can be considered as a (nonlinear) mapping between the network input and output. This mapping is controlled by a set of trainable parameters (weights and biases) that are optimized during the training. More specifically, the output of a network is given by:

$$\mathbf{Out} = \mathcal{N}(\mathbf{In}; \Theta) \tag{A.1}$$

where $\mathcal{N}$ is the network, $\mathbf{x}$ is the network input, $\mathbf{y}$ is the network output, and Θ is the network trainable parameters. Θ is calculated during the training procedure through an optimization process. The training process can be expressed as:

$$\Theta^* = \arg \min_{\Theta} \mathcal{L}[\mathcal{N}(\mathbf{In}; \Theta), \mathbf{Out}^{true}] \tag{A.2}$$

where $\mathbf{y}^{true}$ is the true output value corresponding to input $\mathbf{x}$, which is available in the training dataset that consists in pairs of input and their true outputs, i.e. $(\mathbf{x}, \mathbf{y}^{true})$. The loss function $\mathcal{L}$ is an index that measures the difference between network prediction and true output values ($\mathbf{y}^{true}$), e.g. mean square of prediction error. The training step, in fact, tunes the network trainable parameters in order to obtain network predictions close to $\mathbf{y}^{true}$. Hence, for example, if the $\mathbf{y}^{true}$ is obtained from the analytical solution of a system, then during the training, the network learns to predict output close to the analytical solution.

We used a physical law informed loss function for training the networks. The loss function was defined by the weighted summation of two terms: 1) $\mathcal{L}_{\text{MSE}}$ and 2) $\mathcal{L}_{\delta\mathcal{W}}$. The general expression is given by:

$$\begin{aligned} \mathcal{L} &= \lambda_1 \mathcal{L}_{\text{MSE}} + \lambda_2 \mathcal{L}_{\delta\mathcal{W}} \\ \mathcal{L}_{\text{MSE}} &= \frac{1}{N_b} \sum_{i=1}^{N_b} \left(\mathcal{U}_i^{net} - \mathcal{U}_i^{true}\right)^2 \\ \mathcal{L}_{\delta\mathcal{W}_i} &= \frac{1}{N_b} \sum_{i=1}^{N_b} \left(\frac{\partial \mathcal{W}}{\partial \mathcal{U}_i^{net}}\right)^2 \end{aligned} \tag{A.3}$$

where λ_1 and λ_2 are weight factors, $\mathcal{L}_{\text{MSE}}$ is the traditional mean squared error (MSE) of network predictions, $\mathcal{L}_{\delta\mathcal{W}}$ is the energy term, N_b is the batch size, $\mathcal{U}^{net}$ is the beam displacement vector predicted by the network, $\mathcal{U}^{true}$ is the correct beam displacement vector obtained from dataset, $\mathcal{W}$ is the total energy of the beam (see Eq. (4)) calculated based on network prediction $\mathcal{U}^{net}$. While the first term of the loss function enforces the predictions to be the same as the true values provided by the training samples, the latter term ($\mathcal{L}_{\delta\mathcal{W}}$) enforces the physical law of the problem. More specifically, $\mathcal{L}_{\delta\mathcal{W}}$ enforces the principle of virtual displacements. At the beginning of the training phase, we randomly selected 256 batches of sample problems to determine the weight factors λ_1 and λ_2 using the following relation:

$$\lambda_1 = \frac{1}{\frac{1}{256}\sum_{i=1}^{256} \mathcal{L}_{\text{MSE}}|_{i^{th}\ batch}}, \quad \lambda_2 = \frac{1}{\frac{1}{256}\sum_{i=1}^{256} \mathcal{L}_{\delta\mathcal{W}}|_{i^{th}\ batch}} \tag{A.4}$$

We used the default weight and bias initializer [50,51] in the Keras package. More specifically, the biases were initialized to zero, while the *Glorot uniform initializer* [52] (also known as *Xavier uniform initializer*) was used to initialize the weights. This initializer extracts samples from a uniform distribution within the range $[-a, a]$, where $a = \sqrt{6/(n_{in} + n_{out})}$, n_{in} is the layer input size (number of neurons in a layer), and n_{out} is the layer output size. Using ADAM (adaptive momentum)[53] optimization algorithm, the networks were trained for a total of 1000 epochs. The learning rate was initially set to .0001, and was divided by a factor of two every 250 epochs (implemented via Keras learning rate scheduler).

Further, given that each training iteration must be performed via a batch of data with the same input–output sequence length, we defined and utilized a data generator function to load batches of sample problems with the same size and used the Keras *fit_generator* training method to train the networks. The data generator function loads batches of sample problems (having the same sequence length from the training dataset) that are used by the *fit_generator* at each training iteration. In each training epoch, the data generator function loads all the samples with different

sequence sizes (batch by batch) to be used by *fit_generator*. Also, the order of sequence sizes of the data batches was set to be shuffled at the beginning of each epoch.

