PyTorch-FEA: Autograd-enabled Finite Element Analysis Methods with Applications for Biomechanical Analysis of Human Aorta

Liang Liang[1], Minliang Liu[2], John Elefteriades[3], and Wei Sun[4]

[1]Department of Computer Science, University of Miami, Coral Gables, FL

[2]Department of Biomedical Engineering, Georgia Institute of Technology, Atlanta, GA

[3]Aortic Institute, School of Medicine, Yale University, New Haven, CT

[4]Sutra Medical Inc, Lake Forest, CA

available at

Biorxiv

5/19/2023

For correspondence:

Liang Liang, Ph.D.

Department of Computer Science

University of Miami

Ungar Building, Room 330K

Coral Gables, FL, 33146

Tel: (305) 284-8381; Email: liang@cs.miami.edu

Abstract

Background and Objectives: Finite-element analysis (FEA) is widely used as a standard tool for stress and deformation analysis of solid structures, including human tissues and organs. For instance, FEA can be applied at a patient-specific level to assist in medical diagnosis and treatment planning, such as risk assessment of thoracic aortic aneurysm rupture/dissection. These FEA-based biomechanical assessments often involve both forward and inverse mechanics problems. Current commercial FEA software packages (e.g., Abaqus) and inverse methods exhibit performance issues in either accuracy or speed.

Methods: In this study, we propose and develop a new library of FEA code and methods, named PyTorch-FEA, by taking advantage of autograd, an automatic differentiation mechanism in PyTorch. We develop a class of PyTorch-FEA functionalities to solve forward and inverse problems with improved loss functions, and we demonstrate the capability of PyTorch-FEA in a series of applications related to human aorta biomechanics. In one of the inverse methods, we combine PyTorch-FEA with deep neural networks (DNNs) to further improve performance.

Results: We applied PyTorch-FEA in four fundamental applications for biomechanical analysis of human aorta. In the forward analysis, PyTorch-FEA achieved a significant reduction in computational time without compromising accuracy compared with Abaqus, a commercial FEA package. Compared to other inverse methods, inverse analysis with PyTorch-FEA achieves better performance in either accuracy or speed, or both if combined with DNNs.

Conclusions: We have presented PyTorch-FEA, a new library of FEA code and methods, representing a new approach to develop FEA methods to forward and inverse problems in solid mechanics. PyTorch-FEA eases the development of new inverse methods and enables a natural integration of FEA and DNNs, which will have numerous potential applications.

Keywords: finite element analysis, inverse method, autograd, deep neural network, aortic aneurysm

1. Introduction

Structural Finite-element analysis (FEA) has been widely used to perform stress and deformation analysis of complex structures, for which an analytical solution might be infeasible. In the biomedical domain, FEA has been applied to study the mechanics of human tissues and organs, tissue-medical device interactions, diagnostic and treatment strategies at a patient-specific level [1-11], which may need to solve forward and inverse mechanics problems. For instance, an emerging application of FEA for medical diagnosis is the risk assessment of aortic aneurysms. Aortic aneurysm ranks consistently in the top 20 causes of death in the U.S. population [12]. Thoracic aortic aneurysm (TAA) is manifested as an abnormal bulging of thoracic aortic wall, and it is a leading cause of death in adults with a prevalence of ~ 1% in the general population [13, 14]. From the perspective of biomechanics, TAA rupture/dissection occurs when the stress acting on the aortic wall exceeds the material strength of the wall [15]. To accurately obtain the stress distribution of TAA, FEA is often used as the core solution tool in the risk assessment workflow [8, 9, 16-19]. This workflow typically involves both forward (compute stress given stress-free geometry, boundary conditions, and material properties) and inverse (identify material properties given loaded geometries and boundary conditions) problems. Commercial FEA software packages (e.g., Abaqus) have been used for aortic wall stress analysis of human aorta [8, 9, 19, 20]. The forward problem can be solved by these FEA packages with graphical interfaces. However, inverse problem solvers are not supported by these FEA packages and typically require an optimization framework that calls a FEA package to run FEA simulations iteratively to update parameters. These existing inverse strategies [21-24] heavily rely on FEA-updating that treats FEA as a blackbox, which results in well-known performance issues in either accuracy or speed.

In this study, we propose and develop PyTorch-FEA, a new library of FEA code and methods to solve forward and inverse problems with significantly improved performance. PyTorch [25] is an open source machine learning library, especially for developing deep neural networks (DNNs), and it can run on CPU and GPU. Besides those specific to DNNs, it has a high performance linear algebra module and many extensions [26], which are suitable for FEA implementation. The key benefit from PyTorch is the automatic differentiation mechanism, named autograd, which implements backpropagation from the loss, the fundamental algorithm to train DNNs. It enables the design of new loss functions for solving inverse problems using FEA principles, and the optimization can be done in a backpropagation fashion. It eliminates the need for hand-calculating derivatives (e.g., the stiffness matrix), which simplifies the implementation of FEA methods. Naturally, it enables a seamless integration of FEA and DNNs, as will be shown in this study.

We demonstrate the advantages of PyTorch-FEA in four basic applications for biomechanical analysis of human aorta, which are: (1) forward simulation of aorta inflation from zero-pressure to a target pressure, given a hyperelastic constitutive model with known material parameters; (2) inverse estimation of ex-vivo material parameters, given the zero-pressure geometry and a pressurized geometry; (3) inverse estimation of the zero-pressure geometry, given a pressurized geometry and the material parameters; and (4) inverse estimation of in-vivo material parameters, given two pressurized geometries at two pressure levels. To evaluate the performance in each application, we employ FEA-generated data for which "ground-truth" solutions are known.

The application-1 is the basic forward analysis to obtain the deformation and stress of the aorta at a target blood pressure, e.g., 18kPa representing the systolic pressure at hypertension stage-1 [27]**;** and if the stress is high, the aortic aneurysm rupture risk is also high. In this application, the initial undeformed geometry, blood pressure, and material parameters are given as the inputs, and the analysis outputs are the deformed geometry and stress at the target blood pressure. We compare PyTorch-FEA with the commercial software Abaqus and show that the discrepancy is negligible and PyTorch-FEA is much faster.

The application-2 arises from the study of human aorta material properties in inflation experiments [28]. In such an experiment, an aortic root, which is obtained from a cadaver heart, is inflated from 0 to ~26kPa by smoothly injecting saline solution to the root that is placed inside a container; markers are placed on the root surface to enable camera-based marker tracking; and then pressure-strain responses are obtained from the experiment data. Subsequently, tissue material parameters of a specific hyperelastic constitutive model can be determined by fitting the pressure-strain responses in an inverse analysis. In this application, the initial (i.e., undeformed, zero-pressure) geometry, the deformed geometry at a known blood pressure, and the form of the constitutive model are given as the inputs, and the inverse analysis estimates the material parameters of the constitutive model.

The application-3 arises from the fact that the geometry of the aorta in clinical images is in the pressurized state and therefore cannot be used as the initial, undeformed geometry in a forward FEA (e.g., application-1) to obtain stresses; and therefore the zero-pressure (i.e., undeformed) geometry needs to be estimated by using a deformed geometry at a known blood pressure and a specific constitutive model with known material parameters. This application also arises from the potential use of 3D-printed tissue-engineered aortic root as a replacement of the native aortic root [29]**.** In such case, the engineered aortic root is load-free and needs to match the deformed geometry of the native aortic root when pressurized. In this application, we will show two PyTorch-FEA inverse methods, and one of the methods combines FEA with DNNs to speed up the inverse analysis process while maintaining the same accuracy.

In the application-4, the material parameters of the aortic wall tissue are considered unknown and will be estimated by using two pressurized geometries at two different pressure levels, assuming the form of the constitutive model is given. In practice, the two geometries can be reconstructed from multiphase 3D CT images of a patient's aorta [24], one from the diastolic phase and the other from the systolic phase. By using patient-specific material parameters in a forward FEA, more accurate stress distribution can be obtained, which may improve the accuracy of aortic aneurysm diagnostics. In this application, we will demonstrate the capability of PyTorch-FEA using statical determinacy to solve this inverse problem, and we will use two different loss functions to show that the combination of residual force term and stress error term leads to better accuracy.

2. Materials and Methods

2.1 Material modeling of human aortic wall tissue

Soft biological tissues, such as human aortic wall tissue, comprise bundles of collagen fibers embedded in a ground matrix and can be regarded as fiber-reinforced composites, which exhibit nonlinear hyperelastic behaviors [15-18]. Modeling of the mechanical behavior (a.k.a. constitutive modeling) of hyperelastic tissues is often achieved by specifying the strain energy Ψ per unit undeformed volume as a function of strain invariants and fiber orientations. These strain invariants are derived from the deformation gradient tensor $\boldsymbol{F}$.

Among existing hyperelastic constitutive relations, the GOH model [30] is widely used for the aortic wall tissue. In this model, tissues are composed of a matrix material with two families of embedded fibers, each of which has a preferred direction. The strain energy density function is expressed by:

$$\Psi = \frac{C_{10}}{2}(\overline{\boldsymbol{I}}_1 - 3) + \frac{k_1}{2k_2}\sum_{i=1}^{2}[exp\{k_2[\kappa\overline{\boldsymbol{I}}_1 + (1-3\kappa)\overline{\boldsymbol{I}}_{4i} - 1]^2\} - 1] + \frac{1}{D}[\frac{J^2-1}{2} - \ln J] \tag{1}$$

The parameter C_{10} describes the matrix material. The parameters k_1 and k_2 describe the fiber properties. The deviatoric strain invariants $\overline{\boldsymbol{I}}_1$ and $\overline{\boldsymbol{I}}_{4i}$ characterize the deformation of the matrix and preferred fiber directions, respectively. The parameter κ describes the distribution of the fiber orientation. The parameter θ defines the mean local fiber direction in a local coordinate system. The material is assumed to be nearly incompressible, and D is a penalty constant. $J = \det(\boldsymbol{F})$.

Using a constitutive model, different stresses can be calculated, including the first PK stress tensor $\boldsymbol{P}$ and the Cauchy stress tensor $\boldsymbol{\sigma}$

$$\boldsymbol{P} = \frac{\partial\Psi}{\partial\boldsymbol{F}} \tag{2}$$

$$\boldsymbol{\sigma} = J^{-1}\boldsymbol{P}\boldsymbol{F}^T \tag{3}$$

In this work, we use the GOH model to demonstrate the capabilities of PyTorch-FEA. We select a representative set of the material parameters, $C_{10} = 54.73(kPa), k_1 = 2225.62(kPa), k_2 = 24.61, \kappa = 0.2494, \theta = 32.43(degree)$ from our previous studies [8, 9, 19, 31].

2.2 Finite element formulation in the context of solid mechanics

We provide a brief overview of the finite element formulation, and we try to keep the notations and descriptions to be as consistent as possible with those in the book [32]. The finite element formulation can be established by using the virtual work representation of the equilibrium equation, and the directional derivative of the total potential energy Π yields the principle of virtue work, which is given by

$$\Pi = \int_V \Psi\, dV - \int_V f_0 \cdot \boldsymbol{U} dV - \int_{\partial V} \boldsymbol{t}_0 \cdot \boldsymbol{U} dA \tag{4}$$

The body force f_0 is ignored for the applications of human aorta. $\boldsymbol{U}$ is the displacement of a material point from the initial position $\boldsymbol{X}$ to the current position $\boldsymbol{x}$, i.e., $\boldsymbol{x} = \boldsymbol{X} + \boldsymbol{U}$. V represents the undeformed body of the object. $\boldsymbol{t}_0$ is traction force on the surface ∂V of the object. The total potential energy Π can also be expressed using integrals evaluated on the deformed body. Given the constitutive model, the external forces, and the geometry of the object at the initial state, the total potential energy will decrease and approach a minimum at equilibrium.

In the FEA formulation, the domain of interest is discretized into finite elements. The position $\boldsymbol{x}$ of a material point inside an element with n_e nodes can be interpolated using the node positions $\{\boldsymbol{x}_i, i = 1, \dots, n_e\}$ and a shape function N_i:

$$\boldsymbol{x} = \sum_{i=1}^{n_e} N_i \boldsymbol{x}_i \tag{5}$$

The same interpolation can be applied to the initial position $\boldsymbol{X}$ and the displacement $\boldsymbol{U}$ of the same material point. As a result, the deformation gradient tensor at the material point located at $\boldsymbol{x}$ inside an element can be calculated by

$$\boldsymbol{F} = \sum_{i=1}^{n_e} \boldsymbol{x}_i \otimes \frac{\partial N_i}{\partial \boldsymbol{X}} \tag{6}$$

By setting the directional derivative of the total potential energy to zero (equivalent to the virtual work principal), the internal equivalent nodal force $\boldsymbol{T}_i^{(e)}$ and the external equivalent nodal force $\boldsymbol{G}_i^{(e)}$ can be obtained [32]:

$$\boldsymbol{T}_i^{(e)} = \int_{v^{(e)}} \boldsymbol{\sigma} \frac{\partial N_i}{\partial \boldsymbol{x}} dv \tag{7}$$

$$\boldsymbol{G}_i^{(e)} = \int_{v^{(e)}} N_i \boldsymbol{f} dv + \int_{\partial v^{(e)}} N_i \boldsymbol{t} da \tag{8}$$

In the above equation, $v^{(e)}$ represents an element of the deformed body and its index is e; $\boldsymbol{t}$ is traction force acting on the surface of the element; and $\boldsymbol{f}$ is body force that is ignored for the applications of human aorta. The integrals are evaluated numerically using the Gaussian integration method. For the aorta applications, the traction force vector $\boldsymbol{t}$ becomes $p\boldsymbol{n}$, where p is a scalar pressure value and $\boldsymbol{n}$ is the normal on the surface $\partial v^{(e)}$, and therefore the second term in Eq.(8) is the surface integral of the pressure acting in the normal direction, and its calculation details can be found in chapter 8 of the book [32]. The proximal and distal boundaries of the aorta are fixed with zero-displacement in this study.

