Emulation of cardiac mechanics using Graph Neural Networks

David Dalton*, Hao Gao1, Dirk Husmeier2

School of Mathematics and Statistics, University of Glasgow, University Avenue, Glasgow, G12 8QQ, United Kingdom

Received 8 July 2022; received in revised form 7 September 2022; accepted 7 September 2022
Available online 6 October 2022


Abstract

Recent progress in Graph Neural Networks (GNNs) has allowed the creation of new methods for surrogate modelling, or emulation, of complex physical systems to a high level of fidelity. The success of such methods has yet to be explored however in the context of soft-tissue mechanics, an area of research which has itself seen substantial developments in recent years. The present work explicates on this by introducing an emulation framework based on a multi-scale, message-passing GNN, before applying it to the modelling of passive left-ventricle mechanics. Through numerical experiments, it is demonstrated that the proposed method delivers strong predictive accuracy when benchmarked against the results of the nonlinear finite-element method (FEM), and significantly outperforms an alternative emulator based on a fully connected neural network. Furthermore, large computational gains are achieved at prediction time against the FEM.

© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).

MSC: 92C10; 62P10; 68T99
Keywords: Cardiac mechanics; Emulation; Graph Neural Networks

1. Introduction

We are entering the era of simulation intelligence (SI) [1]. SI describes a new paradigm for quantitative scientific methods, where traditional numerical simulation methods are combined with the latest advances in scientific machine learning. The leverage afforded by machine learning methods allows experiments that were previously computationally intractable to be performed, causal relationships between variables to be studied, model uncertainty to be quantified, and noisy real-world data to be integrated into the model framework. A key element of SI is surrogate modelling, or emulation. An emulator is a statistical machine learning model which approximates a numerical forward model, while incurring much lower computational expense at prediction time [2]. Commonly used emulation approaches include neural networks [3, Chapter 13], polynomial chaos [4] and Gaussian processes [5]. Emulation is widely used for forward [6] and inverse [7] problems, as well as design [8] and optimisation [9] tasks. Traditional methods for emulation however struggle to model high dimensional physical systems, such as those

* Corresponding author.
E-mail address: d.dalton.1@research.gla.ac.uk (D. Dalton).

1 Orcid: 0000-0001-6852-9435
2 Orcid: 0000-0003-1673-7413

https://doi.org/10.1016/j.cma.2022.115645
0045-7825/© 2022 The Author(s). Published by Elsevier B.V. This is an open access article under the CC BY license (http://creativecommons.org/licenses/by/4.0/).
represented by high-fidelity meshes or large numbers of particles, due to the so-called curse of dimensionality. Furthermore, it can be difficult to build known symmetries or invariances of the system under consideration into the structure of traditional surrogate models. In recent years, new approaches have been developed for emulation of physical systems using Graph Neural Networks (GNNs) [3, Chapter 23]. A GNN can be seen as a generalisation of a Convolutional Neural Network (CNN) [3, Chapter 14] to operate on data with a non-Euclidean structure. This is critical for emulating complex or multiscale systems, where the topology of the particles or nodes in the system under consideration can be highly non-Euclidean. Furthermore, by accounting for the intrinsic topology of a given system, GNN emulators have demonstrated the ability to perform accurate modelling to very high levels of fidelity, beyond what is possible with traditional emulation approaches. Example application domains of GNN emulators include computational fluid dynamics [10], particle-based systems [11], cloth and elastic simulations [12], rigid and deformable bodies [13] as well as computer graphics [14].

One potential application domain of GNN emulation that has received less attention to date is the mechanics of biological soft tissue, that is, muscles, neurons, skin, cartilages, which usually exhibit remarkable complexity with multiconstituent, hierarchical and heterogeneous structures, and their mechanical responses to various loading conditions can be critical to maintain the biomechanical homeostasis for optimal mechanical functioning of the organism [15]. When injury or disease occurs in biological soft tissues, an imbalanced stress/strain micro-environment can be induced due to the loss of the complex organisation of biomolecules, such as the loss of myocytes after myocardial infarction. Therefore, there is a critical need for accurate and fast quantification of biomechanical factors, such as stress and strain, in soft tissues. One approach is to develop high-fidelity computational biomechanics models, which integrate knowledge of physiology, pathology and the fundamental laws of mechanics into one framework [16], with the potential to offer patient-specific diagnostic insights beyond what is available from typical in/ex vivo analyses [17]. An excellent example is computational cardiac mechanics, which is now being translated into industry and the clinic [18–20]. Other topics include the modelling of liver [21], mitral valve [22], arteries [23] and brain [24], to list some examples. Personalised biomechanics models need to be solved numerically, one such method being the nonlinear finite element method (FEM) [25]. FEM-based models can require significant computational resources due to potentially millions of unknown variables, interactions among different subsystems, integration across different temporal and spatial scales, and limited experimental data for model calibration [16]. This computational bottleneck constitutes a large obstacle in translating biomechanical models into the clinic for use in real-time estimation, where thousands of model simulations can be required to obtain accurate calibration with experimental data [18].

1.1. Contributions

To address these issues, recent efforts have focused on integrating machine learning and biomechanics modelling together to create robust predictive models, explore massive parameter spaces, and provide real-time solutions [26]. This work adds to the growing literature in this area by presenting a new GNN emulation framework, before evaluating its accuracy in the modelling of passive left ventricle (LV) mechanics. This is a challenging emulation problem, as different LV anatomies can vary substantially in terms of size, asymmetry, and wall thickness, for example, as well as exhibiting different material properties. In addition, the emulator must deliver significant computational savings at prediction time to be useful for real-time estimation. The GNN is implemented using a message-passing approach on an augmented, multi-scale graph representation of the LV. Experimental results demonstrate that the proposed method displays strong emulation accuracy across different LV anatomies, as well as the ability to specialise on a given anatomy of interest in a transfer learning setting. In addition, the emulator can make forward predictions up to five orders of magnitude more quickly than the numerical forward simulator.

The manuscript is laid out as follows; first, Section 2 described the methods we use, including the proposed emulation framework. Section 3 describes the application of this framework to an illustrative two-dimensional beam deformation problem, before Section 4 describes the emulation results for passive LV mechanics. Section 5 finally then concludes.

2. Methods

2.1. General mechanics and graph representation framework

We consider two quasi-static or static mechanical systems in this work. The first is a 2-D beam with linear isotropic material property, clamped at one end and deformed under its own weight (see Section 3). The second
is a real human left ventricular model in diastole, characterised by a nonlinear constitutive law (see Section 4). In general, the two models can be formulated as a nonlinear boundary-value problem (BVP) of elastostatics, i.e.

$$\begin{align*}
\nabla \cdot \sigma + b &= 0 \quad \text{in } \Omega, \\
u = u_d & \quad \text{on } \partial \Omega_d, \\
\sigma n &= t \quad \text{on } \partial \Omega_e,
\end{align*}$$

(1)
in which $\sigma$ is the Cauchy stress tensor that is related to the strain tensor through a chosen constitutive law, $b$ is the body force, $u$ is the displacement field that is the unknown to be solved numerically, $u_d$ is the prescribed displacement boundary conditions on the boundary $\partial \Omega_d$, and $t$ is the applied traction density to the boundary $\partial \Omega_e$.

The whole computational domain $\Omega$ needs to be discretised with $n$ simplicial or quadrilateral/hexahedral elements to allow the above BVP to be solved numerically, denoted as $\{T_h\}_{h>0}$. Specifically,

$$\Omega \approx \{T_h\}_{h>0} = \bigcup_{e_l=1}^n T_{e_l},$$

(2)

where $T \in T_h$ can be tetrahedron or hexahedron for a 3-D finite-element (FE) mesh, and triangle or quadrilateral for a 2-D mesh. For example, Fig. 1(a) displays a simple rectangular domain $\Omega$ discretised with 2 triangles ($T_1$ and $T_2$).

In this work we use Graph Neural Network (GNN) emulators to find an approximation to the solution of the above general BVP for the two mechanical systems considered. In order to apply a GNN emulator to a given geometry, the discretised FE mesh (2) must be represented in the form of a graph $G$. We define a graph to be the following 3-tuple:

$$G \triangleq (V, E, \theta).$$

(3)
The set of nodes of the graph $V$ is defined as

$$V \triangleq \bigcup_{i=1}^{\vert V \vert} \{(n_i, x_i)\},$$

(4)

which are extracted directly from the nodes of the FE mesh. Each node consists of a scalar index value $n_i$, the ordering of which follows that of the underlying FE mesh, and a coordinate vector $x_i$, which gives the spatial coordinates of the corresponding node in the FE mesh, for $i = 1, 2, \ldots, \vert V \vert$. As a shorthand notation, we will use the scalar index variable $n_i$ alone when referring to the nodes in $V$. The creation of $V$ for the simple FE mesh in Fig. 1(a) is illustrated in panel (b), where the four FE nodes are converted to the set of graph nodes

$$V = \{(n_1, x_1), (n_2, x_2), (n_3, x_3), (n_4, x_4)\}.$$
The set \( E \) is the graph topology, each element of which denotes a directed edge relationship between a pair of nodes in \( \mathcal{V} \). A directed edge from node \( n_i \) to node \( n_j \) is denoted \( \{n_i \rightarrow n_j\} \). The directed edges in \( E \) are found by converting each undirected edge in the FE mesh into two directional graph edges which point in opposite directions. This is done by creating a directed edge from node \( n_i \) to all nodes \( n_j \in \mathcal{V} \) that are in its FE neighbourhood, \( \mathcal{N}_{i}^{FE} \), for all \( n_i \in \mathcal{V} \). We define \( \mathcal{N}_{i}^{FE} \) to be the set of all nodes in the graph that share an edge with node \( n_i \) in the FE discretisation from Eq. (2), i.e. the one-ring neighbourhood of \( n_i \). For example, in Fig. 1(a), we have that the FE neighbourhood of node \( n_1 \) is
\[
\mathcal{N}_{1}^{FE} = \{n_2, n_3\}.
\]

We introduce the following shorthand notation to refer to all directed edges that have node \( n_i \) as sender:
\[
\{n_i \rightarrow \mathcal{N}_{i}^{FE}\} = \bigcup_{n_j \in \mathcal{N}_{i}^{FE}} \{n_i \rightarrow n_j\}.
\]

The entire graph topology \( \mathcal{E} \) is then defined to be the union of the above over all nodes \( n_i \in \mathcal{V} \),
\[
\mathcal{E} = \bigcup_{i=1}^{|\mathcal{V}|} \{n_i \rightarrow \mathcal{N}_{i}^{FE}\}.
\]

Thus for the FE mesh from Fig. 1(b), the specific edge set \( \mathcal{E} \) is
\[
\{n_1 \rightarrow n_2, n_2 \rightarrow n_1, n_2 \rightarrow n_4, n_4 \rightarrow n_2, n_3 \rightarrow n_4, n_4 \rightarrow n_3, n_2 \rightarrow n_3, n_3 \rightarrow n_2, n_1 \rightarrow n_3, n_3 \rightarrow n_1\}.
\]

In this example, we have that \(|\mathcal{E}| = 10\), which is almost as high as the cardinality of the completely connected directed graph on four nodes. However, for larger FE meshes, the cardinality of the graph topology \( \mathcal{E} \) extracted in this manner will be significantly lower than that of the completely connected graph. Note that additional edges can be added if desired, beyond those directly extracted from the FE mesh in manner described above. For example, the edge relations \( \{n_1 \rightarrow n_4\}, \{n_4 \rightarrow n_1\} \) could be added to the graph topology \( \mathcal{E} \) derived from the FE mesh in Fig. 1.

The third element of \( \mathcal{G} \) is a vector denoted \( \theta \), which represents any global attributes of interest. In the present work, \( \theta \) represents the material stiffness properties of the geometry under consideration, assumed constant for all nodes \( n_i \in \mathcal{V} \), and hence \( \theta \) is a global attribute. However, this assumption can be relaxed to allow stiffness levels to vary across the graph.

Finally therefore, the entire graph is
\[
\mathcal{G} = \left( \bigcup_{i=1}^{|\mathcal{V}|} \{(n_i, \mathbf{x}_i)\}, \bigcup_{i=1}^{|\mathcal{V}|} \{n_i \rightarrow \mathcal{N}_{i}^{FE}\}, \theta \right).
\]

For both the beam and left-ventricle mechanical systems considered for emulation in this work, the objective of the surrogate model is to be able to generalise to initial states not seen in the training data, in terms of the nodal coordinates from \( \mathcal{V} \) and global material parameters \( \theta \). However, the topology \( \mathcal{E} \) of the graph representations is assumed constant for all graphs in each respective system. For example, each left ventricle anatomy is represented by an FE mesh with the same number of nodes and the same finite element structure. However, individual left ventricles differ geometrically, meaning that their nodal coordinates can differ.

### 2.2. Neural networks

A fully connected neural network, also known as the multi-layer perceptron (MLP) is a powerful tool for function approximation. Recall that an MLP \( f_{\omega} : \mathbb{R}^{D_{in}} \rightarrow \mathbb{R}^{D_{out}} \) is comprised of the sequential composition of affine functions with a nonlinear activation function \( \varphi : \mathbb{R} \rightarrow \mathbb{R} \). A network with more than one unobserved, or hidden layer is referred to as a deep network. A deep network with two hidden layers takes the form
\[
\begin{align*}
\mathbf{z}_1 &= \varphi (\mathbf{W}_0 \mathbf{x} + \mathbf{b}_0) \\
\mathbf{z}_2 &= \varphi (\mathbf{W}_1 \mathbf{z}_1 + \mathbf{b}_1)
\end{align*}
\]

---

3 In the computer graphics literature, “directed edges” are commonly referred to as “half-edges”.
\[ y = W_2 z_2 + b_2, \]

where \( \psi \) acts element-wise, \( W_t \) is the weight matrix of layer \( t \) and \( b_t \) the bias, for \( t = 0, 1, 2 \). Various forms for \( \psi \) have been proposed [27], one of which is the tanh function:

\[
\psi(x) = \tanh(x) = \frac{2}{1 + \exp(-2x)} - 1. \tag{12}
\]

The form of the forward map defined by an MLP is determined by the collection of all weights and biases of the network. A point estimate of these parameters is typically used for prediction, trained with a stochastic gradient-based update scheme on a dataset of input–output exemplars [3, Chapter 13].