We trained the networks using a GPU platform. We used NVIDIA Tesla V100 GPU, which has 5120 CUDA cores and 32 GB HBM2 memory, to train the FNEs. The GPU was installed on a HPC cluster node that had a Xeon Gold 5218R CPU with 40 cores at 2.10 GHz base frequency and 192 GB of memory.

Note that, owing to the stochastic initialization of the network parameters and of the training algorithms as well as to the different training datasets, the values of the trainable parameters (i.e. the weights) of the different $B_{\#}$ networks are different, hence the network models are independent. This aspect has important implications in increasing both accuracy and reliability of the predicted results (Section 3.3).

Fig. A.1 shows the trend of the training and validation loss functions versus the training epoch for each of the three networks. It is seen that the FNEs training and validation losses converge to values very close to each other. Hence, the models are not overfitted on their training datasets. Recall that during the training process, the learning rate scheduler was set to divide the rate by a factor of two every 250 epochs. The changes in the learning rate explain the sudden drops visible in the loss function values.

Before we proceed with the plate FNEs training results, we clarify our rationale for determining the total training epoch and learning rate division step. The training hyperparameters were mainly determined through a trial and error process. However, for the specific task of determining the total number of training epochs and the learning rate division step, we mainly based our decision on the trend of the loss function during training. Specifically, while we monitored the loss function trend to determine the point at which the loss did not further decrease, we also monitored the difference between the training and test loss functions to ensure that the network was not overfitted or underfitted on the training dataset.

We determined the epoch number at which the learning rate was revised for the first time (epoch 250) based on the trend of the training loss. More specifically, as it is seen in Fig. A.1, the loss trend turns into a flat line (which means no further improvement in the accuracy of the network) before the first division. Hence, we selected epoch 250 for the first learning rate division. After each learning rate revision, we trained the networks for the same number of epochs, e.g. 250 epochs per each learning rate value for the beam networks.

To determine the total training epochs, we also considered the gap between the loss values each time the learning rates were revised. The beam network loss (Fig. A.1a) drops about an order of magnitude at the first two revisions (epoch 250 and 500), but it does not decrease appreciably after the third revision (epoch 750). Hence, we did not revise the learning rate further and did not continue the training after epoch 1000. It is worth mentioning that we also tried higher learning rate division factors. Higher factors (e.g., using a division factor of 10) decelerated the training loss reduction rate and resulted in higher loss values which remained flat after the first learning rate revision.

Certainly, the training hyperparameters have not been optimized and there could be potentially other combinations of parameters that would result in (trained) FNEs with similar accuracy on the same test datasets. We highlight that having achieved good accuracy without any particular effort on optimizing the set of hyperparameters is a very positive aspect because it suggests intrinsic robustness of the network elements formulation.

Appendix B. Plate FNE training

A training strategy similar to that described in the previous section for the beam was used also for the plate FNE. In order to train the P_1, P_2, and P_3 networks, one training dataset per each FNE was generated by using the in-house finite element code. The four-noded conforming rectangular element presented in [40] was used for this task. In the following, we describe the dataset generation and the details of the network training.

B.1. Training dataset

The parameters used for the data generation are summarized in Table B.1. The plate occupies the domain $0 \leqslant x \leqslant L$ and $0 \leqslant y \leqslant W$ and has free boundary conditions at $y = 0$ and $y = W$. The boundaries $x = 0$ and $x = L$ were connected to stiffness elements K_z, K_{θ_x}, and K_{θ_y} that were acting on the nodal displacements w, θ_x, and θ_y, respectively. Each plate was uniformly discretized, resulting in a grid of 11×11 nodes. We generated a total of 10^5 samples for each dataset. We also generated 15×10^3 validation samples per each network element. Similar to the beam problem, we used LHS sampling method to guarantee that all parts of the applied load space were properly sampled.

B.2. Network training

As for the beam FNEs, we built the plate networks P_{1-3} using Python, Keras, and Tensorflow packages. The plate FNE architecture presented in Section 3.1.2 has a total of 80,994 trainable parameters (weights and biases). Note that the difference in the size of the input and output layers of the plate and beam FNEs causes the difference in the total number of network trainable parameters in the two networks.