A node may be shared by adjacent elements, and therefore, at each node, the forces from neighbor elements are added together to obtain the assembled internal equivalent nodal force $\boldsymbol{T}_i$ and the assembled external equivalent nodal force $\boldsymbol{G}_i$ at the node-i :

$$\boldsymbol{T}_i = \sum_e \boldsymbol{T}_i^{(e)} \tag{9}$$

$$\boldsymbol{G}_i = \sum_e \boldsymbol{G}_i^{(e)} \tag{10}$$

At the equilibrium state, the nodal residual force $\boldsymbol{R}_i$ is zero:

$$\boldsymbol{R}_i \triangleq \boldsymbol{T}_i - \boldsymbol{G}_i = 0 \tag{11}$$

The relationship among Π, $\boldsymbol{R}_i$, and the displacement $\boldsymbol{U}_i$ of the node-i is given by

$$\Pi'(\boldsymbol{U}_i) \triangleq \frac{\partial \Pi}{\partial \boldsymbol{U}_i} = \boldsymbol{R}_i \tag{12}$$

Commercial FEA software packages (e.g., Abaqus) implement forward FEA solvers to determine the deformed geometry, given the initial (i.e., undeformed) geometry, external loading, boundary conditions, and a specific constitutive model with known parameters. A known variable in the forward problem becomes unknown (i.e., to be solved) in an inverse problem. Solution of inverse problems requires substantial changes to the FEA implementation, which is often not feasible in commercial FEA software packages. The applications 2-4 mentioned in the introduction section are inverse problems and are readily tractable by functionalities in Pytorch-FEA.

2.3 The design of PyTorch-FEA

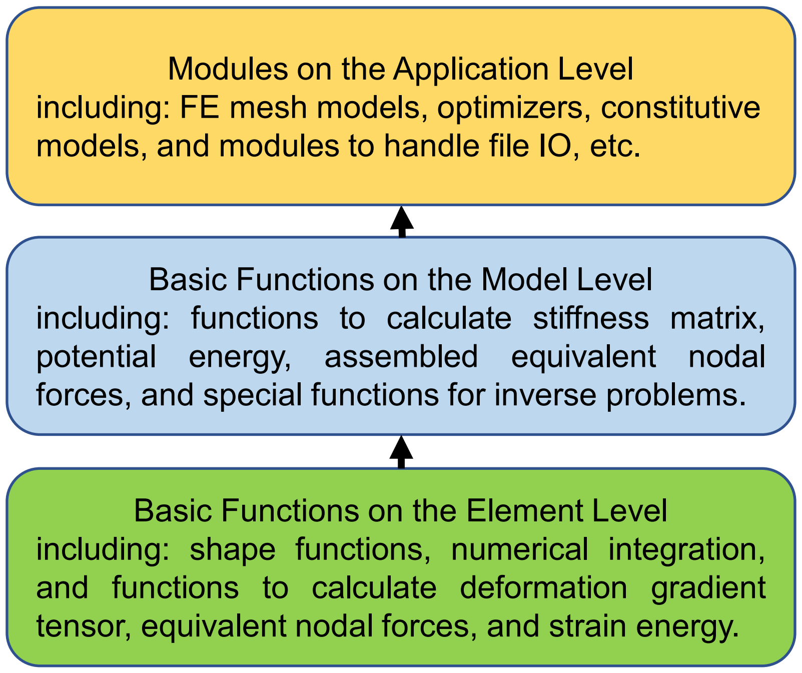
Modules on the Application Level
including: FE mesh models, optimizers, constitutive models, and modules to handle file IO, etc.
Basic Functions on the Model Level
including: functions to calculate stiffness matrix, potential energy, assembled equivalent nodal forces, and special functions for inverse problems.
Basic Functions on the Element Level
including: shape functions, numerical integration, and functions to calculate deformation gradient tensor, equivalent nodal forces, and strain energy.

Figure 1. The three-level design of Pytorch-FEA

The 3-level structure of PyTorch-FEA is shown in Figure 1. On the element level, it has basic functions to perform interpolation using shape functions (Eq. (5)), calculate the deformation gradient tensor (Eq.(6)), perform numerical integration to compute the equivalent nodal forces (Eq.(7) and Eq.(8)) and the strain energy stored in an element (i.e., $\int_{V(e)} \Psi \, dV$). We implemented the hexahedral element that is similar to the C3D8 element in Abaqus. To avoid the volumetric locking issue associated with nearly incompressible materials (e.g., Eq.(1)), we implemented selective-reduced integration [33] for the numerical integrations. We note that there are other methods to stabilize FEA for hyperelastic large deformation problems involving incompressible or nearly incompressible materials [34]. We choose selective-reduced integration because Abaqus uses a similar method. In addition to the hexahedral element, we also implemented functions to handle blood pressure acting on the inner surface of the aortic wall, and functions for surface integrals are implemented for calculating the equivalent pressure (i.e., external) force at each node of a quadrilateral element. It also defines the interfaces to constitutive models, which must provide three functions: one to calculate the strain energy density (e.g., Eq.(1)) at a material point (e.g., integration point) given deformation gradient tensor $\boldsymbol{F}$ and element orientation as inputs, and the other two to calculate the stress tensors $\boldsymbol{P}$ and $\boldsymbol{\sigma}$. We note that PyTorch provides automatic differentiation (a.k.a. autograd) for almost all operations, thus, the stress tensor $\boldsymbol{P}$ can be computed directly by autograd using Eq.(2). These element-level functions are implemented in batch-mode to handle all of the elements at the same time.

On the model level, PyTorch-FEA has basic functions to calculate the total potential energy (Eq.(4)), assemble the equivalent nodal forces (Eq.(9) and Eq.(10)) at each node, and calculate the stiffness matrix that is useful for optimization. The stiffness matrix is the second-order derivative of the total potential energy with respect to the displacement, which is also called Hessian matrix. And therefore, it is the first-order derivative of the residual force with respect to the displacement. The size of the full stiffness matrix is $3N \times 3N$, where N is the total number of nodes. The stiffness matrix is very sparse. A naive implementation would be directly using PyTorch autograd to compute the derivative and obtain a dense matrix that will consume lots of CPU or GPU memory, leading to OOM (out-of-memory) error. Instead, we provide a memory-efficient implementation to assemble the sparse matrix from individual elements.

On the application level, PyTorch-FEA currently only supports the four applications of aorta biomechanical analysis. It has FE models of the aortic wall, which are implemented as Python class, encapsulating all necessary functions to calculate a variety of quantities (e.g., stress, deformation, strain energy, etc). The inner surface needs to be specified, i.e., the quad elements on which the pressure load will be applied. Currently, it only uses one type of boundary condition: zero displacement at each inlet/outlet boundary of the aorta. It also includes optimizers and constitutive models, as well as modules to handle file IO. The four applications are explained in the next sections.

2.4 The application of aorta inflation simulation

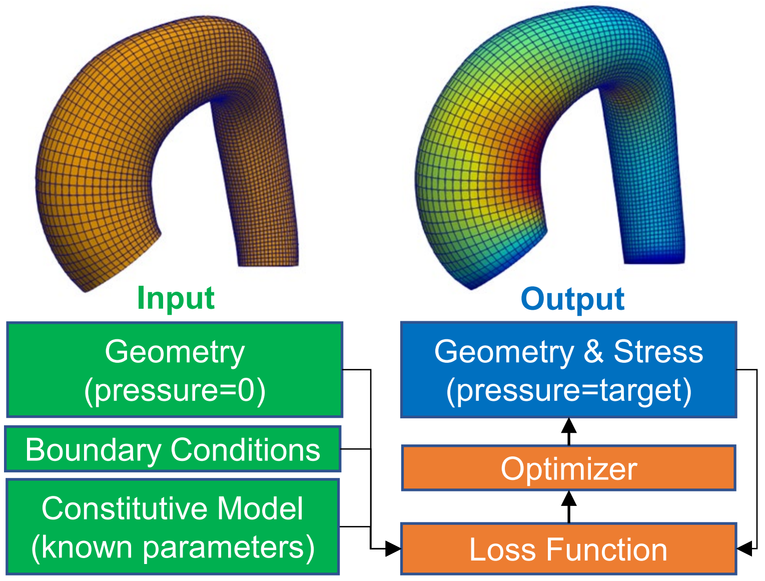

Figure 2. Forward simulation of aorta inflation

As shown in Figure 2, in this application, the initial (i.e., undeformed) geometry, blood pressure, and the constitutive model with known parameters (Section 2.1) are given as the inputs, and the objective is to obtain the deformed geometry at the specified blood pressure and then calculate stress. Here, we consider blood pressure distribution on the inner wall to be uniform, although Pytorch-FEA supports non-uniform distributions. The blood pressure value is set to 18kPa, representing the systolic pressure at hypertension stage-1 [27]. If the stress at the hypertension stage-1 is high, the aneurysm rupture risk is also high.

We use the total potential energy in Eq.(4) as a pseudo loss function of the displacement field $\mathbf{U}$ of the nodes. We note that Eq.(4) is not used to calculated the residual force by autograd; and instead, the residual force is calculated by using Eq.(11). For optimization, we implemented a quasi-newton method based on the L-BFGS optimizer [35]. Briefly, the Taylor expansion could be applied to the total potential energy:

$$\Pi\left(\mathbf{U}_{(k)} + \boldsymbol{d}_{(k)}\right) \approx \Pi\left(\mathbf{U}_{(k)}\right) + \boldsymbol{d}_{(k)}^{T}\Pi'\left(\mathbf{U}_{(k)}\right) + \frac{1}{2}\boldsymbol{d}_{(k)}^{T}\mathbf{H}_{(k)}\boldsymbol{d}_{(k)} \tag{13}$$

In the above equation, $\mathbf{U}_{(k)}$ represents the displacement field (assembled into a vector) of all the free nodes at the current iteration. $\boldsymbol{d}_{(k)}$ is a small increment. $\mathbf{H}_{(k)}$ is known as the tangent stiffness matrix at the current iteration-k, and $\mathbf{H}_{(k)} = \frac{\partial \boldsymbol{R}_{(k)}}{\partial \mathbf{U}_{(k)}}$, where $\boldsymbol{R}_{(k)}$ is the residual force field of the nodes at the current iteration. If the boundary nodes are fixed with zero-displacement, the corresponding rows and columns in $\mathbf{H}_{(k)}$ need to be removed. The optimal increment $\boldsymbol{d}_{(k)}$, which minimizes the quadratic form (i.e., the right-side of Eq.(13)), is known as

$$\boldsymbol{d}_{(k)} = -(\mathbf{H}_{(k)})^{-1}\boldsymbol{R}_{(k)} \tag{14}$$

In practice, the inverse $\left(\mathbf{H}_{(k)}\right)^{-1}$ is never calculated, and the following sparse linear system of equations is solved to obtain $\boldsymbol{d}_{(k)}$:

$$\mathbf{H}_{(k)}\boldsymbol{d}_{(k)} = -\boldsymbol{R}_{(k)} \tag{15}$$

We used a Python package PyPardiso [36] as the sparse solver. Also, a line search is applied to adjust the increment:

$$\mathbf{U}_{(k+1)} \leftarrow \alpha \cdot \boldsymbol{d}_{(k)} + \mathbf{U}_{(k)} \tag{16}$$

where the nonnegative scalar α is auto-adjusted to minimize the loss function.

A full-Newton method will solve Eq.(15) in every iteration, and therefore it will be very computationally expensive and time-consuming. The L-BFGS method [35] avoids Eq.(15) by approximating the Hessian inverse, and therefore each iteration is fast, but it may need numerous iterations to converge due to the imprecise approximation. As a combination of the two methods, we re-initialize L-BFGS with Eq.(15) every 20 iterations, and this combined method works well in this application.