Consider using an MLP to model very high-dimensional data, for example, emulating a complex physical system to a high level of fidelity. To capture the interactions and structure of the data, a network with large depth and/or width may be required. Because the internal weight matrices of the MLP are dense, the number of trainable parameters of the network will also be very large. This can make MLPs difficult to train to achieve good generalisation performance in this case. If however the high dimensional data being modelled has some intrinsic topology, this issue can be alleviated by designing a neural network architecture that accounts for this topology. For example, a convolutional neural network (CNN) applies a translation-invariant, discrete convolution to its internal layers that is applicable to data with Euclidean topology, instead of dense weight matrices. As convolution is a linear operator, a CNN can be equivalently written as a special case of an MLP, where the weight matrices exhibit a sparse, Toeplitz-like structure. This sparsity can lead to a massive reduction in the number of trainable parameters compared to an MLP, allowing CNNs to model high-dimensional image data, for example, without overfitting [3, Chapter 13].

The idea of a CNN is generalised by a Graph Neural Network (GNN), which refers to a class of deep learning architectures that allow graph-structured data to be modelled. GNNs have been applied in a wide range of contexts, including social network analysis [28], drug discovery [29] molecular chemistry [30], traffic prediction [31], and surrogate modelling of physical systems [11,32], to give a few examples. Numerous approaches have been proposed for the design of GNN architectures. One such approach is spatial graph convolutions, which seeks to extend the convolution operations of a CNN, which operate on the regular (grid-like) local neighbourhoods of image data, to the irregular local neighbourhoods of arbitrary graph data. Spectral methods are another approach, where graph convolution operations are performed by operating in the spectral domain of the graph Laplacian matrix. In this work however, we use a GNN emulator that implements a message-passing approach. Message passing GNNs naturally extend to both large graphs and graphs not seen in the training data, and have proven successful for the emulation of physical systems [11,12,14,33]. A message-passing GNN tackles the problem of learning the physical dynamics of a system via an information diffusion mechanism around the graph topology. The nodes of the system are assigned a learned representation using a sequence of processor steps, where messages are exchanged between neighbouring nodes, before a prediction for the forward dynamics of the system is made. A more general form of the message-passing framework was proposed in [34], where representations of nodes, edges, and the entire graph can be learned simultaneously in a three-stage, encode–process–decode approach. In this work a variant of the three-stage approach is used for emulation. Fig. 2 gives a high-level illustration of the framework: for specific details, the reader can see Section 2.4.

The input to the GNN emulator is the initial state of the physical system under consideration in the form of a graph. In Fig. 2, the input is the geometry of a left ventricle at the start of diastole, which is the emulation problem considered in Section 4. The first stage of the emulator is the encoder. This is where any node and edge-wise numerical features of the graph, such as boundary condition information, are embedded into a higher dimensional latent space. The second stage is to process these embeddings by performing a series of message-passing steps, whereby the node and edge representations are sequentially updated by exchanging information between neighbouring nodes in accordance with the graph topology. The last stage is the decoder stage, whereby the final node embeddings are used to predict the end state of the system. Each of the three stages: encode, process and decode, are performed using individual, trainable MLPs. The parameters of the MLPs, which we refer to collectively as \( \omega \), control the predictions of the GNN. A pointwise estimate of \( \omega \) can be found using a gradient-based update scheme on a set of training data, in the same manner as for a single MLP.
Fig. 2. Schematic illustration of GNN emulation. The emulator makes use of a three-stage, encode–process–decode framework to map the initial state of the physical system to its end state.

2.3. Augmented graph generation

The GNN architecture introduced in Section 2.4 uses the message passing framework from Fig. 2 to perform emulation, whereby information is propagated between the graph nodes, before a prediction is made for the forward displacement of the given body. Instead of working with the graph representation \( G = (V, E, \theta) \) directly extracted from the FE mesh in the manner discussed in Section 2.1, improved emulation performance can be achieved by working with an augmented graph representation of the mesh, \( \tilde{G} \). We define an augmented graph as

\[
\tilde{G} = (V \cup \tilde{V}, \tilde{E}, \theta),
\]

where the augmented nodes \( \tilde{V} \) and topology \( \tilde{E} \) are created iteratively over \( L \) steps as in Algorithm 1. The set \( \tilde{V} \) is defined as

\[
\tilde{V} = \bigcup_{l=1}^{L} \text{Roots}^l,
\]

where \( \text{Roots}^l \) is a set of virtual nodes derived from the FE nodes, for \( l = 1, 2, \ldots, L \). The root nodes are defined constructively in Algorithm 1. In summary, each layer of root nodes \( \text{Roots}^l \) is found by clustering the root nodes from the previous layer \( \text{Roots}^{l-1} \) into a set of lower cardinality, meaning that successive layers provide an increasingly coarse representation of the underlying FE mesh. The augmented topology \( \tilde{E} \) is defined as

\[
\tilde{E} = E \cup \bigcup_{l=1}^{L} \{ E_{\text{intra}}^l \cup E_{\text{inter}}^l \}.
\]

At each step \( l \), \( E_{\text{intra}}^l \) describes the intra-layer connections between nodes in \( \text{Roots}^l \), while \( E_{\text{inter}}^l \) describes the inter-layer connections between nodes in \( \text{Roots}^l \) and the nodes in the previous layer, \( \text{Roots}^{l-1} \). Again, exact details of the construction of \( E_{\text{intra}}^l \) and \( E_{\text{inter}}^l \) are given in Algorithm 1. The advantage of the augmented graph representation \( \tilde{G} \) over the original graph \( G \) is that the virtual nodes \( \tilde{V} \) provide a shortcut by which information can propagate along the extended graph topology \( \tilde{E} \supset E \).

Algorithm 1 details the construction of the augmented nodes and topology for a given input graph \( G \). Two other variables are required as inputs, the first of which is \( \kappa \), the number of nearest neighbours to consider when defining the intra-layer connectivity \( E_{\text{intra}}^l \) for each layer of virtual nodes up to the final layer. The second is a decreasing sequence of natural numbers \( \eta \), which gives the cardinality of each layer of virtual nodes that are to
be generated. These variables can be adjusted depending on the application context; specific details for the two experimental domains considered in this work are given in Sections 3.2 and 4.4 respectively. In general, the value of \( \kappa \) is specified so that the average number of neighbours each virtual node has in \( \bar{E}_{\text{intra}} \) is approximately equal to the average number of neighbours each real node has in \( \bar{E} \). The node cardinalities in \( \eta \) are selected so that the number of nodes decreases by approximately the same factor for each additional level, and such that this factor is at least as great as the number of real nodes in each finite element.

Fig. 3 presents a visual explanation of the augmented graph generation algorithm, for one of the 2-D beam geometries considered for emulation in Section 3. Panel (a) of the figure displays the original graph \( \mathcal{G} \). The positions of the \( |\mathcal{V}| = 96 \) nodes in \( \mathcal{V} \) are shown as black squares, and each pair of directed edges in \( \bar{E} \) is shown as a single grey line. Panel (b) then shows the coordinates of the virtual nodes \( \hat{\mathcal{V}} = \text{Roots}^1 \cup \text{Roots}^2 \) generated by Algorithm 1 with \( \eta = [24, 6] \) and \( \kappa = 4 \), along with the original FE nodes. The first layer of \( \eta_1 = 24 \) virtual nodes are displayed as red circles, and the second and final layer of \( \eta_2 = 6 \) nodes are shown as green triangles.

Algorithm 1 begins by initialising the 0th layer of root nodes, \( \text{Roots}^0 \), to be the original FE nodes \( \mathcal{V} \). The multiscale augmented graph is then built over \( L \) steps. At step \( l = 1 \), the initial layer of virtual nodes \( \text{Roots}^1 \) is created by first grouping the coordinates \( \mathbf{x}_i \) of the real nodes \( \mathcal{V} = \text{Roots}^0 = \text{Leaves}^1 \) into the clusters \( \mathcal{C} = \{c_1, c_2, \ldots, c_{\eta_1}\} \) using \( k \)-means. The coordinates of the virtual nodes \( \text{Roots}^1 \) are then found as the centre of these \( \eta_1 \) clusters:

\[
\text{Roots}^1 = \bigcup_{c \in \mathcal{C}} \{(n_c, \mathbf{x}_c)\} \quad \text{where} \quad \mathbf{x}_c = \frac{1}{|c|} \sum_{n_i \in c} \mathbf{x}_i. \tag{16}
\]

Note that each root node is also assigned a scalar index variable, denoted \( n_c \), which is defined to be consistent with the index variables \( n_i \) of the real nodes \( \mathcal{V} \). Once the roots have been found, the inter-layer topology \( \bar{E}_{\text{inter}}^l \) between root and leaf nodes is defined by creating a pair of directed edges \( \{n_i \rightarrow n_c, n_c \rightarrow n_i\} \) between each root node \( n_c \in \text{Roots}^1 \) and all leaf nodes \( n_j \in \text{Leaves}^1 \) which were assigned to cluster \( c \) by \( k \)-means,

\[
\bar{E}_{\text{inter}}^1 = \bigcup_{c \in \mathcal{C}} \{(n_c \rightarrow n_i, n_i \rightarrow n_c \mid n_i \in c\}. \tag{17}
\]
Fig. 3. Illustration of Algorithm 1 applied to a 2-D beam geometry, with \( \kappa = 4 \) and \( \eta = [24, 6] \). Panel (a) shows the nodes and edges of the input graph and panel (b) plots the real nodes in black with the augmented nodes in red and green. The second and third row of figures illustrate steps one and two of the algorithm respectively — for a detailed description, see the text.

Fig. 3(c) displays the construction of the root nodes \( \text{Roots}^1 \) and inter-layer connections \( \tilde{\mathcal{E}}^1_{\text{inter}} \) for the beam example. The coordinates \( x_c \) of each of the \( \eta_1 = 24 \) root nodes are shown as red circles, which were found by clustering the 96 FE nodes. The connections are shown as grey lines between each FE node and its corresponding cluster centre node.

For each root node \( n_c \in \text{Roots}^1 \), its local root neighbourhood \( \mathcal{N}^\text{Roots}_c \) is defined to be its \( \kappa \)-nearest neighbours among other root nodes:

\[
\mathcal{N}^\text{Roots}_c = \{ n_{c'} | x_{c'} \in \kappa-\text{nn}(x_c, \text{Roots}^1) \}.
\]  

The distance metric used in the nearest-neighbour calculations is the usual Euclidean metric. The intra-layer topology \( \tilde{\mathcal{E}}^1_{\text{intra}} \) between the root nodes in \( \text{Roots}^1 \) is then constructed by creating a pair of directed edges \( \{ n_c \rightarrow n_{c'} \}, \{ n_{c'} \rightarrow n_c \} \) between each root node \( n_c \) and its \( \kappa \) nearest neighbours in \( \mathcal{N}^\text{Roots}_c \):

\[
\tilde{\mathcal{E}}^1_{\text{intra}} = \bigcup_{c \in \mathcal{C}} \{ n_c \rightarrow n_{c'}, n_{c'} \rightarrow n_c | n_{c'} \in \mathcal{N}^\text{Roots}_c \}.
\]  

Fig. 3(d) shows the creation of \( \tilde{\mathcal{E}}^1_{\text{intra}} \) for the beam example as grey lines, using \( \kappa = 4 \) nearest neighbours. Note that the neighbourhoods defined by the \( \kappa \)-nn operation are not symmetric, that is \( n_{c'} \in \mathcal{N}^\text{Roots}_c \not\iff n_c \in \mathcal{N}^\text{Roots}_{c'} \). This asymmetry is illustrated in Fig. 4(a), which shows the top right corner of the beam geometry from Fig. 3(d). Two nodes \( n_c, n_{c'} \in \text{Roots}^1 \) are highlighted. The four root nodes in \( \mathcal{N}^\text{Roots}_c \) are shown with grey arrows, while the neighbours \( \mathcal{N}^\text{Roots}_{c'} \) are shown with black arrows. We see that \( n_{c'} \in \mathcal{N}^\text{Roots}_c \) but \( n_c \notin \mathcal{N}^\text{Roots}_{c'} \). The GNN emulation architecture introduced in Section 2.4 requires that each edge relation \( \{ n_i \rightarrow n_j \} \) in \( \tilde{\mathcal{E}} \) has a twin relation.
this manner means that each root node will have connection...{\bar{\mathcal{N}}_c}^\text{Roots}, with black arrows. We see that \( n_c \in \mathcal{N}_c^\text{Roots} \) but \( n_c \notin \mathcal{N}_c^\text{Roots} \). Therefore, an additional edge \( \{n_c \rightarrow n_c\} \) is created, so that the symmetry from Eq. (27) can be enforced in the message-passing stage of the emulator.

\{n_l \rightarrow n_l\}, so that the symmetry from Eq. (27) can be enforced. For this reason, we also add the additional connection \( \{n_c \rightarrow n_c\} \) to \( \mathcal{E}_\text{intra}^l \), shown as a black arrow in Fig. 4(b). The inclusion of additional connections in this manner means that each root node will have at least \( \kappa \) neighbours in \( \mathcal{E}_\text{intra}^l \), for \( l = 1, 2, \ldots, L - 1 \).

