Data-driven synchronization-avoiding algorithms in the explicit distributed structural analysis of soft tissue

Guoxiang Grayson Tong · Daniele E. Schiavazzi

Received: 5 July 2022 / Accepted: 1 November 2022 / Published online: 15 November 2022
© The Author(s), under exclusive licence to Springer-Verlag GmbH Germany, part of Springer Nature 2022

Abstract
We propose a data-driven framework to increase the computational efficiency of the explicit finite element method in the structural analysis of soft tissue. An encoder–decoder long short-term memory deep neural network is trained based on the data produced by an explicit, distributed finite element solver. We leverage this network to predict synchronized displacements at shared nodes, minimizing the amount of communication between processors. We perform extensive numerical experiments to quantify the accuracy and stability of the proposed synchronization-avoiding algorithm.

Keywords Data-driven modeling · Distributed finite element method · LSTM neural network · Structural analysis of soft tissue · Cardiovascular simulation

1 Introduction
A wide range of convergent numerical approaches with rigorously derived error bounds are available from numerical analysis for time integration of ordinary and partial differential equation models. These methods, combined with the increasing availability of high performance computational resources have significantly contributed to the remarkable realism achievable by modern high-fidelity numerical models in many fields.

This paper focuses on distributed explicit time integrators, where time updates are computed through matrix–vector products and are therefore highly scalable and amenable to efficient GPU implementation. Highly scalable GPU solvers for physics-based modelling are already available in the literature [3,20,29,34,53] with GPU-based accelerated explicit finite element structural simulations of soft tissues discussed, for example, in [21,34,53,54]. Unlike implicit time integration, explicit schemes typically do not need element-level quantities to be assembled in a global matrix, leading to memory and runtime savings. However, explicit schemes are only conditionally stable [3,4,11,19] with time step size a few order of magnitude smaller compared to their implicit counterpart. This difference becomes less pronounced for the structural analysis of biological soft tissue where explicit approaches have the potential to be competitive with respect to implicit time integration, for example in the context of cardiovascular modeling.

In explicit schemes, the main costs per time step relate to the computation of element-level quantities (mass, stiffness matrix and load vector) and the cost of communication. While the first can be mitigated by reduced numerical integration [45,57], closed-form representations [37,51] or and more recent data-driven methods [22,23], the second remains a fundamental bottleneck, despite several optimized approaches proposed in the context of GPU-based distributed computation [1,21,31,54]. Since synchronization must be performed at every time step, this problem is also exacerbated, in explicit solvers, by the small size of the stable time steps.

To alleviate the cost of synchronization, recently developed data-driven approaches offer a possible solution. The expressive power of artificial neural networks has been widely demonstrated in the construction of surrogate models for dynamical systems, producing fast emulators that can be integrated in optimization and UQ design loops. In this context, extensive recent work include the use of residual networks (ResNet [16]) for data-driven generalization of explicit Euler time integrators [9,10,12,42,56], showing promising results for both linear and nonlinear dynamical systems. Other methods are based, for example, on physics-
informed neural networks [43], deep operator networks [35] and convolutional networks assembled from encoders and decoders [40]. Others incorporate spectral properties of system dynamics in the design of data-driven models, to realize linearization and handle high-dimensionality [25,33] or use sparse regression to construct parsimonious surrogates with model complexity from an a-priori selected dictionary (sparse identification of nonlinear dynamics or SINDy [6,24,49]). Note that all the approaches above aim to create effective data-driven surrogate models of dynamical systems, rather than leveraging new advances in data-driven architectures to further improve the efficiency of numerical schemes.

In this paper, we combine numerical simulation and data-driven approaches to mitigate the synchronization bottleneck in explicit distributed time integration. We equip each processor with an independent network which models synchronized displacement solutions for the shared nodes of the respective partition, in order to reduce synchronization frequency and to increase the degree of parallelism. Using the proposed approach, substantial savings are obtained for the cost of communication without compromising accuracy and long-term stability. In addition, multiple networks are employed to predict the displacements for the same shared nodes, providing a means by which to assess prediction robustness and to bound approximation error. Our data-driven framework (built based on the PyTorch library [41] and publicly available as a GitHub repository at https://github.com/desResLab/Synchronization-avoiding-algorithms) uses recurrent neural networks (RNN) due to their ability to handle time series data [5,7,13,50]. However, since vanilla RNNs are unable to effectively learn long-term dependence in the data, we employ long short term memory (LSTM) encode-decoder networks [17,36], that have received significant previous attention in the context of dynamical systems [15,18,36,39,55].

This model successfully fits our purpose of approximating dynamical systems only on a subset (shared nodes) of the entire system. Learning the partial rather than the full dynamics by a non-recurrent neural network may require, for example, the construction of a memory kernel of the Mori-Zwanzig type [10,12]. This is, however, not required for the proposed LSTM recurrent network which inherently holds a temporal memory due to its sequential input [5,50]. Additional work on learning the dynamics of a subset of degrees of freedom can be found in [2].

This paper is organized as follows. A brief review of the governing equations for linear elastodynamics and their spatial discretization into finite elements is reported in Sect. 2.1, followed by the formulation for our distributed explicit solver in Sect. 2.2. The proposed data-driven method is discussed by first presenting a single LSTM cell in Sect. 3.1, followed by an encoder–decoder layout in Sect. 3.2, with the details of network training summarized in Sect. 3.2.1. A data-driven synchronization-avoiding algorithm is proposed in Sect. 4.

Numerical tests are performed in Sect. 5, starting with extensive benchmarks on a simple cantilever problem in Sect. 5.1, and a realistic coronary model in Sect. 5.2. In Sect. 6, we discuss a few criteria to measure the prediction error of our proposed framework. Performance is assessed in Sect. 7 for a fixed number of cores and fixed mesh size, respectively, while Sect. 8 contains the conclusions and addresses possible future research directions.

2 Governing equations and discretization

2.1 Equations of linear elastodynamics

2.1.1 Strong form

Consider the following initial-boundary value problem (S) defined over the domain \( \Omega \in \mathbb{R}^3 \) with Lipschitz boundary \( \partial \Omega = \partial \Omega_d \cup \partial \Omega_n \) and \( \bar{\Omega} = \Omega \cup \partial \Omega \) (see, e.g., [19])

\[
\begin{align*}
\text{Given } f_i, g_i, h_i, d_{0i}, \bar{d}_{0i}, & \quad \text{find } d_i, \ i, j = \{x, y, z\}, \ \text{s.t.} \\
\rho \ddot{d}_i &= \frac{\partial \sigma_{ij}}{\partial x_j} + f_i & \text{in } \Omega \times (0, T) \\
d_i &= g_i & \text{on } \partial \Omega_d \times (0, T) \\
\sigma_{ij} n_j &= h_i & \text{on } \partial \Omega_n \times (0, T) \\
d_i &= d_{0i} & \text{in } \Omega, \ \text{at } t = 0 \\
\ddot{d}_i &= \bar{d}_{0i} & \text{in } \Omega, \ \text{at } t = 0,
\end{align*}
\]

where the quantity of interest \( d_i(x, t) : \bar{\Omega} \times [0, T] \rightarrow \mathbb{R} \) denotes the \( i \)-th component of the displacement field \( d \), \( \rho \) is the material density, \( f_i(x, t) : \Omega \times [0, T] \rightarrow \mathbb{R} \) is the \( i \)-th component of the body force, \( g_i(x, t) : \partial \Omega_d \times [0, T] \rightarrow \mathbb{R} \), \( h_i(x, t) : \partial \Omega_n \times [0, T] \rightarrow \mathbb{R} \) are the \( i \)-th components of the prescribed Dirichlet and Neumann boundary conditions on \( \partial \Omega_d \) and \( \partial \Omega_n \), respectively. In addition, initial values \( d_{0i}, \ddot{d}_{0i} \) are set for the displacement and velocity component.

We also assume a linear, elastic and isotropic constitutive model in the small strain regime of the form

\[
\sigma_{ij} = 2\mu \epsilon_{ij} + \lambda \delta_{ij} \epsilon_{kk}, \ i, j = \{x, y, z\},
\]

where \( \sigma = [\sigma_{ij}] \) is the Cauchy stress tensor, \( \epsilon = [\epsilon_{ij}] \) is the infinitesimal strain tensor, \( \delta_{ij} \) is the Kronecker delta, and the Lamé coefficients \( \mu \) and \( \lambda \) are defined as

\[
\mu = \frac{E}{2(1 + \nu)}; \quad \lambda = \frac{E \nu}{(1 + \nu)(1 - 2\nu)},
\]

where \( E \) is the material Young’s modulus, \( \nu \) is the Poisson ratio.
The infinitesimal strain $\epsilon$ is obtained from the symmetric part of the displacement gradient tensor

$$
\epsilon_{ij} = d_{i,j} = \frac{1}{2} \left( \frac{\partial d_i}{\partial x_j} + \frac{\partial d_j}{\partial x_i} \right),
$$

leading to an expression of the Cauchy stress in (2) in terms of displacements as

$$
\sigma_{ij} = c_{ijkl} \epsilon_{kl} = c_{ijkl} d_{(k,l)},
$$

where $C = [c_{ijkl}]$ is a fourth-order elasticity tensor, defined as (see, e.g., [46])

$$
c_{ijkl} = \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) + \lambda\delta_{ij}\delta_{kl}.
$$

2.1.2 Weak form

A weak or variational formulation for problem (1) can be written as

$$
(\mathcal{W}) : \begin{cases}
\text{Given } f, g, h, d_0, \dot{d}_0, \text{ find } d(x, t) \in \mathcal{D}^l, \\
\text{that for any } w \in \mathcal{W}, \text{ s.t.}
\end{cases}
$$

$$
(\rho \ddot{d}, w)_{\Omega} + (a(d, w)_{\Omega} = (l(w)_{\Omega} + l_n(w)_{\partial \Omega_n})
$$

where $(\cdot, \cdot)_{\Omega}$ denotes the standard product in $L^2(\Omega)$, and the linear and bilinear forms $l(\cdot)_{\Omega}, l_n(\cdot)_{\partial \Omega_n}$ and $a(\cdot)_{\Omega}$ are defined, respectively, as

$$
a(d, w)_{\Omega} = \int_{\Omega} w_{(i,j)} c_{ijkl} d_{(k,l)} d\Omega,
$$

$$
l(w)_{\Omega} = \int_{\Omega} w_i f_i d\Omega,
$$

$$
l_n(w)_{\partial \Omega_n} = \int_{\partial \Omega_n} w_i h_i dA.
$$

The weak form (7) relaxes the regularity requirement of displacement solutions of the strong form (1) such that the trial and test spaces only need to satisfy the conditions

$$
\mathcal{D}^l := \left\{ d(x, t) | d(x, t) \in H^1(\Omega); d_i(x, t) = g_i(x, t), \forall x \in \partial \Omega_d; t \in (0, T) \right\}
$$

$$
\mathcal{W} := \left\{ w(x) | w_i(x) \in H^1(\Omega); w_i(x) = 0, \forall x \in \partial \Omega_d \right\},
$$

where $H^1(\Omega)$ is the standard Sobolev space of order 1.