We trained the plate FNEs using ADAM optimization algorithm and the training loss function described in Eq. (A.3). The *Xavier uniform initializer* [52] was used to initialize the weights and the

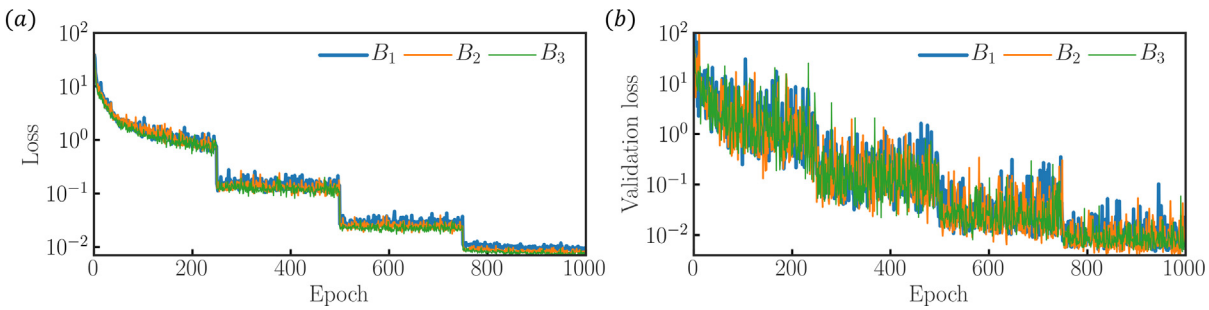

Fig. A.1. Trend of the loss function during the training phase of the beam FNEs: (a) training loss, and (b) validation loss. Results are reported for the three networks B_1, B_2, and B_3.

Table B.1
Parameters used to generate the training dataset for the plate FNE.

parameter	value	parameter	value
$E[MPa]$	30	$K_{\theta_x}[N.m/rad]$	17.17
v	0.3	$K_{\theta_y}[N.m/rad]$	17.17
$L[m]$	1.0	$F_z[N]$	$[-1, 1]$
$W[m]$	1.0	$M_x[N.m]$	$[-0.1, 0.1]$
$t[m]$	0.005	$M_y[N.m]$	$[-0.1, 0.1]$
$K_z[N/m]$	15×10^6	q	121

biases were initialized to zero. We used the same GPU platform used for the beam networks (see Section A.2) to train the plate FNEs. Each network was trained for 2000 epochs with a batch size of 64 and an initial learning rate of .0001, which was divided by a factor of two every 800 epochs by the learning rate scheduler. Again, note that the training parameters were determined via a trial and error procedure, with particular attention placed on the overall prediction performance of the network as well as the trend of the training and validation losses.

Fig. B.1 shows the trend of the training and validation loss functions during training for the three networks P_{1-3}. Results show that all models converge approximately to the same loss value, hence they are expected to have a similar level of accuracy. The train and validation losses converged to very close values, indicating that the network models are not overfitted on their training datasets.

Additionally, as it is seen in B.1, around epoch 750, the loss trends for the plate FNE gradually become flat. Hence, we selected epoch 800 as the plate FNEs learning rate division step. Note that the sudden drops in the losses values are due to the change in the learning rate. Further, there is a minimal decrease in the loss value of the plate networks after the second learning rate revision. Also, the loss trend remains flat in the last 400 epochs. Hence, we did not revise the learning rate further and stopped training the plate networks at epoch 2000.

B.3. DNN-based plate network elements

In order to further illustrate the simulation performance of BRNN-based network elements in FENA compared to other deep-learning-based methodologies, we developed a plate network element based on a deep fully connected feedforward neural network (DNN). We selected DNN since it is the most commonly used network architecture in studies focused on deep learning techniques applied to the simulation of physical systems. However, the flexibility of DNN-based solution methods is significantly limited due to the intrinsic properties of the DNN architecture. The most important limitation is that DNN input and output sizes scale with the size of the physical domain and of the inputs (see [1] for a detailed discussion on why BRNN is a considerably better choice to develop deep-learning-based simulation methodologies). More specifically, DNN has a fixed input and output size and all system inputs must be provided to the DNN simultaneously. Hence, the input size of the plate network element based on DNN is $[3 \times q]$ (q is the number of nodes), that is the applied external nodal load vector $[F_{z_i}, M_{xx_i}, M_{yy_i}], i = 1, 2, \ldots, q$. The DNN output also scales with the number of nodes and has the size of $[4 \times q]$ (four nodal DOF times the total number of nodes). Note that in BRNN, the input and output sizes do not scale with the domain size as we sequentially feed in the applied external load vector to the network element.

The DNN that was tested included 16 hidden layers with 50 neurons in each layer. This architecture was selected to have almost the same number of trainable parameters in both BRNN-based (80994 trainable parameters) and DNN-based plate network elements (81134 trainable parameters). We used E-swish activation function for the hidden layers and linear activation function for the output layer of the DNN. We trained three DNN-based plate network elements and used the (BRNN-based) plate FNEs training and test datasets to train them. We also followed the same training scheme and hyperparameters discussed in Appendix Section B.2.