Subsequent steps \( l = 2, \ldots, L \) of Algorithm 1 proceed in the same manner as the first step, whereby the root nodes at the previous layer, \( \text{Roots}^{d-1} \) become the leaf nodes of step \( l \), \( \text{Leaves}^l \) and are clustered into \( \eta_l \) root nodes. The inter-layer and intra-layer node topologies \( \mathcal{E}_\text{inter}^l \) and \( \mathcal{E}_\text{intra}^l \) are then calculated as before. This is with the exception of the final layer of virtual nodes \( \text{Roots}^L \), which is fully connected:

\[
\mathcal{E}_\text{intra}^L = \bigcup_{c \in \mathcal{C}} \{ n_c \rightarrow n_c \mid n_c, n_c' \in \text{Roots}^L \setminus \{n_c, x_c\} \}.
\]

Once the last layer of nodes has been created, the augmented graph \( \tilde{\mathcal{G}} \) is returned. The third row of plots in Fig. 3 shows the creation of the second and final layer of virtual nodes and connections for the beam example. In Panel (e), the virtual nodes from layer 1 are clustered into \( \eta_1 = 6 \) cluster centres to form \( \text{Roots}^2 \), and \( \tilde{\mathcal{E}}_\text{intra}^2 \) is created as before. Then, the final intra-layer topology \( \tilde{\mathcal{E}}_\text{intra}^L \) in panel (f) is fully connected.

Our use of a multi-scale augmented graph for emulation is conceptually similar to earlier work [13,35]. Recall however that the goal of this work is to build an emulator that can generalise to geometries not seen in the training data. As discussed though in Section 2.1, we assume that the graph representations of the geometries in each system we consider share a common topology, \( \tilde{\mathcal{E}} \). We therefore desire that the augmented graph representations of the FE meshes also share a common topology, \( \tilde{\mathcal{E}} \). However this is not guaranteed to be the case if Algorithm 1 were applied separately to the graph \( \mathcal{G} \) induced by the FE mesh of each individual geometry. For example, consider the graphs \( \mathcal{G}_0 \) and \( \mathcal{G}_1 \) displayed in the top row of Fig. 5. The two graphs clearly share a common topology, that is \( \mathcal{E}_0 = \mathcal{E}_1 \). However, the augmented graphs \( \tilde{\mathcal{G}}_0 \) and \( \tilde{\mathcal{G}}_1 \) that result from applying Algorithm 1 to each graph respectively with \( \eta = [2] \) do not share a common topology. This is illustrated in the second row of plots in Fig. 5, where we can see that \( \tilde{\mathcal{E}}_0 \neq \tilde{\mathcal{E}}_1 \).

The discrepancy occurs because the generation of the virtual nodes depends solely on the (different) coordinate values from the graph nodes \( \mathcal{V}_0 \) and \( \mathcal{V}_1 \), not the common graph topology \( \mathcal{E}_0 = \mathcal{E}_1 \). To ensure then that the augmented graph representations of different geometries have a shared topology \( \tilde{\mathcal{E}} \), this can be constructed with respect to a single, representative graph using Algorithm 1. Any subsequent graphs can then be augmented in a manner similar to Algorithm 1, whereby each layer of virtual nodes is constructed by pooling nodes in the previous layer. However, the pooling is performed with respect to the clusters implied by the augmented topology \( \tilde{\mathcal{E}} \) found on the representative graph, rather than reapplying \( k \)-means clustering, meaning that each augmented graph shares this common topology. The nodes of the reference graph, denoted \( \mathcal{V} \) can be found by “averaging” over the node coordinates of the geometries from the available training data. Specifically, for each \( n_l \in \mathcal{V} \), the corresponding coordinate value \( \tilde{x}_l \) is found as \( \tilde{x}_l = \frac{1}{N_r} \sum_j x^{(j)}_l \), where we have introduced a superscript \( j \) to index over the \( N_r \) available training geometries.
Fig. 5. The upper row of figures shows two graphs $G_0$ and $G_1$ which share a common topology, that is $E_0 = E_1$. The bottom row shows the augmented graphs $\tilde{G}_0$ and $\tilde{G}_1$ found by running Algorithm 1 with $\eta = [2]$. Virtual nodes are shown in red and additional edges are shown as dashed red lines. In this case, we have that $\tilde{E}_0 \neq \tilde{E}_1$. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

The final data-processing stage before the surrogate model can be applied is to assign numerical feature vectors to the individual nodes and edges of the augmented graph representations. As detailed in Algorithm 2, these feature vectors are iteratively processed by the GNN emulator over a series of message passing steps, before the final processed values are used to predict the forward displacement of the body under consideration. Specifically, each real or virtual node $n_i \in V \cup \tilde{V}$ is assigned a feature vector, denoted $v_i$,

$$n_i \rightarrow v_i \text{ for all } n_i \in V \cup \tilde{V}. \quad (21)$$

The real nodes in $V$ are first assigned a feature vector. The form of these features will depend on the application context, but could contain, for example, boundary condition information, or for an inhomogeneous material, local fibre structure information. Importantly, the absolute positions $x_i$ are not encoded in the node features. Instead, relative node positions are encoded in the edge features defined below, ensuring that the outputs of the emulator are invariant to translations of the system in space. Each node in the first layer of virtual nodes is then assigned a feature vector by taking the mean of the features from all real nodes with which it shares an edge relation. Subsequent virtual node layers $l = 2, \ldots, L$ are then iteratively assigned features in the same manner. Note that for boundary node Boolean indicator variables, we could have used min() rather than mean() aggregation to ensure that a virtual node will have non-zero boundary indicator variable only if all of its neighbours in the previous layer are boundary nodes. These neighbourhoods will in general include interior points however and by using mean() aggregation then the virtual nodes would have no boundary condition information in their feature vectors. See, for example, Fig. 3, where no virtual nodes lie on the boundary of the beam geometry. By using mean() aggregation instead, boundary condition information is included in the virtual nodes, where the boundary indicator variable can take values between 0 and 1.

The edge relations $\{n_i \rightarrow n_j\} \in \tilde{E}$ are also assigned numerical features, denoted $e_{i \rightarrow j}$. For reasons discussed in Section 2.4, only one of each pair of directed edges $\{n_i \rightarrow n_j\}, \{n_j \rightarrow n_i\} \in \tilde{E}$ needs to be assigned an edge feature vector. This is achieved by assigning features to those edges where the index of the sender node $n_i$ is greater than
Algorithm 2 DeepGraphEmulator

Input: $\tilde{G} = (\mathcal{V} \cup \tilde{V}, \tilde{E}, \theta, \mathbf{z}^{\text{glob}})$
Output: $\hat{U} = \{\hat{u}_i\}$ for all $n_i \in \mathcal{V}$

Encoder:
$\mathbf{v}_i^0 = f_V (\mathbf{v}_i)$ for all $n_i \in \mathcal{V} \cup \tilde{V}$
$\mathbf{e}_{i \rightarrow j}^0 = f_E (\mathbf{e}_{i \rightarrow j})$ for all $\{n_i \rightarrow n_j\} \in \tilde{E}$ where $i > j$

Processor:
for $k = 1 : K$ do
$\mathbf{m}_{i \rightarrow j}^k = g_E^k \left( \mathbf{e}_{i \rightarrow j}^{k-1}, \mathbf{v}_i^{k-1}, \mathbf{e}_{j \rightarrow i}^{k-1} \right)$ for all $\{n_i \rightarrow n_j\} \in \tilde{E}$ where $i > j$
$\mathbf{m}_{j \rightarrow i}^k = -\mathbf{m}_{i \rightarrow j}^k$ for all $\{n_i \rightarrow n_j\} \in \tilde{E}$ where $i < j$
$\mathbf{v}_i^k = \mathbf{v}_i^{k-1} + g_V \left( \mathbf{v}_i^{k-1}, \sum_{j \in \mathcal{V}} \mathbf{m}_{j \rightarrow i}^k \right)$ for all $n_i \in \mathcal{V} \cup \tilde{V}$
$\mathbf{e}_{i \rightarrow j}^k = \mathbf{e}_{i \rightarrow j}^{k-1} + \mathbf{m}_{j \rightarrow i}^k$ for all $\{n_i \rightarrow n_j\} \in \tilde{E}$ where $i > j$
end for
$\mathbf{z}_{i}^{\text{local}} = (\mathbf{v}_i^K, \sum_{j \in \mathcal{V}} \mathbf{m}_{j \rightarrow i}^K)$ for all $n_i \in \mathcal{V}$

Decoder:
$\mathbf{z}_i^\theta = f_P (\mathbf{\theta})$
$\hat{u}_i^d = h_d (\mathbf{z}_i^\theta, \mathbf{z}_i^{\text{glob}}, \mathbf{z}_i^{\text{local}})$ for all $n_i \in \mathcal{V}$ for $d = 1, ..., D$
$\hat{u}_i = (\hat{u}_i^1, ..., \hat{u}_i^D)$ for all $n_i \in \mathcal{V}$

that of the receiving node $n_j$ (note there are no self-loops in the graph). That is, edge features are assigned as
$$\{n_i \rightarrow n_j\} \rightarrow \mathbf{e}_{i \rightarrow j} \text{ for all } \{n_i \rightarrow n_j\} \in \tilde{E} \text{ where } i > j.$$ (22)
The form of the assigned edge features was the same for both systems considered here (beam and left ventricle);
$$\mathbf{e}_{i \rightarrow j} = (\mathbf{x}_j - \mathbf{x}_i, || \mathbf{x}_j - \mathbf{x}_i ||_2),$$ (23)
containing both the difference and distance between the coordinates of the sender node and receiving node respectively.

2.4. GNN emulation architecture

Algorithm 2 presents the DeepGraphEmulator GNN architecture, which defines a forward map of the form
$$\left( \tilde{G}, \mathbf{z}^{\text{glob}} \right) \mapsto \hat{U}.$$ (24)
The first input variable $\tilde{G}$ is the augmented graph representation of the geometry under consideration, the construction of which is described in Section 2.3 above. The second, optional input is $\mathbf{z}^{\text{glob}}$, a vector-embedding of the global shape of the real nodes of the given geometry. The form of $\mathbf{z}^{\text{glob}}$ will be context-dependent, and could be obtained for example using dimensionality reduction or parameterised methods. The GNN emulator then makes a prediction $\hat{U}$ for the node-wise displacement of the body from its initial to end-state using a three-stage, encode-process-decode approach, which is illustrated schematically in Fig. 2. Each stage is controlled by individual, trainable MLPs. In total, there are $(2K + 3 + D)$ such MLPs internal to the emulator: two encoder MLPs, $f_V$ and $f_E$, two processor MLPs for each of $K$ rounds of message passing, $(g_V^k, g_E^k)_{k=1}^K$, one global parameter embedding MLP, $f_P$, and $D$ node-decode MLPs, $(h^d)_{d=1}^D$, where $D$ is the dimensionality of the system.

The first stage of the DeepGraphEmulator algorithm, the encoder, embeds each node feature vector $\mathbf{v}_i$ and edge feature vector $\mathbf{e}_{i \rightarrow j}$ from the augmented graph representation of the physical system $\tilde{G}$ into a higher $M$-dimensional
inspired by interpreting the messages as forces and then incorporating the elementary concept in classical mechanics that the net force acting on \( \sum \) for example, using GNNs \([34]\). Permutation invariance and the message-passing symmetry could both be maintained at each step by, for remaining edges \( \{n_i \to n_j\} \in \hat{E} \) where \( i < j \), the message value \( m_{j \to i} \) is found by simply negating the message \( m_{i \to j} \) found using \( g_E^k \) above:

\[
m_{j \to i}^k = -m_{i \to j}^k \text{ for all } \{n_i \to n_j\} \in \hat{E} \text{ where } i < j.
\]

Note that a consequence of this method of enforcing the symmetry in Eq. (27) is that the output \( \hat{U} \) of the emulator is not invariant under a permutation of the node and edge indices of the input graph \( \hat{G} \), as is typically the case for GNNs \([34]\). Permutation invariance and the message-passing symmetry could both be maintained at each step by, for example, using \( g_E^k \) to compute an initial message \( m_{i \to j}^0 \) on each edge \( \{n_i \to n_j\} \in \hat{E} \), before calculating the final message values as \( m_{i \to j} = (m_{i \to j}^k - m_{j \to i}^k) / 2 \). We took the alternative approach given in Eqs. (28) and (29) in this work as it is more computationally efficient, because only half of the messages need to be explicitly computed using \( g_E^k \). The updated node embedding vectors \( \mathbf{v}_i^k \) are then found as the sum of the embedding from the previous step \( k - 1 \), plus a learned value from the node-update MLP \( g_V^k \),

\[
\mathbf{v}_i^k = \mathbf{v}_i^{k-1} + g_V^k \left( \mathbf{v}_i^{k-1}, \sum_{j \in N_i^0} m_{j \to i}^k \right) \text{ for all } n_i \in \mathcal{V} \cup \hat{\mathcal{V}}.
\]

The node-update MLP incorporates the message vectors by summing over all incoming messages to each node \( n_i \in \mathcal{V} \cup \hat{\mathcal{V}} \). The set of all nodes \( n_j \in \mathcal{V} \cup \hat{\mathcal{V}} \) which send a directed edge to node \( n_i \) is called its graph neighbourhood \( \mathcal{N}_i^0 \), defined as

\[
\mathcal{N}_i^0 \triangleq \{ n_j \in \mathcal{V} \cup \hat{\mathcal{V}} \mid \{n_j \to n_i\} \in \hat{E} \}.
\]

The use of sum aggregation when updating the node representations is again motivated by interpreting the message vectors as forces, and then incorporating the elementary concept in classical mechanics that the net force acting on an object is the sum of all forces acting upon it. The updated edge embeddings \( \mathbf{e}_{i \to j}^k \) are found as the sum of the embedding from the previous step \( k - 1 \), plus the computed message value \( m_{i \to j}^k \),

\[
\mathbf{e}_{i \to j}^k = \mathbf{e}_{i \to j}^{k-1} + m_{i \to j}^k \text{ for all } \{n_i \to n_j\} \in \hat{E} \text{ where } i > j.
\]

After \( K \) rounds of message passing are complete, the final local learned representation \( \mathbf{z}_{i \text{ local}} \) is found for each real node \( n_i \in \mathcal{V} \) as the concatenation of \( \mathbf{v}_i^K \) with the sum of all incoming messages \( m_{j \to i}^K \) from its local graph

\[4\] An alternative message-aggregation function could however be selected here instead, however the selected function must be both permutation invariant and applicable to a variable number of inputs in order to be suitable for graphs with arbitrary topology \([34, \text{Section } 3.2.2]\).
neighbourhood $\mathcal{N}_i^G$,
\[
z_{i}^{\text{local}} = (v_i^K, \sum_{j \in \mathcal{N}_i^G} m_{j \rightarrow i}^K) \quad \text{for all } n_i \in \mathcal{V}. \tag{33}
\]

The importance of the virtual nodes and edges described in Section 2.3 to the processor stage of the emulator is illustrated in Fig. 6, for an augmented beam geometry generated using Algorithm 1. Because messages can only move by one neighbourhood at a time, it can take a large number of steps for information to disseminate across a dense set of real nodes. By contrast, the augmented nodes are less dense, allowing information to be disseminated more efficiently around the graph, as shown for example by the blue arrows.