2.1.3 Discrete matrix form

The solution of (7) by a Bubnov-Galerkin finite element approach requires the selection of appropriate discrete subspaces $\mathcal{D}_h^{(n)} \subset \mathcal{D}^l$ and $\mathcal{W}_h \subset \mathcal{W}$, such that the projected solution $d_h$ converges to the true solution $d$ in $L^2(\Omega)$ with respect to any $w_h \in \mathcal{W}_h$. We consider the discrete subspace $\mathcal{D}_h^{(n)}$ spanned by the set of linear Lagrange polynomials $\mathcal{P}^1$, leading to the following semi-discrete matrix formulation

$$
(\mathcal{M}) : \begin{cases}
\text{Find } d_h^{(n)} \in \mathcal{D}_h^{(n)}, \ n \in \{1, 2, \ldots, n_T\}, \text{s.t.}
\end{cases}
$$

$$
\mathcal{M} d_h^{(n)} + C_d h^{(n)} + K d_h^{(n)} = f_h^{(n)}
$$

$$
\mathcal{D}_h^{(n)} := \left[ \begin{array}{c}
d_h^{(n)}(x)| d_h^{(n)}(x) \in C^0(\Omega_h); \\
d_h^{(n)}(x)| d_h^{(n)}(x) \in C^0(\Omega_h); \\
d_h^{(n)}(x)| d_h^{(n)}(x) \in P^1(\Omega_h,e); n \in \{1, 2, \ldots, n_T\} \end{array} \right]
$$

$$
\mathcal{W}_h := \left\{ w_h(x) | w_i(x) \in C^0(\Omega_h); w_i(x) = 0, \forall x \in \partial \Omega_h; w_h(x)| e \in P^1(\Omega_h,e) \right\},
$$

where the subscripts $(\cdot)_e, (\cdot)_0$ indicate restriction to a single finite element and $n_T$ is the total number of time steps. The quantities $\mathcal{M}, C, K, f$ denote the mass, damping, stiffness matrices and loading vector, respectively with element-level expressions that are standard in the isoparametric finite element literature (see, e.g., [19]) and are therefore omitted. In this paper, we consider mass-proportional damping, i.e., $C = \alpha \mathcal{M}$ with damping factor $\alpha \in \mathbb{R}$ (see, e.g., [19]).

2.2 A distributed explicit structural finite element solver

2.2.1 Explicit time integration

The algebraic system (10) is discretized in time using a second-order central difference stencil, where structural accelerations and velocities are approximated at every time point $n$ as

$$
a_h^{(n)} = \ddot{d}_h^{(n)} = \frac{d_h^{(n+1)} - 2d_h^{(n)} + d_h^{(n-1)}}{\Delta t^2} + O(\Delta t^3),
$$

$$
v_h^{(n)} = \dot{d}_h^{(n)} = \frac{d_h^{(n+1)} - d_h^{(n-1)}}{2\Delta t} + O(\Delta t^3).
$$

Consistent mass and damping matrices $\mathcal{M}$ and $\mathcal{C}$ are replaced by their lumped counterparts $\bar{\mathcal{M}}$ and $\bar{\mathcal{C}}$, leading to trivial inversion for fully explicit displacement-based time integrators. To initiate the time iterations, the quantity $d_h^{(-\Delta t)}$ is computed using a second order Taylor approximation of $d(-\Delta t)$, for consistency. Once the initial conditions $d_{h,0}$,
The explicit scheme is known to be conditionally stable with respect to the choice of \( \Delta t \), consistent with the well known Courant–Friedrichs–Lewy (CFL) condition (see, e.g., [4])

\[
\Delta t = \alpha_s \frac{h_e}{\sqrt{\frac{E}{\rho(1-v^2)}}},
\]

where \( h_e \) is a characteristic length (here assumed as the diameter of the circumsphere associated with each tetrahedral element), \( n_e \) is the total number of elements in the mesh, and the safety factor \( \alpha_s \in (0, 1) \) is set to \( \alpha_s = 0.9 \). The local displacement solution is updated as shown in (17), introducing the internal force at step \( n \) as \( f^{(n), \text{int}} = Kd^{(n-1)}_h \) and renaming \( f^{(n)} \) as \( f^{(n), \text{ext}} \), i.e., the external force at step \( n \). This is consistent with our implementation in Algorithm 1.

Remark 1 Application of initial conditions or loads at \( t = 0 \) in explicit structural dynamics may lead to the excitation of a broad range of frequencies. To prevent this to occur, a ramp function is applied to the external force, through a time-dependent function \( \mathcal{R}(t, t_{\text{end}}) \), such that

\[
f^{(n)}(t_{\text{end}}) = f^{(n)} \mathcal{R}(t_n, t_{\text{end}}).
\]

This allows for a smooth and quasi-static application of the external loading until time \( t_{\text{end}} \). Although many formulations are available in the literature, we select a simple linear ramp function

\[
\mathcal{R}(t_n, t_{\text{end}}) = \begin{cases} t_n/t_{\text{end}} & t_n \leq t_{\text{end}} \\ 1 & \text{otherwise}. \end{cases}
\]

2.2.2 Distributed solver

The pseudo-code in Algorithm 1 illustrates how our displacement-based parallel finite element elastodynamics solver is implemented based on element-level computation and communication operations [3,4,19]. We consider a computational mesh partitioned and distributed over \( n_c \) processors, labeled as \( i = 1, \ldots, n_c \), each containing \( n_e[i] \) finite elements.

The steps in Algorithm 1 emphasized using boxes denote CPU-to-CPU (or GPU-to-CPU and vice versa) synchronization tasks. These ensure equilibrium to be satisfied within each local partition at every time step, based on communicating internal and external force information at the shared nodes (i.e., nodes belonging to multiple mesh partitions). However, synchronization constitutes one of the main factors responsible for performance degradation in distributed structural analysis codes. This is particularly true for fully explicit time integration, where shared node information needs to be communicated to the root processor at every time step, and therefore millions or tens of millions of times during one simulation. Thus, development of effective synchronization-avoiding strategies would boost the performance of explicit distributed finite element codes, particularly in the context of ensemble multi-GPU finite element solvers, discussed in our previous work [34].

Remark 2 The procedure in Algorithm 1 generalizes different types of structural problems by forming element stiffness matrix \( K_e \) and external loading \( f^{(n), \text{ext}}_e \) at every time step. For isotropic linear elastodynamics and constant external loading, it is instead sufficient to generate the local stiffness matrix and loading vector only once, before the beginning of the time loop, and re-use them at every time step. We refer to this practice as “element pre-assembly”.

2.2.3 Artificial mass scaling

Explicit time integration schemes are stable under condition (14) on \( \Delta t \), where the small time steps increase the computational cost for long-term or steady state simulations, and the frequency of synchronization tasks. In practice, artificial mass scaling is a widely adopted pre-processing technique to increase \( \Delta t \), for situations where the choice of the time step size is dictated by a few small elements in the mesh. The pseudo-code for a typical implementation is shown in Algorithm 2 with more recent approaches discussed, for example, in [38,52].

The price to pay for a larger time step is a non-physical increase in the mass of the system that may potentially affect the system dynamics. Therefore, the scaling factor \( \beta \) has to be carefully selected not to alter the structural response. In this paper, we consider Algorithm 2 applied to the most expensive numerical experiment in Sect. 5.2.
Algorithm 1  Displacement-based distributed linear elastodynamics solver.

Communicate the Dirichlet boundary conditions and initial conditions to each processor
Form and communicate the global lumped mass and damping matrices $\mathbf{M}, \mathbf{C}$ to each processor

\[ \text{for } \eta = 1, 2, \ldots, n_{\eta} \text{ do} \]
\[ \text{Initialize local internal and external forces } f^{(n)}_{\eta, \text{int}}, f^{(n)}_{\eta, \text{ext}} \text{ as zero vectors} \]
\[ \text{for } e = 1, 2, \ldots, n_{e}[i] \text{ do} \]
\[ \text{Form element stiffness matrix } \mathbf{K}_{e} \text{ and external force } f^{(n)}_{e, \text{ext}} \]
\[ \text{Calculate element internal force: } f^{(n)}_{e, \text{int}} = \mathbf{K}_{e} d^{(n-1)}_{h,e} \]
\[ \text{Update local forces } f^{(n)}_{\eta, \text{int}}, f^{(n)}_{\eta, \text{ext}} \text{ by } f^{(n)}_{\eta, \text{int}} \text{ and } f^{(n)}_{\eta, \text{ext}} \text{ based on global element label } e \]
end for

\[ \text{Send local forces } f^{(n)}_{\eta, \text{int}}, f^{(n)}_{\eta, \text{ext}} \text{ to the root processor} \]

\[ \text{Update local forces } f^{(n)}_{\eta, \text{int}}, f^{(n)}_{\eta, \text{ext}} \text{ based on contributions from the shared mesh nodes} \]

\[ \text{Send updated local forces } f^{(n)}_{\eta, \text{int}}, f^{(n)}_{\eta, \text{ext}} \text{ back to each processor} \]

\[ \text{Update local solution } d^{(n)}_{h} \text{ using (17)} \]

\[ \text{Strongly enforce Dirichlet boundary conditions} \]
end for

Algorithm 2  Artificial mass scaling.

