https://doi.org/10.1016/j.cma.2023.116669

Predicting Dynamic Responses of Continuous Deformable Bodies: A Graph-Based Learning Approach

Qianyi Chen[a], Jiannong Cao[a], Wanyu Lin[a,*], Songye Zhu[b], and Sumei Wang[b]

[a]*Department of Computing, The Hong Kong Polytechnic University, Hung Hom, Hong Kong, China*
[b]*Department of Civil and Environmental Engineering, The Hong Kong Polytechnic University, Hung Hom, Hong Kong, China*

Abstract

Predicting dynamic responses of continuous deformable bodies (CDBs) is essential for various research fields, including civil engineering and computer graphics. Machine learning models, unlike traditional physically-based models, learn from data without predefined physical properties and have significantly advanced the simulation of rigid body systems and fluids. Graph neural network (GNN)-based simulators excel in this area due to their ability to naturally represent physical bodies and interactions using graphs composed of nodes and edges. However, predicting the dynamic responses of CDBs remains challenging because interactions in a CDB are complex, heterogeneous, and dynamic. The complexity of interactions arises from the spatial variability of internal stress, making it challenging to encapsulate such multifaceted interactions of one edge into one edge attribute vector. Additionally, those interactions are heterogeneous and dynamic due to the material and geometric nonlinearity of CDBs. CDBs with elastic, plastic, and elastoplastic materials follow different deformation rules during vibrations. These rules also vary under small and large deformations for a single material type. To address these challenges, we introduce a new GNN-based simulator called **p**hysics-**i**nformed **e**dge **r**ecurrent **s**imulator (*Piers*) for learning CDB dynamics. We first formulate the CDB simulation as a sequence-to-sequence input-output relationship modeling problem and incorporate a recurrent neural network (RNN) to learn edge updates within short timesteps, during which stiffness changes are negligible. To accurately capture complex interactions, we initialize the RNN module's hidden states with prior physical knowledge and equip *Piers* with a physics-informed loss function by assigning the physical properties of interactions as the target edge output. Extensive experimental results demonstrate that *Piers* can simulate the dynamics of elastic, plastic, and elastoplastic CDBs with smaller response prediction errors than alternative baselines. *Piers* reduces prediction errors by 78% to 99% across four typical CDB physical systems.

Keywords: Physical system simulation, AI for science, physics-informed machine learning, graph neural network.

1. Introduction

Continuous deformable bodies (CDBs) are solid structures with their substance filling the occupied space and their shapes are continuously changing under dynamic forces. Their dy-

*Corresponding author

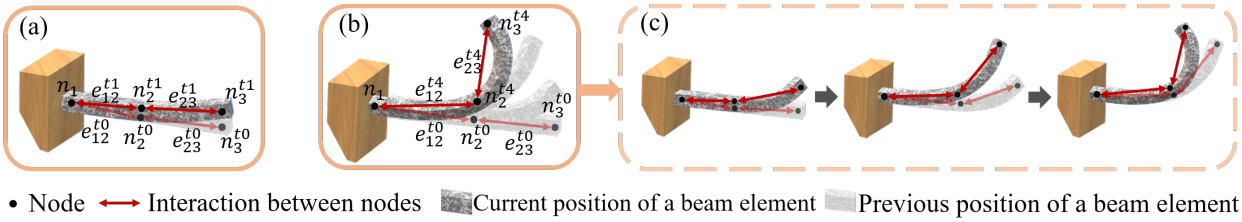


Figure 1: Dynamics of a structure under small and large deformation. (a) A cantilever beam with small deformation. The beam is discretized into 3 nodes and 2 beam elements (interactions). Each node can be represented with a vortex in a graph. Each interaction can be represented with an edge in a graph. (b) A cantilever beam with large deformation. Internal forces in interaction $\mathbf{e}_{23}$ are determined by its relative deformation to the initial position $\mathbf{e}_{23}^{to}$ (rest state). From time t_0 to time t_1, the deformation of interaction $\mathbf{e}^{23}$ is small and its stiffness remains unchanged. However, its stiffness changes dramatically at time t_4 due to the large deformation. (c) The large deformation of CDBs is decomposed into a sequence of small deformations when the time step is short.