The final, decode stage of the emulator makes a prediction $\hat{u}_i$ for the displacement for each real node $n_i \in \mathcal{V}$. This is done by first embedding the global parameters $\theta$ into an $M$-dimensional vector $z^\theta$ using the MLP $f_P$,
\[
z^\theta = f_P(\theta). \tag{34}
\]

Then, a prediction is made for the displacement at each node, using the MLPs $h^1, \ldots, h^D$ for each of the $D$ dimensions of the system
\[
\hat{u}_i^d = h^d(z^\theta, z^{\text{global}}_i, z^{\text{local}}_i) \quad \text{for all } n_i \in \mathcal{V} \text{ for } d = 1, \ldots, D. \tag{35}
\]

The decoder MLPs take as input the global parameter embedding $z^\theta$, global shape embedding vector $z^{\text{global}}$, and the local embedding vector $z^{\text{local}}$. Note how the node-decode MLP makes a prediction for each node individually, and each dimension individually, without explicitly accounting for any correlation structure. This is because any such structure is implicitly accounted for by the processor stage of the emulator, and can hence be ignored in the final decoder stage. The output $\hat{U}$ of the emulator is then the collection of all individual displacement predictions,
\[
\hat{U} = \{\hat{u}_i \text{ for all } n_i \in \mathcal{V}\} \text{ where } \hat{u}_i = (\hat{u}_i^1, \ldots, \hat{u}_i^D). \tag{36}
\]

Performing emulation with DeepGraphEmulator requires in particular the structure of the internal MLPs, the dimensionality $M$ of the internal embedding vectors, and finally the number of message-passing steps $K$ to be
specified. Greater values of $M$ and $K$ allow for more information to be incorporated in the final decoder stage of the emulator. However, this also leads to longer training and prediction times, as well as increases the potential for overfitting. As discussed above, there are $(2K + 3 + D)$ MLPs internal to the GNN. In addition, the outputs of all MLPs with the exception of the final node decode MLPs $\{h_d^d\}_{d=1}$ are processed with a LayerNorm [37]. The trainable parameters denoted $\omega$ of the GNN then consist of the weights and biases of all internal MLPs, as well as the LayerNorm parameters.

Before moving on to describe the experimental results for the two systems considered for emulation, we first discuss the motivation behind two of the design choices of the architecture that are particularly advantageous for emulation of LV mechanics. Firstly, note the inclusion of the global shape embedding vector $z_{\text{global}}$ in the decode stage. This allows the decoder MLP $h$ to assimilate information specific to each individual node $n_i \in V$ ($z_{\text{local}}^i$) together with information about the global shape of the geometry when predicting the forward displacement of the LV, which is useful as each LV anatomy will have a unique shape. Secondly, consider how the parameter vector $\theta$ is only included at the final, decode stage of the GNN. This means that, when considering repeated forward evaluations for a fixed initial geometry under different input parameter values, the encode and process state can be precomputed exactly once initially. Subsequent evaluations then only require the decode step to be performed. Since the processor stage constitutes the bulk of the computational requirements of the emulator, forward evaluations of the surrogate model can be made highly efficiently in this case. Again, this is useful in LV emulation, as the emulator must deliver significant computational savings over the numerical simulator to be useful for real-time estimation tasks, which in general will be required for the fixed LV geometry of a given subject.

3. Beam deformation application

We first illustrate the application of the proposed GNN emulation framework to modelling the displacement of a 2-D beam. Specific details of the mechanics problem are given in Section 3.1, as well as a description of how training, validation and testing datasets were generated. Section 3.2 details how the emulator was implemented, and finally Section 3.3 presents and discusses the emulation results.

3.1. Mathematical details and data generation

In this section, we will introduce a 2-D clamped beam that is deformed under its own weight, this is a simple 2-D linear elasticity model based on a similar example from FEniCS [38]. In brief, Fig. 7(a) shows an example of a beam geometry in its reference configuration, while Fig. 7(b) displays the beam in its current, displaced configuration with the left side fixed. Denote the Cauchy stress tensor $\sigma$, the linear strain tensor $\epsilon = \frac{1}{2} (\nabla u + (\nabla u)^T)$ where $u$ is the displacement field. By assuming the material is isotropic and linear, we have

$$\sigma = \lambda \text{tr}(\epsilon) I + 2\mu \epsilon,$$

where $\theta = (\lambda, \mu)$ gives the Lamé parameters, and $I$ is the identity matrix. From Eq. (1), the balance equations for this beam ($\Omega$) can be written as

$$-\nabla \cdot \sigma = f \quad \text{in} \quad \Omega,$$

$$u = 0 \quad \text{at} \quad \partial \Omega^d,$$

where $f = (0, -\rho g)$ is the gravity force density with $\rho$ the density of the beam and $g$ the gravity acceleration, and $\partial \Omega^d$ is the clamped end that is the left side of the beam. No traction is applied to the rest of the boundaries. Detailed numerical implementation can be found in the FEniCS documentation [38].

Three hundred individual beam geometries were generated for emulation, all of which shared a common triangular finite-element mesh with 96 nodes and 154 elements. Each beam’s length was found by uniformly randomly sampling within the range [40, 70] mm, and its width was set as a randomly chosen fraction of the length within the range [0.20, 0.40]. A forward simulation was run for each beam according to Eq. (38), with a uniformly random sample of $\theta \in [4, 10]^2$, where the Lamé parameters are in units of stress [38]. The first 225 simulations were used as training data on which the emulator could be fit, the following 25 for validation and the remaining 50 were reserved as an independent test set. Note that this sampling procedure ensured each data point had unique beam geometry and Lamé parameter values.
3.2. Implementation details

Emulation was performed for several different values of the number of message-passing steps $K$, dimensionality $M$ of the internal embedding vectors within the GNN, and cardinality of the virtual node layers $\eta$, to quantify the impact of these hyper-parameters on emulation accuracy. In practice, these values could be selected using a held-out validation set of simulations. The global shape embedding input to Algorithm 2 for each individual beam geometry was set to be the two-dimensional vector giving the beam’s length and width. All encoder, processor and decoder MLPs internal to the emulator were implemented with two hidden layers each of width 128, using tanh activation function. For each value of $\eta$ considered, a single augmented graph topology $\tilde{E}_{\text{Beam}}$ for use on all training, validation and test beams was found by applying Algorithm 1 to a reference beam geometry. The reference beam was generated using the “averaging” approach described in Section 2.3. Emulation results for an alternative choice of representative graph are given in Appendix A, with the results obtained being very similar to those presented in Section 3.3. The augmented nodes for each individual training, validation and test beam were then generated in the manner discussed in Section 2.3. For all beam graph representations, each edge $\{n_i \rightarrow n_j\}$ where $i > j$ was assigned a feature vector $e_{i \rightarrow j}$ as given in Eq. (23). Each real node was assigned a single, Boolean feature variable, which indicated if it lay on the Dirichlet boundary $\partial \Omega^d$ from Eq. (38). Augmented nodes were then assigned features sequentially as detailed in Section 2.3. Before training, all input features and output displacement values were normalised to mean zero and unit variance. Training was performed using the Adam optimiser [39] with a batch size of one for 300 epochs using a fixed learning rate of $5 \times 10^{-5}$, after which the loss value on the validation set was found to plateau. The loss function used for training the GNN parameters $\omega$ was the mean prediction error between nodal displacements outputted by the simulator $u_i$ and those predicted by the emulator $\hat{u}_i$,

$$\mathcal{L}(\omega) = \frac{1}{|\mathcal{V}|} \sum_{i=1}^{|\mathcal{V}|} || u_i - \hat{u}_i ||_2,$$

(39)

where $|\mathcal{V}| = 96$ is the number of nodes in the FE representation of each beam. The 25 simulations in the validation dataset were used to monitor emulation performance, with the trained network parameters which gave the lowest error on this data saved for use in predicting the displacements of the 50 test simulations.

3.3. Results and discussion

Fig. 8 compares the test set prediction errors when emulation is performed on the non-augmented graph representations of the beam geometries against results on the augmented graphs, for $K = 1, \ldots, 6$ message passing steps.\(^6\) When operating on the non-augmented graphs, at least $K = 5$ message passing steps are required to ensure

\(^5\) Initial experiments with smaller MLPs gave slightly poorer predictive performance. We found two hidden layers of width 128 to be a good comprise between predictive accuracy and computational efficiency.

\(^6\) For ease of reference, all beam emulation results are tabulated in Appendix A.
the bulk of emulation errors are less than 0.1 mm, whereas almost identical performance can be obtained in only $K = 2$ steps when using the augmented representations. For reference, the average magnitude of the true nodal forward displacement vectors across the test data was 4.2 mm. Fig. 9(a) displays emulation errors for different values of virtual node cardinality input vector $\eta$, with results exhibiting low sensitivity with respect to this hyperparameter vector, and panel (b) displays errors for different choices of latent embedding dimension $M$. Errors tend to fall as $M$ increases, however for $M = 60$, we see a small increase in the distribution of tail prediction errors which could be evidence of slight overfitting. By using arguments similar to Takens’ embedding theorem for dynamical systems (see e.g. [40]), it is reasonable to assume there exists an optimal value for $M$, below which insufficient information underfitting may occur and beyond which more scope for overfitting is introduced.

The disparity between the augmented and non-augmented emulation errors for low values of $K$ from Fig. 8 exists because the emulator requires higher values of $K$ to distinguish between the internal nodes of the beam geometries when using the non-augmented graph representations. In particular, due to the regularity of the nodes in each beam geometry, as illustrated in Fig. 7(a), any nodes more than $K$ steps away from a boundary node will have the exact same local information available to them after $K$ message passing steps. This means they will share a common value of $z^{\text{local}}_i$ (Eq. (33)) and hence will be indistinguishable to the node-decode MLPs (Eq. (35)). Even then for this 2-D beam model, for a high fidelity FE mesh a large value of $K$ would be required to discriminate between internal nodes and achieve accurate emulation. By introducing virtual nodes and edges, this issue is alleviated. The use of augmented graph representations has other advantages also, beyond what can be seen in this simple example. For instance, when considering simulations involving the contact of different bodies, the virtual nodes and edges allow the contact information to be rapidly propagated across the entire body under contact [13].

The top row of Fig. 10 compares the predictions of the GNN emulator with $K = 2$ against the outputs of the simulator for two beam geometries from the independent test set. Note the difference in size between the two beams, which is a result of the random sampling approach used to generate them. The beam in the left column was the simulation for which the emulator obtained the first-quartile value of mean prediction error (0.03 mm), and the beam on the right the third quartile error value (0.05 mm). For both simulations, the emulator delivers accurate predictions across the entire geometry. The violin plots from Fig. 8 illustrate the accuracy/efficiency trade-off of the choice of $K$. That is, $K$ must be chosen such that the node-decode MLPs have sufficient information to make accurate predictions. However, higher values of $K$ also introduce the potential for the network to overfit to its training data. In this case, as more than $K = 2$ steps of message passing are performed using the augmented graph representations, the emulator begins to overfit. Nevertheless, the exact value of $K$ is far less critical than when performing emulation using the non-augmented graphs. The overfitting effect is visualised in the final two rows of Fig. 10, which compares the predictions of the GNN emulator with $K = 2$ against an emulator with $K = 6$. Despite the slight overfitting, the predictions of the two emulators appear qualitatively similar, with increased deviations for nodes farther from the clamped left side.
Fig. 9. Violin plots of test-set nodal GNN emulation prediction errors for different hyperparameter input values. Panel (a) shows the distribution of errors for six different values of $\eta$, with $K = 2$ and $M = 40$. Panel (b) plots the distribution of emulation error for $M = 3, 6, 10, 20, 40, 60$, with $K = 2$ and virtual node layer cardinality vector $\eta = [24, 6]$.