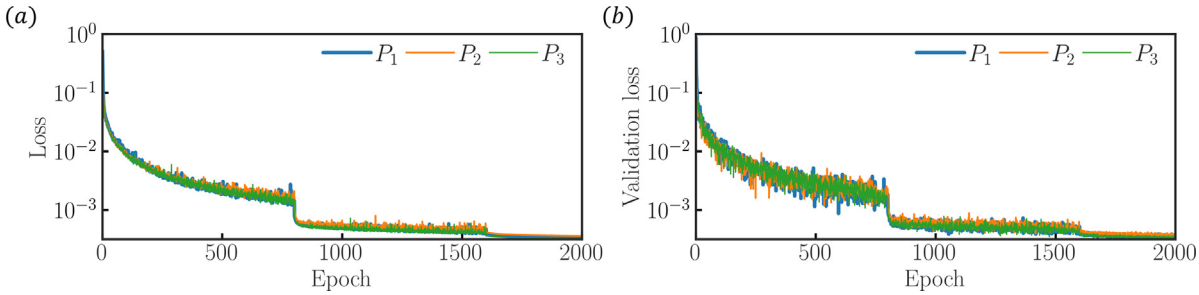

Fig. B.1. Loss function versus epoch number for the plate FNEs: (a) training loss, and (b) validation loss. Results are reported for the three network models P_1, P_2, and P_3. Each plate FNE was trained for 2000 epochs.

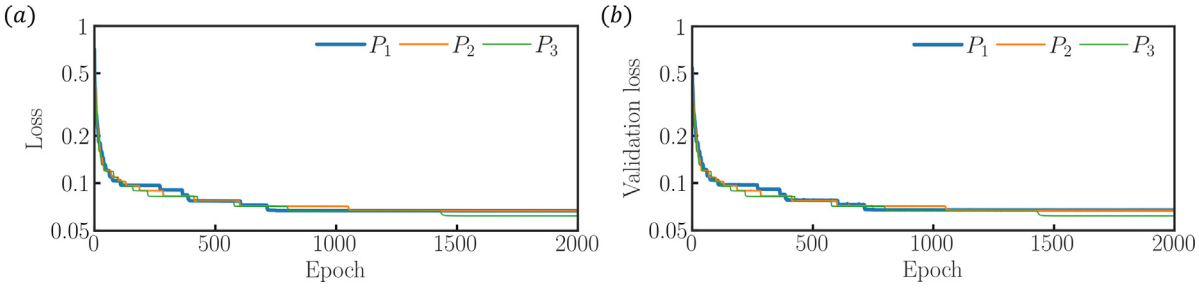

Fig. B.2. Loss function versus epoch number for the DNN-based plate network element: (a) training loss, and (b) validation loss. Results are reported for the three DNN-based network models P_1, P_2, and P_3.

Fig. B.2 shows the trend of training and validation loss of the three DNN-based network elements. It is seen that during training the loss values decreased only about an order of magnitude (compare to the results of Fig. B.1 wherein the loss value decreased by more than two orders of magnitude). Comparing the test loss results of Fig. B.2b and B.1b shows that DNN-based plate network elements have higher test loss values compared to the BRNN-based plate elements. Hence, the developed (BRNN-based) FENA plate network elements have higher simulation accuracy than the DNN-based networks. This can be clearly seen in the simulation results of sample problem *Plate 1* simulated with the DNN-based network elements, reported in Fig. B.3. It is seen that the maximum error values of all the nodal DOF increased at least 500% (compare with the results shown in Fig. 7 where the maximum relative error is 1.2%).

Further, from a computational standpoint, the average DNN-based networks simulation time for *Plate 1* is 0.031[*s*], which is almost the same as the simulation time of BRNN based network elements. Hence, the DNNs do not offer any superiority from a computational point of view while they fail to provide a level of accuracy comparable with the BRNN-based network elements. Note that, for both DNN and BRNN, we did not perform any architecture optimization. In principle, it might be possible to obtain other network architectures with better prediction accuracy for both types of network architectures, but the main characteristics observed in this example will likely remain unaltered. Even considering the case where a DNN-based architecture could deliver similar accuracy to the BRNN-based network elements, the internal architecture of DNNs remains a significantly limiting factor that cannot be overcome.