namic responses are usually described with node displacement, velocity, and acceleration. Predicting the dynamic responses of CDBs (e.g., elastic beams and plates) is the foundation for various scientific and engineering tasks, such as active control [1], seismic-resistant structure design [2], and computer graphics [3]. The associated objects vary in different research disciplines, ranging from mechanical components [4] to large-scale physical entities (e.g., civil engineering structures) [5]. Many physically based models have been proposed for dynamic responses prediction of CDBs, including finite element method (FEM) [6], discrete element method (DEM) [7], boundary element method (BEM) [8], and many more. However, prior works simulate CDBs using pre-defined materials and geometric properties, which might deviate from reality due to the nonlinearity of real-world structures. In addition, their computation complexity grows dramatically when the structure's degrees of freedom (DOF) increases.

As a typical research area in AI for Science, machine learning (ML) based simulators [9, 10, 11] have attracted tremendous attention in recent years. Different from physically based models, they can learn the dynamics of physical systems from data without manually pre-defined properties. Despite their effectiveness in simulating rigid bodies and fluids, they may fail to simulate physical systems with complex geometries (such as an irregular frame structure), as these geometries can hardly be discretized into uniform grids. Fortunately, systems with complex geometries can be explicitly described with graphs, where vertexes [1] can model the nodes [2] of the systems after discretization, and edges denote interactions (elements) between nodes. Considering the superior capability of GNNs in modeling and learning graphs, simulating complex physical systems with GNNs precisely and efficiently is feasible.

However, CDBs are different from rigid bodies and fluids. Simulating CDBs with GNNs is still challenging because of the complex, dynamic, and heterogeneous interactions in CDBs. *Firstly*, interactions in CDBs are complex because the body is continuous. When forces are applied to a CDB, the continuous entities that are simulated with interactions between nodes will be stretched, bent, and twisted. Accordingly, they will pass internal forces and bending moments in various directions, making CDBs' edge hidden states highly complex and hard

[1] Vertexes of a graph

[2] A node represents a component of a system. As vertex is used to model the node, we will use "node" in the entire paper.

to be simulated. In contrast, physical systems simulated by existing ML-based simulators usually consist of much simpler interactions, such as axial interactions in rigid body systems [12] and springs' elastic forces in cloth simulation [13]. *Secondly*, an interaction modeled as an edge could be highly dynamic as its stiffness changes when the CDB deforms, as known as geometric nonlinearity [14]. For example, the stiffness of $\mathbf{e}_{23}$ under small deformation[3] (see in Fig. 1 (a)) is different from that under large deformation[4] (see in Fig. 1 (b)). *Thirdly*, the material of CDBs could be elastic, plastic, or elastoplastic. The correlations between their internal forces and deformation are different, and the dynamics of CDBs with different materials are distinct. For instance, when a continuous elastic body deforms, the generated internal forces will resist the body's relative deformation to its rest state. Its shape will restore after the applied forces vanish. However, the shapes of plastic and elastoplastic bodies will not or only partially restore to their initial shapes when external forces are removed. Moreover, the stiffness of an elastoplastic body is subjected to its trajectories, while that of a hyperelastic body is free from trajectories. Therefore, simulating the heterogeneous interactions in CDBs using a unified framework is challenging.

Those challenges haven't been settled in existing GNN-based simulators [12, 15, 13]. Existing simulators are not designed for CDBs, and they got unsatisfying performance in predicting their dynamic responses, which has been validated with experiments in this paper. They are mainly proposed for fluids [15], rigid body systems [12], and continuous plastic bodies [13] (e.g., clothes). Although MESHGRAPHNETS[13] has been used to simulate a continuous elastic plate, it is for a quasi-static simulation (the influence of damping, inertia, and frequency are neglected) instead of dynamic responses prediction. Specifically, the external loads are applied to a hyperelasticity plate very slowly. The plate's dynamic responses are not excited as it only deforms slowly rather than vibrates periodically.

