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

Guoxiang Grayson Tong and Daniele E. Schiavazzi

Department of Applied and Computational Mathematics and Statistics, University of Notre Dame, 102G Crowley Hall, Notre Dame, 46656, IN, United States.

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 (LSTM) 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 cost 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 Section 2.1, followed by the formulation for our distributed explicit solver in Section 2.2. The proposed data-driven method is discussed by first presenting a single LSTM cell in Section 3.1, followed by an encoder-decoder layout in Section 3.2, with the details of network training summarized in Section 3.2.1. A data-driven synchronization-avoiding algorithm is proposed in Section 4. Numerical tests are performed in Section 5, starting with extensive benchmarks on a simple cantilever problem in Section 5.1, and a realistic coronary model in Section 5.2. In Section 6, we discuss a few criteria to measure the prediction error of our proposed framework. Performance is assessed in Section 7 for a fixed number of cores and fixed mesh size, respectively, while Section 8 contains the conclusions and addresses possible future research directions.

2 Governing equations and discretization

2.1 Equations of linear elastodynamics

Strong form - Consider the following initial-boundary value problem $(\mathcal{S})$ 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])

$$(\mathcal{S}): \begin{cases} \text{Given } f_i, g_i, h_i, d_{0i}, \dot{d}_{0i}, \\ \text{find } d_i, \ i,j = \{x,y,z\}, \ s.t. \\ \rho\ddot{d}_i = \dfrac{\partial\sigma_{ij}}{\partial x_j} + f_i & \text{in } \Omega \times (0,T] \\ d_i = g_i & \text{on } \partial\Omega_d \times (0,T] \\ \sigma_{ij}n_j = h_i & \text{on } \partial\Omega_n \times (0,T] \\ d_i = d_{0i} & \text{in } \Omega, \text{ at } t=0 \\ \dot{d}_i = \dot{d}_{0i} & \text{in } \Omega, \text{ at } t=0, \end{cases} \quad (1)$$

where the quantity of interest $d_i(\boldsymbol{x},t) : \bar{\Omega} \times [0,T] \to \mathbb{R}$ denotes the i-th component of the displacement field $\boldsymbol{d}$, ρ is the material density, $f_i(\boldsymbol{x},t) : \Omega \times (0,T] \to \mathbb{R}$ is the i-th component of the body force, $g_i(\boldsymbol{x},t) : \partial\Omega_d \times (0,T] \to \mathbb{R}$, $h_i(\boldsymbol{x},t) : \partial\Omega_n \times (0,T] \to \mathbb{R}$ 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 $d_{0i}, \dot{d}_{0i}$ 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

$$\sigma_{ij} = 2\mu\epsilon_{ij} + \lambda\delta_{ij}\epsilon_{kk}, \ i,j,k = \{x,y,z\}, \quad (2)$$

where $\boldsymbol{\sigma} = [\sigma_{ij}]$ is the Cauchy stress tensor, $\boldsymbol{\epsilon} = [\epsilon_{ij}]$ is the infinitesimal strain tensor, δ_{ij} is the Kronecker delta, and the Lamé coefficients μ and λ are defined as

$$\mu = \frac{E}{2(1+\nu)}; \quad \lambda = \frac{E\nu}{(1+\nu)(1-2\nu)}, \quad (3)$$

where E is the material Young's modulus, ν is the Poisson ratio.

The infinitesimal strain $\boldsymbol{\epsilon}$ is obtained from the symmetric part of the displacement gradient tensor

$$\epsilon_{ij} = d_{(i,j)} = \frac{1}{2}\left(\frac{\partial d_i}{\partial x_j} + \frac{\partial d_j}{\partial x_i}\right), \quad (4)$$

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

$$\sigma_{ij} = c_{ijkl}\epsilon_{kl} = c_{ijkl}d_{(k,l)}, \quad (5)$$

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

$$c_{ijkl} = \mu(\delta_{ik}\delta_{jl} + \delta_{il}\delta_{jk}) + \lambda\delta_{ij}\delta_{kl}. \quad (6)$$

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

$$(\mathcal{W}): \begin{cases} \text{Given } \boldsymbol{f},\boldsymbol{g},\boldsymbol{h},\boldsymbol{d}_0,\dot{\boldsymbol{d}}_0, \text{ find } \boldsymbol{d}(\boldsymbol{x},t) \in \mathscr{D}^t, \\ \text{that for any } \boldsymbol{w} \in \mathscr{W}, \ s.t. \\ (\rho\ddot{\boldsymbol{d}},\boldsymbol{w})_\Omega + a(\boldsymbol{d},\boldsymbol{w})_\Omega = l(\boldsymbol{w})_\Omega + l_n(\boldsymbol{w})_{\partial\Omega_n} \\ (\rho\boldsymbol{d}(\boldsymbol{x},0),\boldsymbol{w})_\Omega = (\rho\boldsymbol{d}_0,\boldsymbol{w})_\Omega \\ (\rho\dot{\boldsymbol{d}}(\boldsymbol{x},0),\boldsymbol{w})_\Omega = (\rho\dot{\boldsymbol{d}}_0,\boldsymbol{w})_\Omega \end{cases} \quad (7)$$

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 $a(\cdot,\cdot)_\Omega$ are defined, respectively, as

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

$$l(\boldsymbol{w})_\Omega = \int_\Omega w_i f_i d\Omega \quad (8)$$

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

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

$$\mathscr{D}^t := \Big\{\boldsymbol{d}(\boldsymbol{x},t)|d_i(\boldsymbol{x},t) \in H^1(\Omega); \ d_i(\boldsymbol{x},t) = g_i(\boldsymbol{x},t),$$
$$\forall \boldsymbol{x} \in \partial\Omega_d; \ t \in (0,T]\Big\}$$
$$\mathscr{W} := \Big\{\boldsymbol{w}(\boldsymbol{x})|w_i(\boldsymbol{x}) \in H^1(\Omega); \ w_i(\boldsymbol{x}) = 0, \ \forall \boldsymbol{x} \in \partial\Omega_d\Big\}, \quad (9)$$

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

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

$$(\mathcal{M}): \begin{cases} \text{Find } \boldsymbol{d}_h^{(n)} \in \mathscr{D}_h^{(n)}, \ n \in \{1,2,\cdots,n_T\}, s.t. \\ \mathbf{M}\ddot{\boldsymbol{d}}_h^{(n)} + \mathbf{C}\dot{\boldsymbol{d}}_h^{(n)} + \mathbf{K}\boldsymbol{d}_h^{(n)} = \boldsymbol{f}^{(n)} \end{cases}$$
(10)

$$\begin{aligned} \mathscr{D}_h^{(n)} :=& \Big\{ \boldsymbol{d}_h^{(n)}(\boldsymbol{x}) | d_{h,i}^{(n)}(\boldsymbol{x}) \in C^0(\Omega_h); \\ & d_{h,i}^{(n)}(\boldsymbol{x}) = g_{h,i}^{(n)}(\boldsymbol{x}), \forall \boldsymbol{x} \in \partial\Omega_h; \\ & d_{h,i}^{(n)}(\boldsymbol{x})|_e \in \mathcal{P}^1(\Omega_{h,e}); \ n \in \{1,2,\cdots,n_T\} \Big\} \\ \mathscr{W}_h :=& \Big\{ \boldsymbol{w}_h(\boldsymbol{x}) | w_i(\boldsymbol{x}) \in C^0(\Omega_h); \ w_{hi}(\boldsymbol{x}) = 0, \\ & \forall \boldsymbol{x} \in \partial\Omega_h; \ w_{hi}(\boldsymbol{x})|_e \in \mathcal{P}^1(\Omega_{h,e}) \Big\}, \end{aligned}$$
(11)

where the subscripts $(\cdot)|_e$, $(\cdot)_e$ indicate restriction to a single finite element and n_T is the total number of time steps. The quantities $\mathbf{M}, \mathbf{C}, \mathbf{K}, \boldsymbol{f}$ 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., $\mathbf{C} = \alpha\,\mathbf{M}$ with damping factor $\alpha \in \mathbb{R}$ (see, e.g., [19]).

2.2 A distributed explicit structural finite element solver

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

$$\begin{aligned} \boldsymbol{a}_h^{(n)} &= \ddot{\boldsymbol{d}}_h^{(n)} = \frac{\boldsymbol{d}_h^{(n+1)} - 2\boldsymbol{d}_h^{(n)} + \boldsymbol{d}_h^{(n-1)}}{\Delta t^2} + \mathcal{O}(\Delta t^3), \\ \boldsymbol{v}_h^{(n)} &= \dot{\boldsymbol{d}}_h^{(n)} = \frac{\boldsymbol{d}_h^{(n+1)} - \boldsymbol{d}_h^{(n-1)}}{2\Delta t} + \mathcal{O}(\Delta t^3). \end{aligned}$$
(12)

Consistent mass and damping matrices $\mathbf{M}$ and $\mathbf{C}$ are replaced by their lumped counterparts $\mathring{\mathbf{M}}$ and $\mathring{\mathbf{C}}$, leading to trivial inversion for fully explicit displacement-based time integrators. To initiate the time iterations, the quantity $\boldsymbol{d}_h^{(-1)}$ is computed using a second order Taylor approximation of $\boldsymbol{d}(-\Delta t)$, for consistency. Once the initial conditions $\boldsymbol{d}_{h,0}$, $\boldsymbol{v}_{h,0}$ are provided for the displacement and velocity at time $t = 0$, an initial acceleration $\boldsymbol{a}_{h,0}$ is computed by solving the following discrete system (note the use of consistent matrices)

$$\mathbf{M}\,\boldsymbol{a}_{h,0} + \mathbf{C}\,\boldsymbol{v}_{h,0} + \mathbf{K}\,\boldsymbol{d}_{h,0} = \boldsymbol{f}^{(0)}.$$
(13)

The explicit scheme is known to be conditionally stable with respect to the choice of Δt, consistent with the well known Courant-Friedrichs-Lewy (CFL) condition (see, e.g., [4])

$$\Delta t = \alpha_s \frac{\min\limits_{e=1,2,\cdots n_e} h_e}{\sqrt{\frac{E}{\rho(1-\nu^2)}}},$$
(14)

where h_e 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 $\boldsymbol{f}^{(n),\text{int}} = \mathbf{K}\boldsymbol{d}_h^{(n-1)}$ and renaming $\boldsymbol{f}^{(n)}$ as $\boldsymbol{f}^{(n),\text{ext}}$, 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* is applied to the external force, through a time-dependent function $\mathscr{R}(t, t_{\text{end}})$, such that

$$\tilde{\boldsymbol{f}}^{(n)}(t_{\text{end}}) = \boldsymbol{f}^{(n)}\mathscr{R}(t_n, t_{\text{end}}).$$
(15)

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

$$\mathscr{R}(t_n, t_{\text{end}}) = \begin{cases} t_n/t_{\text{end}} & t_n \le t_{\text{end}} \\ 1 & \text{otherwise.} \end{cases}$$
(16)

$$(\tilde{\mathcal{M}}): \begin{cases} \text{Given initial conditions: } \boldsymbol{d}_{h,0}, \boldsymbol{v}_{h,0}, \boldsymbol{a}_{h,0} \text{ and time step size } \Delta t \\ \text{Find } \boldsymbol{d}_h^{(n)} \in \mathscr{D}_h^{(n)}, \text{ for every discrete step } n \in \{1, 2, \cdots, n_T\} \;\; s.t. \\ \boldsymbol{d}_h^{(n)} \;\; = (\mathring{\mathbf{M}} + \frac{\Delta t}{2}\mathring{\mathbf{C}})^{-1} \Big[\Delta t^2 \boldsymbol{f}^{(n)} - (\Delta t^2 \mathbf{K} - 2\mathring{\mathbf{M}})\boldsymbol{d}_h^{(n-1)} - (\mathring{\mathbf{M}} - \frac{\Delta t}{2}\mathring{\mathbf{C}})\boldsymbol{d}_h^{(n-2)} \Big] \\ \qquad\;\; = (\mathring{\mathbf{M}} + \frac{\Delta t}{2}\mathring{\mathbf{C}})^{-1} \Big[\Delta t^2 (\boldsymbol{f}^{(n),\text{ext}} - \boldsymbol{f}^{(n),\text{int}}) + 2\mathring{\mathbf{M}}\boldsymbol{d}_h^{(n-1)} - (\mathring{\mathbf{M}} - \frac{\Delta t}{2}\mathring{\mathbf{C}})\boldsymbol{d}_h^{(n-2)} \Big] \\ \boldsymbol{d}_h^{(0)} \;\; = \boldsymbol{d}_{h,0} \\ \boldsymbol{d}_h^{(-1)} = \boldsymbol{d}_{h,0} - \Delta t \boldsymbol{v}_{h,0} + \frac{\Delta t^2}{2}\boldsymbol{a}_{h,0} \end{cases} \tag{17}$$

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 $i = 1, \cdots, n_c$, 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 $\mathbf{K}_e$ and external loading $\boldsymbol{f}_e^{(n),\text{ext}}$ 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".