We use the following metric as the primary metric to determine the convergence of optimization:

$$R_{max} = \max_{n}\left\{\frac{\|\boldsymbol{R}_n\|}{\operatorname*{mean}_{e,i}\left\|\boldsymbol{T}_i^{(e)}\right\|}, n = 1, \dots, N\right\} \tag{17}$$

We use $\|\blacksquare\|$ to denote scalar absolute value, vector L2 norm, or matrix Frobenius norm in this paper. $\boldsymbol{R}_n$ is the residual force (Eq.(11)) at the node-*n* at the current iteration, and the total number of nodes of an aorta mesh is *N*. If $R_{max} < threshold$, then the optimization is considered converged. In this application, the threshold is 0.005.

According to Abaqus documentation, a similar metric is used by Abaqus FEA solver to determine convergence. We also use two secondary metrics to monitor the optimization process: $\Delta \mathrm{U}_{ratio} = \left\| \mathbf{U}_{(k+1)} - \mathbf{U}_{(k)} \right\| / \left\| \mathbf{U}_{(k+1)} \right\|$ and $\underset{n}{\mathrm{mean}} \|\boldsymbol{R}_n\|$ at the current iteration. In our application, $\Delta \mathrm{U}_{ratio}$ is negligibly small when $R_{max} < 0.005$.

To improve convergence, we use pseudo time steps similar to those in Abaqus. The simulation duration is mapped to a time interval between 0 and 1, and the interval is divided into small steps. The external load is increased gradually by $pressure_{(t)} = t \times pressure$ where $pressure$ is the target pressure (e.g., 18kPa), and t is the current time in the interval. For each time point t, the optimizer runs until $R_{max} < threshold$ to obtain the optimal displacement under $pressure_{(t)}$. If it does not converge at the current time point, the program will go back to the previous time point with a randomly-chosen time step. If it converges, then the time step will increase but not exceed the pre-defined max step (e.g., 0.01 in this application).

To demonstrate the capability of PyTorch-FEA for this application, we use seven representative initial geometries sampled from a statistical shape model (SSM) [19]. The SSM is built on 60 real aorta geometries in our previous studies [31], and it captures the major shape variations. Among the seven geometries, six of those are the mean shape ± 1.5 × mode (mode 1, 2 or 3) of shape variations, and the other one is the mean shape. Figures of the initial (i.e., undeformed, zero-pressure) geometries are provided in the appendix. The results are in Section 3, which show that the discrepancy between PyTorch-FEA and Abaqus is negligible.

2.5 The application of ex-vivo material parameter estimation

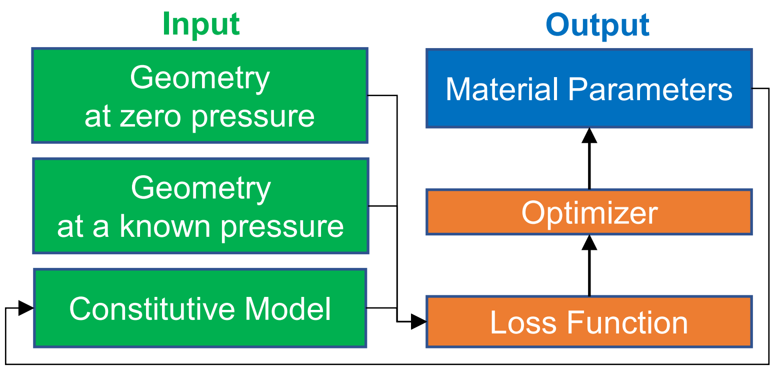

Figure 3. Inverse estimation of ex-vivo material parameters

In this application, the initial (i.e., undeformed, zero-pressure) geometry, the deformed geometry at a known blood pressure, and the form of the constitutive model are given, and the objective is to estimate the material parameters of the constitutive model. We use the same seven test cases in Section 2.4 to demonstrate this application, and the deformed geometries are obtained by the PyTorch-FEA forward method.

To achieve the objective, we define the loss function to be:

$$L(C_{10}, k_1, k_2, \kappa, \theta) = \sum_{n=1}^{N} \|\boldsymbol{R}_n\|^2 \tag{18}$$

We use L-BFGS with line search as the optimizer. The optimization converged in a few seconds for each test case, and the difference between the estimated parameters and the true parameters is negligible. Because of the automatic differentiation capability of Pytorch-FEA, the gradient of the loss with respect to each parameter can be exactly calculated (e.g., $\frac{\partial L}{\partial C_{10}}$), and therefore, we can use gradient-based optimization that runs much faster than non-gradient-based optimization that is typically used in FEA-updating inverse methods (e.g., [21-23]) that may use the following loss function:

$$L_{FEA\_updating}(C_{10}, k_1, k_2, \kappa, \theta) = \sum_{n=1}^{N} \|\widetilde{\boldsymbol{x}}_n - \boldsymbol{x}_n\|^2 \tag{19}$$

where $\{\widetilde{\boldsymbol{x}}_n, n = 1, \dots, N\}$ is an estimation of the deformed-geometry obtained by running a forward FEA with the current estimations of the material parameters. Because the loss $L_{FEA\_updating}$ is not differentiable with respect to the material parameters, the FEA-updating inverse method needs numerous forward FEA simulations.

2.6 The application of zero-pressure geometry estimation

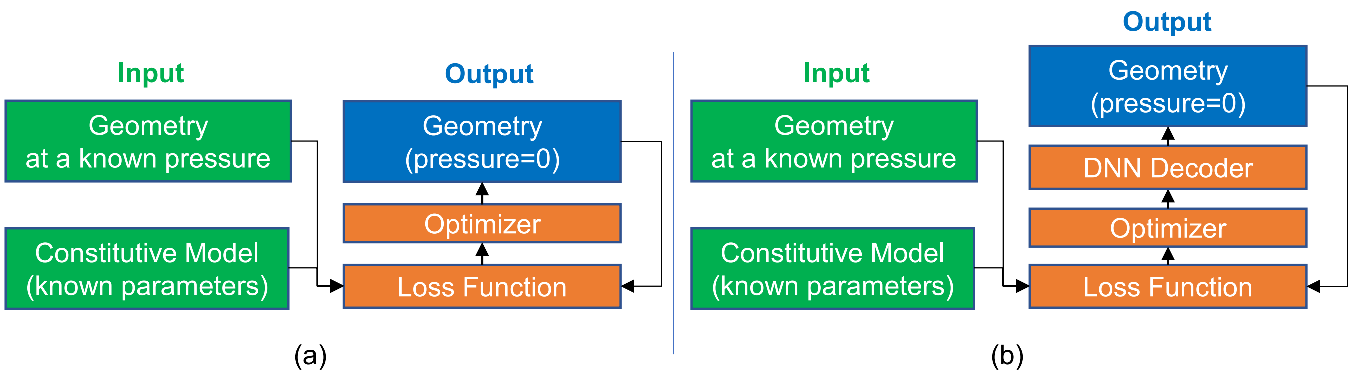
Input
Geometry at a known pressure
Constitutive Model (known parameters)
Output
Geometry (pressure=0)
Optimizer
Loss Function
(a)
Input
Geometry at a known pressure
Constitutive Model (known parameters)
Output
Geometry (pressure=0)
DNN Decoder
Optimizer
Loss Function
(b)

Figure 4. Two PyTorch-FEA inverse methods for the estimation of zero-pressure geometry. (a) The PyFEA-P0 method. (b) The PyFEA-NN-P0 method that has a DNN decoder to generate a displacement field.

In this application, the zero-pressure (i.e., initial, undeformed) geometry needs to be estimated by using the information about the deformed geometry at a known blood pressure and the constitutive model with known parameters. To demonstrate this application, we use the same seven test cases in Section 2.5 and set the pressure to be 10kPa, representing in vivo diastolic pressure.

In a previous study [19], we used the backward-displacement method [37] to obtain the zero-pressure geometry of human aorta, and it was implemented by using Abaqus, Python, and Matlab. The key iteration step of the method is given by:

$$\widetilde{\mathbf{X}}_{(k+1)} = \widetilde{\mathbf{X}}_{(k)} + 0.5(\mathbf{x} - \tilde{\mathbf{x}}_{(k)}) \tag{20}$$

where $\widetilde{\mathbf{X}}_{(k)}$ represents the estimated zero-pressure geometry at the iteration k, $\tilde{\mathbf{x}}_{(k)}$ represents the deformed-geometry obtained by a forward FEA using $\widetilde{\mathbf{X}}_{(k)}$ as the initial geometry, and $\mathbf{x}$ represents the true deformed-geometry that is deformed from the true but unknown zero-pressure geometry. At the initialization step, $\widetilde{\mathbf{X}}_{(1)}$ is equal to $\mathbf{x}$. The total number of iterations is fixed to 20, leading to long computation time based on our previous study [19]. We note that k is the index of an iteration (i.e., a forward FEA simulation), not the index of a node. Although this backward-displacement method is easily implemented into PyTorch-FEA, we employ two better inverse strategies in Pytorch-FEA.

To take advantage of the automatic differentiation capability of Pytorch-FEA, we propose two new methods: PyFEA-P0 and PyFEA-NN-P0, where P0 stands for pressure zero, NN stands for neural network, and PyFEA refers to Pytorch-FEA. As shown in Figure 4, the PyFEA-P0 method is solely based on the FEA principles, and the PyFEA-NN-P0 method has a DNN component to speed up the analysis.

The procedure of PyFEA-P0 is very similar to the forward FEA of the application-1 in Section 2.4. In PyFEA-P0, the analysis process starts with a deformed geometry $\mathbf{x}$ (known), a pressure (known), and a set of material parameters (known); the unknown displacement field $\mathbf{U}$ is the primary variable to be optimized, and the following equations are used to obtain $\widetilde{\mathbf{X}}_{(k+1)}$, an estimation of the zero-pressure geometry at iteration $k+1$:

$$\widetilde{\mathbf{X}}_{(k+1)} = \mathbf{x} - \mathbf{U}_{(k+1)} \tag{21}$$

$$\mathbf{x} = detach\left(\widetilde{\mathbf{X}}_{(k+1)}\right) + \mathbf{U}_{(k+1)} \tag{22}$$

The displacement field $\mathbf{U}_{(k+1)}$ is obtained by using Eq.(16), and the initial values of the displacements are zero. The Eq.(22) does nothing mathematically, but it is necessary to obtain the correct derivatives related to the displacement. The autograd mechanism works with a computational graph that is automatically constructed and represents a sequence of operations to compute the loss. By detaching $\widetilde{\mathbf{X}}_{(k+1)}$ from the computational graph, Eq.(22) will ensure the correct derivatives as if a forward FEA is performed by starting from $\widetilde{\mathbf{X}}_{(k+1)}$. Similarly, the element orientation (a function of $\widetilde{\mathbf{X}}_{(k+1)}$) variable needs to be detached from the graph. To improve convergence, pseudo time steps counting down from 1 to 0 can be used.

The PyFEA-NN-P0 method needs an autoencoder DNN that can be explained by two functions: an encoder function: $\boldsymbol{c} = f_{encoder}(\{\boldsymbol{U}_n, n = 1,2, \dots N\})$ and a decoder function: $\{\boldsymbol{U}_n, n = 1,2, \dots N\} = f_{decoder}(\boldsymbol{c})$. Here, $\boldsymbol{U}_n$ is the displacement of the node-n from the zero-pressure state to the pressurized state, and N is the total number of nodes of an aorta mesh. Briefly, the input to the encoder is the entire displacement field of an aorta mesh, and the displacement field is compressed into a lower dimensional code vector $\boldsymbol{c}$. Given the code vector as input, the decoder recovers the displacement field. In this application, the encoder is implemented by an MLP (multi-layer perceptron) that has two hidden layers with 128 units per layer and Softplus activations, and the dimensions of code vector $\boldsymbol{c}$ is 16; and the decoder is implemented by another MLP mirroring the structure of the encoder MLP. The autoencoder is trained on 273 samples generated by the SSM (Section 2.4), which do not include the seven test cases. After training, the decoder will be used in the optimization process, and the code vector $\boldsymbol{c}$ becomes the primary variable to be optimized. For each of the test cases, the optimal value of $\boldsymbol{c}$ should lead to the minimum of the loss (i.e., total potential energy). The L-BFGS optimizer with line search is used, and the gradient can be calculated by

$$\frac{\partial \Pi}{\partial \boldsymbol{c}} = \sum_{n=1}^{N} \frac{\partial \Pi}{\partial \boldsymbol{U}_n} \frac{\partial \boldsymbol{U}_n}{\partial \boldsymbol{c}} = \sum_{n=1}^{N} \boldsymbol{R}_n \frac{\partial \boldsymbol{U}_n}{\partial \boldsymbol{c}} \tag{23}$$

where $\frac{\partial \boldsymbol{U}_n}{\partial \boldsymbol{c}}$ is the derivative of the decoder output with respect to its input and it is auto-calculated by autograd. We provide more details of this method in the appendix.