In this paper, we propose a physics-informed edge recurrent simulator (*Piers*), a unified GNN-based simulator for learning the dynamics of CDBs. *Piers* can learn the dynamical and heterogeneous interactions in CDBs because we formulate the task as a sequence-to-sequence input-output relationship modeling problem, and we use recurrent neural networks (RNNs) to learn the update of interactions' states. Its advantages are threefold. Firstly, the large deformation of a CDB is decomposed into a sequence of small deformations shown in Fig. 1 (c). Then, the interactions become much more stable as the changes in stiffness become slighter within each small deformation. Secondly, each RNN cell can memorize the state of previous interactions (edges). So, it can infer the updated interactions accurately when internal forces are related to previous trajectories. Then, we can use a unified simulator to simulate CDBs with elastic and plastic materials. Thirdly, the RNN architecture helps reduce the rollout error since it is trained to reduce the loss of multi-step sequence prediction during model training. Nevertheless, most existing GNN-based simulators are trained for one-step prediction, and the accumulation of errors is inevitable during rollouts. It should be noted here that recurrent GNN architecture has been extensively used for various tasks, including traffic prediction [16] and natural language processing [17]. However, to the best of our knowledge, we are the first to adopt this architecture in learning the dynamics of CDBs.

[3]Small deformation describes that changes in structural geometric dimensions are minor and corresponding changes in stiffness are negligible.

[4]Large deformation describes that changes in geometric dimensions are too large to bring negligible changes in stiffness. Small and large deformation are two terminologies used in structural mechanics.

Moreover, *Piers* can model the complex interactions in CDBs precisely by incorporating physics-informed edge hidden states and loss functions. These physical prior knowledge guides *Piers* to generate accurate internal forces while generating accurate node accelerations as node output. Owing to these additional constraints, *Piers* can update its parameters to an optimum solution while learning to simulate more complex interactions. Finally, it makes more physically plausible and accurate predictions. The response prediction accuracy of *Piers* is also validated with extensive experiments in CDBs with different materials. The major contributions of this paper are summarized as follow:

1. We propose *Piers*, the first unified GNN-based simulator dedicated to learning the dynamics of CDBs. It can predict the dynamic responses of elastic, plastic, and elastoplastic CDBs using a unified architecture.
2. We summarize the challenging issues in predicting the dynamic responses of CDBs with ML models. Our proposed *Piers* help solve those challenges by incorporating an optimized problem formulation, RNN as the edge update function, and physical prior knowledge.
3. We validate *Piers'* response prediction accuracy using three self-designed and one public datasets. The simulated CDBs include an elastic beam, an elastoplastic beam, an elastic 3D plate, and a plastic flag waving in the wind. The prediction results show that *Piers* can significantly reduce the response prediction error for the four CDBs by up to 99%, 94%, 99%, and 78% as compared to alternative baselines. We also show that *Piers* can be generalized to predict the dynamic responses of structures under unseen load conditions.

The remaining part of this paper is organized as follows. Section 2 reviews existing ML-based simulators for physical systems and elaborate the differences between them and our proposed *Piers*. Then, our proposed *Piers* are elaborated in Section 3. In Section 4, the effectiveness of *Piers* are validated with four types of CDBs. The prediction error of *Piers* are also compared with several baseline models. We further discussed the prediction results in Section 5 . Finally, the conclusion is drawn in Section 6.

2. Related Work

Existing ML-based simulators are proposed for two categories of physical systems: rigid body systems and deformable bodies.