Finally, Fig. 11 displays the node coordinate values $\{x_i\}_{i=1}^{\nabla}$ of the same geometries from Fig. 10 in the top row of panels, while the bottom row displays two-dimensional projections of the corresponding local embedding vectors $\{z_i^{\text{local}}\}_{i=1}^{\nabla}$. The embeddings were generated using the GNN with $M = 40$ (which implies that each $z_i^{\text{local}}$ was of length 80) and $K = 2$. The projection values were found using Principal Component Analysis (PCA). The PCA projection matrix was learned using as input data the collection of individual values of $z_i^{\text{local}}$ across all the beams in the test data. The first two components, labelled PC 1 and PC 2, were found to explain over 44% of the variance in this data. The columns of the nodal coordinates in geometry space are coloured with a shading that is shared with the PCA projections, allowing for clusters in the projection values to be compared with coordinate position values. In panel (c), four distinct node clusters can be seen. Firstly, the boundary nodes in black are clustered far away from all other nodes, with high values of PC 1 and low values of PC 2. The second cluster consists of blue nodes near the left hand clamped boundary of the beam, which congregate at the top left of the scatter plot. The third cluster near the top centre of the plot consists mainly of red nodes from the right hand side of the beam. The final cluster consists of mostly nodes from the middle of the beam geometry which have low values of PC 1 and PC 2. Interestingly, the blue nodes which are nearest the clamped boundary in geometry space are furthest from the clamped boundary nodes in the projected space. The results in panel (d) are very consistent in relative terms with panel (c), while in absolute terms the projections in (d) tend to have slightly lower PC 1 value.
Fig. 10. Illustration of emulation performance for two test-set beam simulations. The beam in the left column had an initial width/height of 56.5/22.0 mm, and the simulation was run with Lamé parameters $\lambda = 7.03$, $\mu = 5.94$. The beam in the right column had initial dimensions of 66.1/16.7 mm with $\lambda = 6.52$, $\mu = 5.98$. The top row of plots shows the predictions of the GNN emulator with $K = 2$ rounds of message passing as red circles against the outputs of the simulator as black squares. The second row displays emulator predictions with $K = 2$ again as red circles, against predictions with $K = 6$ as blue squares. The final row of figures displays an enlarged version of the nodes inside the black squares from the figures in the second row, with coordinate values provided for reference. (For interpretation of the references to colour in this figure legend, the reader is referred to the web version of this article.)

4. Passive left ventricle mechanics application

This section describes the application of the proposed GNN emulation framework to the mechanics of the left-ventricle (LV) in diastole, the passive stage of the cardiac cycle. Mathematical and numerical details of the mechanics problem are given in Section 4.1. Section 4.2 describes the creation of simulation data on which the emulator was trained, validated and tested. A summary of existing work on the emulation of cardiac-mechanics is given in Section 4.3 before implementation details are given in Section 4.4. Emulation results are then presented in Section 4.5 and finally discussed in Section 4.6.
4.1. Mathematical details

The passive myocardium is modelled as an incompressible, hyperelastic and transversely isotropic material, which is reduced from the Holzapfel–Ogden HO model \([41]\), motivated by the results of previous uncertainty quantification study \([42]\). The adopted strain energy density function is given by

\[
\Psi = \frac{a}{2b} \left( e^{b(I_1 - 3)} - 1 \right) + \frac{a_f}{2b_l} \left( e^{b_l \max(I_{4f}, 1)} - 1 \right) + \frac{1}{2} \mathcal{P}(J - 1)^2, \tag{40}
\]

where \(\theta = (a, b, a_f, b_f)\) are the material parameters. Parameters \(a\) and \(b\) describe the isotropic response of the myocardium, \(a_f\) and \(b_f\) characterise the reinforced stiffness in the myofibre direction. The principal invariant \(I_1\) and transversely isotropic invariant \(I_{4f}\) are defined as \(I_1 = \text{trace}(C)\) and \(I_{4f} = f_0 \cdot (Cf_0)\), with \(C = F^T F\) the right Cauchy–Green tensor, \(F\) the deformation gradient, and \(f_0\) the unit vector in the myofibre direction in the reference configuration. The max() term in Eq. (40) ensures that the myofibres only support extension but not compression, while the final term in the equation enforces the incompressibility constraint with the Lagrange multiplier \(\mathcal{P}\) and \(J = \det(F)\).

The filling process of the LV in diastole is described by a quasi-static, pressure-loaded boundary-value problem over the computational domain \(\Omega\) occupied by the geometry of the LV. A linearly ramped pressure from 0 to 8 mmHg is applied to the endocardial surface \((\Gamma_{\text{endo}})\), and the basal plane of the geometry is fixed in the longitudinal and circumferential directions, with only radial expansion allowed during the filling process. Note a cylindrical coordinate system is introduced to the nodes in the basal plane. The system of equations at the current configuration
(\Omega_t) is given by:

\[
\begin{align*}
\nabla \cdot \sigma &= 0 \quad \text{in} \ \Omega, \\
\sigma \cdot n &= -p \quad \text{on} \ \Gamma^{\text{endo}}, \\
u_\theta = u_z &= 0 \quad \text{on} \ \Gamma^{\text{base}},
\end{align*}
\]

where \(n\) is the normal direction of the endocardial surface \(\Gamma^{\text{endo}}\) and \(p\) is the loaded pressure at \(\Gamma^{\text{endo}}\). The vectors \(u_\theta\) and \(u_z\) are the displacement components along the circumferential direction and \(z\)-axis at the basal surface \(\Gamma^{\text{base}}\), respectively. The above FE-based LV model (41) is then solved using ABAQUS (Simulia, Providence, RI, USA), a general-purpose finite-element package. Further details of the FE simulation of the passive LV dynamics are given in [43].

4.2. Simulation data generation

The emulation study was conducted using 3000 randomly generated, synthetic LV geometries. Synthetic geometries were used in the absence of a large dataset of real geometries. For full details of the generation procedure, see Appendix C. In brief, the synthetic LVs were generated by sampling in a 40-dimensional latent space found using PCA, before being projected back to the full geometry space. Using a latent space of this dimensionality ensured that the synthetic geometries exhibited realism in terms of asymmetry and wall-thickness variation. PCA was trained on a dataset of 65 real LV geometries, extracted from the CMR scans of healthy volunteers. Each LV geometry was represented using a three-layer, hexahedral mesh with 6874 nodes and 4995 finite-elements, with myofibre structure described by a rule based method (RBM) [43]. Fig. 12(a) illustrates both the form of the hexahedral mesh and the RBM for fibre generation.

To create simulation datasets on which the GNN emulator could be trained, validated and tested, a forward mechanics simulation was run for all 3000 synthetic LVs. Panel (b) in Fig. 12 shows an example LV geometry in its reference configuration, while panel (c) shows the form of the geometry after a forward simulation is run. Each simulation was performed with a unique value of the material parameter vector \(\theta = [a, b, a_f, b_f]\). The parameter values were found by running a Sobol sequence of length 3000 within the set \([0, 10]^4\), where \(a\) and \(a_f\) are measured in kPa while \(b\) and \(b_f\) are dimensionless quantities. A Sobol sequence is a low-discrepancy sequence, commonly used in the design of computer experiments to ensure a space-filling coverage of the experimental domain of interest [44]. The Sobol sequence was run on the log-scale rather than the uniform scale. Sampling on the log-scale favours material parameter vectors \(\theta\) with lower stiffness levels, which results in an increase in the magnitude of the simulated displacement of the myocardium from start to end-diastole. By contrast, samples on the uniform scale result in displacements that are smaller in magnitude than would be expected for real data [42]. The first 2250 simulation results were used as training data, and the following 150 were used as a validation set. The remaining simulation results were used as a test set. Note that 29 of the forward mechanics simulations failed to converge. Furthermore, some of the remaining simulations exhibited extremely large displacements beyond what would be expected to be observed in-vivo. For this reason, we excluded from the test set any simulation results where the cavity volume of the deformed geometry exceeded 850 mL, which left 567 test data points. For reference, the mean end-diastolic volume for healthy males has been found to be 160 mL, with standard deviation of 27 mL [45, Table 2].

An additional dataset of 350 simulations was generated for a fixed LV geometry, randomly chosen from the test set. Again, each simulation was run with a different material parameter input vector \(\theta\), the values of which were found by continuing the Sobol sequence beyond the last index from the above data. The fixed LV geometry simulations were used to quantify how transfer learning on a specific geometry of interest can affect emulation performance. Transfer learning involves taking a pre-trained machine learning model and fine-tuning it for a new, but related application domain [46]. Details of the transfer-learning approach used in this instance are given in Section 4.4.2. The first 85 simulations were used for training, with 15 reserved for validation. Four simulations failed to converge, leaving 246 test points.
Fig. 12. Panel (a) shows the hexahedral FE mesh structure of a LV along with the RBM generated myofibre field, where myofibre orientation varies smoothly from endocardium to epicardium. Panel (b) shows the endocardial and epicardial surfaces of an example LV geometry in its initial reference configuration. Panel (c) then shows the deformed LV geometry after a simulation following Eq. (41) is run.

4.3. Existing work on cardiac-mechanic emulation

Emulation has been widely applied in LV mechanics, due to the numerical expense incurred in simulating soft-tissue mechanical systems. As discussed in Section 2.2 however, traditional emulation methods such as MLPs are not suited to emulating LV mechanics due to the excessive dimensionality of the simulations that are required for high fidelity results. Instead, a common approach in the literature has been to emulate certain numerical properties of the displaced geometry outputted by the simulator, such as LV inner cavity volume (LVV), instead of directly emulating the displacement of the myocardium itself. Consider for example the additional simulation data described in the last paragraph of Section 4.2, that were created for a fixed LV geometry. If we calculate the LVV resulting from each simulation, a dataset where inputs are the four material parameters in $\theta$ and outputs being scalar LVV
can be created, which is appropriate to model with a traditional surrogate modelling technique. This is the approach taken in [47], where Gradient Boosted trees were used to emulate blood pressure, myocardial stresses and LVV through the full cardiac cycle. In [48], polynomial chaos emulators were used to conduct a sensitivity analyses of the passive mechanics of the LV, considering outputs such LVV and apex lengthening. Finally, [7,49] used Gaussian process emulators for LVV and circumferential stains to perform fast parameter inference for the material properties of the underlying constitutive law.

All of the above works performed emulation for a single, fixed LV geometry. In order for a traditional emulation method to be able to generalise to different LV anatomies, a low-order representation of the true LV is required as input to the emulator. This representation is typically found using PCA. In [50] for example, a 5 dimensional representation of the LV geometry was found using PCA, while in [51] a 2 dimensional PCA representation of the LV was used. Low-order parameterised representations for the LV geometry have also been considered for training emulators [52]. However, [53] has shown that such parameterised methods incur higher reconstruction error than PCA.

To benchmark the emulation performance of the GNN surrogate model, we compare its predictive accuracy for LVV with that of an MLP emulator on the test data set of 567 simulations described in Section 4.2. The MLP learns a forward map of the form

\[
(\theta, z^{\text{global}}) \mapsto \text{LVV}
\]  

on the training data, whereby a prediction of LVV is made directly given input values for the global material parameters \(\theta\) and global shape coefficients \(z^{\text{global}}\) respectively. We have previously found that MLPs achieve strong performance for emulation of LV mechanics [54]. In addition, the computational efficiency of MLPs allows for gradient-based MCMC approaches to be used for inverse problems, see [50] for example.

### 4.4. Implementation details

Emulation for the LV model in diastole was performed using the same augmented graph topology for each LV geometry, denoted \(\hat{\mathcal{E}}^{\text{LV}}\). A common augmented topology was used as all LV geometries share a common FE mesh structure and hence share a common non-augmented graph topology, \(\mathcal{E}^{\text{LV}}\), which is derived as in Eq. (8). \(\hat{\mathcal{E}}^{\text{LV}}\) was generated by applying Algorithm 1 to a reference graph with nodes \(\hat{\mathcal{V}}\) and graph topology \(\mathcal{E}^{\text{LV}}\), where \(\kappa = 5\) and \(\eta = [813, 126, 13]\). The node set \(\hat{\mathcal{V}}\) was found using the “averaging” approach discussed in Section 2.3. The value of \(\kappa\) was chosen so virtual nodes would have approximately the same number of intra-layer neighbours in \(\hat{\mathcal{E}}^{\text{LV}}\) as each real node has neighbours in the FE mesh (5.46 on average). The value of \(\eta\) was selected so that the cardinality of each successive layer of virtual nodes decreased by approximately a factor of 8, which is the number of real nodes in each hexahedral finite-element. Note that the usual Euclidean metric was used for the operations performed in Algorithm 1. Performance could potentially be improved by instead using a metric which accounts for the non-convexity and curvature of the myocardium, however we leave this to future work. Each real node \(n_i \in \mathcal{V}\) was assigned a feature vector \(v_i\) for processing by the emulator, as discussed in Section 2.3. The feature vector for each node included two Boolean indicator variables, which were equal to one if the node lay on \(I^{\text{base}}\) or \(I^{\text{endo}}\) respectively, and zero otherwise. Additionally, the 3-D fibre orientation vectors generated by the rule-based method at each node were assigned to \(v_i\). Finally, each node feature further included the radial direction of the underlying node in the FE mesh (the vector \(e_i\) in Fig. 12(a)), which helped the emulator learn the boundary condition on \(I^{\text{base}}\). Edge features were assigned as in Eq. (23). Emulation was performed with fixed \(M = 40\), which was the value that gave the lowest prediction error for the beam dataset. The structure of the MLPs internal to the GNN was the same as described in Section 3.2, as were the training details, with the exception that training was performed using a fixed learning rate of 5 \(\times\) 10\(^{-5}\) for 3000 epochs, after which loss on the validation dataset was found to plateau. Because the LV geometries under analysis are synthetic, generated by sampling coefficients in a 40-dimensional latent space obtained using PCA (see Appendix C for details), we used these principal component coefficients for input to the emulator as the global embedding vector \(z^{\text{global}}\). Note that for real LV geometry data, the global coefficients could be found by applying the PCA projection matrix as in Eq. (C.1). To analyse the impact of this hyper-parameter on emulation performance, emulation was performed with \{0, 8, 16, 24, 32, 40\} PC coefficients inputted to \(z^{\text{global}}\). Emulation performance was also quantified for \(K = \{1, 2, 3, 4, 5, 6\}\). Preliminary experiments with regularisation techniques including weight decay and dropout yielded no increase in emulation...
accuracy. Note that our use of a stochastic training approach (Adam [39] with batch size of one) can be seen as an implicit regularisation method. This is because stochastic training is practically similar to training with added noise, which can help prevent overfitting [55] and is equivalent to Tikhonov regularisation [56].