Artificial mass scaling - Explicit time integration schemes are stable under condition (14) on Δt, 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 Δt, 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 β 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 Section 5.2.

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

---

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 $\mathring{\mathbf{M}}$, $\mathring{\mathbf{C}}$ to each processor

for $n = 1, 2, \cdots, n_T$ **do**   ▷ Time loop

    Initialize local internal and external forces $\boldsymbol{f}^{(n),\text{int}}, \boldsymbol{f}^{(n),\text{ext}}$ as zero vectors

    for $e = 1, 2, \cdots, n_{e,[i]}$ **do**   ▷ Element loop

        Form element stiffness matrix $\mathbf{K}_e$ and external force $\boldsymbol{f}_e^{(n),\text{ext}}$

        Calculate element internal force: $\boldsymbol{f}_e^{(n),\text{int}} = \mathbf{K}_e \boldsymbol{d}_{h,e}^{(n-1)}$

        Update local forces $\boldsymbol{f}^{(n),\text{int}}, \boldsymbol{f}^{(n),\text{ext}}$ by $\boldsymbol{f}_e^{(n),\text{int}}$ and $\boldsymbol{f}_e^{(n),\text{ext}}$ based on global element label e

    end for

    | Send local forces $\boldsymbol{f}^{(n),\text{int}}, \boldsymbol{f}^{(n),\text{ext}}$ to the root processor |

    | Update local forces $\boldsymbol{f}^{(n),\text{int}}, \boldsymbol{f}^{(n),\text{ext}}$ based on contributions from the shared mesh nodes |

    | Send updated local forces $\boldsymbol{f}^{(n),\text{int}}, \boldsymbol{f}^{(n),\text{ext}}$ back to each processor |

    Update local solution $\boldsymbol{d}_h^{(n)}$ using (17)

    Strongly enforce Dirichlet boundary conditions

end for

Algorithm 2 Artificial mass scaling.

Loop through all elements to determine Δt by equation (14)

Set a target time step $\widehat{\Delta t} = \beta \Delta t$, $\beta > 1$

Initialize the artificial density vector $\widehat{\boldsymbol{\rho}}$

for $e = 1, 2, \cdots, n_e$ **do**   ▷ Element loop

    Calculate the element time step size $\Delta t_e = \alpha_s h_e / \sqrt{\frac{E}{\rho(1-\nu^2)}}$

    if $\Delta t_e < \widehat{\Delta t}$ **then**

        $\widehat{\boldsymbol{\rho}}[i] = E\widehat{\Delta t}^2 / \alpha_s^2 h_e^2 (1 - \nu^2)$

    else

        $\widehat{\boldsymbol{\rho}}[i] = \rho$

    end if

    Recompute the mass matrix $\widehat{\mathbf{M}}$ based on $\widehat{\boldsymbol{\rho}}$

end for

Compute the total percent mass increase m_r

---

addition, we also introduce the notation

$$\|\boldsymbol{v}\|_2 = \sqrt{\sum_{i=1}^{n} |v_i|^2}, \quad \boldsymbol{v} \in \mathbb{R}^n$$

$$\|\boldsymbol{V}\|_F = \sqrt{\sum_{i=1}^{n}\sum_{j=1}^{m} |V_{ij}|^2}, \quad \boldsymbol{V} \in \mathbb{R}^{n \times m}.$$

3.1 LSTM cell model

As illustrated in Figure 1, a classical LSTM *cell* consists of a hidden state $\boldsymbol{h}$, a cell state $\boldsymbol{c}$, an input $\boldsymbol{d}$, the activation functions $\sigma_1, \cdots, \sigma_5$ and the component-wise operations $\otimes$ and $\oplus$. In each cell, the feedback control feature of the deep recurrent neural network is further reinforced by bringing a number of gated flow controls.
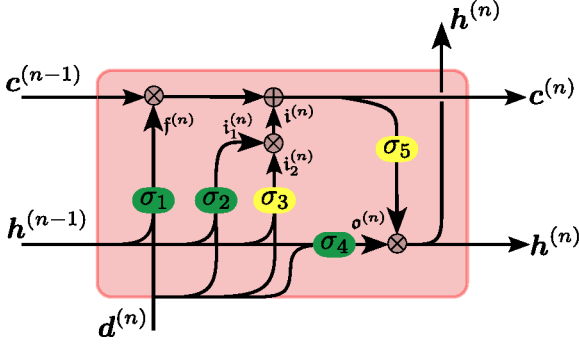
Fig. 1: Schematic representation of LSTM cell.

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 σ_1 is usually a sigmoid function, which provides an output range from 0 to 1. The forget gate implements the expression

$$f^{(n)} = \sigma_1(W_{f,d}d^{(n)} + W_{f,h}h^{(n-1)} + b_f). \quad (18)$$

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

$$i_1^{(n)} = \sigma_2(W_{i_1,d}d^{(n)} + W_{i_1,h}h^{(n-1)} + b_{i_1})$$
$$i_2^{(n)} = \sigma_3(W_{i_2,d}d^{(n)} + W_{i_2,h}h^{(n-1)} + b_{i_2}) \quad (19)$$
$$i^{(n)} = i_1^{(n)} \otimes i_2^{(n)},$$

where σ_2, σ_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

$$c^{(n)} = c^{(n-1)} \otimes f^{(n)} \oplus i^{(n)}, \quad (20)$$

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

$$o^{(n)} = \sigma_4(W_{o,d}d^{(n)} + W_{o,h}h^{(n-1)} + b_o)$$
$$h^{(n)} = \sigma_5(c^{(n)}) \otimes o^{(n)}, \quad (21)$$

where σ_4, σ_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 Figure 2. It is the combination of a k-layer bi-directional 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 $\widehat{Y}$ of length n_f

$$\widehat{Y} = \left(\mathcal{N}_{D,\Theta_D} \circ \mathcal{N}_{E,\Theta_E}\right)X = \mathcal{N}_\Theta X \quad (22)$$
$$X = \left[d^{(1)}, d^{(2)}, \cdots, d^{(n_p)}\right] \in \mathbb{R}^{n_{\text{dof}} \times n_p}$$
$$\widehat{Y} = \left[\widehat{d}^{(n_p+1)}, \widehat{d}^{(n_p+2)}, \cdots, \widehat{d}^{(n_p+n_f)}\right] \in \mathbb{R}^{n_{\text{dof}} \times n_f}. \quad (23)$$

The encoder and decoder models are expressed as $\mathcal{N}_{E,\Theta_E}$, $\mathcal{N}_{D,\Theta_D}$ respectively, and n_{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 $\mathcal{N}_\Theta$, parameterized by $\Theta = \Theta_D \cup \Theta_E$.

In the encoder model, each displacement solution $d^{(i)}, i = 1, \cdots n_p$ of input sequence X is fed to
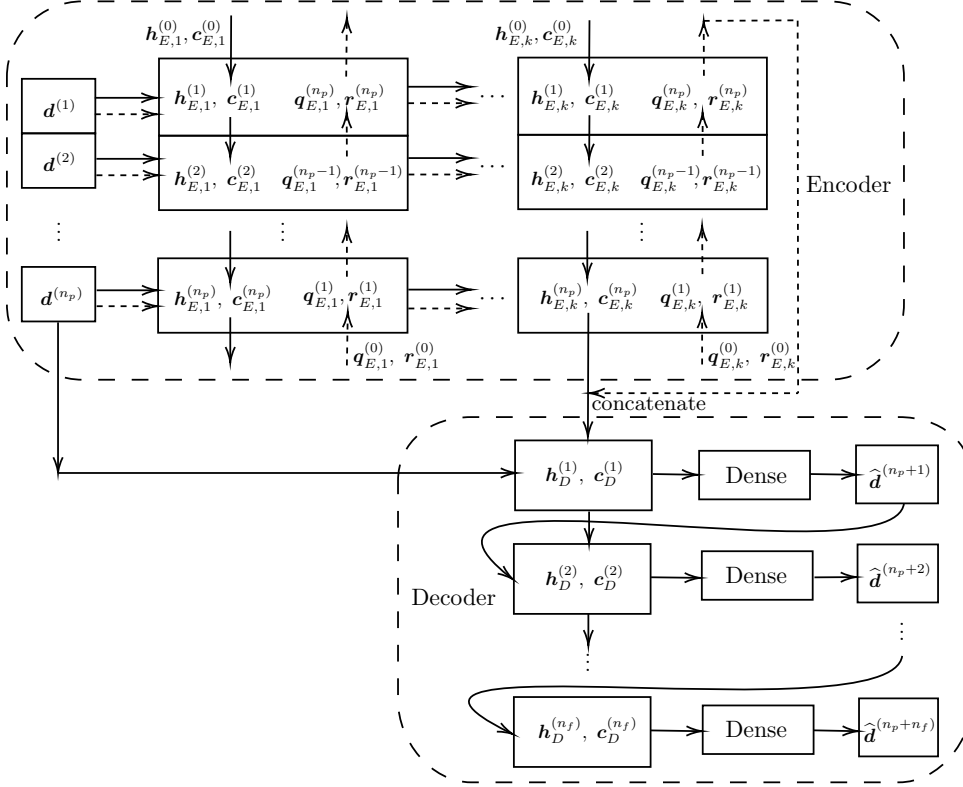
Fig. 2: Schematic representation of the LSTM encoder-decoder model. The Encoder is a stacked bi-directional LSTM network. The Decoder is a single layer uni-directional LSTM network with dense output layers.

a LSTM cell (Figure 1) while having hidden and cell states coming from two opposite directions. We use expressions $\boldsymbol{h}_{E,j}^{(i)}$ and $\boldsymbol{c}_{E,j}^{(i)}$ to represent the hidden and cell states in the direction $1 \rightarrow n_p$ (solid arrow) and $\boldsymbol{q}_{E,j}^{(i)}$ and $\boldsymbol{r}_{E,j}^{(i)}$ along $n_p \rightarrow 1$ (dashed arrow), where $j = 1, \cdots, k$ 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 $\boldsymbol{h}_D^{(0)}$ and $\boldsymbol{c}_D^{(0)}$.

The decoder then receives the final item $\boldsymbol{d}^{(n_p)}$ in $\boldsymbol{X}$ and recursively produce the predictions $\widehat{\boldsymbol{d}}^{(n_p+j)}$, $j = 1, \cdots, n_f$ in $\widehat{\boldsymbol{Y}}$. 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 $\widehat{\boldsymbol{Y}}$ and true numerical solution $\boldsymbol{Y}$ as

$$\mathcal{L}(\boldsymbol{Y}, \widehat{\boldsymbol{Y}}) = \frac{1}{n_f \cdot n_{\text{dof}}} \sum_{j=1}^{n_f} \|\boldsymbol{Y}_j - \widehat{\boldsymbol{Y}}_j\|_2^2$$

$$= \frac{1}{n_f \cdot n_{\text{dof}}} \sum_{j=1}^{n_f} \sum_{i=1}^{n_{\text{dof}}} |Y_{ij} - \widehat{Y}_{ij}|^2 \qquad (24)$$

$$= \frac{1}{n_f \cdot n_{\text{dof}}} \|\boldsymbol{Y} - \widehat{\boldsymbol{Y}}\|_F^2,$$

and perform gradient-based updates for the trainable parameters $\boldsymbol{\Theta}$.

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 Δt is usually in the range $1 \times 10^{-6} \sim 1 \times 10^{-3}$ 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 Section 5.1.3).

This is accomplished through a so-called *sample-refill* strategy during the training and evaluation stages, respectively. First, as illustrated in Figure 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,
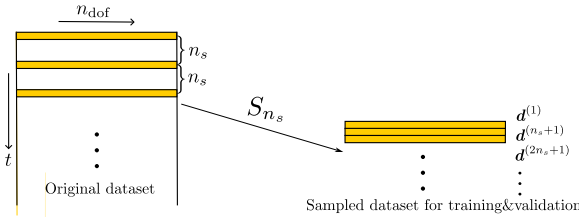
Fig. 3: The sample pre-processing step.

batch and feed into the deep neural network at training and validation stages. Upon successful training, as described in Figure 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 Figure 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 Figure 2, producing a set of optimally trained network models $\mathcal{N}_{\Theta,[j]}, j = \{1, 2, \cdots, n_c\}$. 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 Figure 5 for $n_c = 2$, highlighting the training, validation and prediction stages. We further distinguish between an *Offline* and an *Online* prediction stage in Section 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 Section 3.2.1) is realized by indexing every intermediate steps as lists $\mathbf{N}_p^i$ and $\mathbf{N}_f^i$ 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.
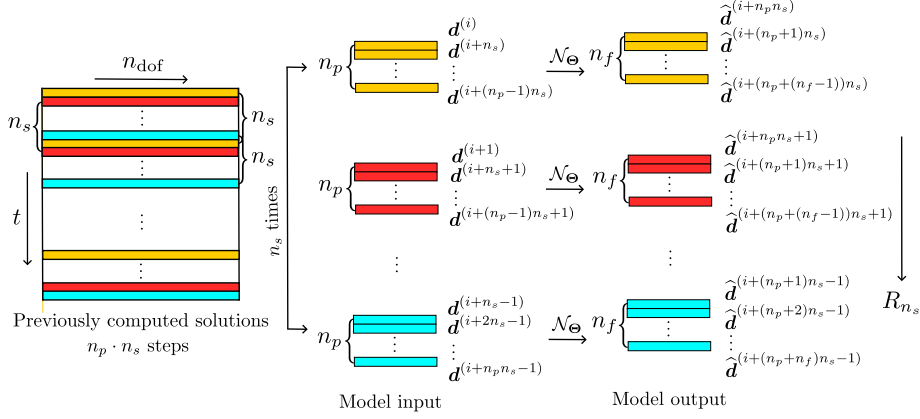
Fig. 4: Schematic illustration for the refill task. The whole operation starts at step i and requires $n_p \cdot n_s$ steps of previously computed solutions to produce $n_f \cdot n_s$ predictions

.