2.1. ML-based Simulator for Rigid Body Systems

A rigid body system (e.g., n-body systems and articulated rigid bodies) consists of interacting rigid bodies with neglectable intrinsic deformation. Particle-based simulation is suitable for a rigid body system as its dynamics can be naturally described with the movements of particles. The pioneering work of [12] represented rigid bodies and their interactions with nodes and edges in graphs. The proposed GNNs learned gravitational attraction, collisions, and springs from measured data. Also, it can generalize to systems with a different number of objects from training sets. Afterward, a general-purpose learnable physics engine was proposed [18] based on graph networks [19]. Owing to the structural inductive bias implemented by graph networks, the engine can simulate more complex, 3D physical systems and accomplish more tasks like system identification and control. Moreover, GNN is used for the continuous control [20] of rigid body systems in reinforcement learning environments.

2.2. ML-based Simulator for Deformable Bodies

Typical deformable bodies include deformable solids, cloths, fluids, etc. In classical dynamic analysis, the movements of deformable bodies are described with partial differential equations (PDEs) formulated with underlying physical theories. Solutions to those PDEs are obtained after deformable bodies are discretized into particles or meshes, and their deformations are described with a finite number of particles in Lagrangian systems [9] or interpolated with shape functions in Eulerian systems [21]. Inspired by the impressive performance of ML models in approximating solutions to PDEs, researchers have started to use ML models such as multilayer perceptron (MLP) [11], convolutional neural network (CNN) [21, 10], GNN[13], and graph convolution network (GCN) [22] to simulate Eulerian systems discretized with meshes. Those methods have achieved high efficiency in solving both forward prediction and inverse problems for fluids.

Nevertheless, the mesh-based simulation above might fail to simulate deformable bodies with large deformation because meshes will be extremely distorted and skewed [23]. As mesh-free methods, Lagrangian particle-based approaches can overcome this challenge by simulating those systems with movable particles because systems with particle-based representation can be explicitly described with graph-shaped data. GNN-based methods have attracted increasing attention in simulating those systems recently. Sanchez-Gonzalez et al. [24] designed a GNN-based simulator with the message passing framework [25]. The proposed GNN has shown high accuracy and strong generalization ability in the forward prediction of fluids with distinct damping and friction properties. Apart from forward prediction, GNN has also been used for controlling deformable and particle-based objects. [15].

To simulate deformable solids and cloths with large deformation (e.g., self-collision), Pfaff et al. [13] integrates both mesh-space and world-space coordinates when simulating systems with mesh-based representation. Because the change of mesh-space coordinates can accurately capture edges' rest states, this framework achieves promising forward prediction performance in Lagrangian and Eulerian systems. Wu et al. [26] found that while higher mesh resolutions enhance rollout accuracy in ML-based physical system simulations, they also elevate computational demands. To address this trade-off, they pioneered a deep learning-based surrogate model that simultaneously learns the evolution model and optimizes spatial resolutions, striking a balance between accuracy and computational efficiency. However, the proposed simulators have only been tested in the quasi-static simulation (the influence of damping, inertia, and frequency are neglected) of a continuous elastic plate. Furthermore, the dynamics of the plate haven't been simulated. Here we proposed *Piers* to simulate the dynamics of CDBs, which haven't been explored in existing ML-based simulators to the best of our knowledge.

3. Piers: Physics-informed Edge Recurrent Simulator

In this section, we first briefly introduce the traditional time integration method used to solve differential equations in structural dynamics. Our proposed *Piers* follows similar integration procedures but skip the formulation of differential equations for a CDB. Instead, the relationship between external forces and the CDB' responses is learnt by *Piers* purely from data. We use vertexes and edges in a graph to represent nodes and elements in a CDB. Then we formulate the dynamic responses prediction task as a sequence-to-sequence relationship modeling problem and use *Piers* to solve it. The architecture of *Piers*, associated equations, and key properties are elaborated as well.