2.7 The application of in-vivo material parameter estimation

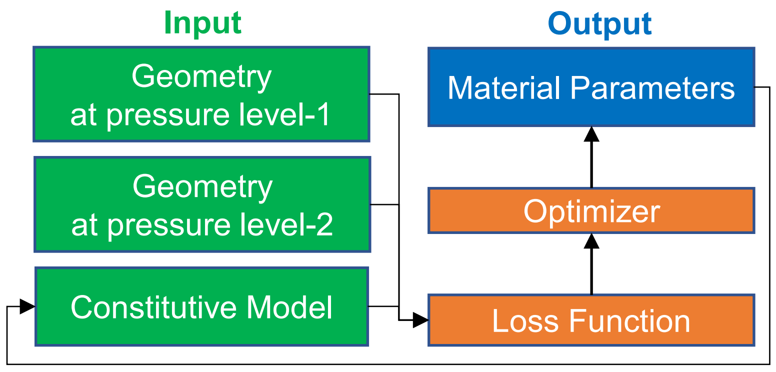

Figure 5. Inverse estimation of in-vivo material parameters

In this application, the material parameters of the aortic wall tissue are considered unknown and need to be estimated by using two pressurized geometries at two different pressure levels, and the form of the constitutive model is given by Eq.(1). In practice, the two geometries can be reconstructed from gated 3D CT scans of a patient's aorta [24], at two cardiac phases (diastole and systole). The identified patient-specific material parameters can be used in a forward FEA to calculate patient-specific wall stress under an elevated blood pressure, which can help to make more accurate risk assessment of TAA [31, 38]. To demonstrate the capability of PyTorch-FEA for this application, we use the same seven test cases in Section 2.4, and for each test case, the two pressurized geometries are obtained by the PyTorch-FEA forward method using pressure values of 10kPa (diastole) and 16kPa (systole).

To achieve the objective, we employ the following loss function by requiring diminishing residual force and matching stresses according to the principle of statical determinacy [39-41]:

$$L\left(C_{10}, k_1, k_2, \kappa, \theta, \left\{\boldsymbol{V}_m^{pL}\right\}_{m=1}^{M}\right) = \sum_{n=1}^{N} \left\|\boldsymbol{R}_n^{pL}\right\|^2 + \sum_{n=1}^{N} \left\|\boldsymbol{R}_n^{pH}\right\|^2$$

$$+\sum_{m=1}^{M}\left\|\boldsymbol{\sigma}_m^{pL}-\widehat{\boldsymbol{\sigma}}_m^{pL}\right\|^2+\sum_{m=1}^{M}\left\|\boldsymbol{\sigma}_m^{pH}-\widehat{\boldsymbol{\sigma}}_m^{pH}\right\|^2 \quad (24)$$

In the loss function, $\boldsymbol{R}_n^{pL}$ is the residual force at the node-n at the diastolic phase (pL stands for pressure low), and $\boldsymbol{R}_n^{pH}$ is the residual force at the node-n at the systolic phase (pH stands for pressure high). $\boldsymbol{\sigma}_m^{pL}$ and $\boldsymbol{\sigma}_m^{pH}$ are stress tensors at the element-m, which are computed by using Eq.(3) at the diastolic phase and the systolic phase, respectively. N is the number of nodes, and M is the number of elements of an aorta mesh. $\boldsymbol{R}_n^{pL}$, $\boldsymbol{R}_n^{pH}$, $\boldsymbol{\sigma}_m^{pL}$, and $\boldsymbol{\sigma}_m^{pH}$ are nonlinear functions of the unknown material parameters $\{C_{10}, k_1, k_2, \kappa, \theta\}$. The stress tensors $\widehat{\boldsymbol{\sigma}}_m^{pL}$ and $\widehat{\boldsymbol{\sigma}}_m^{pH}$ are directly obtained using the principle of statical determinacy [39-41] with the two pressurized geometries and are independent of the material parameters. $\boldsymbol{V}_m^{pL}$ is the left stretch tensor at the element-m from the zero-pressure state to the diastolic phase, and it is unknown. It is related to the deformation gradient tensor $\boldsymbol{F}_m^{pL}$ by the polar-decomposition: $\boldsymbol{F}_m^{pL}=\boldsymbol{V}_m^{pL}\boldsymbol{Q}_m^{pL}$, where $\boldsymbol{Q}_m^{pL}$ is rotation. We use the L-BFGS method with line search as the optimizer for this application.

From the two pressurized geometries, the deformation gradient tensor $\boldsymbol{F}_m^{pLH}$ of the element-m from the diastolic phase to the systolic phase (pLH stands for pressure from low to high) can be directly calculated. Then, the deformation gradient tensor $\boldsymbol{F}_m^{pH}$ of the element-m from the zero-pressure state to the systolic phase can be obtained by $\boldsymbol{F}_m^{pH}=\boldsymbol{F}_m^{pLH}\boldsymbol{V}_m^{pL}\boldsymbol{Q}_m^{pL}$. Although $\boldsymbol{Q}_m^{pL}$ is unknown, it will not affect stress calculations using the constitutive model, and the explanation is provided in the appendix. Thus, the stress tensors ($\boldsymbol{\sigma}_m^{pL}$ and $\boldsymbol{\sigma}_m^{pH}$) and the residual forces ($\boldsymbol{R}_n^{pL}$ and $\boldsymbol{R}_n^{pH}$) can be calculated by using the current estimations of the deformation gradient tensors and the current estimations of the material parameters. The boundary nodes and elements are removed from the loss function because $\widehat{\boldsymbol{\sigma}}_m^{pL}$ and $\widehat{\boldsymbol{\sigma}}_m^{pH}$ are inaccurate near the boundaries due to the known boundary effect.

To obtain $\widehat{\boldsymbol{\sigma}}_m^{pL}$ and $\widehat{\boldsymbol{\sigma}}_m^{pH}$ using the principle of statical determinacy for each test case, we first apply the inverse method PyFEA-P0 (Section 2.6) to recover the zero-pressure geometries, for which the material parameters are set to $C_{10}=10000(kPa), k_1=0(kPa), k_2=1, \kappa=0, \theta=0(degree)$, representing a very stiff material. Then, the zero-pressure geometries are inflated by the PyTorch-FEA forward method (Section 2.4) to the diastolic phase and the systolic phase to obtain the stresses $\hat{\boldsymbol{\sigma}}_m^{pL}$ and $\hat{\boldsymbol{\sigma}}_m^{pH}$, respectively. In the appendix, we provide more explanations about the principle of statical determinacy for this application.

If we remove the residual force terms, i.e., $\sum_{n=1}^{N}\left\|\boldsymbol{R}_n^{pL}\right\|^2 + \sum_{n=1}^{N}\left\|\boldsymbol{R}_n^{pH}\right\|^2$, from the loss function, then the loss function will become similar to the loss in our previous method [42] that has a very complex implementation using co-rotational coordinate system for problem formulation and using Abaqus, Python, and Matlab for programming. In the result section, we will show that the residual force term improves the estimation accuracy.

3. Results

The results of the application-1 in Section 2.4 are reported in Table 1. The differences between Pytorch-FEA and Abaqus for forward simulations in this application are quantified for each test case by using three metrics related to geometry and stress:

$$node\_diff = \frac{\frac{1}{N}\sum_{n=1}^{N}\left\|\boldsymbol{x}_n^{(PyTorchFEA)} - \boldsymbol{x}_n^{(Abaqus)}\right\|}{\max\limits_{n}\left\|\boldsymbol{x}_n^{(Abaqus)} - \boldsymbol{X}_n\right\|} \tag{25}$$

$$stress\_diff = \frac{\frac{1}{M}\sum_{m=1}^{M}\left\|\boldsymbol{s}_m^{(PyTorchFEA)} - \boldsymbol{s}_m^{(Abaqus)}\right\|}{\max\limits_{m}\left\|\boldsymbol{s}_m^{(Abaqus)}\right\|} \tag{26}$$

$$peak\_stress\_diff = \frac{\left|\max\limits_{m}\left\|\boldsymbol{s}_m^{(PyTorchFEA)}\right\| - \max\limits_{m}\left\|\boldsymbol{s}_m^{(Abaqus)}\right\|\right|}{\max\limits_{m}\left\|\boldsymbol{s}_m^{(Abaqus)}\right\|} \tag{27}$$

We use von Mises stress as $\boldsymbol{s}_m^{(*)}$ in Eq.(26) and Eq.(27) because the stresses in Abaqus are always in the local coordinate systems of the elements in the deformed geometry, and the stresses in Pytoch-FEA are always in the global coordinate system. Pytorch-FEA and Abaqus were tested on the same computer equipped with an Intel i7-8700 CPU and two Nvidia Titan V GPUs. Abaqus only uses the CPU with FP64 data type (i.e., double-precision). PyTorch-FEA uses the CPU and one GPU with FP64 data type. The sparse solver in Pytorch-FEA runs on CPU and consumes about 1/3 of the total time cost. We also measured the relative time cost of Pytorch-FEA for each test case: relative time cost = (time cost of Pytorch-FEA)/(time cost of Abaqus), which is shown in Table 1.

Table 1: The differences between PyTorch-FEA and Abaqus in the application-1 in Section 2.4

Test Case Index	$node\_diff$	$stress\_diff$	$peak\_stress\_diff$	relative time cost
1	0.00107	0.00197	0.00988	0.66373
2	0.00081	0.00172	0.00264	0.67728
3	0.00044	0.00164	0.01510	0.63004
4	0.00125	0.00119	0.00041	0.63752
5	0.00194	0.00201	0.00412	0.62740
6	0.00232	0.00187	0.00114	0.64171
7	0.00176	0.00146	0.00108	0.62408

The results of the application-2 in Section 2.5 are reported in Table 2. The error of each estimated material parameter is quantified by the following metric:

$$mat\_error = |a_{estimation} - a_{true}|/a_{max} \tag{28}$$

where a represents a parameter and a_{max} is the upper bound of the parameter. The lower bound of each parameter is close to zero. According to our previous studies [31, 38], $C_{10\_\max} = 120$ (kPa), $k_{1\_\max} = 6000$ (kPa), $k_{2\_\max} = 60$, $\kappa_{max} = 1/3$, and $\theta_{max} = 90$ (degree). For each test case, the time cost is about 10 seconds.

Table 2: The errors of the estimated material parameters in the application-2 in Section 2.5

Test Case Index	C_{10} error	k_1 error	k_2 error	κ error	θ error
1	5.25×10^{-6}	1.35×10^{-5}	7.82×10^{-4}	1.18×10^{-5}	2.74×10^{-7}
2	2.97×10^{-5}	4.07×10^{-5}	2.37×10^{-4}	1.86×10^{-5}	5.69×10^{-8}
3	2.65×10^{-5}	2.11×10^{-5}	1.51×10^{-6}	3.00×10^{-6}	4.79×10^{-8}
4	2.82×10^{-5}	6.74×10^{-5}	6.58×10^{-4}	4.02×10^{-5}	4.21×10^{-7}

5	4.96×10^{-5}	2.61×10^{-7}	4.61×10^{-4}	8.40×10^{-6}	1.73×10^{-7}
6	1.12×10^{-4}	8.09×10^{-5}	2.84×10^{-4}	2.15×10^{-5}	4.36×10^{-8}
7	4.30×10^{-5}	1.73×10^{-5}	2.55×10^{-5}	5.34×10^{-7}	8.22×10^{-8}

The results of the application-3 in Section 2.6 are reported in Table 3, comparing the performance of the three inverse methods, including backward-displacement (BD), PyFEA-p0, and PyFEA-NN-p0. For each test case, we use the following metrics to measure the performance of a method:

$$node\_error = \frac{\frac{1}{N}\sum_{n=1}^{N}\left\|\boldsymbol{X}_n^{(inverse)} - \boldsymbol{X}_n\right\|}{\max_n \|\boldsymbol{x}_n - \boldsymbol{X}_n\|} \tag{29}$$

$$stress\_error = \frac{\frac{1}{M}\sum_{m=1}^{M}\left\|\boldsymbol{s}_m^{(inverse)} - \boldsymbol{s}_m\right\|}{\max_m |\boldsymbol{s}_m|} \tag{30}$$

$$peak\_stress\_error = \frac{\left|\max_m \left\|\boldsymbol{s}_m^{(inverse)}\right\| - \max_m \|\boldsymbol{s}_m\|\right|}{\max_m \|\boldsymbol{s}_m\|} \tag{31}$$

$\boldsymbol{x}_n$ is the position of the node-*n* of the aorta mesh model at the diastolic phase. $\boldsymbol{X}_n^{(inverse)}$ is the estimated position of the node-*n* at the zero-pressure state. $\boldsymbol{s}_m$ is the true stress at the element-*m* at the diastolic phase, and $\boldsymbol{s}_m^{(inverse)}$ is the estimated stress from an inverse method. To be consistent with the evaluation in application-1, we use von Mises stress in Eq.(30) and Eq.(31). To compare the methods, we calculate the average of each metric (averaged across the seven cases) as showing in Table 3. The performance of each individual method for the seven test cases is reported in the Appendix. The fastest method is PyFEA-NN-P0 that only takes less than 10 seconds to handle a test case. The slowest method is PyFEA-P0 that may take hours for a test case. The speed of the BD method is between the other two, but its accuracy is relatively lower.