Algorithm 3 A synchronization-avoiding algorithm for distributed linear elastodynamics.

Step 1:   Partition the mesh over n_c processors and compute Δt

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_{cri}, n_p, n_f, n_s

while $t \leq T$ **do**

    if $n \leq n_{\mathrm{cri}}$ **then**

        Apply Algorithm 1 to compute $\boldsymbol{d}^{(n)}$

        $t = t + \Delta t$

        $n = n + 1$

    else

        for $i = 1, 2 \cdots, n_s$ **do**   ▷ refill step

            $\mathbf{N}_p^i = [i + n - n_p n_s - 1 : n_s : i + n - n_s - 1]$ ▷ Index set of NN model input

            $\mathbf{N}_f^i = [i + n - 1 : n_s : i + n + n_f n_s - n_s - 1]$ ▷ Index set of NN model output

            Forward pre-trained NN model: $\boldsymbol{d}^{(\mathbf{N}_p^i)}[\text{shared}] \xrightarrow{\mathcal{N}_{\boldsymbol{\Theta},[j]}} \widehat{\boldsymbol{d}}^{(\mathbf{N}_f^i)}[\text{shared}]$

        end for

        Gather all predictions on shared nodes: $\widehat{\boldsymbol{d}}^{(\mathbf{N}_f)}[\text{shared}] = \{\widehat{\boldsymbol{d}}^{(\mathbf{N}_f^i)}[\text{shared}], i = 1, 2, \cdots, n_s\}$

        for $m = n, n + 1, \cdots, n + n_s n_f - 1$ **do**

            Apply Algorithm 1 to compute $\boldsymbol{d}^{(m)}$ **without** "boxed" steps (synchronization)

            Update $\boldsymbol{d}^{(m)}$ at shared nodes by the corresponding modeled values in $\widehat{\boldsymbol{d}}^{(\mathbf{N}_f)}[\text{shared}]$

        end for

        Impose Dirichlet boundary conditions to $\boldsymbol{d}^{(\mathbf{N}_f)}$

        $t = t + n_s n_f \Delta t$

        $n = n + n_s n_f$

    end if

end while

---

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.
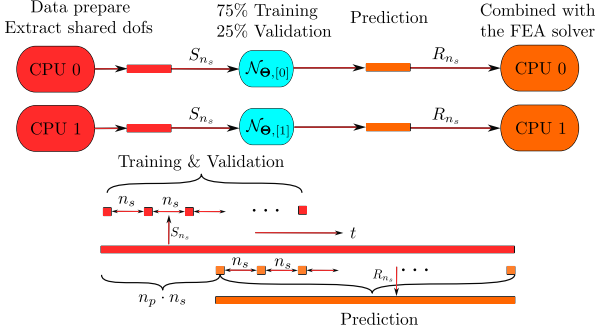
Fig. 5: Top: Flow chart of the data-driven synchronization-avoiding method. Bottom: Training-validation-testing dataset decomposition and sample-refill operation.

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

The time step size is set to 2.48×10^{-4} via equation (14). The mesh partitioning is realized by `mgmetis` [8] based on the `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 Figure 7.
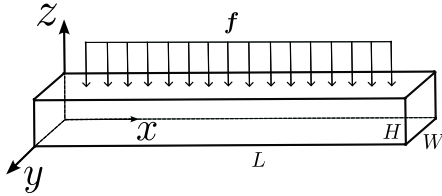
Fig. 6: Simple cantilever beam model.

5.1.1 Hyperparameters and network training

In our numerical experiments, we optimize over a number of selected hyperparameter realizations

Length (L)	25 (cm)
Width (W)	1 (cm)
Height (H)	1 (cm)
Young's modulus (E)	1×10^6 (dynes/cm^2)
Density (ρ)	1 (g/cm^3)
Poisson's ratio (ν)	0.3

Table 1: Geometric and material parameters for the cantilever beam model.
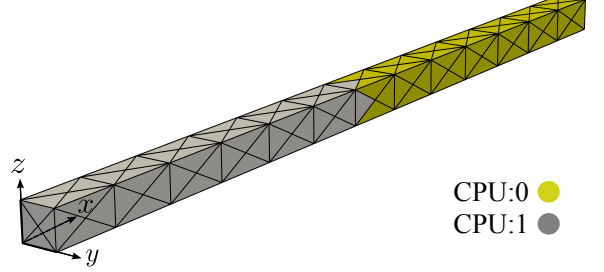
Fig. 7: Coarse mesh partitioned over 2 CPUs, labelled 0 and 1, respectively.

by performing a *grid search* on the mini-batch size n_B, the hidden unit size n_H and the initial learning rate η_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 η_0, we set a minimum learning rate $\eta_{\min} = 5 \times 10^{-7}$ and keep the decay rate fixed at $\gamma = 0.9995$. As a result, the total number of epochs n_{epoch} can be calculated as:

$$n_{\text{epoch}} = \lfloor \log_\gamma (\frac{\eta_{\min}}{\eta_0}) \rfloor. \qquad (25)$$

The selected grid of hyperparameter realizations includes $n_B = 5, 10, 20, 50$, $n_H = 20, 50, 100$ and $\eta_0 = 5 \times 10^{-3}$ ($n_{\text{epoch}} = 18416$), 5×10^{-4} ($n_{\text{epoch}} = 13812$), 5×10^{-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 η_0, we introduce the following criterion, rather than simply looking at the loss curves produced by MSE (24) during training and

validation

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

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

$$\left. + \sum_{j=i+n_f}^{i+2n_f-1} \|\boldsymbol{d}^{(j)} - \widehat{\widehat{\boldsymbol{d}}}^{(j)}\|_2^2 + \cdots \right),$$

that is, after the network is trained, starting from step i, we go ahead and use the trained network model $\mathcal{N}_{\boldsymbol{\Theta}}$ for N times. This is referred to as the *testing* phase. Then, by definition, E_{mse} quantifies a MSE error of displacement predictions on all shared nodes over $N \cdot n_f$ steps (note that a lag of n_s time steps is still present between a collection of n_f successive shared nodes predictions in the testing phase, and this gap is only "refilled" at the offline and online prediction stages (see Section 5.1.2.) Note that during the calculation of E_{mse}, after the first time ($N = 1$), we no longer have exact inputs. In other words, $\widehat{\widehat{\boldsymbol{d}}}$ is the model output ($N = 2$) with model input $\widehat{\boldsymbol{d}}$, predicted from the last step. Here we abuse the notation $\widehat{(\cdot)}$ to avoid stacking multiple "hats". Trivially, (26) collapses to (24) when $N = 1$.

Using E_{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 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-recurrent 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_{mse} as a measure of accuracy per degree of freedom and per prediction. All the results are reported in Figure 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 Figure 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 "$\triangle$" in Figure 8c) is finally selected as $\mathbf{n_B = 5, n_H = 100, \eta_0 = 5 \times 10^{-4}}$, 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 1×10^{-5}, 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 section 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 Δt, problem type, etc., where larger n_s are associated with increasing cost savings during training. In the experiments shown in Figure 8, these parameters are fixed as $n_p = n_f = 20$, $n_{ts} = 50\%, n_s = 100$.

Further, our training is performed through the `Adam` 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 Section 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,
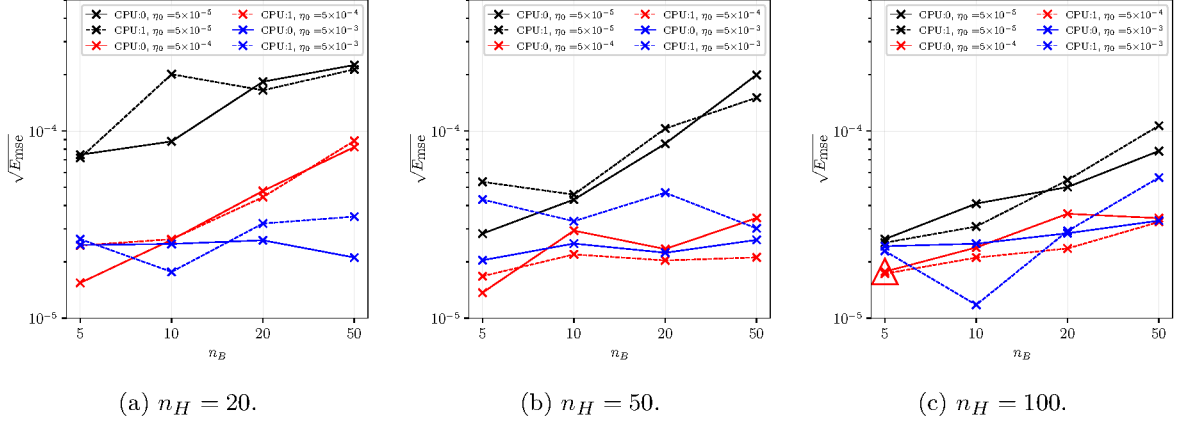
(a) $n_H = 20$.   (b) $n_H = 50$.   (c) $n_H = 100$.

Fig. 8: Values of $\sqrt{E_{\mathrm{mse}}}$ from grid search on mini-batch size n_B, hidden unit size n_H and initial learning rate η_0. The optimal hyperparameter combination is selected as $n_B = 5$, $n_H = 100$, $\eta_0 = 5 \times 10^{-4}$, marked by "△".

which differs from the previous testing stage, as missing predictions are refilled. Offline prediction performance using the hyperparameters determined in Section 5.1.1 is shown in Figure 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 seconds on the lateral x and y components.

A vertical line in each subplot of Figure 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* modelled 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,
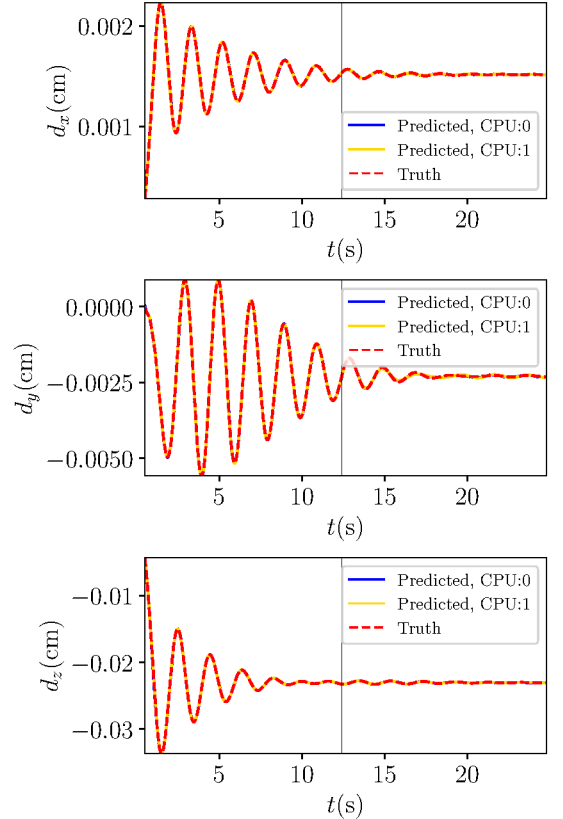
Fig. 9: Offline shared node displacement predictions for the cantilever model. Predicted steps: 98000. Node coordinate: (11.54, 0, 1).

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 propagate from share nodes to other nodes during each explicit update, due to a non-diagonal stiffness matrix $\mathbf{K}$.