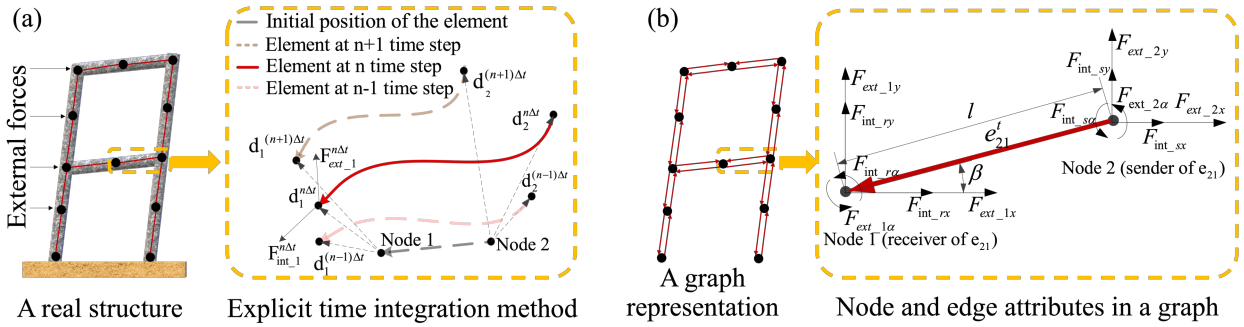


Figure 2: (a) Update of nodes' displacements using explicit time integration method. $\mathbf{d}_1^{n\Delta t}$ and $\mathbf{d}_2^{n\Delta t}$ denote the displacement vectors of node 1 and node 2 at n_{th} time step, respectively. $\mathbf{F_{ext\_1}}$ and $\mathbf{F_{int\_1}}$ denote the external and internal force vectors at node 1 respectively. (b) Graph representation of structures. $F_{\mathbf{ext\_x}}, F_{\mathbf{ext\_y}}, F_{\mathbf{ext\_}\alpha}$ denote the external forces in x axis direction, y-axis direction and external moment respectively. $F_{\mathbf{int\_sx}}, F_{\mathbf{int\_sy}}, F_{\mathbf{int\_s}\alpha}$ denote the internal forces and moments at the sender of an edge respectively while $F_{\mathbf{int\_rx}}, F_{\mathbf{int\_ry}}, F_{\mathbf{int\_r}\alpha}$ denote those at the receiver.

3.1. Computational Procedures of Physically Based Models

FEMs are the mainstream physically based models used to solve structure dynamics. In a traditional FEM, structures are first meshed into large quantities of basic elements whose mass is assigned to the nodes on each element. The movement of the structure are then with the second-order differential equations governed by Newton's second law, where the variable is the vector formed by displacement of nodes in basic elements. The differential equations are shown as

$$\mathbf{M}\ddot{\mathbf{d}} + \mathbf{C}\dot{\mathbf{d}} + \mathbf{K}\mathbf{d} = \mathbf{F_{ext}}, \tag{1}$$

where $\mathbf{M} \in \mathbb{R}^{N\times N}$, $\mathbf{C} \in \mathbb{R}^{N\times N}$, and $\mathbf{K} \in \mathbb{R}^{N\times N}$ denote mass matrix, damping matrix, and stiffness matrix respectively of the discretized structure with N degrees of freedom. $\mathbf{d} \in \mathbb{R}^{N\times 1}$ denotes the displacement vector. $\dot{\mathbf{d}}$ and $\ddot{\mathbf{d}}$ denote the velocity and acceleration respectively. $\mathbf{F_{ext}} \in \mathbb{R}^{N\times 1}$ denotes the external forces vector.

Explicit time integration method is widely used for solving the above differential equations. It can can be expressed as

$$\dot{\mathbf{d}}^{n\Delta t} = \dot{\mathbf{d}}^{(n-1)\Delta t} + \frac{1}{2}\Delta t\ddot{\mathbf{d}}^{(n-1)\Delta t} + \frac{1}{2}\Delta t\ddot{\mathbf{d}}^{n\Delta t}, \tag{2}$$

$$\mathbf{d}^{n\Delta t} = \mathbf{d}^{(n-1)\Delta t} + \Delta t\dot{\mathbf{d}}^{(n-1)\Delta t} + \frac{1}{2}\Delta t^2\ddot{\mathbf{d}}^{(n-1)\Delta t}, \tag{3}$$