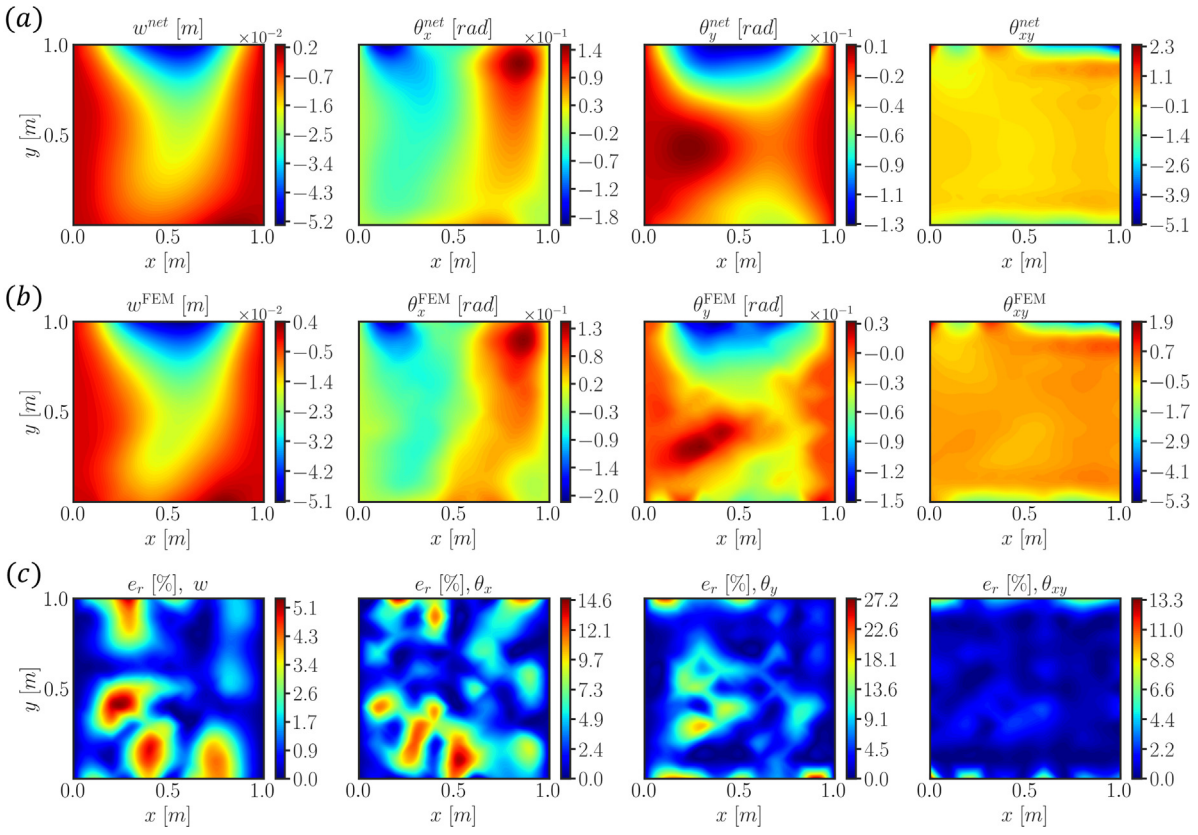

Fig. B.3. Static response of *Plate 1* subject to a spatially distributed load with random amplitude simulated with the DNN-based plate network elements. (a) Results for *Plate 1*. The predicted fields w, θ_x, θ_y, and θ_{xy} obtained by averaging predictions from the three DNN-based network elements. (b) Reference solution of each nodal degree of freedom calculated via FE approach. (c) Distribution of the percentage relative error of the nodal degrees of freedom. The errors are calculated with respect to the FE results.

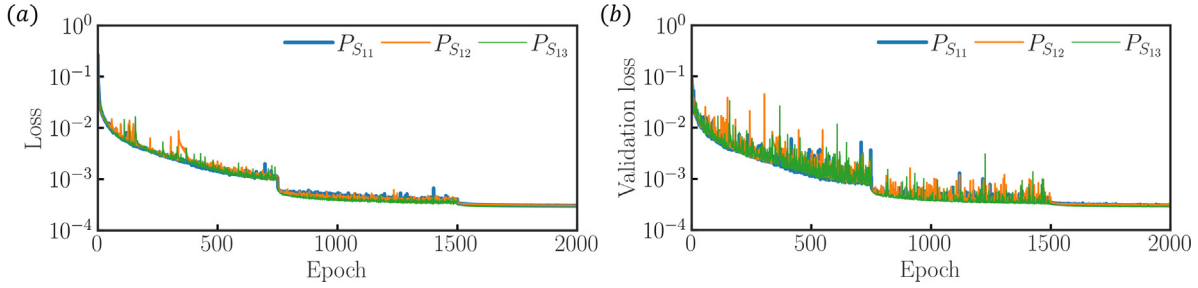

Fig. B.4. Trend of the loss function during the training phase of the first group of plate stress–strain networks. (a) training loss, and (b) validation loss. This group of networks maps the applied external loads to the stress and strain fields. Results are reported for the three networks $P_{S_{11}}, P_{S_{12}}$, and $P_{S_{13}}$.