Table 3: Performance comparison of the three inverse methods in the application-3 in Section 2.6

Method	average $node\_error$	average $stress\_error$	average $peak\_stress\_error$

PyFEA-NN-P0	0.00190	0.00035	0.00066
PyFEA-P0	0.01254	0.00026	0.00070
BD	0.08942	0.00334	0.00911

The results of the application-4 in Section 2.7 are reported in Table 4, comparing the performance of the two inverse methods, one with the residual force terms in the loss (Method-A, the second row in Table 4), and the other using only the stress terms in the loss (Method-B, the third row in Table 4). The metric of Eq.(28) is also used in this application. The performance of each individual method for the seven test cases is reported in the Appendix. PyFEA-p0 is used in this application, and it only takes a few minutes to handle a test case. The difference between the estimated stresses ($\widehat{\boldsymbol{\sigma}}_m^{pL}$ and $\widehat{\boldsymbol{\sigma}}_m^{pH}$) and the true stresses is about 3%, and the details are reported in the Appendix. With residual force terms in the loss function, Method-A has lower errors in C_{10}, k_2, κ, and θ. The two methods have roughly the same time cost.

Table 4: Performance comparison of the two inverse methods in the application-4 in Section 2.7

Method	C_{10} error (average)	k_1 error (average)	k_2 error (average)	κ error (average)	θ error (average)
with residual force terms	0.03921	0.02140	0.03015	0.01755	0.00551
without residual force terms	0.11146	0.01825	0.03438	0.01836	0.00840

4. Discussion

The capability of PyTorch-FEA has been demonstrated on the four applications. The application-1 shows that PyTorch-FEA forward analysis solution is almost the same as Abaqus, and the small discrepancy (< 0.5% in stress as shown in Table 1) could be caused by the differences in the optimizers. PyTorch-FEA eliminates the need for manual-calculating derivatives (e.g., calculating stiffness matrix, calculating stress by differentiating the strain energy density function, etc), and therefore the implementation is clean and efficient for GPU. The current implementation of the PyTorch-FEA forward method is about 1.5 times faster than Abaqus for the seven test cases, and the speed is limited by the sparse solver that runs on CPU and accounts for 1/3 of the time cost. We will upgrade the sparse solver with a GPU implementation [43] in our future work to further reduce the time cost. Also, if provided with the current generation of GPUs, e.g., Nvidia H100, which is magnitudes faster than Titan V GPU used in this study, PyTorch-FEA can become much faster. The applications 2-4 show the advantage of PyTorch-FEA for solving the inverse problems. For each inverse problem, we only need to define an appropriate loss function, and the optimization is similar to training a DNN by backpropagation from the loss, which enables us to define new or improved loss functions (Eq.(18), Eq.(23), and Eq.(24)) to achieve better accuracy or faster speed.

PyTorch-FEA enables a natural marriage of FEA and DNNs, which enjoys the benefits of both: accurate and fast, as shown in the application-3 for inverse estimation of the zero-pressure geometry. In our inverse method PyFEA-NN-P0, the DNN decoder is a generative model that can generate a displacement field from an input code vector. Then, the solution space is reduced from the dimension of 30000 (1000 nodes in a mesh model, and each node has three displacement values in 3D) to a much lower dimension of 16. As a result, the optimization can complete in seconds, magnitudes faster than the other methods. A common issue in ML is that an ML model may not work well if the input sample is out of the distribution (OOD) of the training data. The generalization capability of an ML model is largely limited by the training data. The application-3 in this study is a feasibility demonstration of combining DNN with FEA. In this study, we used the SSM to generate the zero-pressure geometries for the training and test cases, which are all from the same distribution; and the pressurized geometries have the same loading and boundary conditions. To improve the generalization ability of the method to handle cases that are not covered by the training data, a larger training dataset with diverse geometries and blood pressure levels will be needed. If a much larger training dataset is available, the decoder could be replaced by a generative adversarial network (GAN) [44] or a diffusion model [45], which are state-of-the-art generative models for image and natural language applications. Besides the PyFEA-NN-P0 method in this application, new FEA-DNN integrations could be developed for the other applications [46-48] to improve speed or accuracy, or both.

Recently, we and other research groups have developed constitutive models based on DNNs [49-55], which perform better than the expert-prescribed models in some applications. However, it can be difficult to implement a complex DNN as a user subroutine in the programming language that is compatible with commercial software packages, e.g., FORTRAN for Abaqus UMAT. PyTorch-FEA could potentially serve as a platform for developing and testing DNN-based constitutive models.

Physics-informed neural networks (PINNs) [56], which attracted a lot of attentions in the field of scientific machine learning, offers a direct solution to PDEs by using DNNs and physics principles. Although PINNs are intriguing, current PINNs have not led to performance improvement compared with commercial FEA methods in the solid mechanics domain and suffer from the error/instability [57] in the deformation gradient tensor calculated by differentiating a DNN with respect to the input: $\boldsymbol{F} = \boldsymbol{I} + \frac{\partial h(\boldsymbol{X})}{\partial \boldsymbol{X}}$ where $h(\boldsymbol{X})$ is a DNN to output displacement. The instability is not a numerical defect of autograd, and it is closely related to the well-known issue of DNN robustness [58], and it is known that [59, 60] gradients of a DNN can change dramatically if the input changes only a little or even if the same DNN is trained twice on the same data.

Several open-source FEA software packages/programs are available, such as FEBio [61], FEAP [62], and FEniCS [63], which employ traditional Newton-Rapson forward FEA solvers. FEAP and FEniCS are general-purpose FEA programs. FEBio specializes in solving nonlinear large deformation (forward) problems in biomechanics and biophysics, and it works for the application-1 but does not have functionalities for the inverse applications. The official FEniCS code does not have off-the-shelf functionalities to support complex hyperelastic constitutive models with selective-reduced integration, geometry, loading and boundary condition in this study. A recent work enhanced FEniCS capability for a compressible hyperelastic model [64] to solve inverse problems using the FEA updating method (Eq. (19)). FEBio and FEAP do not support autograd. There has been an attempt to combine FEniCS and PyTorch to solve simple PDEs [65], but it is unclear if it could be extended to complex cases, such as the applications in this study.

The current PyTorch-FEA only supports the four applications for biomechanical analysis of the human aorta. The basic functions on the element level and the model level are general FEA functions, not specifically designed for the four applications, which enable extensions of PyTorch-FEA, especially for vascular biomechanics (e.g., growth and remodeling [66, 67]). There could be two future extensions of PyTorch-FEA inverse methods: one to build robustness against noises, which requires the modification of the loss function to model the noise properties; and the other to estimate nonhomogeneous elastic property distribution, as studied by Mei et al. in [68, 69], which requires the modification of the loss function to take into account the properties of the distribution (e.g., spatial smoothness). We will continue developing PyTorch-FEA for biomechanics applications.

5. Conclusion

We have presented PyTorch-FEA, a new library of FEA code and methods, representing a new approach to develop FEA methods to forward and inverse problems in solid mechanics. We have demonstrated its capabilities in the four basic applications in aorta biomechanics. PyTorch-FEA eases the development of new inverse methods and enables a natural integration of FEA and DNNs, which will have numerous potential applications.

Code Availability

Once the paper is published, the source code will be uploaded to https://github.com/liangbright/pytorch_fea

Acknowledgement

This work was supported in part by the NIH grant R01HL158829.

Appendix

1. The zero-pressure geometries of the seven test cases

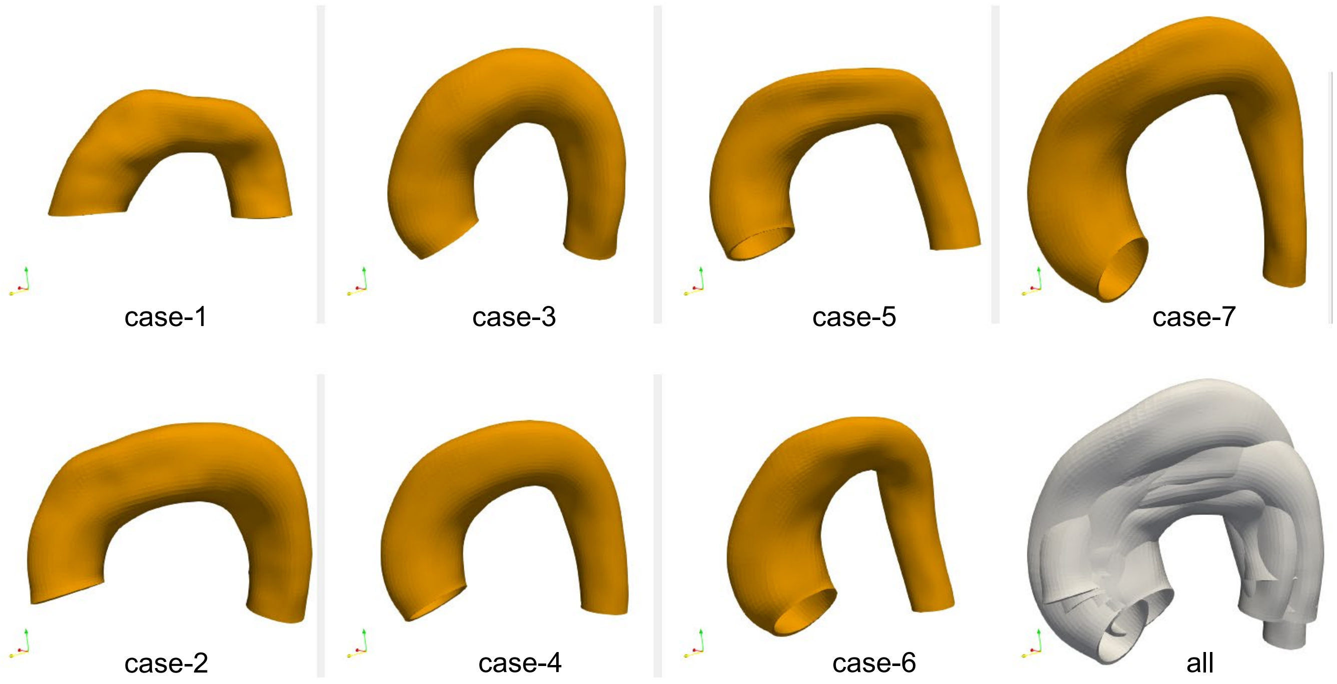

Figure A1. The zero-pressure geometries of the seven test cases. Case-4 is the mean shape.

2. Explanation of $\boldsymbol{Q}_m^{pL}$ in the application-4 in Section 2.7

In many models (e.g., Eq.(1)), the strain energy density Ψ is an explicit function of the strain invariants of the right Cauchy–Green tensor $\boldsymbol{C} \triangleq \boldsymbol{F}^T\boldsymbol{F}$, and the strain invariants are independent of any rotation. We can calculate $\boldsymbol{C}_m^{pH} \triangleq \boldsymbol{F}_m^{pH^T}\boldsymbol{F}_m^{pH} = \left(\boldsymbol{Q}_m^{pL}\right)^T\left(\boldsymbol{F}_m^{pLH}\boldsymbol{V}_m^{pL}\right)^T\boldsymbol{F}_m^{pLH}\boldsymbol{V}_m^{pL}\boldsymbol{Q}_m^{pL}$, and therefore the strain invariants of $\boldsymbol{C}_m^{pH}$ are independent of the rotation $\boldsymbol{Q}_m^{pL}$. Similarly, the strain invariants of $\boldsymbol{C}_m^{pL}$ are independent of the rotation $\boldsymbol{Q}_m^{pL}$.