In Figure 10 we plot the online predicted displacement dynamics at two non-shared nodes, one for each partition. The predictions appear accurate and stable such that the previously mentioned error propagated from shared nodes to non-shared nodes does not grow unbounded. A l^2 error is shown in Figure 11 with its definition at time t as

$$e_{l^2}^{(t)} = \|\boldsymbol{d}^{(t)} - \widehat{\boldsymbol{d}}^{(t)}\|_2. \tag{27}$$

From Figure 11, we observe an initial increase in the error once we start to use the trained surrogate model. As time evolves, e_{l^2} gradually reduces before the vertical bar due to the increasingly smaller oscillation amplitude. After the vertical bar, e_{l^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 Figure 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 Figure 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 Figure 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 (see Figure 13), accuracy improves for longer training periods and approximately three cantilever oscillations are needed to accurately learn the damped response in the z-direction.

Sample size - In this section, we explore the performance of our approach for varying sample size n_s. The results are presented in Figure 14, where a reduced accuracy is observed, with respect to $n_s = 100$, for sparser ($n_s = 200$) or denser ($n_s = 50, 10$) training datasets.

Another side effect of a smaller n_s is the proliferation of training examples, which reduces the training efficiency. As shown in Figure 14, a larger sample size $n_s = 200$ could still produce stable and accurate predictions for the (dominant) z-component response. Therefore, in general, we prefer to select n_s as large as possible.

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 Figure 15.

Refined mesh with additional processors - Next, we extend our data-driven framework to a
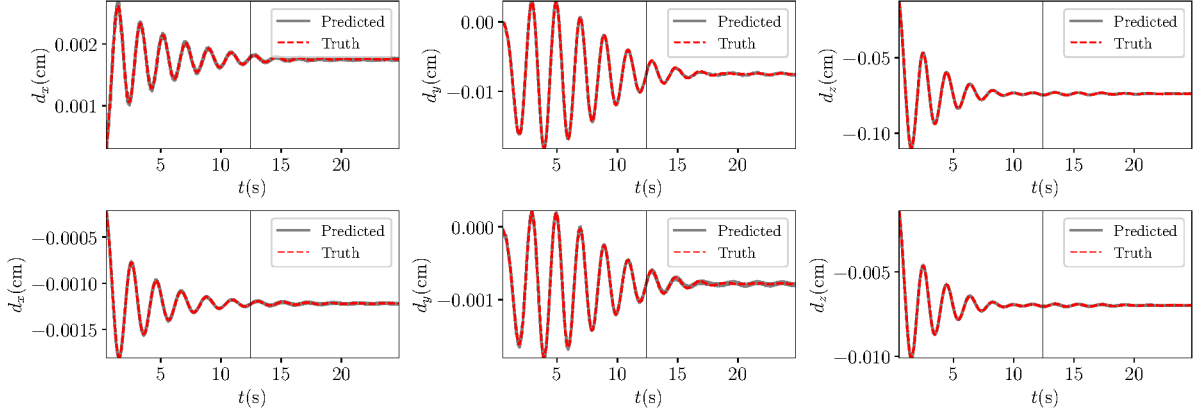
Fig. 10: Online displacement predictions for the cantilever model. CPU label: 0,1. Training is based on shared degrees of freedom at each partition, but plotted nodes are not shared. Predicted steps: 98000. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0).
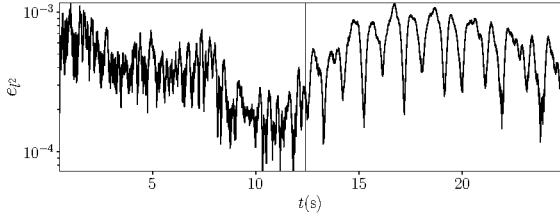

Fig. 11: Time history of l^2 error in online predictions.

refined cantilever model partitioned over 6 CPUs and shown in Figure 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^3, resulting in extra non-zeros in the update for the displacement solutions (17), and therefore with additional complexity imposed to the network training.

Results are shown in Figure 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: $\boldsymbol{f}(t) = [0, 0, -f_z \mathbb{I}_{t<3}(t)]^T$, where the indicator function $\mathbb{I}_{t<3}(t)$ is defined as

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

As shown in Figure 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) $\boldsymbol{d}^{(0)}$ 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 approximating dynamical systems. To generate training data, we random perturb the steady solution $\bar{\boldsymbol{d}}$ up to 25% and use it as the
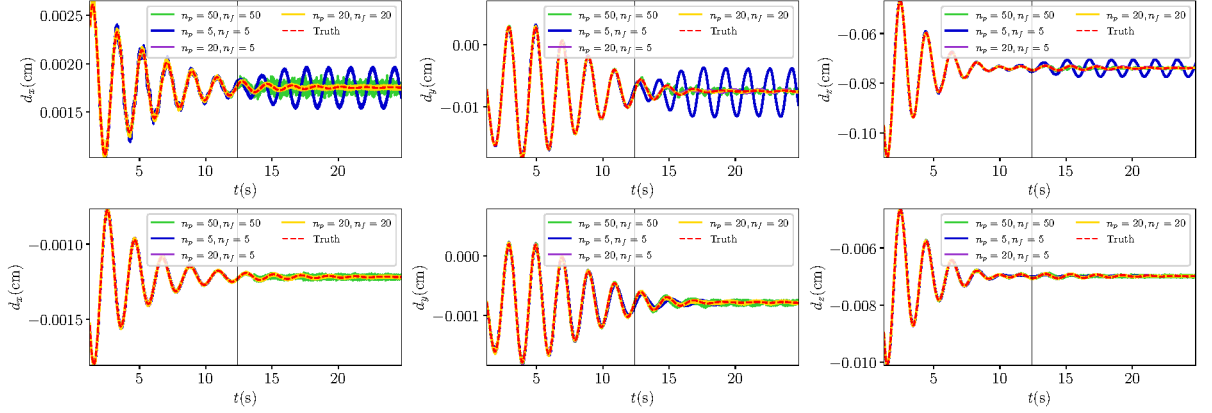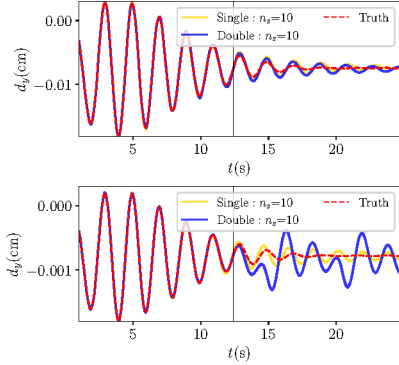
Fig. 12: Predicted dynamics for the cantilever model trained with different input-output sequence length (n_p, n_f). CPU label: 0,1. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 95000. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0).
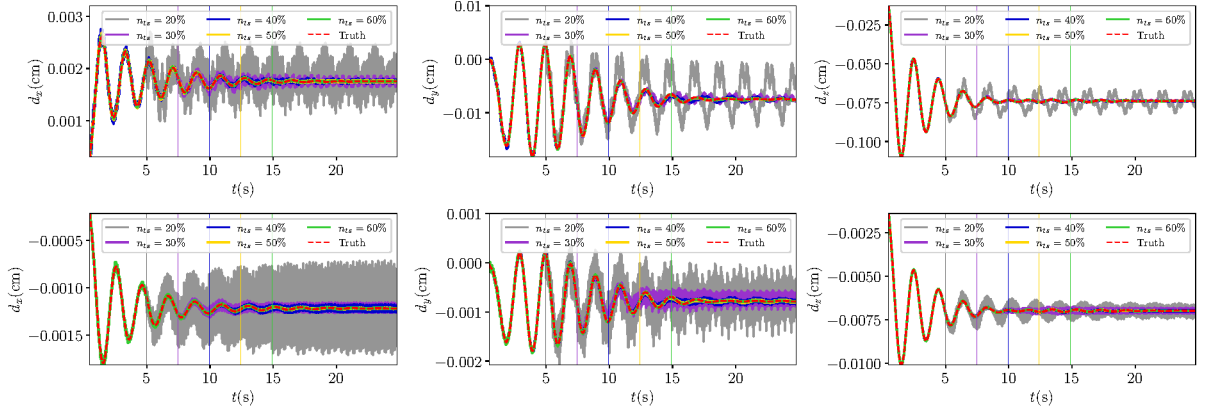
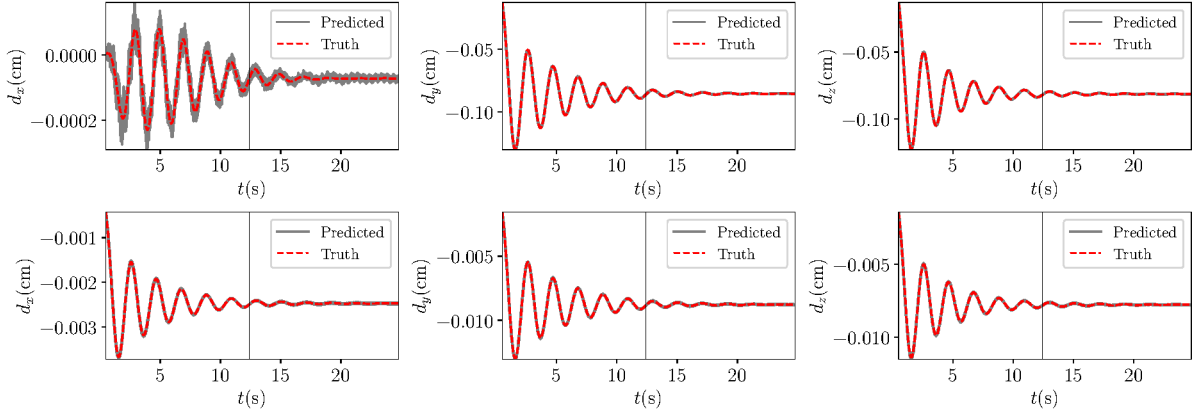**Fig. 13**: Predicted dynamics for the cantilever model trained with different training data size (n_{ts}). CPU label: 0,1. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 98000. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0).

initial displacement, i.e.

$$\boldsymbol{d}^{(0)} = (1+u)\bar{\boldsymbol{d}}, \qquad (29)$$

where u is a uniformly distributed random variable $u \sim \mathcal{U}(-0.25, 0.25)$. 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 $\boldsymbol{f}$ can severely affect the dynamic system response (see, e.g., *forced* vibrations [11]). Similarly to the previous section, we expand the training data set by adding displacement ensembles generated by our distributed finite element solver through the application of multiple uniform loads (the number of loads is denoted
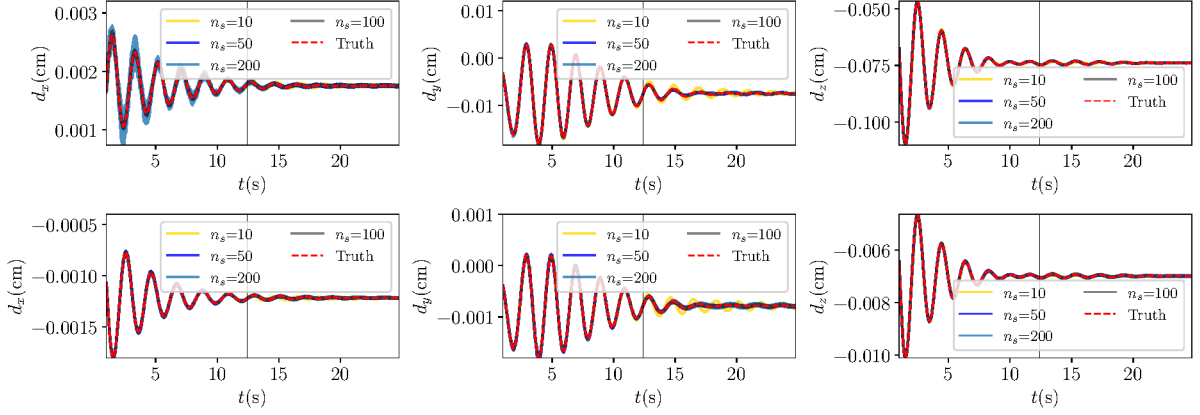
Fig. 14: Predicted dynamics for the cantilever model trained with different sample size n_s. CPU label: 0,1. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 96000. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0).


Fig. 15: Predicted y-displacement dynamics for the cantilever model trained with single and double precision, $n_s = 10$.
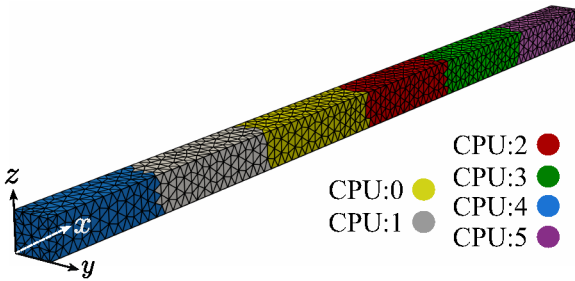
Fig. 16: Refined mesh partitioned over 6 CPUs.

as F_n). A parametric family of loading conditions is obtained by introducing a uniform random variable to the z-component of $\boldsymbol{f}$

$$\boldsymbol{f} = [0, 0, -\alpha_f]^T; \quad \alpha_f \sim \mathcal{U}(\alpha_{f,\min}, \alpha_{f,\max}), \quad (30)$$

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

However, simply increasing the size of the dataset is not sufficient in this case to produce accurate predictions (see Figure 21). We therefore modify the network architecture using a *conditional* decoder [14, 30] (e.g. see Figure 22) , where the loading condition is concatenated to the last item in the input sequence, i.e., $\boldsymbol{d}^{(n_p)}$. It is also worth noting that, by stacking multiple identical α_f values to $\boldsymbol{d}^{(n_p)}$, we can further improve the accuracy. In our experiment, we concatenate 12 copies of α_f to $\boldsymbol{d}^{(n_p)}$.