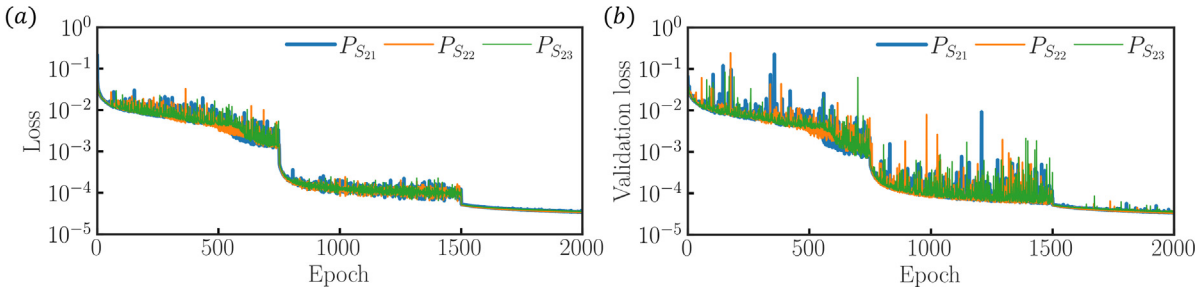

Fig. B.5. Loss function trend during the training of the second group of plate stress–strain networks: (a) training loss, and (b) validation loss. Results are reported for the three networks $P_{S_{21}}, P_{S_{22}}$, and $P_{S_{23}}$. This group of networks maps the deformation fields predicted by the plate FNEs (P_{1-3}) to the stress and strain fields.

B.4. Stress and Strain Fields Predictions by Networks

As we discussed in Section 4.2.1, in order to assess the prediction performance and demonstrate the feasibility of the network-based approach, we developed two groups of networks dedicated to stress and stress fields calculations. The first group directly maps the external loads applied onto the structure to stress and strain fields. The second group receives the displacement field predicted by the plate FNEs and maps it to the stress and strain fields.

We used the same network architecture employed for the plate FNEs (see Section 3.1.2). To build the training dataset, we calculated the stress and strain fields for the plate training samples via the FEA elements shape functions. Note that we decided to use FEA because it was readily available from the previous calculations but, as extensively discussed above, any other available source of data could be used to train stress–strain networks. Specifically, stress–strain solutions based on either experimental data or analytical solutions could be used to train the networks (or to fine-tune pre-trained stress–strain networks parameters), whenever available. The major takeaway from the above results is that the stress–strain networks trained on FEA data can return solutions very close to the FEA, hence they do not contribute additional errors to the stress and strain calculations. This latter comment is in contrast with the stress–strain calculations based on numerical derivatives of the displacement field. The training loss function was defined based on the MSE of the predictions. We used the same training scheme and hyperparameters to perform the training of both groups of networks (Appendix Section B). Three networks were trained for each group. The networks in the first group are labeled $P_{S_{11}}, P_{S_{12}}$, and $P_{S_{13}}$ while those in the second group are labeled $P_{S_{21}}, P_{S_{22}}$, and $P_{S_{23}}$. The average of the predictions of the three networks will be used to define the stress or strain fields for a given problem.

Figs. B.4 and B.5 show the trend of both the training and validation losses for the two stress–strain networks groups, respectively. All the three networks in each group converged approximately to the same training and validation loss value, hence confirming the absence of any overfitting in their training datasets. Note that the sudden drops in the loss function value are due to the change in the learning rate (see Appendix Section A.2 and Section B.2).

We highlight that the second group of networks is trained to learn a relatively less complex mapping (compared to the first group), as they have access to the physical solution of the system (i.e. the displacement field resulting from the applied external load). On the other hand, the first group has to learn a relatively more complex mapping, that is the direct mapping from external loads to stress and strain fields. Hence, the second group is expected to achieve a higher prediction accuracy, which is confirmed by comparing the converged loss values of the two groups (recall that the output channels and the loss functions of both groups are the same). Specifically, the converged loss values of the networks in the second group are approximately one order of magnitude smaller than the first group of networks, which shows higher accuracy of the second group (compare Figs. B.4 and B.5). The maximum prediction error of the first group is within the range $[2\%, 5\%]$ while the second group error is typically below 1%. We do not present the prediction results of the first group due to their relatively lower accuracy. Nevertheless, it was pointed out in Section 4.2.1 that the second group still provides the most logical approach because it maps the calculated solution (in terms of displacements) to stress and strain fields; hence treating them as derived quantities similarly to classical FEA.