Loop through all elements to determine $\Delta t$ by equation (14)
Set a target time step $\Delta t = \beta \Delta t$, $\beta > 1$
Initialize the artificial density vector $\hat{\rho}$

\[ \text{for } e = 1, 2, \ldots, n_{e} \text{ do} \]
\[ \text{Calculate the element time step size } \Delta t_{e} = \alpha_{e} h_{e} / \sqrt{\frac{E}{\rho(1 - \nu^{2})}} \]
\[ \text{if } \Delta t_{e} < \Delta t \text{ then} \]
\[ \hat{\rho}[i] = E \Delta t_{e} / \alpha_{e}^{2} h_{e}^{2}(1 - \nu^{2}) \]
\[ \text{else} \]
\[ \hat{\rho}[i] = \rho \]
end if
\[ \text{Recompute the mass matrix } \hat{\mathbf{M}} \text{ based on } \hat{\rho} \]
end for

\[ \text{Compute the total percent mass increase } m_{\eta} \]

3 Data driven model

In this section, we introduce data driven models based on artificial neural networks designed to learn the dynamics of discrete systems generated through the finite element method, specifically focusing on LSTM networks. Hochreiter and Schmidhuber introduced the LSTM deep neural network in their 1997 seminal paper [17] to overcome the problems with vanishing and exploding gradients in vanilla RNN.

In what follows, we will drop the subscript $(\cdot)_{h}$ since only discrete solutions will be considered. In addition, we also introduce the notation

\[ \|v\|_{2} = \sqrt{\sum_{i=1}^{n} |v_{i}|^{2}}, \quad v \in \mathbb{R}^{n} \]
\[ \|V\|_{F} = \sqrt{\sum_{i=1}^{n} \sum_{j=1}^{m} |V_{ij}|^{2}}, \quad V \in \mathbb{R}^{n \times m} \]

![Fig. 1 Schematic representation of LSTM cell](image)

3.1 LSTM cell model

As illustrated in Fig. 1, a classical LSTM cell consists of a hidden state $h$, a cell state $c$, an input $d$, the activation functions $\sigma_{1}, \ldots, \sigma_{5}$ and the component-wise operations $\otimes$ and $\oplus$. In each cell, the feedback control feature of the deep
A recurrent neural network is further reinforced by bringing a number of gated flow controls.

The Forget gate learns how information from the previous cell state \( c^{(n-1)} \) will be kept or discarded. It relates to the weighting matrices \( W_{f,d}, W_{f,h} \), associated with the current input \( d^{(n)} \) and the previous hidden state \( h^{(n-1)} \), respectively, plus the bias \( b_f \). The activation \( \sigma_1 \) is usually a sigmoid function, which provides an output range from 0 to 1. The forget gate implements the expression

\[
\scriptstyle f^{(n)} = \sigma_1 (W_{f,d}d^{(n)} + W_{f,h}h^{(n-1)} + b_f). \tag{18}
\]

The Input gate adds new information to the cell state from the current input \( d^{(n)} \), previous hidden state \( h^{(n-1)} \) and their corresponding input gates weighting matrices \( W_{i_{1,2},d} \), and biases \( b_{i_{1,2}} \), through the operations

\[
\begin{align*}
&i_1^{(n)} = \sigma_2 (W_{i_{1,2},d}d^{(n)} + W_{i_{1,2},h}h^{(n-1)} + b_{i_1}) \\
&i_2^{(n)} = \sigma_3 (W_{i_{2},d}d^{(n)} + W_{i_{2},h}h^{(n-1)} + b_{i_2}) \\
&\{i^{(n)}\} = i_1^{(n)} \otimes i_2^{(n)}.
\end{align*} \tag{19}
\]

where \( \sigma_2, \sigma_3 \) are a sigmoid and a hyperbolic tangent activations, respectively, and \( \otimes \) is the Hadamard product. Then, the cell state \( c^{(n)} \) is updated as

\[
c^{(n)} = c^{(n-1)} \otimes f^{(n)} \oplus i^{(n)}, \tag{20}
\]

where \( \oplus \) denotes component-wise sum. Finally, the output gate updates the hidden state \( h^{(n)} \) as

\[
\begin{align*}
s^{(n)} &= \sigma_4 (W_{o,d}d^{(n)} + W_{o,h}h^{(n-1)} + b_o) \\
h^{(n)} &= \sigma_5 (c^{(n)} \otimes s^{(n)}),
\end{align*} \tag{21}
\]

where \( \sigma_4, \sigma_5 \) are again the sigmoid and hyperbolic tangent functions and \( W_{o,d}, b_o \) are weights and bias associated with the output gate.

In our application, \( d^{(n)} \) will be the discrete displacement solution at step \( n \), at the shared nodes. Since we are interested in sequence-to-sequence learning, we would have multiple LSTM cells like the one introduced above, sharing the same set of weights and biases.
3.2 LSTM encoder–decoder network

The design of our deep neural network model is demonstrated in Fig. 2. It is the combination of a k-layer bidirectional LSTM encoder and a single-layer unidirectional LSTM decoder, inspired and implemented based on [32]. Stacking encoder layers ensures that more information is extracted from the input sequence while limiting the total number of parameters. The encoder–decoder structure is also designed to handle variable sequence lengths, which is a distinctive feature in language translation and other sequence-to-sequence models [15]. In addition, our encoder is enriched by a bidirectional structure, which helps to capture dependency across the whole input sequence [15]. Note that a bidirectional LSTM network was also used in [18] to approximate time-dependent differential equations over discrete lattices through a Many-to-Many recurrent architecture [15]. This is different from the Many-to-One architecture used for the encoder in the present study. Additionally, unlike [18], our decoder remains uni-directional. Further, each item of our input sequence varies spatially, while the input of [18] considers time series at each collocation point.

Our proposed deep neural network model can be simply expressed as the following operation with the input sequence \( X \) of length \( n_p \) and predicted output sequence \( \hat{Y} \) of length \( n_f \).

\[
\hat{Y} = \left( N_{D, \Theta_D} \circ N_{E, \Theta_E} \right) X = N_{\Theta} X \tag{22}
\]

\[
X = [d^{(1)}, d^{(2)}, \ldots, d^{(n_p)}] \in \mathbb{R}^{n_{\text{ dof}} \times n_p}
\]

\[
\hat{Y} = [\hat{d}^{(n_p+1)}, \hat{d}^{(n_p+2)}, \ldots, \hat{d}^{(n_p+n_f)}] \in \mathbb{R}^{n_{\text{dof}} \times n_f} \tag{23}
\]

The encoder and decoder models are expressed as \( N_{E, \Theta_E} \), \( N_{D, \Theta_D} \) respectively, and \( n_{\text{dof}} \) stands for the total number of degrees of freedom in all displacement solutions at the shared nodes. We further refer to the overall LSTM encoder–decoder model as the composition \( N_{\Theta} \), parameterized by \( \Theta = \Theta_D \cup \Theta_E \).

In the encoder model, each displacement solution \( d^{(i)} \), \( i = 1, \ldots, n_p \) of input sequence \( X \) is fed to a LSTM cell (Fig. 1) while having hidden and cell states coming from two opposite directions. We use expressions \( h_{E,j}^{(i)} \) and \( c_{E,j}^{(i)} \) to represent the hidden and cell states in the direction \( 1 \to n_p \) (solid arrow) and \( q_{E,j}^{(i)} \) and \( r_{E,j}^{(i)} \) along \( n_p \to 1 \) (dashed arrow), where \( j = 1, \ldots, k \) is the layer index. At the final encoder layer, hidden and cell states from two directions are concatenated separately and provided to the decoder model as initial states \( h_{D}^{(0)} \) and \( c_{D}^{(0)} \).

The decoder then receives the final item \( d^{(n_p)} \) in \( X \) and recursively produce the predictions \( \hat{d}^{(n_p+j)} \), \( j = 1, \ldots, n_f \) in \( \hat{Y} \). At each decoding step, the previous prediction will be forwarded to the next step as an input, with a dense neural network bridging the different size between the hidden state and model output.

During training (and validation) we use a Mean Squared Error (MSE) loss function built from the predicted output \( \hat{Y} \) and true numerical solution \( Y \) as

\[
L(Y, \hat{Y}) = \frac{1}{n_f \cdot n_{\text{dof}}} \sum_{j=1}^{n_f} \| Y_j - \hat{Y}_j \|_2^2
\]

\[
= \frac{1}{n_f \cdot n_{\text{dof}}} \sum_{j=1}^{n_f} \sum_{i=1}^{n_{\text{dof}}} |Y_{ij} - \hat{Y}_{ij}|^2 \tag{24}
\]

and perform gradient-based updates for the trainable parameters \( \Theta \).

3.2.1 Network training and evaluation

We tailor the training and evaluation of the proposed network to the specific application of interest, i.e., dynamical systems simulated through explicit numerical solution algorithms in time. In the structural analysis of soft biological tissue, the time step \( \Delta t \) is usually in the range \( 1 \times 10^{-6} - 1 \times 10^{-3} \) due to the stability condition (14). As a result, to completely describe the full dynamic response, we might need millions of data points.

Such small time step will also lead to limited changes between displacement solutions at two successive time steps, and therefore almost identical model input \( X \) and true output \( Y \). However, for effective training, we would like each of our training sample to contain enough information of the underlying dynamics. In other words, the input \( X \) and output \( Y \) should be sufficiently different for the network to learn a relevant mapping and not just an identity operator, typical of mere steady state conditions (See additional discussion in Sect. 5.1.3).