4.4.1. MLP emulator implementation details

As discussed in Section 4.3, we compare the performance of the GNN emulator in predicting LVV on the varying geometry dataset against an MLP emulator, which directly predicts LVV using a forward map of the form of Eq. (42). The MLP was implemented with two hidden layers, each of width 128, and tanh activation function. A point estimate for the weights and biases was found by optimising the squared error between the true LVV values and those predicted by the MLP on the training data set of 2250 simulations. Optimisation was performed using the Adam algorithm [39] with a batch size of 200 for 1000 steps, after which validation set loss was found to plateau. We have found that the predictive accuracy of the MLP emulator can be significantly improved by performing pre-transformations of the two inputs \( \theta \) and \( z^{\text{global}} \) before training and evaluation. Firstly, the material parameters \( \theta \) were inputted on normalised log-scale, rather than uniform scale. Secondly, the individual principal component coefficients in \( z^{\text{global}} \) were not normalised independently before training. Instead, all coefficients were normalised with respect to the variance in the training data along the direction of the second principal component. These transformations were informed by our previous experience in emulating LV mechanics [54].

4.4.2. Transfer learning details

The GNN emulator is designed to be able to perform forward evaluations (without retraining) for arbitrary values of the LV geometry and material parameter inputs respectively. In practice however, particularly in clinical applications, predictions for different material parameters may only be required for one specific LV geometry of interest. In this case, the original emulator can be fine-tuned using additional forward simulations of this specific LV geometry, which is referred to as transfer learning. Transfer learning is more efficient and can lead to more accurate predictions than re-training the emulator from scratch on the additional simulations [46]. We performed transfer learning by taking a GNN emulator that was pre-trained on the original data, and fine-tuning its parameters on the dataset of 85 forward simulations for a randomly chosen, fixed LV geometry. See Section 4.2 for details of this fixed geometry dataset. The transfer learning was performed with the Adam optimiser, using a learning rate of \( 1 \times 10^{-5} \) for 1500 epochs. We used a lower learning rate than used on the original dataset to decrease the stochasticity observed in the trace plots of the loss on the validation set of 15 simulations, however we found that the learning rate does not have a significant impact on emulation performance. Furthermore, only the weights and biases of the three node-decode MLPs in the final stage of the GNN were updated during the transfer learning. All other trainable parameters were held fixed to the values determined by the original dataset.

4.5. Results

4.5.1. General dataset

Fig. 13 displays violin plots of the test-set nodal prediction errors (in mm) for the GNN emulator for different hyperparameter input values.\(^7\) Panel (a) shows the distribution of errors with 0, 8, 16, 24, 32 PCs retained in \( z^{\text{global}} \) with \( K = 5 \) message passing steps. We see that emulation error decreases monotonically as more components are included. Panel (b) shows the distribution of errors for \( K = 1, \ldots, 6 \) with 32 coefficients retained in \( z^{\text{global}} \). Again we see a decrease in error as more message passing steps are performed, but this effect is less pronounced than in panel (a). For subsequent experiments, we proceeded with the GNN emulator with \( K = 5 \) and 32 PC coefficients retained in \( z^{\text{global}} \). This emulator obtained a median nodal-prediction error of 0.27 mm on the independent test set, with interquartile range [0.15, 0.50] mm. For reference, the average magnitude of the true nodal forward displacement vectors across the test data was 11.0 mm. We chose \( K = 5 \) as the difference in emulation performance was marginal compared with \( K = 6 \). We do not retain all 40 PCs in \( z^{\text{global}} \), as we believe this better represents the case for real data where a perfect low order representation of the LV anatomy will not be available.

\(^7\) For ease of reference, all LV emulation results are tabulated in Appendix B.
4.5.2. Comparison with MLP emulation

Fig. 14 presents a comparison of test set LVV prediction errors for the MLP and GNN emulators. Performance is evaluated in terms of the absolute percentage error between the predicted and true LVV values. The GNN emulators predict the value of LVV by explicitly calculating the volume inside the deformed endocardial surface derived from the forward prediction of the GNN. The MLP emulators directly predict the value of LVV at end-diastole with a forward map following Eq. (42), where 32 PCs were retained inside $z^{\text{global}}$. The violin plots of MLP 1 and GNN 1 give emulation results when the pre-transformations of inputs $\theta$ and $z^{\text{global}}$ detailed in Section 4.4.1 are not applied before training, while the violin plots of MLP 2 and GNN 2 give the emulation results when these transformations are applied. Without the input transformations, the MLP incurs very large errors in the prediction of LVV, with median absolute error of 20.3%. When the input transformation is performed, error for the MLP approach is significantly reduced, with median error of 4.4% and interquartile range [1.95%, 7.8%]. However, these errors are still substantially larger than those of the two GNN emulators. In particular, GNN 2 achieves median prediction error of 0.49% and interquartile range [0.24%, 0.89%]. This again presents an improvement over the case where the input pre-transformations are not applied, however the difference however is much less pronounced in this case, with GNN 1 achieving median error of 0.60%. The main difference between the two GNN results is that the pre-transformation helps to reduce the spread of the outlier points in the tail of the error distribution. Comparing the emulation results of the MLP 2 and GNN 2, we see the GNN achieves prediction errors approximately one order of magnitude lower than the MLP emulator. In addition, the prediction errors of the GNN emulator were less heavily skewed. The largest error for the GNN was 6.8%, whereas the MLP incurred an error of over 50% for one test.
Fig. 14. Comparison of test set LVV prediction errors for MLP and GNN emulation. Performance is evaluated in terms of the absolute percentage error between the predicted and true LVV values. The violin plots of MLP 1 and GNN 1 give emulation results when the pre-transformations of inputs $\theta$ and $z_{\text{global}}$ detailed in Section 4.4.1 are not applied before training. The violin plots of MLP 2 and GNN 2 then give the emulation results when these transformations are applied.

Fig. 15. Comparison of emulation performance of the general emulator and the emulator transfer-learned on the specific LV geometry of interest. Panel (a) shows violin of prediction errors in the nodal displacement of the deformed geometry (mm), and (b) shows violin plots of errors in the prediction of LVV (%).

point. This outlier point was run with a large value of material parameter $a$ (9.03 kPa) and very low value of $b$ (0.10). For reference, the error of the GNN emulator for this point was 1.6%.

4.5.3. Transfer learning on a fixed LV geometry

We also explore the effect of transfer-learning on the accuracy of the GNN emulator. Initialising on the trained parameter values from the above GNN emulator with $K = 5$ and 32 PCs retained in $z_{\text{global}}$, we transfer-learned the GNN on a set of 85 simulations for a randomly chosen, fixed LV from the held out set of 567 geometries. Fig. 15(a) compares the test-set nodal prediction accuracy of the original and transfer-learned emulators and Fig. 15(b) compares their predictive accuracy for LVV. Nodal prediction errors tend to fall by greater than a factor of 2, with the transfer-learned emulator incurring a median node prediction error of 0.11 mm compared to 0.29 mm for the original emulator. The increase in LVV prediction accuracy was more slight, with the transfer-learned emulator achieving a median error of 0.34%, compared to 0.43% for the original emulator.

To further characterise emulation performance, we consider the distribution of prediction errors across the LV geometry surface in Fig. 16. We selected for plotting the three test simulations where the transfer learned GNN attained the 25th percentile (0.2%), median (0.3%) and 75th percentile (0.6%) error values in the prediction of LVV. The original GNN incurred errors of 1.4%, 0.1% and 0.2% respectively for these simulations. Fig. 16 displays the
three ground truth LV geometries outputted from the simulator in red, against the predictions of the original and transfer-learned GNNs respectively in blue. The first row is for the predictions of the original GNN, and the second for the transfer-learned GNN. The first column shows the simulation where the transfer-learned surrogate achieved the 25th percentile error in LVV, the second the median, and the third the 75th percentile. For the original GNN, we tend to see slightly higher prediction errors towards the apex of the LV in panel (a). For all other predictions for both the original and transfer-learned GNNs, it is difficult to discern a spatial pattern in the predictions errors, as predictions are highly accurate across the entire geometry.

Using the same fixed LV geometry as above, additional forward simulations were run over a grid of values in $[a, b]$ space with fixed $a_t = b_t = 1$ (approximately the median values from the training data), and predictions were
Fig. 17. Distribution of emulation error over \( [a, b] \) space, with fixed \( a_l = b_l = 1 \) and fixed LV geometry. The top row shows heatmaps of mean displacement errors over \( [a, b] \) space (in mm), while the second row shows heatmaps of absolute percentage error in LVV prediction. The left column shows the results when using the original emulator for predictions, while the second column shows results when using the emulator transfer-learned on the fixed LV anatomy.

made using both the general and transfer-learned emulators. Constraining the input space to be two dimensional allows plots of the distribution of prediction errors to be produced, which are displayed in Fig. 17. The first column of Fig. 17 shows the results for the original emulator trained on the varying-geometry dataset, and the second column the results for the emulator transfer-learned on this specific LV anatomy. The top row displays displacement prediction error (in mm) over \( [a, b] \) space, and the second row shows absolute percentage LVV prediction error. The transfer learned emulator tends to attain higher accuracy across the entire space in both cases. For displacement error, both emulators incur highest errors at the corner of the space where \( a \) and \( b \) are low. For LVV, the distribution of emulation errors over \( [a, b] \) space is less smooth. For the general emulator, highest errors occur for very low values of \( a \) and \( b \), while for the transfer learned emulator, highest errors are incurred for low values of \( a \) where \( b \) is in the range [5, 10].

4.5.4. Visual analysis of latent learned embeddings

Fig. 18 displays a comparison of node coordinate values \( \{x_i\}_{i=1}^{V} \) against 2-D projections of the corresponding local embedding vectors \( \{z^{local}_j\}_{j=1}^{V} \), for two randomly selected LV geometries from the test set. The top row of panels shows short-axis views of the two geometries, where the slice is taken mid-way up the myocardium. This plot is analogous to Fig. 11 for the beam data, however in this case we restrict our analysis to the epicardial points.
on a single short-axis slice rather than considering all nodes in the FE mesh, allowing for 2-D plots to be made. PCA was applied to the local embedding vectors $\mathbf{z}^\text{local}_i$ along this slice of the epicardium, across all 567 geometries from the varying geometry test data set. The local embedding vectors were found using the GNN with $K = 5$ message passing steps and $M = 40$. The first two PCs accounted for over 50% of the variance observed in the data. The projection values in the bottom row of panels are coloured with the same shading as the nodal coordinate values in the top row. For both LV geometries, the projection values exhibit a similar periodicity to that seen in the nodal coordinate values. For the blue points with the lowest values of PC1 however, we see in both cases a more acute curve in the surface of the projection values than in the nodal coordinate values. Note however that the projection values also account for the myofibre structure of the LV, which is not shown in the nodal coordinate plots.

4.5.5. Prediction speed

Finally, the wall-clock forward prediction times of the GNN surrogate model and the numerical simulator are compared in Table 1. Note that simulation times for the numerical solver are dependent on the values of the material stiffness parameters, with lower stiffness configurations generally leading to longer prediction times. The values
Table 1

Prediction times (seconds) for forward simulator and GNN emulators. The first row shows emulator prediction times when run on CPU, and the second row shows GPU prediction times.

<table>
<thead>
<tr>
<th>Simulator</th>
<th>Speedup factor</th>
</tr>
</thead>
<tbody>
<tr>
<td>$a = b = a_t = b_t = 1$</td>
<td></td>
</tr>
<tr>
<td>$2.0 \times 10^{-3}$</td>
<td>$9.6 \times 10^{-3}$</td>
</tr>
<tr>
<td>$4.5 \times 10^{-3}$</td>
<td>$2.4 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

indicated were run with $\theta = [1, 1, 1, 1]$, approximately the median values from the simulation data set. The first row of Table 1 compares the prediction times of the emulator when run on a CPU (Dual Intel Xeon Gold 6138 2.0 GHz), and the second row gives GPU (Dual Quadro RTX 8000) prediction times. In addition, two forward evaluation times for the emulator are presented for each type of hardware. The column “Varying Geom.” in Table 1 shows the prediction time when all three stages from Algorithm 2 are performed. We have however designed the emulator such that repeated predictions for a fixed LV geometry over different material parameter values $\theta$ are particularly efficient. Repeated forward evaluations of this type are required for example in the context of an inverse problem where the material stiffness properties of a patient’s LV are estimated from experimental data. Stiffness estimates obtained in this manner have been found to deliver insights into cardiac function that are relevant for clinical diagnostic purposes [57]. As discussed in Section 2.4, for a fixed LV geometry, the message passing stage of the network can be pre-computed exactly once to find $z_{\text{local}}^i$ for each node $n_i \in V$, which constitutes the bulk of the computations of the emulator. The column “Fixed Geom.” in Table 1 shows the evaluation times of the emulator once this pre-computation has been performed, where only the final node-prediction MLP has to be evaluated. On CPU, the varying-geometry emulator requires $2.0 \times 10^{-3}$ s to make a forward evaluation, while in the case of a fixed geometry, predictions can be made in only $9.6 \times 10^{-3}$ s. By contrast, 78 s are typically required for the numerical simulator to converge. Forward evaluations of the emulator are even faster on the GPU, requiring $4.5 \times 10^{-4}$ s in the general case, and $2.4 \times 10^{-4}$ s for a fixed LV geometry. In particular for the fixed geometry case, this gives a speed up of over five orders of magnitude compared to the simulator.