Results produced by this conditional encoder-decoder LSTM network are illustrated in Figure 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 Figure 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
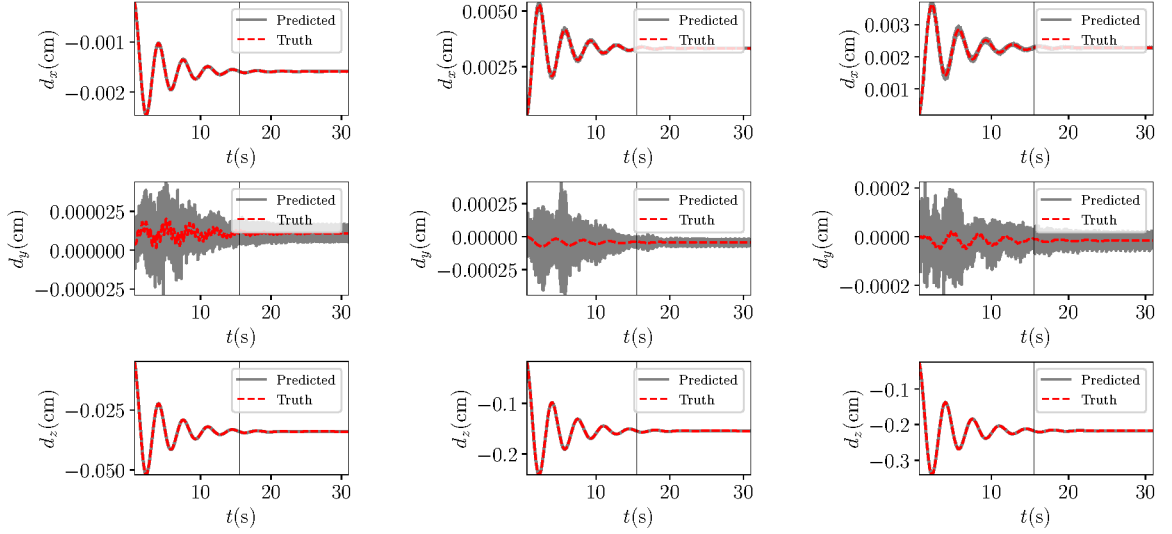
(a) CPU:1, node: $(7.45, 0.69, 0.31)$.   (b) CPU:3, node: $(18.6, 1, 0.77)$.   (c) CPU:5, node: : $(23.66, 1, 0.68)$.

Fig. 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: 490000.

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).

combination of the following loads

$$\boldsymbol{f}_{1*}(\boldsymbol{x}) = [0, 0, -\frac{x}{50}]^T; \quad \boldsymbol{f}_{2*}(\boldsymbol{x}) = [0, 0, -\frac{x^2}{500}]^T,$$
(31)

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

$$\boldsymbol{f}_*(\boldsymbol{x}) = [0, 0, -(1 - \beta_f)\boldsymbol{f}_{1*} - \beta_f \boldsymbol{f}_{2*}]^T, \quad (32)$$

where $\beta_f \sim \mathcal{U}(0,1)$. Similarly to the uniform loading scenario, we condition the predictions by concatenating nodal forces from $\boldsymbol{f}_*$ to $\boldsymbol{d}^{(n_p)}$.

Testing results are shown in Figure 24, using an $\boldsymbol{f}_*$ that does not belong to the training set. Similar to the previous section, training based on $5 \sim 10$ datasets is sufficient for our conditional network to accurately learn the dynamics under a general $\boldsymbol{f}_*$.
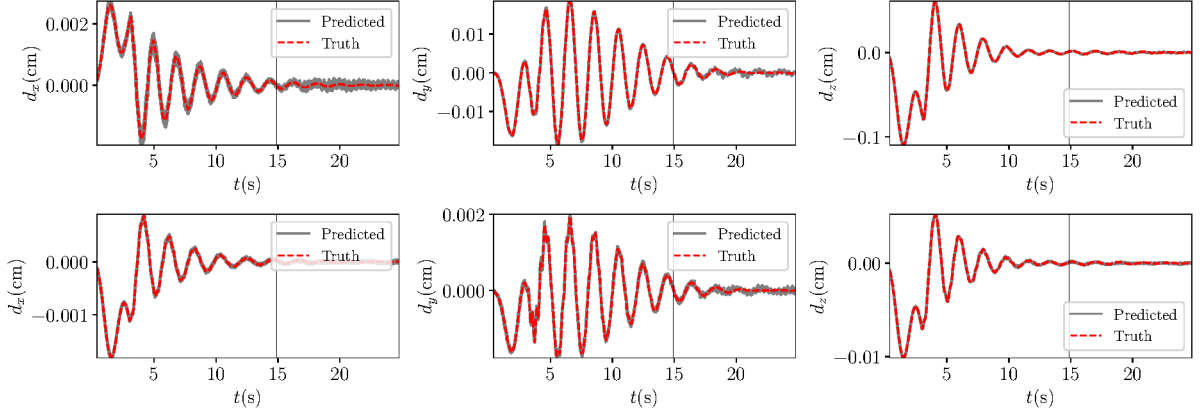
Fig. 19: Predicted dynamics for the cantilever model with discontinuous loading. CPU label: 0,1. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 98400. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0).
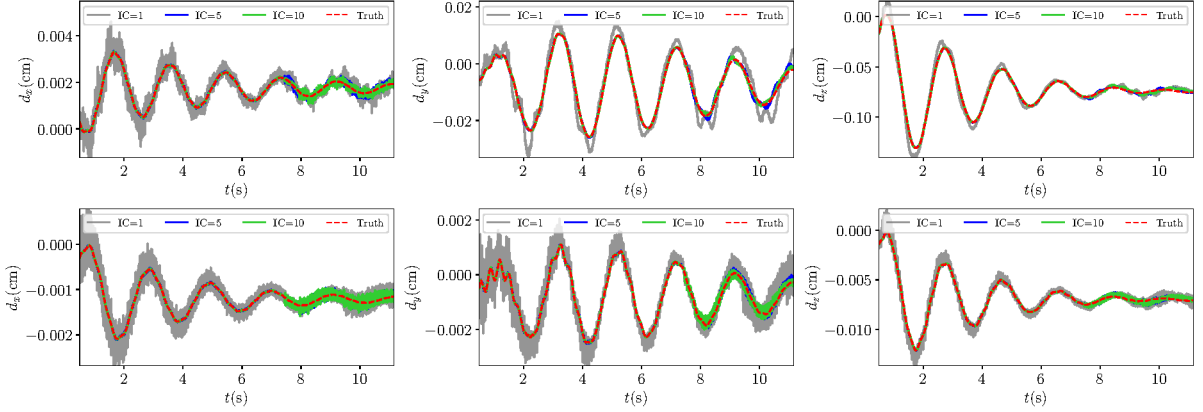
Fig. 20: Predicted dynamics for the cantilever model trained with multiple initial conditions. CPU label: 0,1. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 43000. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0). The tested initial condition is not in the training dataset.

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 $n_B = 5$, $n_H = 100$, $\eta_0 = 5 \times 10^{-4}$, $n_p = n_f = 20$, $n_s = 100$, $n_{ts} = 0.5$ and the external loading is $\boldsymbol{f} = [0, 0, -f_z]^T$. 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
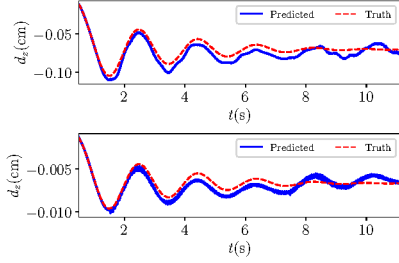
Fig. 21: Predicted z-displacement dynamics for the cantilever model trained with 5 ($F_n = 5$) external loadings. The network is **not** conditioned. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0). The tested external loading is not in the training dataset.
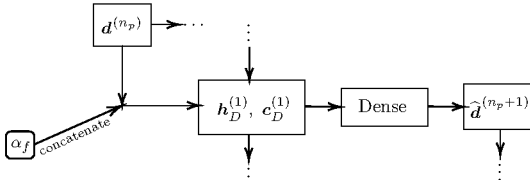
Fig. 22: Modified conditional decoder.

in Figure 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 -1.56×10^{-7} is sufficiently small.

Finally, Figure 27 shows the evolution of the l^2 error using Equation (27), where we observe a trend similar to that reported in Figure 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 Figure 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, $t_{\text{end}} = 1s$). It fluctuates from a systolic maximum of 1.6×10^5 baryes (120 mmHg) to a diastolic minimum of 1.067×10^5 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 $\rho = 1$g/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 ($O_1, \cdots, O_6$), as shown in Figure 28a. 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 2.02×10^{-6}, and we have increased it up to 5×10^{-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 Figure 29 that this increase corresponded to a marginal effect on the model dynamics. We also include a comparison test in Figure 29 using the original time step size 2.02×10^{-6}, current size 5×10^{-6} and a further slightly-increased size 5.1×10^{-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 hyperparamter 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 modelled.

6 Error control

In this section, we discuss metrics for error estimation that are naturally provided by the proposed
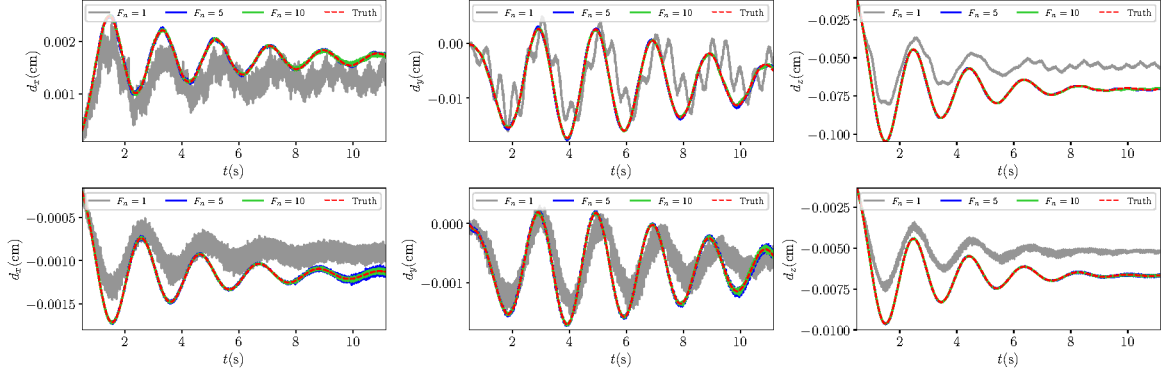
Fig. 23: Predicted dynamics for the cantilever model trained with multiple uniform external loadings. The network is conditioned. CPU label: 0,1. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 43000. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0). The tested external loading is not in the training dataset.
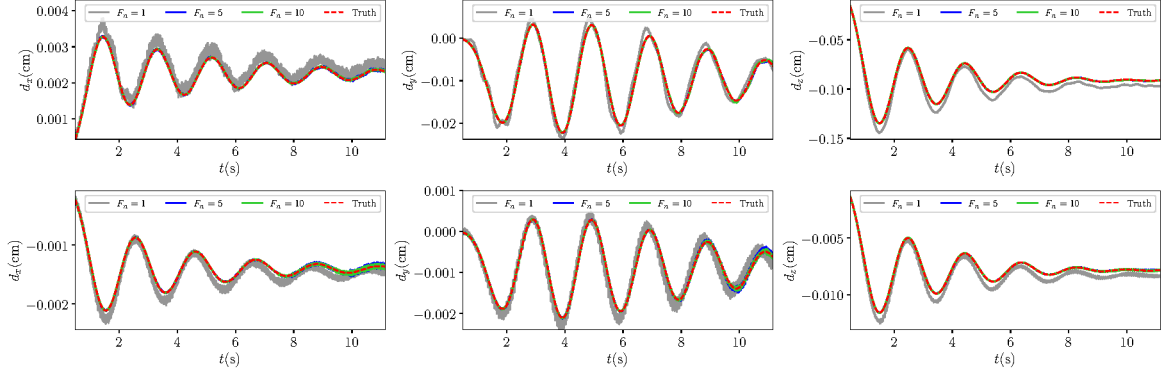

Fig. 24: Predicted dynamics for the cantilever model trained with multiple distributed external loadings. The network is conditioned. CPU label: 0,1. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 43000. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0). The tested distributed external loading is not in the training dataset.