This is accomplished through a so-called sample-refill strategy during the training and evaluation stages, respectively. First, as illustrated in Fig. 3, instead of using the full dataset, we only pick a displacement solution every \( n_s \) steps (operation \( S_{n_s} \)).
This sampled data is what we group, batch and feed into the deep neural network at training and validation stages. Upon successful training, as described in Fig. 2, the network will take \( n_p \) displacement solutions in the past and use them to predict \( n_f \) future steps. Because of the pre-processing, the time lag between each predicted solution will still be \( n_s \).

To fill these gaps, we leverage a refill operation during the evaluation (or prediction) stage. Given enough steps computed in the past, we use the model \( n_s \) times to produce \( n_s \cdot n_f \) predictions. After the first time, the input \( X \) is shifted \( n_s - 1 \) times forward such that the model can generate the missing displacement predictions at all shared mesh nodes, as illustrated in Fig. 4 (operation \( R_{n_s} \)).

More details of the refill task can be found in Algorithm 3 below. The sample-refill strategy can adequately improve the training efficiency, since less training examples will be used for an increasing sample size \( n_s \).

### 4 A data-driven synchronization-avoiding algorithm

We propose a data-driven methodology to minimize synchronization in distributed, explicit in time, finite element structural analysis, which starts with data preparation. This consists in running Algorithm 1 in parallel, and gathering the sequential displacement solutions for each of the \( n_c \) processors. We then identify all degrees of freedom associated with the shared nodes and form a training dataset using samples spaced by \( n_s \) time steps. Next, we train \( n_c \) independent replicas of the LSTM network illustrated in Fig. 2, producing a set of optimally trained network models \( \mathcal{N}_{\theta_j}[j] \), \( j = \{1, 2, \ldots, n_c\} \). We finally apply Algorithm 3 where the synchronized displacements at the shared nodes are modeled by the network at each processor instead of being communicated across partitions.

The parameter \( n_{cri} \) in Algorithm 3 is used to switch between the synchronization process and using displacement predicted by the network for all shared degrees of freedom. Clearly \( n_{cri} \) is expected to be set as low as possible to attain the most speedup but it cannot be less than \( n_p \cdot n_s + 1 \) since the model requires the first \( n_p \cdot n_s \) steps to start. For additional clarity, we sketch the sample-refill approach in Fig. 5 for \( n_c = 2 \), highlighting the training, validation and prediction stages. We further distinguish between an Offline and an Online prediction stage in Sect. 5.1.2. Offline prediction are performed only on the shared nodes, while both shared and internal node displacements are updated in the online prediction stage.

Further note that the previously mentioned refill stage (see Sect. 3.2.1) is realized by indexing every intermediate
Initially we consider a distributed mesh on 2 processors, with are managed through the Message Passing Interface (MPI). The damping with factor \( \alpha \) under-damped oscillatory regime using a mass proportional

Specifically, we focus on the simple cantilever beam model containing only 110 vertices and 256 tetrahedral elements. To test the proposed computational framework, we use a computational model, compared with synchronization costs, execution times for evaluating pre-trained network models at every time step are negligible.

5 Numerical examples

5.1 Cantilever model

To test the proposed computational framework, we use a simple cantilever beam model containing only 110 vertices and 256 tetrahedral elements. Specifically, we focus on the under-damped oscillatory regime using a mass proportional damping with factor \( \alpha \). As shown in Fig. 6, the cantilever has fully fixed restraints \( d|_{x=0} = 0 \) at one end, and external load \( f \) consists of a ramp of 1 s (see Remark 1), followed by a constant distributed body force equal to \( f_z = 0.5 \text{ dynes/cm}^3 \) in the \( z \)-direction, i.e., \( f = [0, 0, -f_z]^T \). Further, homogeneous initial conditions are considered here as \( d^{(0)} = v^{(0)} = 0 \), and geometric and material model parameters are listed in Table 1.

The time step size is set to \( 2.48 \times 10^{-4} \) via Eq. (14). The mesh partitioning is realized by \texttt{mgmetis} [8] based on the \texttt{ParMETIS} library [26], and parallel computations are managed through the Message Passing Interface (MPI). Initially we consider a distributed mesh on 2 processors, with 8 shared mesh nodes each. The result of mesh partitioning is shown in Fig. 7.

Table 1 Geometric and material parameters for the cantilever beam model

| Parameter                  | Value                        |
|----------------------------|------------------------------|
| Length (\( L \))           | 25 (cm)                      |
| Width (\( W \))            | 1 (cm)                       |
| Height (\( H \))           | 1 (cm)                       |
| Young’s modulus (\( E \))  | \( 1 \times 10^9 \) (dynes/cm²) |
| Density (\( \rho \))       | 1 (g/cm³)                    |
| Poisson’s ratio (\( \nu \))| 0.3                          |

steps as lists \( N^i_p \) and \( N^i_f \) such that the model inputs are properly shifted for continuous predictions. For large computational models, compared with synchronization costs, execution times for evaluating pre-trained network models at every time step are negligible.

5.1.1 Hyperparameters and network training

In our numerical experiments, we optimize over a number of selected hyperparameter realizations by performing a grid search on the mini-batch size \( n_B \), the hidden unit size \( n_H \) and the initial learning rate \( \eta_0 \). In addition, we utilize an exponential learning rate scheduler, where \( \eta(x) = \eta_0 \gamma^x \) for a given epoch \( x \). In the search of a good initial learning rate \( \eta_0 \), we set a minimum learning rate \( \eta_{\text{min}} = 5 \times 10^{-7} \) and keep the decay rate fixed at \( \gamma = 0.9995 \). As a result, the total number of epochs \( n_{\text{epoch}} \) can be calculated as:

$$ n_{\text{epoch}} = \lceil \log_\gamma \left( \frac{\eta_{\text{min}}}{\eta_0} \right) \rceil. $$ (25)

The selected grid of hyperparameter realizations includes \( n_B = 5, 10, 20, 50, n_H = 20, 50, 100 \) and \( \eta_0 = 5 \times 10^{-3} \) \( (n_{\text{epoch}} = 18, 416), 5 \times 10^{-4} \) \( (n_{\text{epoch}} = 13, 812), 5 \times 10^{-5} \) \( (n_{\text{epoch}} = 9208), \) leading to 36 cases in total.

In terms of how to pick the best combination of \( n_B, n_H \) and \( \eta_0 \), we introduce the following criterion, rather than simply looking at the loss curves produced by MSE (24) during training and validation

$$ E_{\text{mse}} = \frac{1}{N \cdot n_f \cdot n_{\text{dof}}} \sum_{j=i}^{i+Nn_f-1} \sum_{j=i}^{i+n_f-1} \| \tilde{d}^{(j)} - d^{(j)} \|_2^2 $$

$$ = \frac{1}{N \cdot n_f \cdot n_{\text{dof}}} \left( \sum_{j=i}^{i+n_f-1} \| d^{(j)} - \tilde{d}^{(j)} \|_2^2 + \cdots \right) $$ (26)

that is, after the network is trained, starting from step \( i \), we go ahead and use the trained network model \( \mathcal{N}_\Theta \) for \( N \) times. This is referred to as the testing phase. Then, by definition, \( E_{\text{mse}} \) quantifies a MSE error of displacement predictions on all shared nodes over \( N \cdot n_f \) steps (note that a lag of \( n_t \) time steps is still present between a collection of \( n_f \) successive
Algorithm 3 A synchronization-avoiding algorithm for distributed linear elastodynamics.

Step 1: Partition the mesh over \( n_p \) processors and compute \( \Delta t \)

Step 1a: If needed, apply mass scaling following Algorithm 2

Step 2: Identify Dirichlet nodes and initial conditions for each processor

Step 3: For the generic \( j \)-th processor, set \( t = \Delta t, n = 1, n_{crit}, n_p, n_f, n_s \)

\[ \text{while } t \leq T \text{ do} \]

\[ \text{if } n \leq n_{crit} \text{ then} \]

\[ \text{Apply Algorithm 1 to compute } d^{(n)} \]

\[ t = t + \Delta t \]

\[ n = n + 1 \]

\[ \text{else} \]

\[ \text{for } i = 1, 2, \ldots, n \text{ do} \]

\[ N'_p = \{ i + n - n_p n_s - 1 : n_s: i + n - n_s - 1 \} \]

\[ N'_f = \{ i + n - 1 : n_s: i + n + n_f n_s - n_s - 1 \} \]

\[ \text{Forward } \text{pre-trained NN model: } d^{(N'_p)}[\text{shared}] \xrightarrow{N(i)/j} \hat{d}^{(N'_f)}[\text{shared}] \]

\[ \text{end for} \]

\[ \text{Gather all predictions on shared nodes: } \hat{d}^{(N'_f)}[\text{shared}] = [\hat{d}^{(N'_f)}[\text{shared}], i = 1, 2, \ldots, n_s] \]

\[ \text{for } m = n, n + 1, \ldots, n + n_f n_s - 1 \text{ do} \]

\[ \text{Apply Algorithm 1 to compute } d^{(m)} \text{ without } \text{“boxed” steps (synchronization)} \]

\[ \text{Update } d^{(m)} \text{ at shared nodes by the corresponding modeled values in } \hat{d}^{(N'_f)}[\text{shared}] \]

\[ \text{end for} \]

\[ \text{Impose Dirichlet boundary conditions to } d^{(N'_f)} \]

\[ t = t + n_s \Delta t \]

\[ n = n + n_s n_f \]

\[ \text{end if} \]

\[ \text{end while} \]

shared nodes predictions in the testing phase, and this gap is only “refilled” at the offline and online prediction stages (see Sect. 5.1.2). Note that during the calculation of \( E_{\text{mse}} \), only “refilled” at the offline and online prediction stages (see Sect. 5.1.2). Note that during the calculation of \( E_{\text{mse}} \), after the first time \( (N = 1) \), we no longer have exact inputs. In other words, \( \hat{d} \) is the model output \( (N = 2) \) with model input \( d_s \) predicted from the last step. Here we abuse the notation \( \gamma \) to avoid stacking multiple “hats”. Trivially, (26) collapses to (24) when \( N = 1 \).