4.6. Discussion

In this work we have introduced an emulation framework based on a novel GNN architecture. The GNN uses a three stage, encode-process-decode approach to perform emulation (Algorithm 2). The processor stage involves the processing and propagation of messages around an augmented form of the graph structure induced by the underlying FE mesh. The augmented structure involves the use of additional virtual nodes and edges, generated as in Algorithm 1, to increase message-passing efficiency. We then applied the framework to two separate mechanics problems, the first of which was the toy problem of a 2-D beam with linear and isotropic elastic properties, clamped at one end and deformed under its own weight due to gravity. The second was a 3-D model of the LV in diastole, carried out using synthetic LV geometries with myocardium characterised by a non-linear and transversely isotropic constitutive law, and myofibre field generated by an RBM. Experiments were performed to evaluate the accuracy and computational efficiency of the proposed emulation framework.

4.6.1. General dataset

Fig. 13 shows that the prediction errors of the GNN emulator decrease both as more PCs are included in $z_{\text{global}}$ and as more message passing steps $K$ are performed. This effect is more pronounced for increasing PCs than for increasing $K$. The importance of including additional PCs in $z_{\text{global}}$ shows that information about the global shape of the LV geometry helps to improve the generalisation performance of the emulator, which is likely due to the high level of variation observed between different LV anatomies.

4.6.2. LVV benchmark against MLP

The GNN emulation approach is substantially more accurate than an MLP emulator in the prediction of LVV, achieving a reduction in prediction errors of approximately one order of magnitude, as illustrated in Fig. 14. Note also in Fig. 14 the sensitivity of MLP emulation performance to an application specific pre-transformation of the
input variables. Without this transformation, informed by our experience in modelling this type of simulation data, the emulation performance is extremely poor. On the other hand, the GNN emulation results are much less sensitive to this transformation, suggesting that the GNN requires less domain specific feature engineering to obtain accurate results than a traditional emulator. Note that the GNN and MLP emulators also differ substantially both in terms of number of trainable parameters and training times. Specifically, the MLP emulator has 21,377 parameters and took approximately 1 min to train (Dual Intel Xeon Gold 6138 CPU), while the GNN has over 520,000 parameters and took 27 h to train (Dual Quadro RTX 8000 GPU).

4.6.3. Transfer learning on fixed LV geometry

The use of transfer learning in the case of a fixed LV geometry can lead to improved emulation performance. Fig. 15(a) demonstrates that nodal emulation prediction errors decrease by approximately a factor of three when using less than a hundred additional training simulations. LVV predictions from Fig. 15(b) also improve, but the effect was less pronounced. Recall that LVV was not incorporated in the loss function used when transfer learning. If a quantity such as LVV is of particular interest, the loss function could be adjusted to target this quantity specifically. The additional training input locations were chosen using a space-filling design. This could be improved using an active-learning approach, where the input locations are selected to target those areas of the input space where the emulator is less accurate [5, Chapter 6]. For example, considering the heatmap in the bottom right of Fig. 17. An active-learning approach that accounts for prediction errors over this space would select additional simulation data $a < 1$ and $5 < b < 10$ to target the region of higher error.

4.6.4. Prediction speed

The objective of the present work is to allow for patient-specific computational cardiac models to be eventually applied in clinical decision support roles, for which rapid evaluation times are required. Note that while generating the simulation datasets and training the GNN emulator is computationally expensive, the advantage of our approach is that these computations can be pre-computed in advance as no re-training is required for the emulator to be applied to an arbitrary LV anatomy of interest. A further increase in performance can be achieved however in the case of a fixed LV of interest using a small number of additional forward simulations, as discussed in the previous subsection. The results in Table 1 show that the emulator delivers significant gains in computation time over the simulator at the prediction stage, i.e. once training is complete. In the case of performing simulations for a fixed LV geometry, the GNN emulator can perform over 4000 forward evaluations per second on GPU. The GNN emulator can also leverage automatic differentiation to compute exact derivatives (to machine precision) at negligible additional computational cost. This combination of rapid and accurate predictions available with gradient information means that our approach is suitable for performing numerical experiments such as forward and inverse problems in real-time.

4.6.5. Limitations and future work

The approach presented in this work could be extended for more realistic and patient-specific cardiac simulations. Specifically, the myofibre field of each synthetic LV geometry considered here was assigned by an RBM, see Appendix C. Each real LV geometry however will have a unique myofibre field, the structure of which can have a significant impact on the mechanical response of the LV in diastole [58]. In addition, we have assumed that the material properties of each LV are constant across all nodes. Future work could relax this assumption to allow stiffness values to vary for different regions of the LV, which can be achieved by simply inputting a local value $\theta_i$ in the decoder stage of Algorithm 2 rather than a global value $\theta$. This would be especially useful for emulation of the passive mechanics of post myocardial infarction (MI) patients, as the myocardium can exhibit significantly higher stiffness levels in the MI region. In this case however, removing the global material parameter information from the message passing stage of the GNN as we have done in Algorithm 2 may produce sub-optimal results. An indicator variable in each node feature vector $v_i$ indicating healthy or diseased status could be sufficient though to achieve accurate emulation. Furthermore, this study has restricted its analysis the passive phase of the cardiac cycle. Future work could consider the active phase and model the electrophysiologically driven contraction of the heart.

In addition, the architecture of the GNN itself could be more specifically tailored to the LV model. Note how we have used the exact same architecture for both the 2-D beam model and 3-D LV model. For both models, the operation of running a forward simulation of a given geometry commutes with the operation of translation of the
geometry in space. We have designed the emulator to also satisfy this commutation property, through the use of relative displacements in the edge features from Eq. (23). For the LV model, we have an additional commutation relationship, which the GNN architecture from Algorithm 2 does not satisfy. That is, the operation of running a forward simulation of the LV in diastole commutes with the operation of rotating the geometry around the z-axis. Future work could develop a GNN architecture which does satisfy this commutation relation, potentially improving emulation performance. Further insight into the possibilities offered by a more application specific emulator design can be drawn by considering the results in Fig. 18, which show that the emulator has learned a periodicity in the local embedding vectors $z_{\text{local}}^i$ that mirrors the periodicity seen in the coordinate values $x_i$ for a slice of nodes on the epicardium. The periodicity in the latent space in turn helps to ensure a periodicity in the displacement predictions of the emulator, which we know to be true in advance as we do not consider ruptures of the myocardium. While it is promising that the emulator has learned this periodicity property automatically from the simulation data, emulation performance could potentially be improved by instead explicitly enforcing periodicity by adjusting the design of the emulator itself. This could be done by working with prolate spheroidal coordinates for example when performing emulation, rather than using the Cartesian coordinate system.

Extending the modelling framework to more complex and patient-specific simulations makes the general emulation problem more challenging in turn, as it increases the dimensionality of the input space of the emulator. In this case, it is likely that transfer-learning the general emulator for a specific LV anatomy and myofibre field of interest will be required to maintain highly accurate emulator predictions. However, transfer-learning in a data-driven manner as we consider in this paper would require numerical forward simulations to be performed, which are expensive to obtain. An alternative would be to transfer-learn in a physics-informed manner [59]. With physics-informed training, the parameters of the surrogate model are learned by minimising a potential-energy [60] or PDE residual based loss function [61]. The advantage of physics informed loss functions is that they can be evaluated without requiring any numerical simulation data, potentially allowing for transfer learning to be performed rapidly in-clinic. Physics-informed training has been applied in cardiac mechanics using non-GNN based surrogate models [60,62].

5. Conclusion

This work has introduced a novel GNN framework for emulation of FE based numerical simulators. The GNN uses a three-stage, encode-process-decode approach to perform emulation, implemented on an augmented graph representation of the underlying FE mesh. The GNN can easily generalise to different systems without retraining, both in terms of the geometry of the underlying mesh and any material constitutive parameters. The accuracy and efficiency of the emulator was compared against numerical simulations for two mechanics problems; a 2-D beam model and a 3-D model of the left-ventricle in diastole. Results are presented showing the emulator delivers accurate out of sample predictions comparable to the simulator, while being able to make forward evaluations several orders of magnitude more quickly. These results illustrate the promise of our GNN emulation framework compared to traditional approaches, and constitute an initial step towards eventual application of GNN emulation for clinical decision support.

Declaration of competing interest

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

Data and code availability

The GNN (Algorithm 2) was implemented in Python using the JAX [63] and Flax [64] libraries, while the MLP emulator from Section 4 was implemented using the scikit-learn machine learning library [65]. All GNN emulation code is available at https://github.com/dodaltuin/passive-lv-gnn-emul, along with the beam and fixed LV geometry simulation data. The varying LV geometry data is available at https://doi.org/10.5281/zenodo.7075055 [66].

Acknowledgements

This work as funded by the Engineering and Physical Sciences Research Council (EPSRC) of the United Kingdom, grant reference numbers EP/T017899/1, EP/S030875/1 and EP/S020950/1.
Appendix A. Additional beam emulation results

The emulation results for the non-augmented graph representations from Fig. 8 are tabulated in Table A.2, while the results for the augmented graph representations are given in Table A.3. Table A.5 displays emulation results for different values of the node cardinality input vector to Algorithm 1, $\eta$, from Fig. 9(a), while Table A.4 gives the results from Fig. 9(b), displaying emulation errors for different values of internal embedding dimension $M$. Note that for these tables the representative graph was chosen using the “averaging” method described in Section 2.3.

Tables A.6–A.8 display the analogous results to Tables A.3–A.5 except that a template beam generated using FEniCS was used for representative graph. All triangular elements of the template had height and width of 10 mm each, giving total height/width of 70/110 mm. The variation in emulation performance for the two choices of reference graph (on the order of $1 \times 10^{-2}$ mm) is very slight relative to the magnitude of the mean nodal forward displacement of 4.2 mm from the test set.

Appendix B. Additional LV emulation results

Table B.9 gives emulation error statistics for different numbers of PCs retained in $z^{\text{global}}$ from Fig. 13(a), while Fig. 13(b), which displays emulation error against number of message passing steps $K$, is tabulated in Table B.10. Table B.11 displays the comparison between GNN and MLP prediction errors for LVV from Fig. 14. Finally, the comparison of prediction accuracy for the general and transfer-learned GNN emulators from Fig. 15 is given in Table B.12. In addition, learning curves for the GNN and MLP emulators are displayed in Fig. B.19, for both training and validation data.

Tables B.13 and B.14 display MLP emulation error statistics when early stopping using patience is applied during training. When using patience, either the parameters for which the patience level is exceeded can be returned, or back-tracking can be applied to return the network parameters for which validation loss is minimised. Table B.13 displays results when back-tracking is applied, and Table B.14 shows results when back-tracking is not applied. For higher patience levels, emulation results are slightly more accurate when back-tracking is used, while the converse

---

### Table A.2

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>Number of message passing steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K = 1$</td>
<td>$K = 2$</td>
</tr>
<tr>
<td>25th</td>
<td>$4.16 \times 10^{-1}$</td>
</tr>
<tr>
<td>50th</td>
<td>$1.47$</td>
</tr>
<tr>
<td>75th</td>
<td>$2.69$</td>
</tr>
</tbody>
</table>

### Table A.3

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>Number of message passing steps</th>
</tr>
</thead>
<tbody>
<tr>
<td>$K = 1$</td>
<td>$K = 2$</td>
</tr>
<tr>
<td>25th</td>
<td>$4.75 \times 10^{-2}$</td>
</tr>
<tr>
<td>50th</td>
<td>$1.61 \times 10^{-1}$</td>
</tr>
<tr>
<td>75th</td>
<td>$1.79$</td>
</tr>
</tbody>
</table>

### Table A.4

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>Node cardinality input vector to Algorithm 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>25th</td>
<td>$1.23 \times 10^{-2}$</td>
</tr>
<tr>
<td>50th</td>
<td>$2.37 \times 10^{-2}$</td>
</tr>
<tr>
<td>75th</td>
<td>$4.58 \times 10^{-2}$</td>
</tr>
</tbody>
</table>
Table A.5
Emulation error (in mm) statistics for different values of the dimensionality $M$ of the embedding vectors internal to the GNN, with $K = 2$ and $\eta = [24, 6]$.

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>Dimensionality of internal embedding vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$M = 3$</td>
</tr>
<tr>
<td>25th</td>
<td>$5.63 \times 10^{-2}$</td>
</tr>
<tr>
<td>50th</td>
<td>$1.02 \times 10^{-1}$</td>
</tr>
<tr>
<td>75th</td>
<td>$1.96 \times 10^{-1}$</td>
</tr>
</tbody>
</table>

Table A.6
Emulation error (in mm) statistics for different values of message passing steps $K$, when using augmented graph representations when using FEniCS Template as reference graph with $\eta = [24, 6]$ and $M = 40$.

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>Number of message passing steps</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$K = 1$</td>
</tr>
<tr>
<td>25th</td>
<td>$3.90 \times 10^{-2}$</td>
</tr>
<tr>
<td>50th</td>
<td>$1.04 \times 10^{-1}$</td>
</tr>
<tr>
<td>75th</td>
<td>$9.02 \times 10^{-1}$</td>
</tr>
</tbody>
</table>

Table A.7
Emulation error (in mm) statistics for different values for $\eta$, the node cardinality input vector to Algorithm 1 when using FEniCS Template as reference graph with $K = 2$ and $M = 40$.