where $\mathbf{d^{(n-1)\Delta t}}$, $\dot{\mathbf{d}}^{\mathbf{(n-1)\Delta t}}$, and $\ddot{\mathbf{d}}^{\mathbf{(n-1)\Delta t}}$ denote the displacement, velocity, and acceleration matrix at $(n-1)^{th}$ time step respectively. $\mathbf{d^{n\Delta t}}$, $\dot{\mathbf{d}}^{\mathbf{n\Delta t}}$, and $\ddot{\mathbf{d}}^{\mathbf{n\Delta t}}$ denote the displacement, velocity, and acceleration at n^{th} time step respectively. Δt denotes the length of each time step.

After substituting Eqs. (2) and (3) into Eq. (1), we obtain the acceleration at n^{th} step using its displacement in current time step and state (displacement, velocity and acceleration) at $(n-1)^{th}$ step. The obtained acceleration $\ddot{\mathbf{d}}^{n\Delta t}$ is then used to update nodes' displacements

using Eq. (3). After solving above equations iteratively in consecutive time steps, we finally solve the dynamics of displacements using FEM.

In this paper, we also obtain the acceleration for n^{th} step at first and then update the displacement use the acceleration. Different from FEMs that solve equations with predefined stiffness matrix, mass matrix, and damping matrix, our method predict nodes' acceleration in a data-driven way. As Fig. 2 (a) shows, the movement of each node is triggered by the external forces and internal forces applied to it. Specifically, the acceleration of node 1 at time t is determined by its mass, internal forces F_{int} generated by elements' relative deformation, and external forces F_{ext} applied to the node directly. Then, the updating of a CDB's states can be interpreted as following three steps:

Step 1: Update the internal forces in *element*_21 (node 2 as the sender and node 1 as the receiver) at n^{th} time step using its internal forces at $(n-1)^{th}$ time step and corresponding deformation happened in between the two time steps.

Step 2: Compute the sum of internal forces and external forces applied to node 1. Then use the resultant forces to calculate node 1's acceleration.

Step 3: Update node 1's displacement to $(n-1)^{th}$ time step. The new displacements are fed back to step 1 for the next loop.

The above three steps can be exactly mapped to message computing, message passing, and node updating in a typical message passing neural network. Inspired by such mapping, we propose *Piers* to learn the dynamics of a CDB purelly from data.

3.2. Piers

3.2.1. Graph Representations for CDBs

Before using *Piers* to learn the dynamics of a CDB, we have to discretize the CDB into a finite number of basic elements (e.g., beam elements and triangular elements) and represent the discretized CDB with a graph. Specifically, each basic element has multiple vertexes, which are directly used as nodes in its graph representation. The states of each vertex are encapsulated in the node attributes within the graph. To model the interactions between nodes belonging to the same element, we introduce a pair of edges with opposite orientations to link every two connected nodes. In our study, we adopt a bidirectional graph representation, instead of an undirected one, based on the mandatory edge directionality within a message passing neural network (MPNN) [25]. MPNN is an extensively used GNN framework, and our proposed *Piers* is designed based on it. For alignment with the MPNN's operational framework, undirected graphs are usually consistently described as bidirectional. This approach is consistent with Newton's Third Law, which posits that every action has an equal and opposite reaction, thereby capturing the mutual forces between the connected nodes. The states of elements are carried by edge attributes in the graph. Meanwhile, an adjacency matrix is formulated for the graph, describing the existence of edges between any two nodes. Finally, the node attributes, edge attributes, and adjacency matrix form the graph representation, carrying the states of the interior and vertex of all elements in the CDB.