Using \( E_{\text{mse}} \) is more consistent with our objective, in the prediction stage, to use previously predicted displacements as inputs to predict synchronized displacements at future times. A standard MSE loss (24) on the other hand, would only account for the performance of a single model application \( (n_f \text{ steps}) \), thus delivering less information about accuracy and stability on a longer time horizon. Note that alternative ways to ensure robust long-term predictions for non-recursive networks are proposed in [9, 10, 44], where either a recurrent loss function similar to (26) was used in the context of ResNet network training, or the artificial random-walk noise was added to the input data to counter error accumulation, in graph neural network training.

In practice, we set \( N = 45 \) and calculate the square root of \( E_{\text{mse}} \) as a measure of accuracy per degree of freedom and per prediction. All the results are reported in Fig. 8, where the lowest learning rate \( \eta_0 = 5 \times 10^{-5} \) provides the worst accuracy.

In addition, we train a number of networks equal to the number of mesh partitions (two in this case, see Fig. 7), so we can quantify the variability in the prediction accuracy produced by multiple network replicas, and favor models producing robust predictions across processors. This observation penalizes networks with a larger learning rate \( \eta_0 = 5 \times 10^{-3} \), characterized by large inter-processor variability. The optimal hyperparameter combination (marked by “\( \Delta \)” in Fig. 8c) is finally selected as \( n_B = 5, n_H = 100, \eta_0 = 5 \times 10^{-4} \), since it offers the best compromise between a moderate training cost, a sufficient accuracy and limited model discrepancy. It is worth noting that one could further refine this choice by expanding the number of hyperparameter combinations in the grid search. For example, we expect better accuracy for even smaller mini-batch size \( n_B \) when \( \eta_0 = 5 \times 10^{-4} \) via the observation of a roughly monotonic behavior. However, the selected accuracy metric \( \sqrt{E_{\text{mse}}} \) is already close to \( 1 \times 10^{-3} \), regarded as satisfactory.

We also study the effects of the sequence lengths \( n_p, n_f \), the training set size \( n_{ts} \), and the sample size \( n_s \) in Sect. 5.1.3. The parameters \( n_p, n_f \) specify the amount of past and future displacement observations used in training, whereas the training set size \( n_{ts} \) controls the total amount of pre-computed numerical solutions fed into the network, i.e., temporally speaking, we use \( n_{ts} \) (and later sampled by \( n_s \)) of all finite element solutions at each partition to train our surrogate models in parallel. The sample size \( n_s > 1 \) is also regarded as a hyperparameter that may be dependent on the time step size \( \Delta t \), problem type, etc., where larger \( n_s \) are
associated with increasing cost savings during training. In the experiments shown in Fig. 8, these parameters are fixed as \( n_f = n_s = 20, n_{ts} = 50\% \), \( n_s = 100 \).

Further, our training is performed through the AdaGrad optimizer [28], shuffling the order and choice of the mini-batches at each epoch.

### 5.1.2 Offline and online data-driven model evaluation

We proceed to define two ways to use our data-driven model. First, as discussed in Sect. 4, we train several network models in parallel based on the shared degrees of freedom at each partition. Next, we inspect whether displacement evolution on each set of shared nodes are ideally learned, which we refer to as the offline prediction stage, which differs from the previous testing stage, as missing predictions are refilled. Offline prediction performance using the hyperparameters determined in Sect. 5.1.1 is shown in Fig. 9, along with the finite element solutions, denoted as the truth. Both the damped dynamics and the convergence to the steady state are accurately learned. Besides a satisfactory accuracy, two separately trained network models on the same set of shared degrees of freedom show very good agreement, with hardly noticeable discrepancies after 15 s on the lateral \( x \) and \( y \) components.

A vertical line in each subplot of Fig. 9 separates a region on the left where \( 1/n_s \) of the simulated data is used for training (75\% of which is used to compute the parameter updates), from a pure prediction region on the right, where the system’s response is completely modeled based on the learned dynamics. However, the line does not indicate where we start evaluating the surrogate model which instead occurs after \( n_p \cdot n_s \) steps from the beginning of the simulation. While an overlap between the prediction and training regions may seem inappropriate, this is only apparent, as the network inputs in the prediction phase are the results of previous network evaluations, instead of the true data.

The offline prediction discussed above serves as a preliminary error monitor for an online prediction task, which constitutes the basis for the synchronization-avoiding strategy shown in Algorithm 3. When predictions are performed online, we start to combine the pre-trained neural network surrogates with the distributed finite element solver. This inevitably causes the error of data-driven model to propa-
From Fig. 11, we observe an initial increase in the error once we start to use the trained surrogate model. As time evolves, $e_2$ gradually reduces before the vertical bar due to the increasingly smaller oscillation amplitude. After the vertical bar, $e_2$ increases again due to the lack of familiarity of the data unseen during training, but eventually drops, suggesting a stable prediction of the steady state.

In the current cantilever problem, small-valued predictions in the lateral directions $(x, y)$ would be more susceptible to get polluted by an error of the magnitude shown in Fig. 11, as we can see from the following sections. However, this should be of less concern since the error remains bounded and the dynamics is governed by the dominant $z$-component.

**Remark 3** The vertical bar shown in Fig. 10 and in many other figures in this paper indicates the end of the training-validation period for the shared dofs. However, we often plot the dynamics of internal (non-shared) dofs to show how they are affected by the propagation of errors.

### 5.1.3 Network performance tests

**Input–output sequence length** We investigate the effects produced on the network accuracy by changing the parameters $n_p$ and $n_f$, indicating the number of time steps included in the network input and output, respectively. We consider three symmetric cases with an increasing number of steps $n_p = n_f = (5, 20, 50)$ and an asymmetric case with $n_p = 20$ and $n_f = 5$. As shown in Fig. 12, the best results are obtained for an intermediate number of steps either symmetric or asymmetric while an excessive or limited number of steps seems to reduce the flexibility of the network predictions. A sufficient number of inputs in the past is required for accurate long-term predictions. However, for $n_p = n_f = 50$, the non-optimal performance is due to an insufficiently complex model, since learning a much longer input–output dependence would require more hidden units, deeper encoder etc. Therefore, without making unnecessary model refinements, a choice of $n_p = n_f = 20$ should suffice for the current dynamical system.

Finally, it is also worth mentioning that $n_p, n_f$ also affect the overall speedup of our data-driven framework, since a smaller $n_f$ means more network usage. This is the reason why we prefer $n_p = n_f = 20$ over $n_p = 20, n_f = 5$, although their resulting accuracy seems comparable.

**Training set size** In this section we perturb the hyperparameter $n_{ts}$, to see how the total number of training examples affects the accuracy of the network predictions. As expected...
our approach for varying sample size

Fig. 12 Predicted dynamics for the cantilever model trained with different input–output sequence length \((n_p, n_f)\). CPU label: 0.1. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 95000. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0)

As discussed above, we conjecture that the poor performance under a small \(n_s\) is due to the limited variation of adjacent solutions produced by the explicit FEA solver. This might be amplified by using single precision in the neural network computations. However, experiments using double precision data types did not produce any improvements in Fig. 15.

**Refined mesh with additional processors** Next, we extend our data-driven framework to a refined cantilever model partitioned over 6 CPUs and shown in Fig. 16. The mesh contains 4615 tetrahedral elements, we select \(\Delta t = 6.2 \times 10^{-5}\) and use the previously discussed optimal combination of hyperparameters \(n_B = 5, n_H = 100, \eta_0 = 5 \times 10^{-4}, n_P = n_f = 20, n_{ts} = 0.5\). We set the sample size \(n_s\) to 500, selected based on the ratio of time step size between the two mesh resolutions,
and then increased it to reduce the training cost. Each model learns the dynamics of approximately 105 degrees of freedom on average (35 shared nodes). Even though some differences are observed in the lateral y-direction for all models, the dominant response in the z-direction is accurately learned.

**Test with skewed forcing** We add an additional external loading component in the y-direction, i.e., $f_y = f_z = 0.5$ dynes/cm$^2$, resulting in extra non-zeros in the update for the displacement solutions (17), and therefore with additional complexity imposed to the network training (Fig. 17).

Results are shown in Fig. 18 where the network successfully learns the correct dynamics in the $y$-$z$ plane.

**Test on a discontinuous loading** Next, we switch the external loading back to the z-direction but consider a discontinuous load with respect to time: $f(t) = [0, 0, -f_z I_{t < 3}(t)]^T$, where the indicator function $I_{t < 3}(t)$ is defined as

$$I_{t < 3}(t) = \begin{cases} 1 & t < 3s \\ 0 & t \geq 3s \end{cases}.$$ (28)

As shown in Fig. 19, having part of the training data associated with a forced rather than free dynamic response does not negatively affect the ability of the network to reach a steady state with zero displacements. However, training for this task is harder and requires more samples, so we set $n_s = 80$ and $n_{ts} = 0.6$.

**General initial conditions** For more general applications, we wish our proposed LSTM network to learn and predict the evolution of a class of dynamical systems rather than a very specific case. To do so, we first propose a data-driven model trained on a collection of displacement solutions, generated from different initial conditions (IC) $d^{(0)}$ and check if this model is able to evolve the correct dynamics from an initial condition unseen at training. Note that this strategy has also been used in [9,16,56] for training general residual networks.
with 10 sets of initial conditions, the dynamics in the global conditions included in the training dataset. For networks trained proposed approach is affected by the number of initial con-

Fig. 17 Predicted dynamics of the refined cantilever model. CPU labels are 1,3,5. Training is based on shared degrees of freedom of each partition and the plotted nodes are not shared. Predicted steps: 490,000

(a) CPU:1, node: (7.45, 0.69, 0.31). (b) CPU:3, node: (18.6, 1, 0.77). (c) CPU:5, node: (23.66, 1, 0.68).