The GOH model in Eq.(1) has terms related to fiber orientation $\boldsymbol{a}$ (a vector in the global coordinate system). $\boldsymbol{a} = \mathrm{R}_m^{p0}\mathbf{a}^{p0}$, where R_m^{p0} is the coordinate transform matrix (a.k.a. orientation of the element-*m*) from a local coordinate system to the global coordinate system. $\mathbf{a}^{p0}$ is the fiber orientation in the local coordinate system of the zero-pressure geometry, and it is determined by the material parameter θ. The GOH model needs to calculate

$\boldsymbol{a}^T\boldsymbol{C}_m^{pL}\boldsymbol{a}$ and $\boldsymbol{a}^T\boldsymbol{C}_m^{pH}\boldsymbol{a}$ along with several other strain invariants of $\boldsymbol{C}_m^{pL}$ and $\boldsymbol{C}_m^{pH}$. By using the polar decomposition, we have $\boldsymbol{C}_m^{pL} = (\boldsymbol{Q}_m^{pL})^T(\boldsymbol{V}_m^{pL})^T\boldsymbol{V}_m^{pL}\boldsymbol{Q}_m^{pL}$, and $\boldsymbol{C}_m^{pH} = (\boldsymbol{Q}_m^{pH})^T(\boldsymbol{V}_m^{pH})^T\boldsymbol{V}_m^{pH}\boldsymbol{Q}_m^{pH}$. Thus, we obtain $\boldsymbol{a}^T\boldsymbol{C}_m^{pL}\boldsymbol{a} = (\mathbf{a}^{p0})^T(\mathrm{R}_m^{p0})^T\boldsymbol{C}_m^{pL}\mathrm{R}_m^{p0}\mathbf{a}^{p0} = (\boldsymbol{Q}_m^{pL}\mathrm{R}_m^{p0}\mathbf{a}^{p0})^T(\boldsymbol{V}_m^{pL})^T\boldsymbol{V}_m^{pL}\boldsymbol{Q}_m^{pL}\mathrm{R}_m^{p0}\mathbf{a}^{p0}$. Let R_m^{pL} denote the element orientation at the low-pressure state, and then we have $\mathrm{R}_m^{pL} = \boldsymbol{Q}_m^{pL}\mathrm{R}_m^{p0}$ in theory (it may have numerical errors) because $\boldsymbol{Q}_m^{pL}$ is the rotation from the zero-pressure state to the low-pressure state. Thus, $\boldsymbol{a}^T\boldsymbol{C}_m^{pL}\boldsymbol{a} = (\mathrm{R}_m^{pL}\mathbf{a}^{p0})^T(\boldsymbol{V}_m^{pL})^T\boldsymbol{V}_m^{pL}\mathrm{R}_m^{pL}\mathbf{a}^{p0}$, which is independent of $\boldsymbol{Q}_m^{pL}$. Similarly, it can be shown that $\boldsymbol{a}^T\boldsymbol{C}_m^{pH}\boldsymbol{a}$ is independent of $\boldsymbol{Q}_m^{pL}$.

In summary $\boldsymbol{Q}_m^{pL}$ does not affect stress calculations using the constitutive model. R_m^{pL} is used as the element orientation input for the constitutive model.

3. The performance of the three inverse methods on individual cases in application-3 in Section 2.6

Table A1. Performance of PyFEA-NN-p0 in the application-3

Test Case Index	$node\_error$	$stress\_error$	$peak\_stress\_error$
1	0.001603687	0.000505738	0.000738402
2	0.001609170	0.000380529	0.000390757
3	0.001657492	0.000350096	0.001384317
4	0.001324879	0.000219505	0.000480732
5	0.002291675	0.000376495	0.000533053
6	0.002071800	0.000311183	0.000127871
7	0.002709812	0.000312835	0.000984382

Table A2. Performance of PyFEA-p0 in the application-3

Test Case Index	$node\_error$	$stress\_error$	$peak\_stress\_error$
1	0.013546776	0.000472815	0.001112776

2	0.010208852	0.000231789	0.000156242
3	0.008520643	0.000258453	0.001786788
4	0.007296290	0.000188554	0.000659276
5	0.026151690	0.000238295	0.000080900
6	0.005821294	0.000222569	0.000575342
7	0.016261040	0.000236795	0.000507866

Table A3. Performance of Backward-Displacement in the application-3

Test Case Index	$node\_error$	$stress\_error$	$peak\_stress\_error$
1	0.050443336	0.002912068	0.007059155
2	0.099223304	0.004954056	0.015746499
3	0.097816046	0.004365298	0.028571723
4	0.088351709	0.002286753	0.005556760
5	0.093116105	0.002741628	0.005139562
6	0.096268331	0.002496273	0.001162921
7	0.100715101	0.003628388	0.000532503

4. The performance of the two inverse methods on individual cases in application-4 in Section 2.7

Table A4: The performance of the Method-A in the application-2 in Section 2.5

Test Case Index	C_{10} error	k_1 error	k_2 error	κ error	θ error
1	0.129247378	0.042951700	0.064761514	0.044543974	0.019333531
2	0.000622965	0.022907529	0.051915732	0.026669237	0.009406890

3	0.006252636	0.005289165	0.024329692	0.000735397	0.000075500
4	0.055838874	0.011756801	0.006615650	0.008653229	0.001687726
5	0.015083297	0.023531461	0.022397862	0.014278869	0.001958761
6	0.025283741	0.017563589	0.016656474	0.010606607	0.000387365
7	0.042121716	0.025813757	0.024350831	0.017392986	0.005729018

Table A5: The performance of the Method-B in the application-2 in Section 2.5

Test Case Index	C_{10} error	k_1 error	k_2 error	κ error	θ error
1	0.192870881	0.057150165	0.126684566	0.054823763	0.021151645
2	0.180261673	0.036096861	0.043758695	0.039467320	0.019296666
3	0.118783674	0.008901363	0.011829653	0.008886206	0.003399421
4	0.138038347	0.009141652	0.023494923	0.002153123	0.000827182
5	0.082047669	0.003260920	0.021085351	0.008920925	0.005473181
6	0.029286580	0.011930520	0.004169498	0.012040075	0.006770422
7	0.038954431	0.001235340	0.009627799	0.002200260	0.001858200

Table A6: The accuracy of $\widehat{\boldsymbol{\sigma}}_m^{pL}$ and $\widehat{\boldsymbol{\sigma}}_m^{pH}$ in application-4 in Section 2.7

Test Case Index	$\widehat{\boldsymbol{\sigma}}_m^{pL}$ error	$\widehat{\boldsymbol{\sigma}}_m^{pH}$ error
1	0.042568215	0.042068338
2	0.034829138	0.032056521
3	0.030086746	0.028548364
4	0.023311289	0.022395104

5	0.024368868	0.022454050
6	0.022995795	0.020644430
7	0.019870676	0.018354943

Note: the error is calculated using the metric in Eq.(26)

The estimation errors shown in Tables A4 and A5 are caused by the errors in $\hat{\boldsymbol{\sigma}}_m^{pL}$ and $\hat{\boldsymbol{\sigma}}_m^{pH}$.

5. The statical determinacy principle

As discussed in the literature [39-41], there exist a family of structures in which the stresses depend on the load, the boundary conditions, and the geometry at the loaded state, but not the material property, and such structures are considered statically determinate. The simplest case would be a thin cylinder wall with uniform internal pressure, and the hoop stress is equal to pressure × radius / thickness (i.e., Law of Laplace), which does not depend on the material property. Although truly statically determinate structures are rare in the real world, blood vessels, such as aorta, are approximately statically determinate so that the stress distribution depends weakly on the material properties. A study of aortic aneurysm stress [39-41] shows that: given the pressure and the loaded geometry, the inversely-computed aortic wall stress field on the loaded geometry is insensitive to constitutive models and material parameters. Thus, to obtain aortic wall stress field, the patient-specific (i.e., true) material property is unnecessary.

To obtain $\hat{\boldsymbol{\sigma}}$ (representing $\hat{\boldsymbol{\sigma}}_m^{pL}$ at diastolic phase or $\hat{\boldsymbol{\sigma}}_m^{pH}$ at systolic phase) using the principle of statical determinacy for each test case, we first apply the inverse method PyFEA-P0 (Section 2.6) to recover a zero-pressure geometry, for which the material parameters are $C_{10} = 10000(kPa), k_1 = 0(kPa), k_2 = 1, \kappa = 0, \theta = 0(degree)$. Then, the zero-pressure geometry is inflated by the PyTorch-FEA forward method (Section 2.4) with the corresponding pressure level (diastolic or systolic pressure) to obtain the stress field $\hat{\boldsymbol{\sigma}}$ on the loaded/deformed geometry. The used material parameters are not the true parameters, and the recovered zero-pressure geometry is not the true geometry, but the computed stress $\hat{\boldsymbol{\sigma}}$ will be accurate to some extent. Because aorta is not truly statically determinate, there are errors in $\hat{\boldsymbol{\sigma}}$ obtained by using the statical determinacy principle, as shown in Table A.6. An example is shown in Figure A2 to visualize the stress fields.

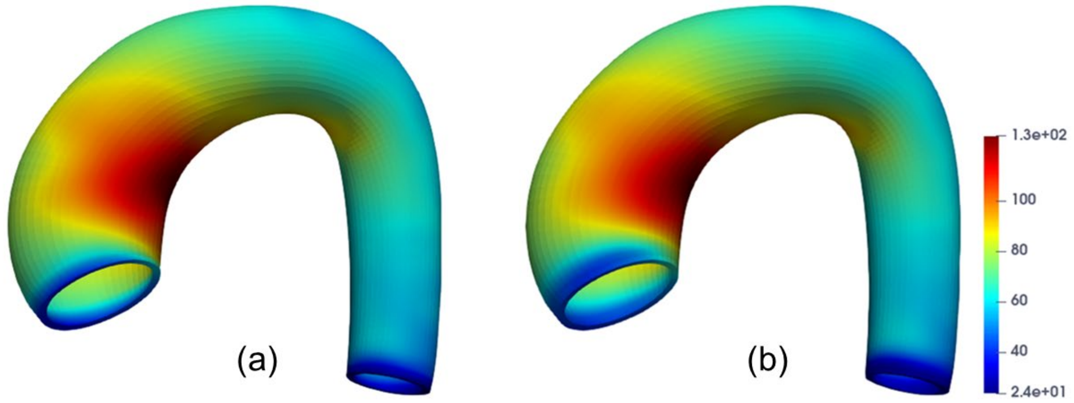

Figure A2. An example of using statical determinacy to obtain the stress field of an aorta at a loaded state with the pressure of 10 kPa. (a) The true Von Mises stress field obtained by a forward FEA with true zero-pressure geometry and true material parameters. (b) The Von Mises stress field obtained by using the principle of statical determinacy. The two stress fields are very similar, but there are noticeable discrepancies near the two boundaries. The unit is kPa in the color bar.

6. The steps of the PyFEA-NN-P0 inverse method

The DNN inside the PyFEA-NN-P0 method needs to be trained before the method can be used to recover zero-pressure geometries. The first step is to generate data for training and testing. We used a statistical shape model (SSM) to generate zero-pressure geometries, and the generated geometries are divided into a training set (273 cases) and a test set (7 cases). The SSM is built on 60 real aorta geometries in our previous studies [31], and it captures the major shape variations. Among the seven geometries in the test set, six of those are the mean shape $\pm$ 1.5 $\times$ mode (mode 1, 2 or 3) of shape variations, and the other one is the mean shape, which are shown in Figure A1.

The second step is to train the neural network. The structure of the neural network is shown in Figure A3, which has an encoder and a decoder. The input to the encoder is a displacement field, $\boldsymbol{U} = \boldsymbol{x} - \boldsymbol{X}$, and the output from the decoder is a displacement field $\widehat{\boldsymbol{U}}$ that should be identical to the input $\boldsymbol{U}$. The loss function for training the neural network is simply the difference between $\widehat{\boldsymbol{U}}$ and $\boldsymbol{U}$, which can be implemented using the standard mean-squared-error loss.