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 Figure 28a, there are exactly two, separately trained, deep neural networks 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 Figure 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 Section 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

$$es_{l^2}^{(t)} = \frac{1}{3 \cdot N_a} \sum_{i=1}^{N_a} \|\widehat{\boldsymbol{d}}_{(i)}^{(t)} - \boldsymbol{d}_{(i)}^{(t)}\|_2. \qquad (33)$$

Therefore, $es_{l^2}^{(t)}$ represents a global average l^2 error per shared degree of freedom at time t.

(a) Node: $(0.96, 1, 0.5)$.   (b) Node: $(12.5, 1, 0.5)$.   (c) Node: $(25, 0.5, 0.5)$.
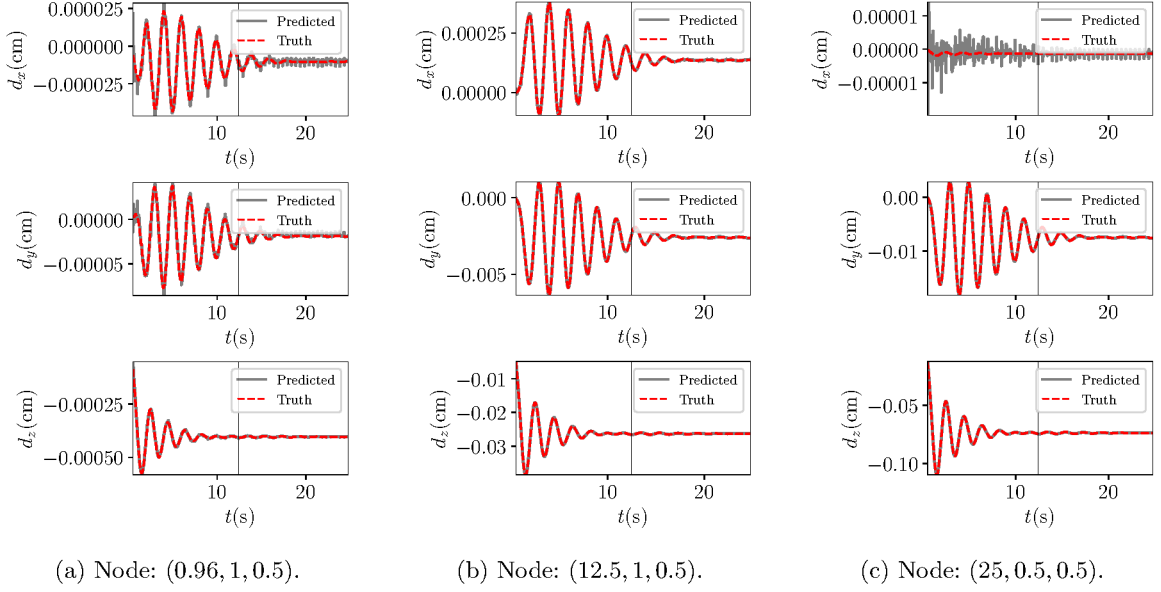
Fig. 25: Predicted dynamics at three distinctive nodes for the cantilever model. Training is based on all degrees of freedom. Predicted steps: 98000.

(a) Exact [cm].
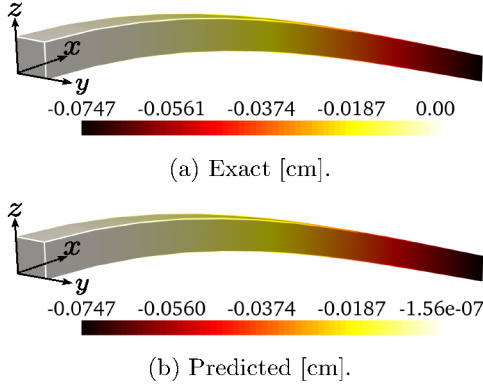
(b) Predicted [cm].

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.

Moreover, since there are exactly two processors associated with each shared node, we denote their arithmetic average as $\bar{es}_{l^2}^{(t)}$. We also calculate the difference between two predictions for the same shared node by replacing the exact solution $\boldsymbol{d}_{(i)}^{(t)}$ in equation (33) with the solution predicted by the second network. The resulting quantity is denoted as $\widehat{es}_{l^2}^{(t)}$. As shown in both Figure 31 and Figure 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 Figure 33 to show the temporal average of such variability. We further define

$$et_{(j),l^2} = \frac{1}{3 \cdot n_T} \sum_{i=1}^{n_T} \|\widehat{\boldsymbol{d}}_{(j)}^{(i)} - \boldsymbol{d}_{(j)}^{(i)}\|_2, \qquad (34)$$

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 $et_{(j),l^2}$ to indicate an average of two generic processors that share
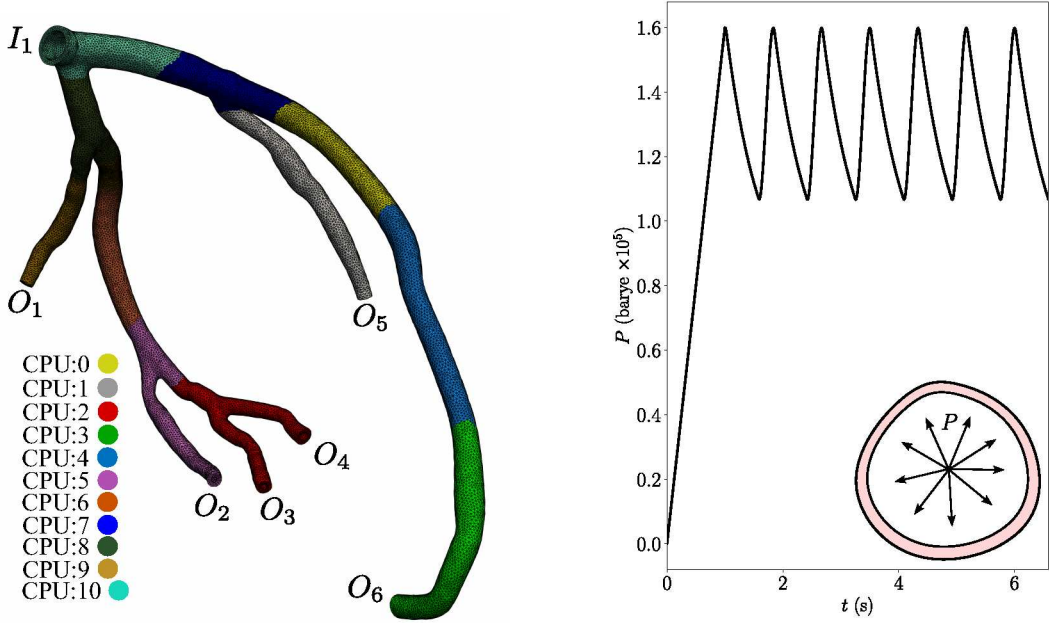
(a) Geometry of the left coronary artery model. The mesh is partitioned over 11 CPUs, labelled 0 to 10.

(b) Pressure waveform acting on the vessel lumen.

Fig. 28: Geometry, mesh and loading condition for the coronary artery model.
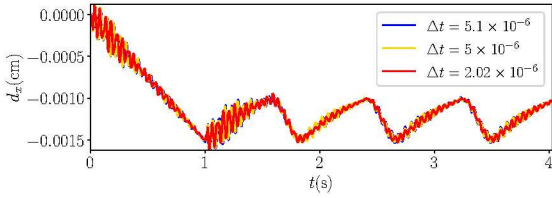
Fig. 29: Comparison of x-component displacement dynamics with and without artificial mass scaling. Plotted node: (-13.7, 6.4, -9.6)

node j, and denote $\widehat{et}_{(j),l^2}$ for the difference in two predictions, at the same shared node j.

From Figure 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 $\mathbf{K}_e$, $\boldsymbol{f}_e^{(n),\text{ext}}$, (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 quantities in Table 2, where the superscript $\bar{(\cdot)}$ means the average with respect to time and over multiple processors. The speedup factor $\bar{\zeta}$ is simply the ratio between $\bar{t}_t$ and $\bar{t}_t'$, where $(\cdot)'$ denotes whether the data-driven surrogate is used to reduce the synchronization cost.
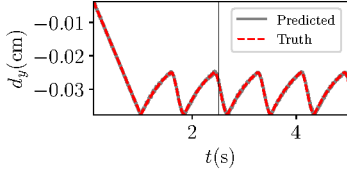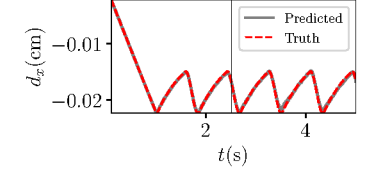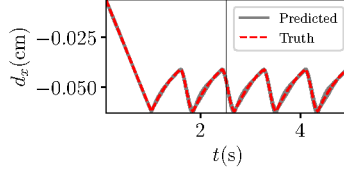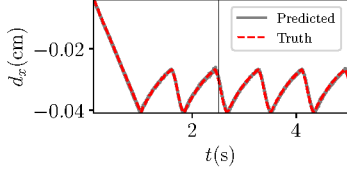
We are interested in two scenarios. The first considers a fixed number of cores and several models of increasing size. The second considers a fixed-size mesh partitioned over an increasing number of
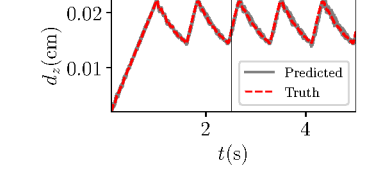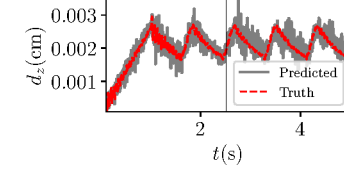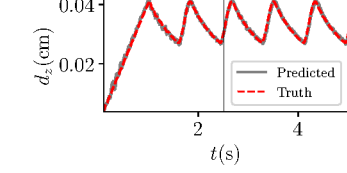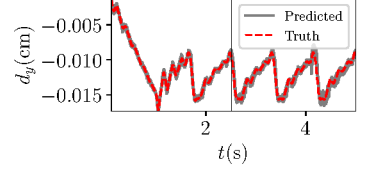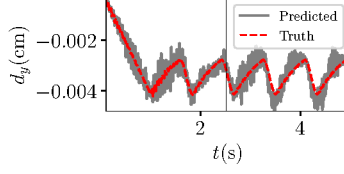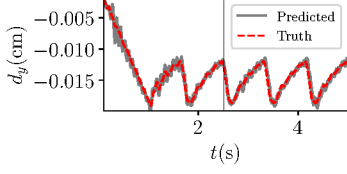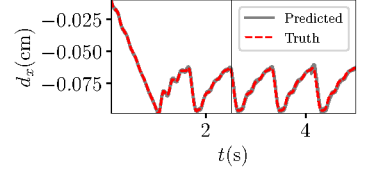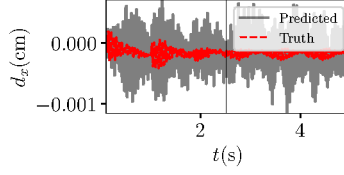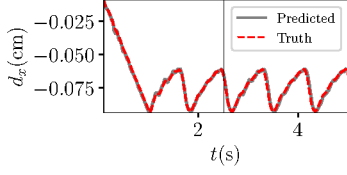
(a) CPU:0, node: (-11.9,10.4,-5.4).

(b) CPU:2, node: (-13.6,6.2,-9.7).

(c) CPU:4, node:(-12.3,13.0,-9.1).
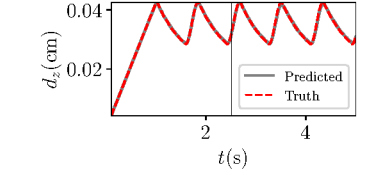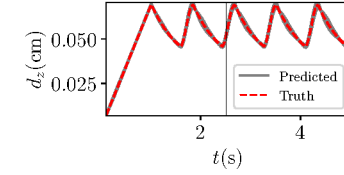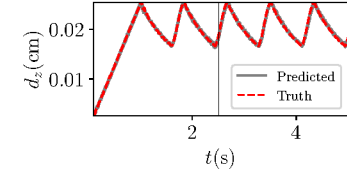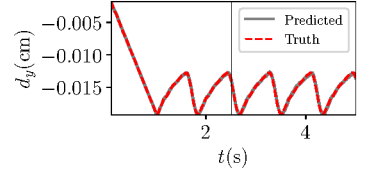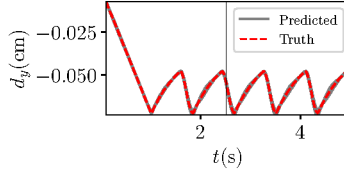
(d) CPU:6, node:(-12.3,4.7,-7.2).

(e) CPU:8, node:(-10.9,6.3,-4.8)

(f) CPU:10, node:(-10.7,7.5,-3.6).