Fig. 18 Predicted dynamics for the cantilever model with skewed loading. CPU label: 0,1. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 98000. Top row: CPU:0, node (25.0, 0.0, 1.0). Bottom row: CPU:1, node (5.77, 0.0, 0.0)

approximating dynamical systems. To generate training data, we random perturb the steady solution \( \tilde{d} \) up to 25% and use it as the initial displacement, i.e.

\[
d^{(0)} = (1 + u) \tilde{d},
\]

where \( u \) is a uniformly distributed random variable \( u \sim U(-0.25, 0.25) \). Figure 20 shows how the accuracy of the proposed approach is affected by the number of initial conditions included in the training dataset. For networks trained with 10 sets of initial conditions, the dynamics in the global \( z \)-direction is captured with satisfactory accuracy by both network replicas.

**General uniform forcing** Next, we seek to achieve generalization with respect to a uniform external load. This task is intrinsically more complicated than varying the initial conditions, as a time-dependent load \( f \) can severely affect the dynamic system response (see, e.g., forced vibrations [11]). Similarly to the previous section, we expand the training dataset by adding displacement ensembles generated by our distributed finite element solver through the application of multiple uniform loads (the number of loads is denoted as \( L_n \)). A parametric family of loading conditions is obtained
by introducing a uniform random variable to the $z$-component of $f$

$$f = [0, 0, -\alpha_f]^T; \quad \alpha_f \sim \mathcal{U}(\alpha_{f,\text{min}}, \alpha_{f,\text{max}}),$$

(30)

where $\alpha_{f,\text{min}} = 0.3$, $\alpha_{f,\text{max}} = 0.7$ are the selected prior bounds.

However, simply increasing the size of the dataset is not sufficient in this case to produce accurate predictions (see Fig. 21).

We therefore modify the network architecture using a conditional decoder [14,30] (e.g. see Fig. 22), where the loading condition is concatenated to the last item in the input sequence, i.e., $d^{(np)}$. It is also worth noting that, by stacking multiple identical $\alpha_f$ values to $d^{(np)}$, we can further improve
the accuracy. In our experiment, we concatenate 12 copies of $\alpha_f$ to $d^{(np)}$.

Results produced by this conditional encoder–decoder LSTM network are illustrated in Fig. 23, where training with 10 different external loads is sufficient to achieve accurate predictions for an external load not seen at training. Finally, note how the current conditional structure can be trivially extended to the previous test case on multiple initial conditions and we expect better performance than Fig. 20, especially for long-time behaviour of small lateral (i.e., $x$ or $y$) displacements.

**General distributed load** We can extend the conditional network introduced in the previous section for training under a random convex combination of the following loads

$$f_{1u}(x) = \begin{bmatrix} 0,0, -\frac{x}{50} \end{bmatrix}^T ; \quad f_{2u}(x) = \begin{bmatrix} 0,0, -\frac{x^2}{500} \end{bmatrix}^T ,$$

so the network can produce accurate predictions for a general forcing of the form

$$f_s(x) = \begin{bmatrix} 0,0, -(1 - \beta_f) f_{1u} - \beta_f f_{2u} \end{bmatrix}^T ,$$

where $\beta_f \sim U(0,1)$. Similarly to the uniform loading scenario, we condition the predictions by concatenating nodal forces from $f_s$ to $d^{(np)}$.

Testing results are shown in Fig. 24, using an $f_s$ that does not belong to the training set. Similar to the previous section, training based on 5–10 datasets is sufficient for our conditional network to accurately learn the dynamics under a general $f_s$.

**Full system modeling** While tests in the previous sections focus on predicting the system dynamics at a small number of shared locations, we would like to see how the accuracy of the proposed network is affected for an increasing number of such locations. Thus, we use the proposed LSTM network to predict the dynamic response for a coarse discretization of the entire cantilever beam. The hyperparameters are still selected as $n_B = 5, n_H = 100, n_0 = 5 \times 10^{-4}, n_p = n_f = 20, n_s = 100, n_t = 0.5$ and the external loading is $f = [0,0,-f_z]^T$.

Clearly in this case, once we start to use the trained network model, no more finite element computations are required, leading to a substantial reduction in the computational effort.

We first show the evolution of displacement predictions at three distinctive locations in the mesh, i.e., near the clamped end, in the middle and at the tip. Figure 25 shows a satisfactory accuracy in all cases, except for the $x$-displacement at the tip, but, in this case, the displacement is practically zero and therefore the absolute error still acceptable. We also show a comparison of the exact and predicted $z$-displacement contours in Fig. 26. In such a case, the surrogate model has no knowledge on how to strongly enforce a Dirichlet boundary condition at the clamped end. However, the predicted displacement of $-1.56 \times 10^{-7}$ is sufficiently small.
The model dynamics is driven by a pulsatile pressure acting and plotted nodes are not shared. Predicted steps: 43,000. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0). The tested distributed external loading is not in the training dataset

Finally, Fig. 27 shows the evolution of the $l^2$ error using Eq. (27), where we observe a trend similar to that reported in Fig. 11.

### 5.2 Coronary model

In this section, we extend the proposed computational framework to a realistic cardiovascular simulation. We adopt a patient-specific human left coronary artery model, which was investigated in previous FSI and UQ studies [47,48]. The model dynamics is driven by a pulsatile pressure acting on the vessel lumen, whose periodic waveform is illustrated in Fig. 28b. The pressure is gradually applied to the model through a linear ramp active during the first 1 second of the simulation (cf. Remark 1, $t_{end} = 1$s). It fluctuates from a systolic maximum of $1.6 \times 10^5$ baryes (120 mmHg) to a diastolic minimum of $1.067 \times 10^5$ baryes (80 mmHg) with a period of approximately 0.833 s (72 beats per minute), as per the normal systemic pressure and heart rate at rest of a healthy adult. Additionally, we consider an elastic modulus of $E = 6.26 \times 10^6$ dynes/cm$^2$, a density of the vascular tissue of $\rho = 1g/cm^3$ [27] and a Poisson ratio equals to $\nu = 0.4$.

Homogeneous Dirichlet boundary conditions are strongly enforced at the main inlet ($I_1$) and 6 outlets in the bottom
Fig. 26 Network prediction for the full cantilever model. Comparison of exact and predicted $z$-displacement solution at $t \sim 12.4s$.

Fig. 27 Time history of the $l^2$ error for the predicted solutions of the full cantilever model.

Fig. 28 Geometry, mesh and loading condition for the coronary artery model.

Fig. 29 Comparison of $x$-component displacement dynamics with and without artificial mass scaling. Plotted node: $(-13.7, 6.4, -9.6)$. The model is discretized using 373,435 tetrahedral elements with 250,659 degrees of freedom and is partitioned over 11 cores, where each core shares about 262 nodes with its neighbors. The original time step size calculated by equation (14) is about $2.02 \times 10^{-6}$, and we have increased it up to $5 \times 10^{-6}$ via artificial mass scaling (e.g. Algorithm 2), saving 60% of the computational time, but leading to a sensible increase in the total mass of the model equal to 21.13%. However, we verified in Fig. 29 that this increase corresponded to a marginal effect on the model dynamics. We also include a comparison test in Fig. 29 using the original time step size $2.02 \times 10^{-6}$, current size $5 \times 10^{-6}$ and a further slightly-increased size $5.1 \times 10^{-6}$, which corresponds to 24.7% of mass increment.

Note that a model of the coronary circulation offers an ideal benchmark for the proposed approach, since the small size of the coronary arteries leads to a lower bound in the explicit time step with respect to other anatomical regions.
To test our methodology, we run Algorithm 3 using the optimal hyperparameter combination as discussed in the previous sections, with an increased sample size $n_s = 1000$, and train 11 separate deep neural network surrogates. Figure 30 shows the predicted displacement dynamics at a non-shared node of a few partitions and confirms that the displacement evolution of the system is sufficiently learned even from the limited available data. Note that the discontinuity brought by the linear ramp condition is also correctly learned. Larger relative errors can be observed for nodes with small displacement amplitudes but the dominant displacement components appear to be accurately modeled.
6 Error control

In this section, we discuss metrics for error estimation that are naturally provided by the proposed approach. As mentioned above, provided the discretization error from the Galerkin method is neglected, then the only source of error comes from the predicted dynamics at the shared nodes. If we consider, for example, the coronary model in Fig. 28a, there are exactly two processors associated with each shared node, that ideally should provide identical predictions. However, in practice, these predictions may differ, providing a means for estimating the approximation error in Algorithm 3.

In Fig. 31, we illustrate the prediction variability for the same shared node produced by networks associated with two different partitions, following offline evaluation, as discussed in Sect. 5.1.2. It is shown that CPU6 produces errors smaller than CPU5 in most instances.

We then introduce the following criterion to quantify such variability globally, i.e., over all the shared nodes

\[ \epsilon_{\text{avg}}^{(t)} = \frac{1}{3 \cdot N_p} \sum_{i=1}^{N_p} \| \hat{d}^{(i)}_t - d^{(i)}_t \|_2. \]  

(33)

Therefore, \( \epsilon_{\text{avg}}^{(t)} \) represents a global average error per shared degree of freedom at time \( t \). Moreover, since there are exactly two processors associated with each shared node, we denote their arithmetic average as \( \epsilon_{\text{avg}}^{(t)} \). We also calculate the difference between two predictions for the same shared node by replacing the exact solution \( d^{(i)}_t \) in equation (33) with the solution predicted by the second network. The resulting quantity is denoted as \( \epsilon^{(t)}_2 \).

As shown in both Figs. 31 and 32, the error increases around regions of high curvature in the displacement response. It can also be observed that the discrepancy between two predictions at the same shared node is highly correlated with the displacement error and is greater in most cases. Moreover, it is easy to compute and therefore particularly appropriate for error monitoring.