We take the frame structure in Fig. 2 (a) as an example. The frame structure is discretized into 12 beam elements. Fig. 2 (b) shows its graph representation, consisting of 12 nodes and 24 edges. The nodes are exactly the vertexes of all beam elements. Two edges with opposite directions are added to connect every two vertexes from the same beam element. Each pair of edges represents the mutual interactions between two nodes. Node attributes include node displacements (d_x, d_y, d_α), node boundary conditions (b_x, b_y, b_α) and node external forces$(F_{\mathbf{ext_x}}, F_{\mathbf{ext_y}}, F_{\mathbf{ext}_\alpha})$. Edge attributes include $(\mathbf{cos}\beta, \mathbf{sin}\beta, \Delta l)$, and internal forces

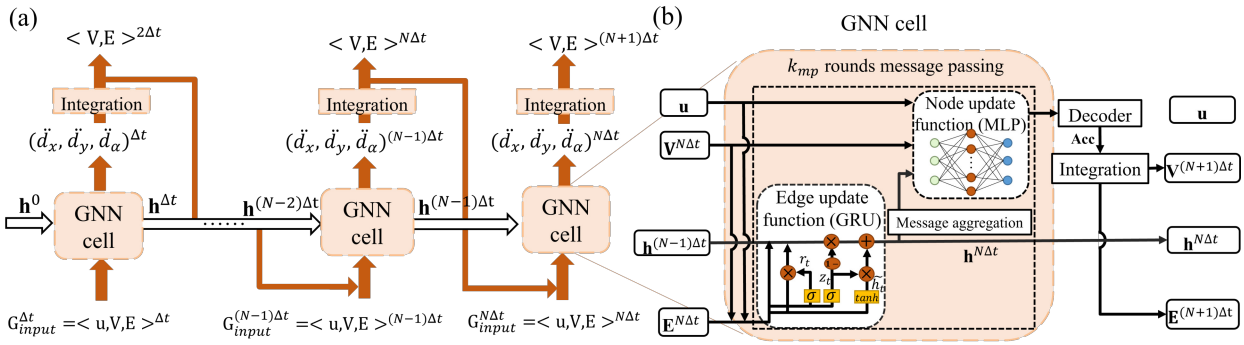


Figure 3: (a) Architecture of *Piers*. It uses a RNN skeleton consisting of recurrent GNN cell. (b) The architecture of one GNN cell. The GNN cell follows the message passing framework. A GRU is used as the edge update function, and MLP is used as the node update function. The output of the node update function in each GNN cell is decoded into the predicted acceleration in each time step.

$(F_{\mathbf{int\_sx}}, F_{\mathbf{int\_sy}}, F_{\mathbf{int\_s}\alpha}, F_{\mathbf{int\_rx}}, F_{\mathbf{int\_ry}}, F_{\mathbf{int\_r}\alpha})$. Δl denotes the change of element length to the initial length, and β denotes the element's angle down from the positive x axis. Global attributes include gravitational acceleration g and time step Δt. It should be noted here that, node displacements in node attributes are not node position measured under the global coordinate system. Instead, they are nodes' relative displacements to their initial position respectively. We use relative displacements here to train a coordinate-free GNN-based simulator. Node position will change under different global coordinate systems. Then, the GNN model trained with data from one global coordinate system might not be applied to other coordinate systems. Consequently, we use nodes' relative displacement in node attributes.

3.2.2. Overall Architecture of Piers

As Fig. 3 shows, *Piers* adopts a recurrent neural network skeleton. We take beam elements shown in Fig. 2 as an example. The overall operation of *Piers* is defined as:

$$\{(\ddot{d}_x, \ddot{d}_y, \ddot{d}_\alpha)_k^{\Delta t}, ..., (\ddot{d}_x, \ddot{d}_y, \ddot{d}_\alpha)_k^{N\Delta t}\}_{k=1,...,n_n} = \mathbf{Piers}(\mathbf{G}_{\mathbf{input}}^{\Delta t}, ..., \mathbf{G}_{\mathbf{input}}^{N\Delta t}, \mathbf{h}^0), \tag{4}$$