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>Node cardinality input vector to algorithm 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>25th</td>
<td>$1.49 \times 10^{-2}$</td>
</tr>
<tr>
<td>50th</td>
<td>$2.80 \times 10^{-2}$</td>
</tr>
<tr>
<td>75th</td>
<td>$5.23 \times 10^{-2}$</td>
</tr>
</tbody>
</table>

Table A.8
Emulation error (in mm) statistics for different values for the dimensionality $M$ of the embedding vectors internal to the GNN when using FEniCS Template as reference graph with $\eta = [24, 6]$ and $K = 2$.

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>Dimensionality of internal embedding vectors</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$M = 3$</td>
</tr>
<tr>
<td>25th</td>
<td>$2.94 \times 10^{-2}$</td>
</tr>
<tr>
<td>50th</td>
<td>$5.71 \times 10^{-2}$</td>
</tr>
<tr>
<td>75th</td>
<td>$1.08 \times 10^{-1}$</td>
</tr>
</tbody>
</table>

Table B.9
LV emulation error statistics (mm) for different number of principal components (PCs) retained in $z_{\text{global}}$, with $K = 5$ and $M = 40$.

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>Number of PCs retained</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 PCs</td>
</tr>
<tr>
<td>25th</td>
<td>$2.80 \times 10^{-1}$</td>
</tr>
<tr>
<td>50th</td>
<td>$5.27 \times 10^{-1}$</td>
</tr>
<tr>
<td>75th</td>
<td>$1.05$</td>
</tr>
</tbody>
</table>

is true for lower patience levels. In both cases, if the patience level is less than 100, the early stopping routine halts training before the plateau in validation loss is reached, leading to less accurate results. Note that training using back-tracking with a patience level of 100 leads to the same parameter values being chosen as training for 1000 epochs and selecting the parameters with optimal validation loss.
Table B.10
LV emulation error statistics (mm) for different values of message passing steps $K$ with $M = 40$.

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>$K = 1$</th>
<th>$K = 2$</th>
<th>$K = 3$</th>
<th>$K = 4$</th>
<th>$K = 5$</th>
<th>$K = 6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>25th</td>
<td>$1.72 \times 10^{-1}$</td>
<td>$1.62 \times 10^{-1}$</td>
<td>$1.59 \times 10^{-1}$</td>
<td>$1.60 \times 10^{-1}$</td>
<td>$1.51 \times 10^{-1}$</td>
<td>$1.56 \times 10^{-1}$</td>
</tr>
<tr>
<td>50th</td>
<td>$2.98 \times 10^{-1}$</td>
<td>$2.81 \times 10^{-1}$</td>
<td>$2.78 \times 10^{-1}$</td>
<td>$2.78 \times 10^{-1}$</td>
<td>$2.66 \times 10^{-1}$</td>
<td>$2.67 \times 10^{-1}$</td>
</tr>
<tr>
<td>75th</td>
<td>$5.42 \times 10^{-1}$</td>
<td>$5.20 \times 10^{-1}$</td>
<td>$5.25 \times 10^{-1}$</td>
<td>$5.10 \times 10^{-1}$</td>
<td>$4.99 \times 10^{-1}$</td>
<td>$4.90 \times 10^{-1}$</td>
</tr>
</tbody>
</table>

Table B.11
LVV emulation error statistics (%) for two MLP emulators and two GNN emulators.

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>MLP 1</th>
<th>MLP 2</th>
<th>GNN 1</th>
<th>GNN 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>25th</td>
<td>9.51%</td>
<td>1.95%</td>
<td>0.28%</td>
<td>0.24%</td>
</tr>
<tr>
<td>50th</td>
<td>20.34%</td>
<td>4.35%</td>
<td>0.60%</td>
<td>0.49%</td>
</tr>
<tr>
<td>75th</td>
<td>36.13%</td>
<td>7.78%</td>
<td>1.13%</td>
<td>0.89%</td>
</tr>
</tbody>
</table>

Table B.12
Emulation error statistics for general and transfer-learned emulators. The “Displacement” columns give prediction errors on the nodal displacement vectors (in mm), while the “LVV” columns give percentage errors in LVV predictions.

<table>
<thead>
<tr>
<th>Error percentile</th>
<th>General</th>
<th>Transfer-learned</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Displacement</td>
<td>LVV</td>
</tr>
<tr>
<td>25th</td>
<td>$1.85 \times 10^{-1}$ mm</td>
<td>0.18%</td>
</tr>
<tr>
<td>50th</td>
<td>$2.95 \times 10^{-1}$ mm</td>
<td>0.43%</td>
</tr>
<tr>
<td>75th</td>
<td>$5.13 \times 10^{-1}$ mm</td>
<td>0.76%</td>
</tr>
</tbody>
</table>

Table B.13
MLP LVV emulation error statistics (in %) for different choices of patience level, where back-tracking is applied to find the optimal network weights on the validation data.

<table>
<thead>
<tr>
<th>Error level</th>
<th>10 (93)</th>
<th>25 (142)</th>
<th>50 (319)</th>
<th>100 (620)</th>
<th>None (1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25th</td>
<td>2.85%</td>
<td>2.69%</td>
<td>2.31%</td>
<td>1.95%</td>
<td>1.95%</td>
</tr>
<tr>
<td>50th</td>
<td>6.31%</td>
<td>5.86%</td>
<td>5.05%</td>
<td>4.35%</td>
<td>4.35%</td>
</tr>
<tr>
<td>75th</td>
<td>12.11%</td>
<td>10.73%</td>
<td>9.38%</td>
<td>7.78%</td>
<td>7.78%</td>
</tr>
</tbody>
</table>

Table B.14
MLP LVV emulation error statistics (in %) for different choices of patience level, where back-tracking is not applied to find the optimal network weights on the validation data.

<table>
<thead>
<tr>
<th>Error level</th>
<th>10 (93)</th>
<th>25 (142)</th>
<th>50 (319)</th>
<th>100 (620)</th>
<th>None (1000)</th>
</tr>
</thead>
<tbody>
<tr>
<td>25th</td>
<td>2.90%</td>
<td>2.60%</td>
<td>2.52%</td>
<td>2.52%</td>
<td>1.95%</td>
</tr>
<tr>
<td>50th</td>
<td>6.51%</td>
<td>5.71%</td>
<td>5.55%</td>
<td>5.33%</td>
<td>4.35%</td>
</tr>
<tr>
<td>75th</td>
<td>12.03%</td>
<td>10.73%</td>
<td>10.50%</td>
<td>9.28%</td>
<td>7.78%</td>
</tr>
</tbody>
</table>
Appendix C. Synthetic LV geometry generation

As discussed in Section 4.2, the LV emulation study was performed with synthetic LV geometries that were generated by sampling in a latent space found using Principal Components Analysis (PCA). PCA was performed on a dataset of \( N = 65 \) real LV geometries \( \{G_n\}_{n=1}^{N} \), which were extracted from in-vivo cardiac magnetic resonance (CMR) images of healthy volunteers, obtained at early-diastole. Both long and short axial CMR cine images were used in the extraction procedure. Full details of the study design, imaging protocol and extraction procedure used are given in previous work [57]. Each geometry is represented by 1696 nodes in both the endocardial and epicardial surfaces respectively, or a total of \( N_v = 3392 \) nodes, each of dimension 3. Before applying PCA, each representation \( G_n \) was flattened to a row vector \( g_n \) of width \( D = 3392 \times 3 = 11796 \), where the coordinates of each vector were centred with respect to the mean geometry vector \( \bar{g} \) evaluated over the \( N \) samples. The objective of PCA is to find an orthonormal matrix \( B \in \mathbb{R}^{D \times P} \) from which an \( P \ll D \) dimensional compression of the data is computed as follows:

\[
z_{n}^{\text{PCA}} = B^\top g_n.
\]
**B** is found as the projection for which the reconstructions in the original space,

\[ \hat{g}_n = BB^\top g_n, \]

minimise the following mean squared error loss over the training data:

\[ \mathcal{L}_{\text{PCA}}(B) = \frac{1}{N} \sum_{n=1}^{N} \|g_n - \hat{g}_n\|^2_2 = \frac{1}{N} \sum_{n=1}^{N} \|g_n - BB^\top g_n\|^2_2. \]

Consider the covariance matrix of the data, and its associated eigendecomposition

\[ S = \frac{1}{N-1} \sum_{n=1}^{N} g_n g_n^\top = \sum_{n=1}^{N} \lambda_n b_n b_n^\top, \]

where the eigenvalues \( \lambda_1 > \lambda_2 > \cdots > \lambda_N \) give the variances of the data along their associated eigenvectors \( b_1, b_2, \ldots, b_N \). It can be shown that \( B \) can be selected to minimise Eq. (C.3) by setting its \( p \)th column equal to the eigenvector \( b_p \), for \( p = 1, 2, \ldots, P \). Because these eigenvectors are the components along which the data varies the most, they are referred to as the principal component vectors. Note that in our case, the covariance matrix \( S \) is rank deficient, as we have \( D > N \). In addition, its rank is further reduced by one as we have mean centred the data. Therefore, \( S \) has only \( N - 1 \) non-zero eigenvalues, meaning that up to \( N - 1 \) principal components can be extracted.

Applying PCA simply requires the number of leading PCs retained \( P \) to be selected. This value is typically specified by considering the cumulative proportion of variance explained by the first \( P \) PCs, \( \tau_P^2 \), which is defined as follows

\[ \tau_P^2 = \frac{\sum_{p=1}^{P} \lambda_p}{\sum_{n=1}^{N-1} \lambda_n} \]

for \( P = 1, 2, \ldots, N-1 \). The value of \( P \) can be chosen as the smallest number of components such that \( \tau_P^2 \) exceeds some threshold value that is deemed sufficient to obtain an acceptable representation of the LV geometry. This is the approach taken for example in [51] to select the use of 2 principal components for representation of the LV anatomy. The value of \( \tau_P^2 \) on our dataset is displayed in Fig. C.20(a) for \( P = 1, 2, \ldots, 20 \). As has been observed in previous analyses of LV geometry data [51], \( \tau_P^2 \) quickly approaches 100\% for increasing values of \( P \). The proportion of variance explained measure is widely applied and is useful for the selection of \( P \) for arbitrary datasets. In the case of the specific dataset considered here however, it does not account for the fact that the latent vectors \( z_{ij}^{\text{PCA}} \) represent the geometry of a LV embedded in 3 dimensional space. For this reason, we take a different approach here to select the number of leading PCs \( P \) to retain, by considering the mean node-wise Euclidean distance between the real LV geometries \( G_n \) and the corresponding geometry reconstructions \( \hat{G}_n \) from the \( P \)-dimensional PC space:

\[ \delta_P = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{N_v} \sum_{j=1}^{N_v} \|G_i[j] - \hat{G}_i[j]\|_2 \]

The notation \( G_i[j] \) means we extract the \( j \)th node from the \( i \)th geometry. Using this measure allows the performance of PCA for different values of \( P \) to be evaluated in the same geometry space from which simulations are run. High values of \( \delta_P \) mean that the synthetic LV geometries are failing to capture the anatomical features present in the real data. This in turns means that simulations run using the synthetically generated geometries may exhibit bias with respect to simulations using the true LV geometries, indicating that more principal components should be retained. Fig. C.20(b) plots the value \( \delta_P \) (in mm) on our dataset for the first 40 principal components. In this case, we observe that \( \delta_P \) does not go to zero as rapidly as \( \tau_P^2 \) goes to 1.

Consequently, we retained a large number \( P = 40 \) leading principal components when generating the synthetic LV geometries for this study, where the mean nodal reconstruction error \( \delta_{40} \) was 0.12 mm. By sampling in a latent space of this dimensionality, the synthetic geometries that are generated exhibit greater realism than samples from a lower-dimensional space, in terms of asymmetry, skewness and wall thickness variation. This effect is illustrated in Fig. C.21. Panel C.21(a) displays a randomly selected, real LV geometry against its reconstruction from a 2-dimensional PC space, as used in [51]. Panels (b) and (c) display the analogous plots for reconstructions from a 5-dimensional space (as used in [50]) and a 40 dimensional space respectively. The reconstruction from the 40
dimensional latent space incurs an average nodal reconstruction error of 0.11 mm, significantly lower than that of the 2 dimensional reconstruction (1.60 mm) and the 5 dimensional reconstruction (0.81 mm). Higher values of $P$ could be considered, but run the risk of overfitting to the dataset of only $N = 65$ real anatomies.

Once PCA has been performed, a synthetic LV geometry vector can be generated by first assigning a probability distribution over the $P$ dimensional latent PC space, from which a random sample $z^*$ is drawn. The sample is re-projected back to the full $D$ dimensional space using $B$, before adding the mean geometry vector from the training data $\bar{g}$:

$$g^* = \bar{g} + Bz^*$$

Creating a synthetic LV geometry $g^*$. We used a $P = 40$-dimensional multivariate Gaussian distribution to generate the random samples. The variances of each dimension were found as the variances from the training data, increased by 10%, while the cross-covariance terms and mean values of the distribution were set to zero. The variances were increased by 10% to produce greater variation in the sampled geometries than was observed in the real geometries, while still ensuring that the majority of the sampled geometries were valid. Note that the sampling procedure outlined...
above is not guaranteed to produce a valid LV geometry, as for example the endocardial and epicardial surfaces of the reconstructed geometry may intersect for some values of $z^*$ in Eq. (C.7). We therefore discarded any synthetic geometries for which wall thickness, length and width did not fall within ranges similar to those seen in the real LV data. The two outer surfaces of the synthetic geometries were further processed into a three-layer, hexahedral mesh with 6784 nodes and 4995 finite-elements. Finally, a rule based method (RBM) [43] was adopted to describe the myofibre structure of each geometry, where the fibre angle varied linearly from $\alpha_{\text{endo}} = 60^\circ$ at endocardium to $\alpha_{\text{epi}} = -60^\circ$ at epicardium. 3000 synthetic LV geometries were generated in this manner, a similar order to the number of real LV geometries available at UK Biobank [67].

References