We finally provide histograms in Fig. 33 to show the temporal average of such variability. We further define

\[ e_{l^2}^{(j)} = \frac{1}{3 \cdot n_T} \sum_{i=1}^{n_T} \| \hat{d}^{(i)}_{(j)} - d^{(i)}_{(j)} \|_2. \]  

(34)

to quantify this error at shared node \( j \), where \( n_T \) is the total number of predicted time steps. Again, we put a bar on top of \( e_{l^2}^{(j)} \) to indicate an average of two generic processors that share node \( j \), and denote \( \epsilon^{(j)}_2 \) for the difference in two predictions, at the same shared node \( j \).

From Fig. 33 and consistent with our discussion above, the time-averaged model differences provide an upper bound for the approximation error at the shared nodes, providing an effective error control mechanism for cases where the true numerical solution is not available.

7 Performance analysis

As presented in Algorithm 1, at each time iteration of the proposed distributed solver, the main computational tasks are (1) the evaluation of element-level quantities \( \mathbf{K}_e, f^{(n)}_e, \) \( f^{(n), \text{ext}}_e \), (2) matrix–vector product for displacement update and (3) displacement synchronization at the shared nodes. Application of the proposed synchronization-avoiding Algorithm 3 allows one to avoid most of the synchronization cost by leveraging pre-trained data-driven surrogates. To better quantify the cost of the above operations, we introduce several
The speedup factor $\bar{\zeta}$ denotes whether the data-driven surrogate is used to reduce the synchronization cost.

We are interested in two scenarios. The first considers a fixed number of cores and several models of increasing size. The second considers a fixed-size mesh partitioned over an increasing number of cores. We further analyze these cases with or without pre-assembling element-level quantities (see Remark 2). In this section, the model geometry consists of a unit cube, which amplifies synchronization costs compared to the slender cantilever model used in the previous sections.

### 7.1 Test with pre-assembly of element-level quantities

For linear structural problems in the small strain regime, the matrices $\mathbf{M}$, $\mathbf{K}$ in problem (17) are constants in time and therefore can be assembled only once before the time loop, stored and re-used. In such a case, the cost of displacement synchronization dominates over the relatively inexpensive matrix–vector product, making the proposed approach particularly appealing.

#### 7.1.1 Fixed number of partitions

We consider a series of pre-assembled explicit structural simulations with increasing mesh size, solved by an 8-core machine. Since the number of shared nodes for each partition increases with the mesh size, the amount of communication also increases. This is clearly shown for all cases in Fig. 34, as $\bar{\zeta}$ grows rapidly with respect to $\bar{\eta}_s$ and takes more than 97% of the total cost. In such a case our data-driven synchronization algorithm is particularly effective, resulting in a significant speed-up as shown in Figs. 35 and 36, since $\bar{\eta}_d$ is orders of magnitude less expensive than $\bar{\eta}_s$.

Generally speaking, as shown in Fig. 2, by fixing the network depth (number of encoder layers), the sequence length $n_p/n_f$, and the number of hidden units $n_H$, the model cost $\bar{\eta}_d$ only depends on the size of input $\mathbf{d}$, which essentially scales with $\bar{\eta}_d$, see Fig. 35.

#### 7.1.2 Fixed mesh size

In this section, we fix the mesh size and distribute it over an increasing number of processors. In Fig. 37, the synchronization cost remains approximately stable for up to 20 processors but increases sensibly with 40 and 80 processors. This relates to the different cost of communication...
in computational environments with shared memory rather than distributed memory. In other words, we need 2 and 4 24-core machines to realize the final two cases characterized by 40 and 80 processors, respectively.

A significant reduction in the computational costs is also achieved for this scenario, as shown in Figs. 38 and 39. On a single machine with shared memory \( (n_c = 8, 12, 20) \), the model cost slightly increases, while it significantly decreases after transition to a distributed memory architecture on multiple machines \( (n_c = 40, 80) \), as more memory becomes available. Finally, we see promising speed-up factors \( \xi \) shown in Fig. 39.

7.2 Test without pre-assembly of element-level quantities

For general nonlinear structural simulations, the element-level matrices and vectors need to be re-computed at each iteration in time, with a cost that dominates over the remaining components. Here, for simplicity, we mimic this scenario via Algorithm 1 but still solving a linear problem.

7.2.1 Fixed number of partitions

Without the pre-assembly, from Figs. 40 and 41, the majority of cost is occupied by the evaluation of element quantities. Again, due to load imbalance and the presence of a barriers in the code preceding inter-processor communication, the synchronization time also increases. Finally, for an increasing mesh size, the relative importance of the synchronization cost \( r_s' \) is reduced while \( r_e' \) increases, clearly reducing the performance of our surrogate model (e.g. see Fig. 42).

7.2.2 Fixed mesh size

For an increasing number of processors, from \( \bar{r}_s \) and \( \bar{r}_e \) in Fig. 43, we observe a more efficient evaluation for the
element quantities is accompanied by an increasing synchronization cost (Fig. 44). The speed-up in this case, shown in Fig. 45, is roughly monotonic with the increasing number of partitions.

In general, the performance of our surrogate model is limited by the dominant cost of element assembly, but can be improved with the techniques discussed in Sect. 1.

7.3 Effect of model geometry

Most examples we have tested in this paper consider a finite element model of a slender cantilever, where the number of shared degrees of freedom remains modest for each partition, as opposed to the cubic geometry we used above for algorithmic performance assessment. We now would like to provide a direct comparison between the speedup factor $\bar{\zeta}$ for these two geometries.

To make the comparison fair, we plot the speed-up factor $\bar{\zeta}$ with respect to the average element number per partition $N_e$ in Fig. 46. For higher mesh resolutions, the cube model holds a similar number of element per partition as the cantilever model while the shared degrees of freedom significantly increase. As a result, the cost for applying our surrogate model also increases, causing a decline in the speedup ratio $\bar{\zeta}$ for the scenario with pre-assembly, due to an increased sensitivity to the small denominator $\bar{t}'$. Without pre-assembly, when the cost of our data-driven model is insignificant comparing to the cost of element formation, after avoiding the synchronization cost, we have a slightly better speedup tendency for the model with cubic geometry. Finally, it is also worth mentioning that most cardiovascular simulations consist of slender geometries, which would therefore benefit from an increased runtime reduction.
7.4 Network training time

In this section we provide a brief discussion on how the selection of hyperparameters affect network training time. We first observe that all networks are trained independently on one or multiple GPUs, and therefore the total number of networks and therefore partitions has limited effect on the overall training time. Second, we compare network training times for three of the models analyzed in the previous sections, i.e., the cantilever models with coarse and fine meshes and the coronary model. For these models, the same total number of examples (and therefore the same number of mini-batches for a constant mini-batch size $n_B = 5$) is obtained by dividing the number of simulation steps by an appropriate $n_s$. For example, $n_s = 100$ is selected for the coarse cantilever mesh which has $10^5$ total simulation steps, while $n_s = 500$ for the fine cantilever mesh with $5 \times 10^5$ total steps. The training examples are then extracted as a subset of such examples.

Even though the average number of shared nodes $N_o$ differs wildly across models, the effect on the training time appears to be minimal, suggesting it is not sensitive to $N_o$ as the number of trainable network parameters does not scale proportionally to the number of shared nodes. This is promising, in view of using this technique for large cardiovascular models.

Further, we believe most of the training effort is used to increase the accuracy for negligible axial or out-of-plane displacement components. For example, in Fig. 14, we showed that training the network using twice the sample size (i.e., $n_s = 200$) can still yield an accurate representation of the dominant model response in the $z$-direction. With this in mind, we may also sacrifice some accuracy by tuning $\gamma$ (learning rate decay rate), $n_B$, $n_H$ and other hyperparameters to minimize the training cost, while preserving accuracy mostly on quantities associated with the dominant structural response. In this context, we re-trained the skewed forcing test case for the coarse cantilever model with hyperparameters $n_B = 10$, $n_H = 50$, $n_s = 200$, $\gamma = 0.998$ using only 6% of the original training time ($\sim 30$ min versus $\sim 5$ h). In the results shown in Fig. 47 the displacements in the $y$-$z$ plane remain comparable to those in Fig. 18 and the solution remains stable even on a long time horizon.

8 Conclusions and future work

In this paper, we have developed a novel data-driven approach to speed up the structural analysis of soft biological tissue. Our approach minimizes the amount inter-processor communication by replacing shared node synchronization tasks with predictions from an optimally trained artificial neural network. As verified through extensive numerical experiments, the LSTM encoder–decoder network model proposed in this study accurately approximates the displacement values at the shared nodes and maintains stability and accuracy even for long integration times.

The proposed network significantly reduces synchronization times in large scale simulations. This is demonstrated in Sect. 5.2 using a realistic coronary artery model used in previous studies for fluid–structure interaction problems. In addition, when modified to include conditional predictions and trained accordingly, our model is robust to the choice of initial condition and external loading. Note that this conditional structure can also be extended to material proprieties, e.g. predicting shared node dynamics under arbitrary elastic modulus, Poisson ratio and, in future work, constitutive laws.

Reduction of synchronization times is crucial for ensemble distributed finite element solvers where multiple realizations of geometry, boundary conditions or material properties
are solved simultaneously. An explicit-in-time multi-GPU implementation of an ensemble solver has been proposed in our previous work [34], where GPU-to-CPU synchronization represents a challenging computational bottleneck. Therefore, the proposed approach represents an interesting acceleration, particularly for the periodic response of cardiovascular models.

In addition, the well known poor bending performance of constant strain tetrahedral elements may significantly degrade the reliability of patient-specific cardiovascular structural models. To overcome this problem, we will explore recently proposed stabilized and variational multiscale finite element formulations [46]. Finally, future work will be devoted to apply the proposed approach in the context of non-linear problems.

Acknowledgements This work was supported by a NSF CAREER award #1942662 (PI DES), a NSF CDS&E award #2104083 (PI DES) and used computational resources provided through the Center for Research Computing at the University of Notre Dame. The authors would like to thank Prof. Zhiliang Xu and Prof. Guosheng Fu for their comments and suggestions that contributed to improve the quality of the present manuscript. The authors would also like to thank the anonymous reviewers for their insightful comments.