References

[1] Jokar Mehdi, Semperlotti Fabio. Finite element network analysis: a machine learning based computational framework for the simulation of physical systems. Comput Struct 2021;247:106484.
[2] Takeuchi Jun, Kosugi Yukio. Neural network representation of finite element method. Neural Networks 1994;7(2):389–95.
[3] Noakoasteen O, Wang S, Peng Z, Christodoulou C. Physics-informed deep neural networks for transient electromagnetic analysis. IEEE Open J Antennas Propag 2020;1:404–12.
[4] Baiges Joan, Codina Ramon, Castañar Inocencio, Castillo Ernesto. A finite element reduced-order model based on adaptive mesh refinement and artificial neural networks. Int J Numer Meth Eng 2020;121(4):588–601.
[5] Parish Eric J, Duraisamy Karthik. A paradigm for data-driven predictive modeling using field inversion and machine learning. J Comput Phys 2016;305:758–74.
[6] Yang Kai, Xu Xinyi, Yang Benjamin, Cook Brian, Ramos Herbert, Anoop Krishnan NM, et al. Predicting the young's modulus of silicate glasses using high-throughput molecular dynamics simulations and machine learning. Scient Rep 2019;9(1):8739.
[7] Schubert Philipp, Dorkenwald Sven, Januszewski Michał, Jain Viren, Kornfeld Joergen. Learning cellular morphology with neural networks. Nat Commun 2019;10(1):2736.
[8] Geist Moritz, Petersen Philipp, Raslan Mones, Schneider Reinhold, Kutyniok Gitta. Numerical solution of the parametric diffusion equation by deep neural networks. J Sci Comput 2021;88(1):1–37.
[9] Brunton Steven L, Noack Bernd R, Koumoutsakos Petros. Machine learning for fluid mechanics. Annu Rev Fluid Mech 2020;52:477–508.
[10] Kharazmi Ehsan, Zhang Zhongqiang, Karniadakis George Em. hp-vpinns: Variational physics-informed neural networks with domain decomposition. Comput Methods Appl Mech Eng 2021;374:113547.
[11] Hornik Kurt, Stinchcombe Maxwell, White Halbert, et al. Multilayer feedforward networks are universal approximators. Neural Networks 1989;2(5):359–66.
[12] Gopalakrishnan Srinivasan, Chakraborty Abir, Mahapatra Debiprosad Roy. Spectral finite element method: wave propagation, diagnostics and control in anisotropic and inhomogeneous structures. Springer Science & Business Media; 2007.
[13] Aliabadi Mohammad H. The boundary element method, volume 2: applications in solids and structures, volume 2. John Wiley & Sons; 2002.
[14] Babuska Ivo, Whiteman John, Strouboulis Theofanis. Finite elements: an introduction to the method and error estimation. Oxford University Press; 2010.
[15] Bathe Klaus-Jürgen. Finite element procedures. 2 edition. Watertown, Massachusetts: Klaus-Jurgen Bathe; 2014.
[16] Hughes Thomas JR. The finite element method linear static and dynamic finite element analysis. Newburyport: Dover Publications; 2012.