Fig. 30: Predicted dynamics of the human left coronary artery model. CPU labels are 0,2,4,6,8,10. Training is based on shared degrees of freedom of each partition and plotted nodes are not shared. Predicted steps: 980000.

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.
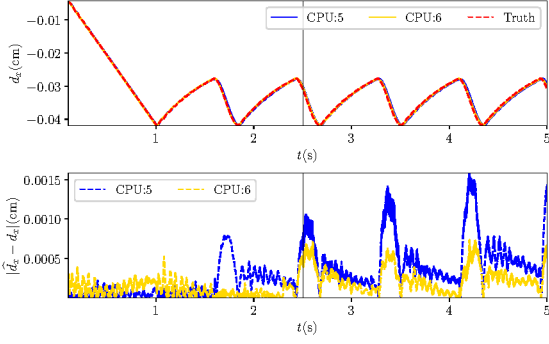
Fig. 31: Offline predicted d_x displacement and time history of absolute error between two neighbor processors for coronary model. The plotted node (-12.8, 4.9, -7.5) is shared by CPU5 and CPU6.
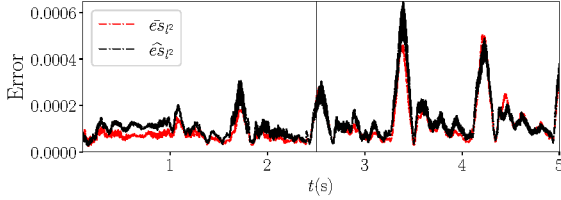

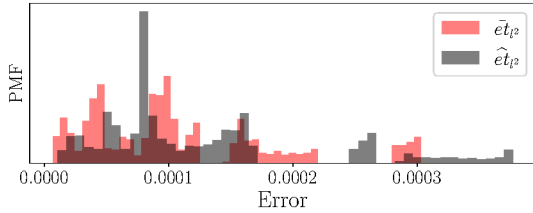**Fig. 32**: Evolution of space-averaged l^2 error per shared degree of freedom of the coronary model.

Fig. 33: PMF of time-averaged l^2 error per shared degree of freedom of the coronary model.

In such a case, the cost of displacement synchronization dominates over the relatively inexpensive matrix-vector product, making the proposed approach particularly appealing.

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 Figure 34, as $\bar{t}_s$

Total time	$\bar{t}_t$ (s)
Time for element quantity evaluation	$\bar{t}_e$ (s)
Time for synchronization	$\bar{t}_s$ (s)
Time for matrix-vector product	$\bar{t}_m$ (s)
Time for applying data-driven model	$\bar{t}_d$ (s)
Ratio of element quantity formation cost	$\bar{r}_e(\%)$
Ratio of synchronization cost	$\bar{r}_s(\%)$
Ratio of matrix-vector product cost	$\bar{r}_m(\%)$
Ratio of data-driven model cost	$\bar{r}_d(\%)$
Number of shared nodes	$\bar{N}_a$
Number of elements	$\bar{N}_e$
Number of processor used	n_c
Speedup factor	ζ

Table 2: Definition of quantities for performance analysis.

grows rapidly with respect to $\bar{N}_a$ 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 Figures 35 and 36, since $\bar{t}'_d$ is orders of magnitude less expensive than $\bar{t}_s$.

Generally speaking, as shown in Figure 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 $\bar{t}'_d$ only depends on the size of input $\boldsymbol{d}$, which essentially scales with $\bar{N}_a$, see Figure 35.
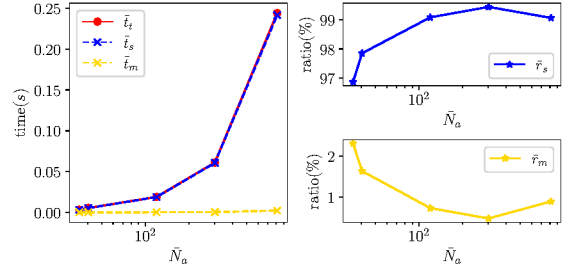

Fig. 34: Timing statistics with pre-assembly and without data-driven model. The number of partitioning is fixed at $n_c = 8$. Left: average cost per step vs. average number of shared nodes. Right: average ratio per step vs. average number of shared nodes.

Fixed mesh size - In this section, we fix the mesh size and distribute it over an increasing number of processors. In Figure 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
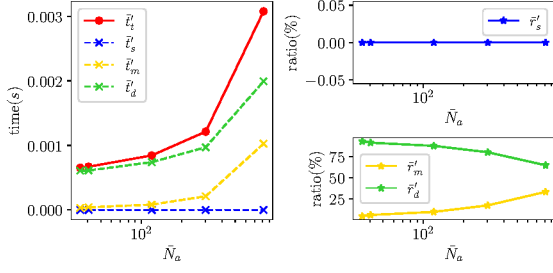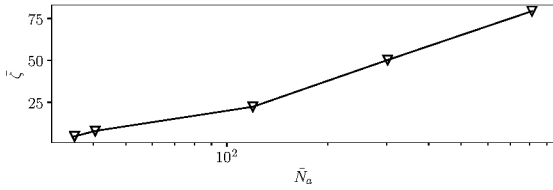
Fig. 35: Timing statistics with pre-assembly and with data-driven model. The number of partitioning is fixed at $n_c = 8$. Left: average cost per step vs. average number of shared nodes. Right: average ratio per step vs. average number of shared nodes.

Fig. 37: Timing statistics of the test with pre-assembly and without data-driven model. The mesh resolution is fixed. Left: average cost per step vs. number of partitioning. Right: average ratio per step vs. number of partitioning.

Fig. 36: Data driven model speed-up with pre-assembly for $n_c = 8$.

Fig. 38: Timing statistics of the test with pre-assembly and with data-driven model. The mesh resolution is fixed. Left: average cost per step vs. number of partitioning. Right: average ratio per step vs. number of partitioning.

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 Figure 38 and Figure 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 $\bar{\zeta}$ shown in Figure 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 recomputed 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.
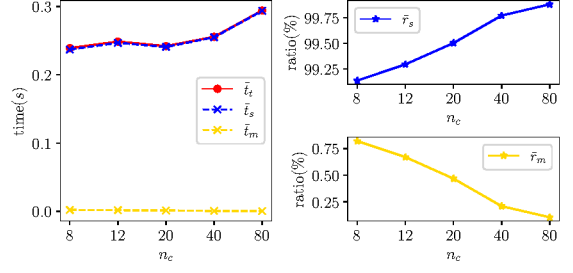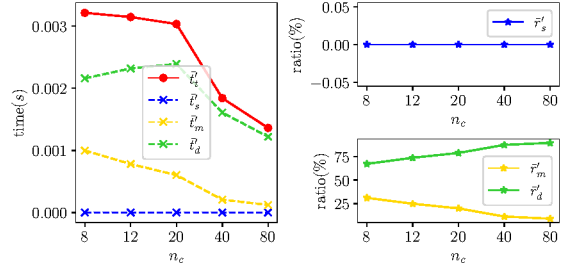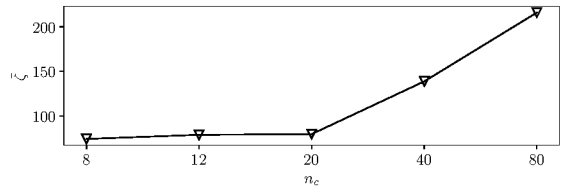

Fig. 39: Data driven model speed-up with pre-assembly for an increasing number of processors.

Fixed number of partitions - Without the pre-assembly, from Figure 40 and Figure 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 $\bar{r}'_s$ is reduced while $\bar{r}'_e$ increases, clearly reducing the performance of our surrogate model (e.g. see Figure 42).
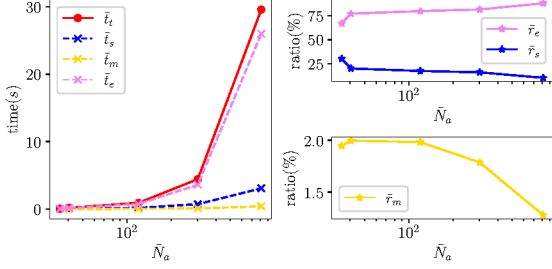

Fig. 40: Timing statistics without pre-assembly and without data-driven model. The number of partitioning is fixed at $n_c = 8$. Left: average cost per step vs. average number of shared nodes. Right: average ratio per step vs. average number of shared nodes.
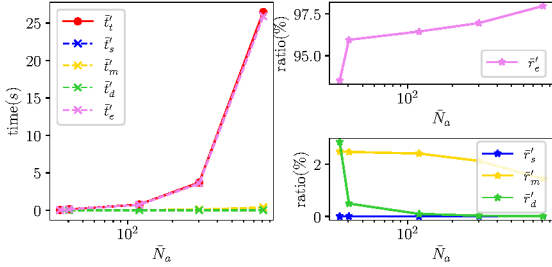

Fig. 41: Timing statistics without pre-assembly and with data-driven model. The number of partitioning is fixed at $n_c = 8$. Left: average cost per step vs. average number of shared nodes. Right: average ratio per step vs. average number of shared nodes.
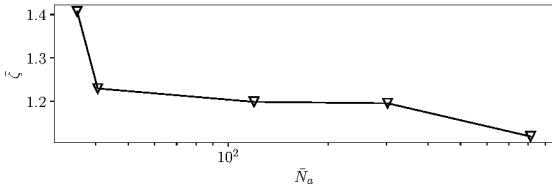

Fig. 42: Data driven model speed-up without pre-assembly for $n_c = 8$.

Fixed mesh size - For an increasing number of processors, from $\bar{r}_s$ and $\bar{r}_e$ in Figure 43, we observe a more efficient evaluation for the element quantities is accompanied by an increasing synchronization cost. The speed-up in this case, shown in Figure 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 Section 1.
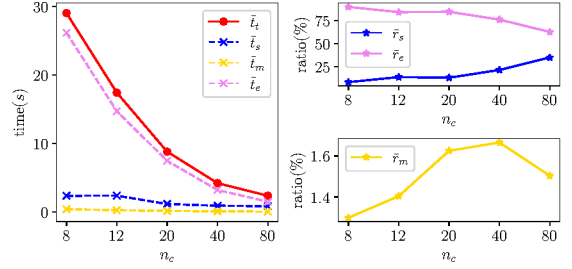

Fig. 43: Timing statistics without pre-assembly and without data-driven model. The mesh resolution is fixed. Left: average cost per step vs. number of partitions. Right: average ratio per step vs. number of partitions.

Fig. 44: Timing statistics without pre-assembly and with data-driven model. The mesh resolution is fixed. Left: average cost per step vs. number of partitions. Right: average ratio per step vs. number of partitions.
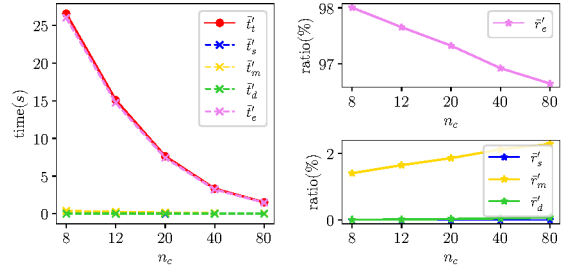
7.3 Effect of model geometry

Most examples we have tested in this paper consider a finite element model of a slender cantilever,
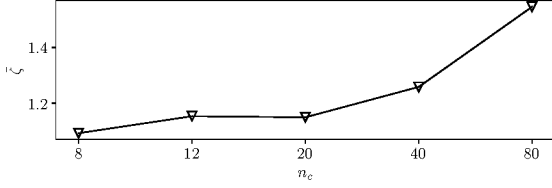
Fig. 45: Data driven model speed-up without pre-assembly for an increasing number of processors.

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.
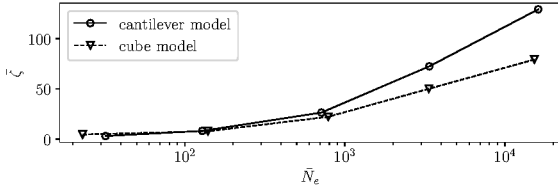
(a) Comparison of speedup factor with pre-assembly.
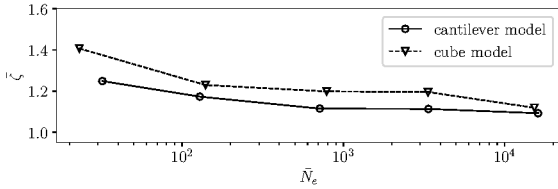
(b) Comparison of speedup factor without pre-assembly.

Fig. 46: Comparison of speedup factors for a cubic and slender cantilever geometry. The number of partitioning is fixed at $n_c = 8$.