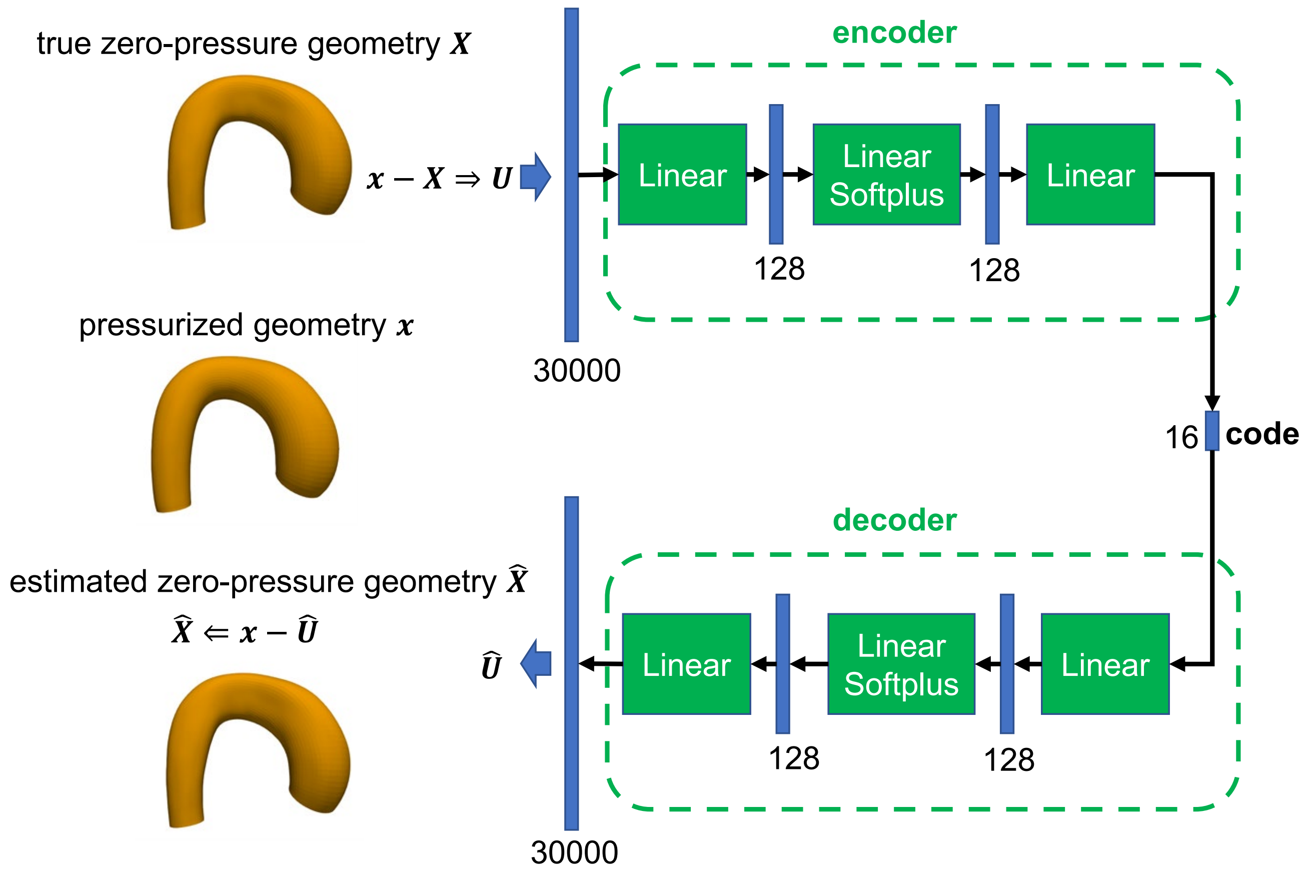
true zero-pressure geometry $\boldsymbol{X}$
$\boldsymbol{x}-\boldsymbol{X}\Rightarrow\boldsymbol{U}$
pressurized geometry $\boldsymbol{x}$
estimated zero-pressure geometry $\widehat{\boldsymbol{X}}$
$\widehat{\boldsymbol{X}}\Leftarrow\boldsymbol{x}-\widehat{\boldsymbol{U}}$
$\widehat{\boldsymbol{U}}$
encoder
Linear
Linear Softplus
Linear
128
128
30000
16
code
decoder
Linear
Linear Softplus
Linear
128
128
30000

Figure A3. The encoder-decoder structure of the neural network. An aorta mesh has 10000 nodes, and each node has a 3D position, and therefore the input to the encoder is a vector with length equal to 30000, containing all the values of the displacement field. "Linear" denotes a fully connected layer. "Softplus" denotes the activation function.

After the neural network is trained on the training set, the encoder will be discarded, and only the decoder will be used in the PyFEA-NN-P0 method. The decoder is the "DNN" in Figure 4(b). When applying the method to a test case, the code is the primary variable to be optimized, which is a vector with length of 16. Given a code as the input to the decoder, the decoder will output a displacement field that is used to recover the zero-pressure geometry. In the PyFEA-NN-P0 method shown in Figure 4(b), the task of the optimizer is to find the optimal code such that the loss function (i.e., the total potential energy) is minimized.

Reference

[1] F. Auricchio, M. Conti, S. Morganti, and A. Reali, "Simulation of transcatheter aortic valve implantation: a patient-specific finite element approach," *Computer Methods in Biomechanics and Biomedical Engineering,* vol. 17, no. 12, pp. 1347-57, 2014, doi: 10.1080/10255842.2012.746676.

[2] C. Capelli, G. M. Bosi, E. Cerri, J. Nordmeyer, T. Odenwald, P. Bonhoeffer, F. Migliavacca, A. M. Taylor, and S. Schievano, "Patient-specific simulations of transcatheter aortic valve stent implantation," *Medical & Biological Engineering & Computing,* vol. 50, no. 2, pp. 183-92, Feb 2012, doi: 10.1007/s11517-012-0864-1.

[3] P. de Jaegere, G. De Santis, R. Rodriguez-Olivares, J. Bosmans, N. Bruining, T. Dezutter, Z. Rahhab, N. El Faquir, V. Collas, B. Bosmans, B. Verhegghe, C. Ren, M. Geleinse, C. Schultz, N. van Mieghem, M. De Beule, and P. Mortier, "Patient-Specific Computer Modeling to Predict Aortic Regurgitation After Transcatheter Aortic Valve Replacement," *JACC: Cardiovascular Interventions,* vol. 9, no. 5, pp. 508-12, Mar 14 2016, doi: 10.1016/j.jcin.2016.01.003.

[4] H. A. Dwyer, P. B. Matthews, A. Azadani, L. Ge, T. S. Guy, and E. E. Tseng, "Migration forces of transcatheter aortic valves in patients with noncalcific aortic insufficiency," *Journal of Thoracic and Cardiovascular Surgery,* vol. 138, no. 5, pp. 1227-33, Nov 2009, doi: 10.1016/j.jtcvs.2009.02.057.

[5] S. Morganti, N. Brambilla, A. S. Petronio, A. Reali, F. Bedogni, and F. Auricchio, "Prediction of patient-specific post-operative outcomes of TAVI procedure: The impact of the positioning strategy on valve performance," *Journal of Biomechanics,* vol. 49, no. 12, pp. 2513-9, Aug 16 2016, doi: 10.1016/j.jbiomech.2015.10.048.

[6] S. Morganti, M. Conti, M. Aiello, A. Valentini, A. Mazzola, A. Reali, and F. Auricchio, "Simulation of transcatheter aortic valve implantation through patient-specific finite element analysis: two clinical cases," *J Biomech,* vol. 47, no. 11, pp. 2547-55, Aug 22 2014, doi: 10.1016/j.jbiomech.2014.06.007.

[7] W. Sun, C. Martin, and T. Pham, "Computational modeling of cardiac valve function and intervention," *Annual Review of Biomedical Engineering,* vol. 16, pp. 53-76, Jul 11 2014, doi: 10.1146/annurev-bioeng-071813-104517.

[8] C. Martin, W. Sun, and J. Elefteriades, "Patient-specific finite element analysis of ascending aorta aneurysms," *American Journal of Physiology-Heart and Circulatory Physiology,* vol. 308, no. 10, pp. H1306-H1316, 2015, doi: 10.1152/ajpheart.00908.2014.

[9] C. Martin, W. Sun, T. Pham, and J. Elefteriades, "Predictive biomechanical analysis of ascending aortic aneurysm rupture potential," *Acta Biomaterialia,* vol. 9, no. 12, pp. 9392-9400, 2013/12/01/ 2013, doi: https://doi.org/10.1016/j.actbio.2013.07.044.

[10] B. A. Zambrano, N. A. McLean, X. Zhao, J.-L. Tan, L. Zhong, C. A. Figueroa, L. C. Lee, and S. Baek, "Image-based computational assessment of vascular wall mechanics and hemodynamics in pulmonary arterial hypertension patients," *Journal of Biomechanics,* vol. 68, pp. 84-92, 2018/02/08/ 2018, doi: https://doi.org/10.1016/j.jbiomech.2017.12.022.

[11] S. Jamaleddin Mousavi, R. Jayendiran, S. Farzaneh, S. Campisi, M. Viallon, P. Croisille, and S. Avril, "Coupling hemodynamics with mechanobiology in patient-specific computational models of ascending thoracic aortic aneurysms," *Computer Methods and Programs in Biomedicine,* vol. 205, p. 106107, 2021/06/01/ 2021, doi: https://doi.org/10.1016/j.cmpb.2021.106107.

[12] CDC. "Centers for Disease Control and Prevention, National Center for Injury Prevention and Control, WISQARS Leading Causes of Death Reports, 1999 - 2018: https://webappa.cdc.gov/cgi-bin/broker.exe." (accessed Sept 15, 2020).

[13] T. Faggion Vinholo, M. A. Zafar, B. A. Ziganshin, and J. A. Elefteriades, "Nonsyndromic Thoracic Aortic Aneurysms and Dissections—Is Screening Possible?," *Seminars in Thoracic and Cardiovascular Surgery,* 2019/06/15/ 2019, doi: https://doi.org/10.1053/j.semtcvs.2019.05.035.

[14] A. Verstraeten, I. Luyckx, and B. Loeys, "Aetiology and management of hereditary aortopathy," *Nature Reviews Cardiology,* Review Article vol. 14, p. 197, 01/19/online 2017, doi: 10.1038/nrcardio.2016.211.

[15] S. Sherifova and and G. A. Holzapfel, "Biochemomechanics of the thoracic aorta in health and disease," *Progress in Biomedical Engineering,* vol. 2, no. 3, p. 032002, 2020/07/14 2020, doi: 10.1088/2516-1091/ab9a29.

[16] T. C. Gasser, "The Biomechanical Rupture Risk Assessment of Abdominal Aortic Aneurysms—Method and Clinical Relevance," in *Biomedical Technology: Modeling, Experiments and Simulation*, P. Wriggers and T. Lenarz Eds. Cham: Springer International Publishing, 2018, pp. 233-253.

[17] T. C. Gasser, "Biomechanical Rupture Risk Assessment: A Consistent and Objective Decision-Making Tool for Abdominal Aortic Aneurysm Patients," *Aorta (Stamford),* vol. 4, no. 2, pp. 42-60, 2016, doi: 10.12945/j.aorta.2015.15.030.

[18] S. Sherifova and G. A. Holzapfel, "Biomechanics of aortic wall failure with a focus on dissection and aneurysm: A review," *Acta Biomaterialia,* vol. 99, pp. 1-17, 2019/11/01/ 2019, doi: https://doi.org/10.1016/j.actbio.2019.08.017.

[19] L. Liang, M. Liu, C. Martin, J. A. Elefteriades, and W. Sun, "A Machine Learning Approach to Investigate the Relationship between Shape Features and Numerically Predicted Risk of Ascending Aortic Aneurysm," *Biomechanics and Modeling in Mechanobiology,* vol. 16, no. 5, pp. 1519--1533, 2017.

[20] F. Lorandon, S. Rinckenbach, N. Settembre, E. Steinmetz, L. Salomon Du Mont, and S. Avril, "Stress Analysis in AAA does not Predict Rupture Location Correctly in Patients with Intraluminal Thrombus," *Annals of Vascular Surgery,* vol. 79, pp. 279-289, 2022/02/01/ 2022, doi: https://doi.org/10.1016/j.avsg.2021.08.008.

[21] A. Wittek, K. Karatolios, P. Bihari, T. Schmitz-Rixen, R. Moosdorf, S. Vogt, and C. Blase, "In vivo determination of elastic properties of the human aorta based on 4D ultrasound data," *Journal of the Mechanical Behavior of Biomedical Materials,* vol. 27, pp. 167-183, 11// 2013, doi: http://dx.doi.org/10.1016/j.jmbbm.2013.03.014.

[22] A. Wittek, W. Derwich, K. Karatolios, C. P. Fritzen, S. Vogt, T. Schmitz-Rixen, and C. Blase, "A finite element updating approach for identification of the anisotropic hyperelastic properties of normal and diseased aortic walls from 4D ultrasound strain imaging," *Journal of the Mechanical Behavior of Biomedical Materials,* vol. 58, pp. 122-138, 5// 2016, doi: http://dx.doi.org/10.1016/j.jmbbm.2015.09.022.

[23] M. Liu, L. Liang, and W. Sun, "Estimation of in vivo mechanical properties of the aortic wall: A multi-resolution direct search approach," *Journal of the Mechanical Behavior of Biomedical Materials,* vol. 77, pp. 649-659, 2018/01/01/ 2018, doi: https://doi.org/10.1016/j.jmbbm.2017.10.022.

[24] M. Liu, L. Liang, F. Sulejmani, X. Lou, G. Iannucci, E. Chen, B. Leshnower, and W. Sun, "Identification of in vivo nonlinear anisotropic mechanical properties of ascending thoracic aortic aneurysm from patient-specific CT scans," *Scientific Reports,* vol. 9, no. 1, p. 12983, 2019.

[25] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Z. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," *Neural Information Processing Systems,* pp. 8024-8035, 2019.

[26] M. Fey and J. E. Lenssen, "Fast Graph Representation Learning with PyTorch Geometric," *ICLR Workshop on Representation Learning on Graphs and Manifolds,* 2019.

[27] A. Hooker, K. G. Buda, and M. Pasha, "Managing stage 1 hypertension: Consider the risks, stop the progression," *Cleveland Clinic Journal of Medicine,* vol. 89, no. 5, p. 244, 2022, doi: 10.3949/ccjm.89a.21101.

[28] K. Li, Q. Wang, T. Pham, and W. Sun, "Quantification of structural compliance of aged human and porcine aortic root tissues," *Journal of Biomedical Materials Research Part A,* vol. 102, no. 7, pp. 2365-74, Jul 2014, doi: 10.1002/jbm.a.34884.