[17] Reddy Junuthula N. An Introduction to Nonlinear Finite Element Analysis: with applications to heat transfer, fluid mechanics, and solid mechanics. Oxford University Press; 2014.
[18] Zienkiewicz Olek C, Taylor Robert L, Zhu Jian Z. The finite element method its basis and fundamentals. Elsevier Butterworth-Heinemann; 2013.
[19] Ihlenburg Frank, Babuska Ivo. Finite element solution of the helmholtz equation with high wave number part ii: the hp version of the fem. SIAM J Numer Anal 1997;34(1):315–58.
[20] Yagawa G, Yoshioka A, Yoshimura S, Soneda N. A parallel finite element method with a supercomputer network. Comput Struct 1993;47(3):407–18.
[21] Ramuhalli Pradeep, Udpa Lalita, Udpa Satish S. Finite-element neural networks for solving differential equations. IEEE Trans Neural Networks 2005;16 (6):1381–92.
[22] Lagaris Isaac Elias, Likas Aristidis C, Papageorgiou Dimitrios G. Neural-network methods for boundary value problems with irregular boundaries. IEEE Trans Neural Networks 2000;11(5):1041–9.
[23] Chao Xu, Wang Changlong, Ji Fengzhu, Yuan Xichao. Finite-element neural network-based solving 3-D differential equations in mfl. IEEE Trans Magn 2012;48(12):4747–56.
[24] Brevis Ignacio, Muga Ignacio, van der Zee Kristoffer G. A machine-learning minimal-residual (ml-mres) framework for goal-oriented finite element discretizations. Comput Math Appl 2021;95:186–99.
[25] Reichstein Markus, Camps-Valls Gustau, Stevens Bjorn, Jung Martin, Denzler Joachim, Carvalhais Nuno, et al. Deep learning and process understanding for data-driven earth system science. Nature 2019;566(7743):195–204.
[26] Patnaik Sansit, Jokar Mehdi, Semperlotti Fabio. Variable-order approach to nonlocal elasticity: theoretical formulation, order identification via deep learning, and applications. Comput Mech 2021:1–32.
[27] Kailiang Wu, Xiu Dongbin. Data-driven deep learning of partial differential equations in modal space. J Comput Phys 2020;408:109307.
[28] Kim Kyungdoc, Kang Seokho, Yoo Jiho, Kwon Youngchun, Nam Youngmin, Lee Dongseon, et al. Deep-learning-based inverse design model for intelligent discovery of organic molecules. npj Comput Mater 2018;4(1): 1–7.
[29] Raissi Maziar, Perdikaris Paris, Karniadakis George E. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 2019;378:686–707.
[30] Karumuri Sharmila, Tripathy Rohit, Bilionis Ilias, Panchal Jitesh. Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks. J Comput Phys 2020;404:109120.
[31] Haghighat Ehsan, Raissi Maziar, Moure Adrian, Gomez Hector, Juanes Ruben. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. Comput Methods Appl Mech Eng 2021;379:113741.
[32] Liu Zeliang. Deep material network with cohesive layers: Multi-stage training and interfacial failure analysis. Comput Methods Appl Mech Eng 2020;363:112913.
[33] Willard Jared, Jia Xiaowei, Xu Shaoming, Steinbach Michael, Kumar Vipin. Integrating physics-based modeling with machine learning: A survey. arXiv preprint arXiv:2003.04919; 2020.
[34] Rai Rahul, Sahu Chandan K. Driven by data or derived through physics? a review of hybrid physics guided machine learning techniques with cyber-physical system (cps) focus. IEEE Access 2020;8:71050–73.
[35] Goodfellow Ian, Bengio Yoshua, Courville Aaron. Deep Learning. MIT Press; 2016. http://www.deeplearningbook.org.
[36] Kharazmi Ehsan, Zhang Zhongqiang, Karniadakis George Em. Variational physics-informed neural networks for solving partial differential equations. arXiv preprint arXiv:1912.00873; 2019.
[37] Wang Sifan, Yu Xinling, Perdikaris Paris. When and why pinns fail to train: A neural tangent kernel perspective. arXiv preprint arXiv:2007.14527; 2020.
[38] Wang Hengjie, Planas Robert, Chandramowlishwaran Aparna, Bostanabad Ramin. Train once and use forever: Solving boundary value problems in unseen domains with pre-trained deep learning models. arXiv preprint arXiv:2104.10873; 2021.
[39] Schuster Mike, Paliwal Kuldip K.. Bidirectional recurrent neural networks. IEEE Trans Signal Process 1997;45(11):2673–81.
[40] Narasimha Reddy Junuthula. Theory and analysis of elastic plates and shells. CRC Press; 2006.
[41] Narasimha Reddy Junuthula. An Introduction to the Finite Element Method. McGraw-Hill Education; 2005.
[42] Alcaide Eric. E-swish: Adjusting activations to different network depths. arXiv preprint arXiv:1801.07145; 2018.
[43] Gers Felix A., Schmidhuber Jürgen, Cummins Fred. Learning to forget: Continual prediction with lstm. In: IET Conference Proceedings 1999;5:850–5.
[44] Krogh Anders, Vedelsby Jesper. Neural network ensembles, cross validation, and active learning. In: Advances in neural information processing systems; 1995. p. 231–8.
[45] Tan Chuanqi, Sun Fuchun, Kong Tao, Zhang Wenchang, Yang Chao, Liu Chunfang. A survey on deep transfer learning. In: International conference on artificial neural networks. Springer; 2018. p. 270–9.
[46] Rih-Teng Wu, Jokar Mehdi, Jahanshahi Mohammad R, Semperlotti Fabio. A physics-constrained deep learning based approach for acoustic inverse scattering problems. Mech Syst Signal Process 2022;164:108190.
[47] Shah Shital, Dey Debadeepta, Lovett Chris, Kapoor Ashish. Airsim: High-fidelity visual and physical simulation for autonomous vehicles. In: Field and service robotics. Springer; 2018. p. 621–35.
[48] Tobin Josh, Fong Rachel, Ray Alex, Schneider Jonas, Zaremba Wojciech, Abbeel Pieter. Domain randomization for transferring deep neural networks from simulation to the real world. In: 2017 IEEE/RSJ international conference on intelligent robots and systems (IROS). IEEE, 2017. p. 23–3.0.
[49] McKay Michael D, Beckman Richard J, Conover William J. A comparison of three methods for selecting values of input variables in the analysis of output from a computer code. Technometrics 2000;42(1):55–61.
[50] Recurrent layer. https://faroit.com/keras-docs/1.1.1/layers/recurrent/#lstm. Accessed: 2021-08-18.
[51] Dense layer. https://faroit.com/keras-docs/1.1.1/layers/core/#dense. Accessed: 2021-08-18.
[52] Glorot Xavier, Bengio Yoshua. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the thirteenth international conference on artificial intelligence and statistics. JMLR Workshop and Conference Proceedings; 2010. p. 249–6.
[53] Kingma Diederik P, Ba Jimmy. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980; 2014.