To make the comparison fair, we plot the speed-up factor $\bar{\zeta}$ with respect to the average element number per partition $\bar{N}_e$ in Figure 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 $\bar{t}'_t$. 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 multipe 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 minibatches 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 5×10^5 total steps. The training example are then extracted as a subset of such examples

Even though the average number of shared nodes $\bar{N}_a$ differs wildly across models, the effect on the training time appears to be minimal, suggesting it is not sensitive to $\bar{N}_a$ 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 Figure 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 γ (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 minutes versus $\sim$5 hours). In the results shown in Figure 47 the displacements in the y-z plane remain comparable to those in Figure 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 Section 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 we will be devoted to apply the proposed approach in the context of non-linear problems.

Declarations

The authors declare that they have no conflicts of interest.

References

[1] M. Aslam, O. Riaz, S. Mumtaz, and A.D. Asif. Performance comparison of GPU-based Jacobi solvers using CUDA provided synchronization methods. *IEEE Access*, 8:31792–31812, 2020.

[2] J. Bakarji, K. Champion, J.N. Kutz, and S.L. Brunton. Discovering governing equations from partial measurements with deep delay autoencoders. *arXiv preprint arXiv:2201.05136*, 2022.

[3] A Bartezzaghi, M. Cremonesi, N. Parolini, and U. Perego. An explicit dynamics GPU structural solver for thin shell finite elements. *Computers & Structures*, 154:29–40, 2015.

[4] T. Belytschko, W.K. Liu, B. Moran, and K. Elkhodary. *Nonlinear finite elements for continua and structures*. John wiley & sons, Hoboken, NJ, 2014.
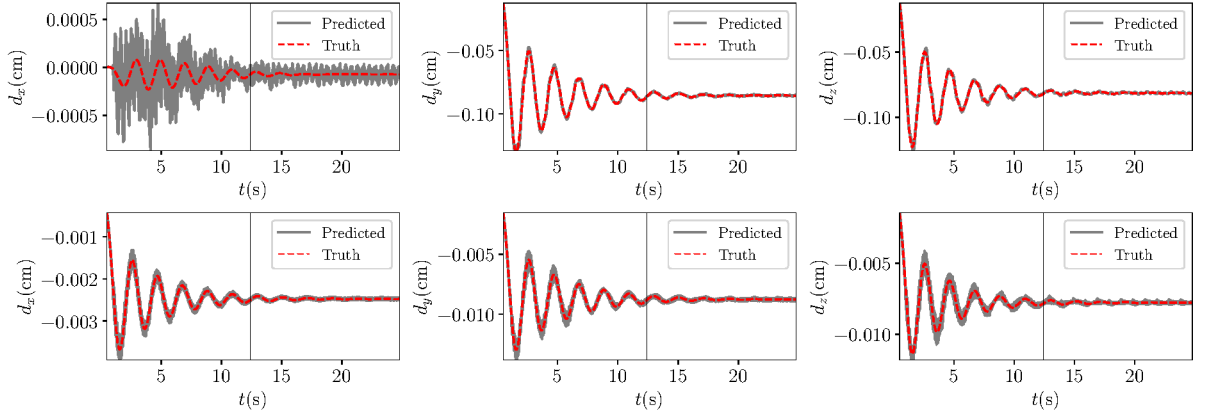
Fig. 47: Predicted dynamics for the cantilever model with skewed loading. CPU label: 0,1. Hyperparameters are optimized for minimize training time: $n_B = 10, n_H = 50, \eta_0 = 0.998, n_s = 200$. Top row: CPU:0, node (25.0,0.0,1.0). Bottom row: CPU:1, node (5.77,0.0,0.0).

[5] U. Bhat and S.B. Munch. Recurrent neural networks for partially observed dynamical systems. *Phys. Rev. E*, 105:044205, Apr 2022.

[6] S.L. Brunton, J.L. Proctor, and J.N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[7] B. Chang, M.M. Chen, E. Haber, and E.H. Chi. AntisymmetricRNN: A dynamical system view on recurrent neural networks, 2019.

[8] Q. Chen. MGMETIS—mesh & graph METIS partitioning. *GitHub repository:* https://github.com/chiao45/mgmetis, 2020.

[9] Z. Chen, V. Churchill, K.L. Wu, and D.B. Xiu. Deep neural network modeling of unknown partial differential equations in nodal space. *Journal of Computational Physics*, 449:110782, 2022.

[10] V. Churchill and D.B. Xiu. Deep learning of chaotic systems from partially-observed data. *arXiv preprint arXiv:2205.08384*, 2022.

[11] R.W. Clough and J. Penzien. *Dynamics of Structures*. Civil engineering series. McGraw-Hill, 1993.

[12] X.H. Fu, L.B. Chang, and D.B. Xiu. Learning reduced systems via deep neural networks with memory. *Journal of Machine Learning for Modeling and Computing*, 1(2), 2020.

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

[14] M. Garnelo, D. Rosenbaum, C. Maddison, T. Ramalho, D. Saxton, M. Shanahan, Y. W. Teh, D. Rezende, and S.M.A. Eslami. Conditional neural processes. In *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 1704–1713. PMLR, 10–15 Jul 2018.

[15] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, Cambridge, MA, 2016.

[16] K.M. He, X.Y. Zhang, S.Q. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

[18] Y.H. Hu, T. Zhao, S.X. Xú, Z.L. Xu, and L.Z. Lin. Neural-PDE: a RNN based neural network for solving time dependent PDEs. *Communications in Information and Systems*, 2022.

[19] T.J.R. Hughes. *The finite element method: linear static and dynamic finite element analysis.* Dover Publications, INC., Mineola, New York, 2012.

[20] P. Huthwaite. Accelerated finite element elastodynamic simulations using the GPU. *Journal of Computational Physics*, 257:687–707, 2014.

[21] G.R. Joldes, A. Wittek, and K. Miller. Real-time nonlinear finite element computations on GPU–application to neurosurgical simulation. *Computer methods in applied mechanics and engineering*, 199(49-52):3305–3314, 2010.

[22] J. Jung, H. Jun, and P.S. Lee. Self-updated four-node finite element using deep learning. *Computational Mechanics*, 69(1):23–44, 2022.

[23] J. Jung, K. Yoon, and P.S. Lee. Deep learned finite elements. *Computer Methods in Applied Mechanics and Engineering*, 372:113401, 2020.

[24] K. Kaheman, J.N. Kutz, and S.L. Brunton. SINDy-PI: a robust algorithm for parallel implicit sparse identification of nonlinear dynamics. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 476(2242):20200279, 2020.

[25] E. Kaiser, J.N. Kutz, and S.L Brunton. Data-driven discovery of Koopman Eigenfunctions for control. *Machine Learning: Science and Technology*, 2(3):035023, june 2021.

[26] George Karypis and Vipin Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0. http://www.cs.umn.edu/~metis, 2009.

[27] H.J. Kim, I.E. Vignon-Clementel, J.S. Coogan, C.A. Figueroa, K.E. Jansen, and C.A. Taylor. Patient-specific modeling of blood flow and pressure in human coronary arteries. *Annals of biomedical engineering*, 38(10):3195–3209, 2010.

[28] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] D. Komatitsch, G. Erlebacher, D. Göddeke, and D. Michéa. High-order finite-element seismic wave propagation modeling with MPI on a large GPU cluster. *Journal of computational physics*, 229(20):7692–7714, 2010.

[30] A. Kovacs, L. Exl, A. Kornell, J. Fischbacher, M. Hovorka, M. Gusenbauer, L. Breth, H. Oezelt, M. Yano, N. Sakuma, A. Kinoshita, T. Shoji, A. Kato, and T. Schrefl. Conditional physics informed neural networks. *Communications in Nonlinear Science and Numerical Simulation*, 104:106041, Jan 2022.

[31] M. Kronbichler and K. Ljungkvist. Multigrid for matrix-free high-order finite element computations on graphics processors. *ACM Trans. Parallel Comput.*, 6(1), may 2019.

[32] L. Kulowski. LSTM_encoder_decoder. *GitHub repository:* https://github.com/lkulowski/LSTM_encoder_decoder, 2020.

[33] J.N. Kutz, S.L. Brunton, B.W. Brunton, and J.L. Proctor. *Dynamic mode decomposition: data-driven modeling of complex systems.* SIAM, 2016.

[34] X. Li and D.E. Schiavazzi. An ensemble solver for segregated cardiovascular FSI. *Computational Mechanics*, 68(6):1421–1436, 2021.

[35] L. Lu, P.Z. Jin, G.F. Pang, Z.Q. Zhang, and G.E. Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[36] P. Malhotra, A. Ramakrishnan, G. Anand, L. Vig, P. Agarwal, and G. Shroff. LSTM-based encoder-decoder for multi-sensor anomaly detection, 2016.

[37] S.E. McCaslin, P.S. Shiakolas, B.H. Dennis, and K.L. Lawrence. Closed-form stiffness matrices for higher order tetrahedral finite elements. *Advances in Engineering Software*, 44(1):75–79, 2012.

[38] L. Olovsson, K. Simonsson, and M. Unosson. Selective mass scaling for explicit finite element analyses. *International Journal for Numerical Methods in Engineering*, 63(10):1436–1445, 2005.

[39] S.H. Park, B.D. Kim, C.M. Kang, C.C. Chung, and J.W. Choi. Sequence-to-sequence prediction of vehicle trajectory via LSTM encoder-decoder architecture. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 1672–1678. IEEE, 2018.

[40] L. Partin, G. Geraci, A. Rushdi, M.S. Eldred, and D.E. Schiavazzi. Multifidelity data fusion in convolutional encoder/decoder networks, 2022.

[41] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z.M. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32:8026–8037, 2019.

[42] T. Qin, K.L. Wu, and D.B. Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 395:620–635, 2019.

[43] M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

[44] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P.W. Battaglia. Learning to simulate complex physics with graph networks. In *Proceedings of the 37th International Conference on Machine Learning*, ICML'20. JMLR.org, 2020.

[45] D. Schillinger, J.A. Evans, F. Frischmann, R.R. Hiemstra, M.C. Hsu, and T.J.R. Hughes. A collocated C0 finite element method: Reduced quadrature perspective, cost comparison with standard finite elements, and explicit structural dynamics. *International Journal for Numerical Methods in Engineering*, 102(3-4):576–631, 2015.

[46] G. Scovazzi, B. Carnes, X. Zeng, and S. Rossi. A simple, stable, and accurate linear tetrahedral finite element for transient, nearly, and fully incompressible solid dynamics: a dynamic variational multiscale approach. *International Journal for Numerical Methods in Engineering*, 106(10):799–839, 2016.

[47] J. Seo, D.E. Schiavazzi, A.M. Kahn, and A.L. Marsden. The effects of clinically-derived parametric data uncertainty in patient-specific coronary simulations with deformable walls. *International journal for numerical methods in biomedical engineering*, 36(8):e3351, 2020.

[48] J. Seo, D.E. Schiavazzi, and A.L. Marsden. Performance of preconditioned iterative linear solvers for cardiovascular simulations in rigid and deformable vessels. *Computational mechanics*, 64(3):717–739, 2019.

[49] D.E. Shea, S.L. Brunton, and J.N. Kutz. SINDy-BVP: Sparse identification of nonlinear dynamics for boundary value problems. *Phys. Rev. Research*, 3:023255, Jun 2021.

[50] A. Sherstinsky. Fundamentals of recurrent neural network (RNN) and long short-term memory (LSTM) network. *Physica D: Nonlinear Phenomena*, 404:132306, 2020.

[51] P.S. Shiakolas, R.V. Nambiar, K.L. Lawrence, and W.A. Rogers. Closed-form stiffness matrices for the linear strain and quadratic strain tetrahedron finite elements. *Computers & structures*, 45(2):237–242, 1992.

[52] S.K.F. Stoter, T.H. Nguyen, R.R. Hiemstra, and D. Schillinger. Variationally consistent mass scaling for explicit time-integration schemes of lower- and higher-order finite element methods, 2022.

[53] V. Strbac, D.M. Pierce, J. Vander Sloten, and N. Famaey. GPGPU-based explicit finite element computations for applications in biomechanics: the performance of material models, element technologies, and hardware generations. *Computer Methods in Biomechanics and Biomedical Engineering*, 20(16):1643–1657, 2017.

[54] Z.A. Taylor, M. Cheng, and S. Ourselin. High-speed nonlinear finite element analysis for surgical simulation using graphics processing units. *IEEE Transactions on Medical Imaging*, 27(5):650–663, 2008.

[55] P.R. Vlachas, W. Byeon, Z.Y. Wan, T.P. Sapsis, and P. Koumoutsakos. Data-driven forecasting of high-dimensional chaotic systems with long short-term memory networks. *Proceedings of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 474(2213):20170844, 2018.

[56] K.L. Wu and D.B. Xiu. Data-driven deep learning of partial differential equations in modal space. *Journal of Computational Physics*, 408:109307, 2020.

[57] O.C. Zienkiewicz, R.L. Taylor, and J.M. Too. Reduced integration technique in general analysis of plates and shells. *International Journal for Numerical Methods in Engineering*, 3(2):275–290, 1971.