[29] Z. Qian, K. Wang, S. Liu, X. Zhou, V. Rajagopal, C. Meduri, J. R. Kauten, Y. H. Chang, C. Wu, C. Zhang, B. Wang, and M. A. Vannan, "Quantitative Prediction of Paravalvular Leak in Transcatheter Aortic Valve Replacement Based on Tissue-Mimicking 3D Printing," *JACC Cardiovasc Imaging,* vol. 10, no. 7, pp. 719-731, Jul 2017, doi: 10.1016/j.jcmg.2017.04.005.

[30] T. C. Gasser, R. W. Ogden, and G. A. Holzapfel, "Hyperelastic modelling of arterial layers with distributed collagen fibre orientations," *Journal of The Royal Society Interface,* vol. 3, no. 6, pp. 15-35, 2005/09/28 2005, doi: 10.1098/rsif.2005.0073.

[31] M. Liu, L. Liang, Q. Zou, Y. Ismail, X. Lou, G. Iannucci, E. P. Chen, B. G. Leshnower, J. A. Elefteriades, and W. Sun, "A probabilistic and anisotropic failure metric for ascending thoracic aortic aneurysm risk assessment," *Journal of the Mechanics and Physics of Solids,* vol. 155, p. 104539, 2021/10/01/ 2021, doi: https://doi.org/10.1016/j.jmps.2021.104539.

[32] J. Bonet and R. D. Wood, *Nonlinear Continuum Mechanics for Finite Element Analysis*. Cambridge University Press, 2008.

[33] S. Doll, K. Schweizerhof, R. Hauptmann, and C. Freischläger, "On volumetric locking of low‐order solid and solid‐shell elements for finite elastoviscoplastic deformations and selective reduced integration," *Engineering Computations,* vol. 17, pp. 874-902, 2000.

[34] A. M. Maniatty, Y. Liu, O. Klaas, and M. S. Shephard, "Higher order stabilized finite element method for hyperelastic finite deformation," *Computer Methods in Applied Mechanics and Engineering,* vol. 191, no. 13, pp. 1491-1503, 2002/01/18/ 2002, doi: https://doi.org/10.1016/S0045-7825(01)00335-8.

[35] D. C. Liu and J. Nocedal, "On the limited memory BFGS method for large scale optimization," *Mathematical Programming,* vol. 45, no. 1, pp. 503-528, 1989/08/01 1989, doi: 10.1007/BF01589116.

[36] A. Haas, "PyPardiso," *GitHub repository https://github.com/haasad/PyPardisoProject,* 2023.

[37] J. Bols, J. Degroote, B. Trachet, B. Verhegghe, P. Segers, and J. Vierendeels, "A computational method to assess the in vivo stresses and unloaded configuration of patient-specific blood vessels," *Journal of Computational and Applied Mathematics,* vol. 246, pp. 10-17, 2013/07/01/ 2013, doi: https://doi.org/10.1016/j.cam.2012.10.034.

[38] M. Liu, L. Liang, Y. Ismail, H. Dong, X. Lou, G. Iannucci, E. P. Chen, B. G. Leshnower, J. A. Elefteriades, and W. Sun, "Computation of a probabilistic and anisotropic failure metric on the aortic wall using a machine learning-based surrogate model," *Computers in Biology and Medicine,* vol. 137, p. 104794, 2021/10/01/ 2021, doi: https://doi.org/10.1016/j.compbiomed.2021.104794.

[39] K. Miller and J. Lu, "On the prospect of patient-specific biomechanics without patient-specific properties of tissues," *Journal of the Mechanical Behavior of Biomedical Materials,* vol. 27, pp. 154-166, 2013/11/01/ 2013, doi: https://doi.org/10.1016/j.jmbbm.2013.01.013.

[40] G. R. Joldes, K. Miller, A. Wittek, and B. Doyle, "A simple, effective and clinically applicable method to compute abdominal aortic aneurysm wall stress," *Journal of the Mechanical Behavior of Biomedical Materials,* vol. 58, pp. 139-148, 2016/05/01/ 2016, doi: https://doi.org/10.1016/j.jmbbm.2015.07.029.

[41] J. D. Humphrey and S. K. Kyriacou, "The use of Laplace's equation in aneurysm mechanics," *Neurological Research,* vol. 18, no. 3, pp. 204-208, 1996/06/01 1996, doi: 10.1080/01616412.1996.11740404.

[42] M. Liu, L. Liang, and W. Sun, "A new inverse method for estimation of in vivo mechanical properties of the aortic wall," *Journal of the Mechanical Behavior of Biomedical Materials,* vol. 72, pp. 148-158, 8// 2017, doi: https://doi.org/10.1016/j.jmbbm.2017.05.001.

[43] L. Pineda, T. Fan, M. Monge, S. Venkataraman, P. Sodhi, R. T. Q. Chen, J. Ortiz, D. DeTone, A. S. Wang, S. Anderson, J. Dong, B. Amos, and M. Mukadam, "Theseus: A Library for Differentiable Nonlinear Optimization," *Advances in Neural Information Processing Systems,* 2022.

[44] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. C. Courville, and Y. Bengio, "Generative Adversarial Nets," *Advances in Neural Information Processing Systems,* pp. 2672-2680, 2014.

[45] J. Sohl-Dickstein, E. A. Weiss, N. Maheswaranathan, and S. Ganguli, "Deep unsupervised learning using nonequilibrium thermodynamics," presented at the Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37, Lille, France, 2015.

[46] S. Baek and A. Arzani, "Current state-of-the-art and utilities of machine learning for detection, monitoring, growth prediction, rupture risk assessment, and post-surgical management of abdominal aortic aneurysms," *Applications in Engineering Science,* vol. 10, p. 100097, 2022/06/01/ 2022, doi: https://doi.org/10.1016/j.apples.2022.100097.

[47] B. Bisighini, M. Aguirre, M. E. Biancolini, F. Trovalusci, D. Perrin, S. Avril, and B. Pierrat, "Machine learning and reduced order modelling for the simulation of braided stent deployment," *Frontiers in Physiology,* Original Research vol. 14, 2023-March-29 2023, doi: 10.3389/fphys.2023.1148540.

[48] S. Kim, Z. Jiang, B. A. Zambrano, Y. Jang, S. Baek, S. Yoo, and H. J. Chang, "Deep Learning on Multiphysical Features and Hemodynamic Modeling for Abdominal Aortic Aneurysm Growth Prediction," *IEEE Transactions on Medical Imaging,* vol. 42, no. 1, pp. 196-208, 2023, doi: 10.1109/TMI.2022.3206142.

[49] M. Liu, L. Liang, and W. Sun, "A generic physics-informed neural network-based constitutive model for soft biological tissues," *Computer Methods in Applied Mechanics and Engineering,* vol. 372, p. 113402, 2020/12/01/ 2020, doi: https://doi.org/10.1016/j.cma.2020.113402.

[50] V. Tac, V. D. Sree, M. K. Rausch, and A. B. Tepole, "Data-driven modeling of the mechanical behavior of anisotropic soft biological tissue," *Engineering with Computers,* vol. 38, no. 5, pp. 4167-4182, 2022/10/01 2022, doi: 10.1007/s00366-022-01733-3.

[51] P. Chen and J. Guilleminot, "Polyconvex neural networks for hyperelastic constitutive models: A rectification approach," *Mechanics Research Communications,* vol. 125, p. 103993, 2022/10/01/ 2022, doi: https://doi.org/10.1016/j.mechrescom.2022.103993.

[52] V. Tac, F. Sahli Costabal, and A. B. Tepole, "Data-driven tissue mechanics with polyconvex neural ordinary differential equations," *Computer Methods in Applied Mechanics and Engineering,* vol. 398, p. 115248, 2022/08/01/ 2022, doi: https://doi.org/10.1016/j.cma.2022.115248.

[53] Y. Leng, V. Tac, S. Calve, and A. B. Tepole, "Predicting the mechanical properties of biopolymer gels using neural networks trained on discrete fiber network data," *Computer Methods in Applied Mechanics and Engineering,* vol. 387, p. 114160, 2021/12/15/ 2021, doi: https://doi.org/10.1016/j.cma.2021.114160.

[54] T. Gärtner, M. Fernández, and O. Weeger, "Nonlinear multiscale simulation of elastic beam lattices with anisotropic homogenized constitutive models based on artificial neural networks," *Computational Mechanics,* vol. 68, no. 5, pp. 1111-1130, 2021/11/01 2021, doi: 10.1007/s00466-021-02061-x.

[55] Z. Jiang, J. Choi, and S. Baek, "Machine learning approaches to surrogate multifidelity Growth and Remodeling models for efficient abdominal aortic aneurysmal applications," *Computers in Biology and Medicine,* vol. 133, p. 104394, 2021/06/01/ 2021, doi: https://doi.org/10.1016/j.compbiomed.2021.104394.

[56] 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,* vol. 378, pp. 686-707, 2019/02/01/ 2019, doi: https://doi.org/10.1016/j.jcp.2018.10.045.

[57] D. W. Abueidda, S. Koric, E. Guleryuz, and N. A. Sobh, "Enhanced physics-informed neural networks for hyperelasticity," *International Journal for Numerical Methods in Engineering,* vol. 124, no. 7, pp. 1585-1601, 2023/04/15 2023, doi: https://doi.org/10.1002/nme.7176.

[58] J. Wang, C. Wang, Q. Lin, C. Luo, C. Wu, and J. Li, "Adversarial attacks and defenses in deep learning for image recognition: A survey," *Neurocomputing,* vol. 514, pp. 162-181, 2022/12/01/ 2022, doi: https://doi.org/10.1016/j.neucom.2022.09.004.

[59] A.-K. Dombrowski, M. Alber, C. J. Anders, M. Ackermann, K.-R. Müller, and P. Kessel, "Explanations can be manipulated and geometry is to blame," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*: Curran Associates Inc., 2019, p. Article 1217.

[60] J. Heo, S. Joo, and T. Moon, "Fooling neural network interpretations via adversarial model manipulation," in *Proceedings of the 33rd International Conference on Neural Information Processing Systems*: Curran Associates Inc., 2019, p. Article 263.

[61] S. A. Maas, B. J. Ellis, G. A. Ateshian, and J. A. Weiss, "FEBio: finite elements for biomechanics," *Journal of Biomechanical Engineering,* vol. 134, no. 1, p. 011005, Jan 2012, doi: 10.1115/1.4005694.

[62] O. C. Zienkiewicz and R. L. Taylor, *The Finite Element Method: Its Basis and Fundamentals*. Elsevier, 2013.

[63] A. Logg, K.-A. Mardal, and G. Wells, *Automated Solution of Differential Equations by the Finite Element Method: The FEniCS Book*. Springer Berlin, Heidelberg, 2012.

[64] A. Elouneg, D. Sutula, J. Chambert, A. Lejeune, S. P. A. Bordas, and E. Jacquet, "An open-source FEniCS-based framework for hyperelastic parameter estimation from noisy full-field data: Application to heterogeneous soft tissues," *Computers & Structures,* vol. 255, p. 106620, 2021/10/15/ 2021, doi: https://doi.org/10.1016/j.compstruc.2021.106620.

[65] S. K. Mitusch, S. W. Funke, and M. Kuchta, "Hybrid FEM-NN models: Combining artificial neural networks with the finite element method," *Journal of Computational Physics,* vol. 446, p. 110651, 2021/12/01/ 2021, doi: https://doi.org/10.1016/j.jcp.2021.110651.

[66] S. Baek and J. D. Humphrey, "Computational Modeling of Growth and Remodeling in Biological Soft Tissues: Application to Arterial Mechanics," in *Computational Modeling in Biomechanics*, S. De, F. Guilak, and M. Mofrad R. K Eds. Dordrecht: Springer Netherlands, 2010, pp. 253-274.

[67] S. J. Mousavi, S. Farzaneh, and S. Avril, "Patient-specific predictions of aneurysm growth and remodeling in the ascending thoracic aorta using the homogenized constrained mixture model," *Biomechanics and Modeling in Mechanobiology,* vol. 18, no. 6, pp. 1895-1913, 2019/12/01 2019, doi: 10.1007/s10237-019-01184-8.

[68] Y. Mei, B. Stover, N. Afsar Kazerooni, A. Srinivasa, M. Hajhashemkhani, M. R. Hematiyan, and S. Goenezen, "A comparative study of two constitutive models within an inverse approach to determine the spatial stiffness distribution in soft materials," *International Journal of Mechanical Sciences,* vol. 140, pp. 446-454, 2018/05/01/ 2018, doi: https://doi.org/10.1016/j.ijmecsci.2018.03.004.

[69] S. Lalitha Sridhar, Y. Mei, and S. Goenezen, "Improving the sensitivity to map nonlinear parameters for hyperelastic problems," *Computer Methods in Applied Mechanics and Engineering,* vol. 331, pp. 474-491, 2018/04/01/ 2018, doi: https://doi.org/10.1016/j.cma.2017.11.028.