Declarations

Conflict of interest The authors declare that they have no conflicts of interest.

References

1. Aslam M, Riaz O, Mumtaz S, Asif AD (2020) Performance comparison of GPU-based Jacobi solvers using CUDA provided synchronization methods. IEEE Access 8:31792–31812

2. Bakarji J, Champion K, Kutz JN, Brunton SL (2022) Discovering governing equations from partial measurements with deep delay autoencoders. arXiv preprint arXiv:2201.05136

3. Bartezzaghi A, Cremonesi M, Parolini N, Perego U (2015) An explicit dynamics GPU structural solver for thin shell finite elements. Comput Struct 154:29–40

4. Belytschko T, Liu WK, Moran B, Elkhodary K (2014) Nonlinear finite elements for continua and structures. Wiley, Hoboken

5. Bhat U, Munch SB (2022) Recurrent neural networks for partially observed dynamical systems. Phys Rev E 105:044205

6. Brunton SL, Proctor JL, Kutz JN (2016) Discovering governing equations from measurements using Morales–Zublieke methods. arXiv preprint arXiv:1610.04566

7. Chang B, Chen MM, Haber E, Chi EH (2019) AntisymmetricRNN: a dynamical system view on recurrent neural networks

8. Chen Q (2020) MGMETIS: mesh & graph METIS partitioning. GitHub repository. https://github.com/chiao45/mgmetis

9. Chen Z, Churchill V, Wu KL, Xi DB (2022) Deep neural network modeling of unknown partial differential equations in nodal space. J Comput Phys 449:110782

10. Churchill V, Xi DB (2022) Deep learning of chaotic systems from partially-observed data. arXiv preprint arXiv:2205.08384

11. Clough RW, Penzien J (1993) Dynamics of structures. Civil engineering series. McGraw-Hill, New York

12. Fu XH, Chang LB, Xiu DB (2020) Learning reduced systems via deep neural networks with memory. J Mach Learn Res Comput 1(2):97

13. Funahashi K, Nakamura Y (1993) Approximation of dynamical systems by continuous time recurrent neural networks. Neural Netw 6(6):801–806

14. Garnelo M, Rosenbaum D, Maddison C, Ramalho T, Saxton D, Shanahan M, Teh YW, Rezende D, Eslami SMA (Jul 2018) Conditional neural processes. In: Proceedings of the 35th international conference on machine learning, volume 80 of Proceedings of Machine Learning Research, pp 1704–1713. PMLR, 10–15

15. Goodfellow I, Bengio Y, Courville A (2016) Deep learning. MIT Press, Cambridge

16. He KM, Zhang XY, Ren SQ, Sun J (2016) Deep residual learning for image recognition. In: Proceedings of the IEEE conference on computer vision and pattern recognition, pp 770–778

17. Hochreiter S, Schmidhuber J (1997) Long short-term memory. Neural Comput 9(8):1735–1780

18. Hu YH, Zhao T, Xu SX, Xu ZL, Lin LZ (2022) Neural-PDE: a RNN based neural network for solving time dependent PDEs. Commun Inf Syst

19. Hughes TJR (2012) The finite element method: linear static and dynamic finite element analysis. Dover Publications, INC., Mineola

20. Huthwaite P (2014) Accelerated finite element elastodynamic simulations using the GPU. J Comput Phys 257:687–707

21. Jolides GR, Wittek A, Miller K (2010) Real-time nonlinear finite element computations on GPU-application to neurosurgical simulation. Comput Methods Appl Mech Eng 199(49–52):3305–3314

22. Jung J, Jun H, Lee PS (2022) Self-updated four-node finite element using deep learning. Comput Mech 69(1):23–44

23. Jung J, Yoon K, Lee PS (2020) Deep learned finite elements. Comput Methods Appl Mech Eng 372:113401

24. Kaheman K, Kutz JN, Brunton SL (2020) SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics. Proc R Soc A Math Phys Eng Sci 476(2242):20200279

25. Kaiser E, Kutz JN, Brunton SL (2021) Data-driven discovery of Koopman Eigenfunctions for control. Mach Learn Sci Technol 2(3):035023

26. Karpathy George, Kumar Vinip (2009) MeTis: unstructured graph partitioning and sparse matrix ordering system, Version 4.0. http://www.cs.umn.edu/~metis

27. Kim HJ, Vignon-Clementel IE, Coogan JS, Figueroa CA, Jansen KE, Taylor CA (2010) Patient-specific modeling of blood flow and pressure in human coronary arteries. Ann Biomed Eng 38(10):3195–3209

28. Kingma DP, Ba J (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980

29. Komatisch D, Erlebacher G, Göddeke D, Michéa D (2010) High-order finite element computations on graphics processors. ACM Trans Parallel Comput 6(1):1–32

30. Kulowski L (2020) LSTM_encoder_decoder. GitHub repository: https://github.com/lkulowski/LSTM_encoder_decoder

31. Kutz JN, Brunton SL, Brunton BW, Proctor JL (2016) Dynamic mode decomposition: data-driven modeling of complex systems. SIAM

32. Kulkowski L (2020) LSTM_encoder_decoder. GitHub repository: https://github.com/lkulowski/LSTM_encoder_decoder

33. Kutz JN, Brunton SL, Brunton BW, Proctor JL (2016) Dynamic mode decomposition: data-driven modeling of complex systems. SIAM

34. Li X, Schiavazzi DE (2021) An ensemble solver for segregated cardiovascular FSI. Comput Mech 68(6):1421–1436
35. Lu L, Jin PZ, Pang GF, Zhang ZQ, Karniadakis GE (2021) Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nat Mach Intell 3(3):218–229
36. Malhotra P, Ramakrishnan A, Anand G, Vig L, Agarwal P, Shroff G (2016) LSTM-based encoder-decoder for multi-sensor anomaly detection
37. McCaslin SE, Shiakolas PS, Dennis BH, Lawrence KL (2012) Closed-form stiffness matrices for higher order tetrahedral finite elements. Adv Eng Softw 44(1):75–79
38. Olovsson L, Simonsson K, Unosson M (2005) Selective mass scaling for explicit finite element analyses. Int J Numer Meth Eng 63(10):1436–1445
39. Park SH, Kim BD, Kang CM, Chung CC, Choi JW (2018) Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture. In: 2018 IEEE intelligent vehicles symposium (IV). pp 1672–1678. IEEE
40. Partin L, Geraci G, Rushdi A, Eldred MS, Schiavazzi DE (2022) Multifidelity data fusion in convolutional encoder/decoder networks
41. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin ZM, Gimelshein N, Antiga L et al. (2019) Pytorch: an imperative style, high-performance deep learning library. Adv Neural Inf Process Syst 32:8026–8037
42. Qin T, Wu KL, Xiu DB (2019) Data driven governing equations approximation using deep neural networks. J Comput Phys 395:620–635
43. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
44. Sanchez-Gonzalez A, Godwin J, Pfaff T, Ying R, Leskovec J, Battaglia PW (2020) Learning to simulate complex physics with graph networks. In: Proceedings of the 37th international conference on machine learning. ICML’20. JMLR.org
45. Schillinger D, Evans JA, Frischmann F, Hiemstra RR, Hsu MC, Hughes TJR (2015) A collocated C0 finite element method: Reduced quadrature perspective, cost comparison with standard finite elements, and explicit structural dynamics. Int J Numer Meth Eng 102(3–4):576–631
46. Scovazzi G, Carnes B, Zeng X, Rossi S (2016) A simple, stable, and accurate linear tetrahedral finite element for transient, nearly, and fully incompressible solid dynamics: a dynamic variational multiscale approach. Int J Numer Meth Eng 106(10):799–839
47. Seo J, Schiavazzi DE, Kahn AM, Marsden AL (2020) The effects of clinically-derived parametric data uncertainty in patient-specific coronary simulations with deformable walls. Int J Numer Methods Biomed Eng 36(8):e3351
48. Seo J, Schiavazzi DE, Marsden AL (2019) Performance of preconditioned iterative linear solvers for cardiovascular simulations in rigid and deformable vessels. Comput Mech 64(3):717–739
49. Shea DE, Brunton SL, Kutz JN (2021) SINDy-BVP: sparse identification of nonlinear dynamics for boundary value problems. Phys Rev Res 3:023255
50. Sherstinsky A (2020) Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. Phys D 404:132306
51. Shiakolas PS, Nambiar RV, Lawrence KL, Rogers WA (1992) Closed-form stiffness matrices for the linear strain and quadratic strain tetrahedron finite elements. Comput Struct 45(2):237–242
52. Stoter SKF, Nguyen TH, Hiemstra RR, Schillinger D (2022) Variationally consistent mass scaling for explicit time-integration schemes of lower- and higher-order finite element methods
53. Strbac V, Pierce DM, Vander Sloten J, Famaey N (2017) GPGPU-based explicit finite element computations for applications in biomechanics: the performance of material models, element technologies, and hardware generations. Comput Methods Biomech Biomed Engin 20(16):1643–1657
54. Taylor ZA, Cheng M, Ourselin S (2008) High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. IEEE Trans Med Imaging 27(5):650–663
55. Vlachas PR, Byeon W, Wan ZY, Sapsis TP, Koumoutsakos P (2018) Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. Proc R Soc A Math Phys Eng Sci 474(2213):20170844
56. Wu KL, Xiu DB (2020) Data-driven deep learning of partial differential equations in modal space. J Comput Phys 408:109307
57. Zienkiewicz OC, Taylor RL, Too JM (1971) Reduced integration technique in general analysis of plates and shells. Int J Numer Meth Eng 3(2):275–290

Publisher’s Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations. Springer Nature or its licensor (e.g. a society or other partner) holds exclusive rights to this article under a publishing agreement with the author(s) or other rightsholder(s); author self-archiving of the accepted manuscript version of this article is solely governed by the terms of such publishing agreement and applicable law.