where $\ddot{d}_x$ and $\ddot{d}_y$ denote node acceleration along x-axis and y-axis direction respectively. $\ddot{d}_\alpha$ denotes the angular acceleration. n_n denotes the number of nodes in the graph. $\mathbf{G}_{\mathbf{input}}^{n\Delta t} =< \mathbf{u}, \mathbf{V}, \mathbf{E} >^{n\Delta t}$ $(n = 1, ..., N)$ denotes the graph representation for the state of a CDB at n^{th} time step. $\mathbf{h}^0$ denotes the initial hidden states for all edges. $\mathbf{u} = \{g, \Delta t\}$ denotes the global attributes. $\mathbf{V}^{n\Delta t}$ denotes node input at n^{th} step. It can be written as

$$\mathbf{V}^{n\Delta t} = \{(d_x, d_y, d_\alpha)_k^{(n-1)\Delta t}, \mathbf{Vd}_k^{n\Delta t}, \mathbf{Ve}_k^{n\Delta t}\}_{k=1,...,n_n}, \tag{5}$$

where d_x and d_y denote node displacements along x-axis and y-axis direction respectively. d_α denotes the angle of rotation.$\mathbf{Vd}_k^{n\Delta t} = (d_x, d_y, d_\alpha, b_x, b_y, b_\alpha)_k^{n\Delta t}$, where $b_{x/y/\alpha} = 0$ if unfixed or $b_{x/y/\alpha} = 1$ if fixed denotes the node boundary condition in x-axis, y-axis direction, and rotation respectively. $\mathbf{Ve}_k^{n\Delta t} = (F_{\mathbf{ext\_x}}, F_{\mathbf{ext\_y}}, F_{\mathbf{ext}\_\alpha})_k^{n\Delta t}$ denote the external forces along the positive x-axis, positive y-axis, and external moment respectively.

$\mathbf{E}^{n\Delta t}$ denotes edge input and it includes edge attributes at both $(n-1)^{th}$ and n^{th} steps.

$$\mathbf{E}^{n\Delta t} = \{(\mathbf{cos}\beta, \mathbf{sin}\beta, \Delta l)_k^{(n-1)\Delta t}, (\mathbf{cos}\beta, \mathbf{sin}\beta, \Delta l)_k^{n\Delta t}\}_{k=1,...,n_e}, \tag{6}$$

where n_e denotes the number of edges in the graph.

$\mathbf{h}^{n\Delta t}$ in Fig. 3 denotes edge hidden state at n^{th} step. It can be expressed as

$$\mathbf{h}^{n\Delta t} = \{(F_{\mathbf{int\_sx}}, F_{\mathbf{int\_sy}}, F_{\mathbf{int\_s}\alpha}, F_{\mathbf{int\_rx}}, F_{\mathbf{int\_ry}}, F_{\mathbf{int\_r}\alpha})_k^{n\Delta t}\}_{k=1,\ldots,n_e}, \tag{7}$$

where $F_{\mathbf{int\_sx}}$ and $F_{\mathbf{int\_sy}}$ denote the element's internal forces at one edge's sender node along the positive x-axis and y-axis, respectively. $F_{\mathbf{int\_s}\alpha}$ denotes the element's internal moment at the sender node. $F_{\mathbf{int\_rx}}$, $F_{\mathbf{int\_ry}}$, and $F_{\mathbf{int\_r}\alpha}$ denote the element's internal forces and moment at the receiver node of the edge.

In *Piers*, one GNN cell is used to update the state of a CDB using its state in the current time step and its edge hidden state in the previous time step. The same GNN cell processes a sequence of graph-shaped data $\mathbf{G}_{\mathbf{input}}^{n\Delta t}$ recurrently for N times, generating the predicted acceleration for each time step. The n^{th} GNN cell are described as

$$\{(\ddot{d}_x, \ddot{d}_y, \ddot{d}_\alpha)_k^{n\Delta t}\}_{k=1,\ldots,n_n}, \mathbf{h}^{n\Delta t} = \mathbf{GNNcell}(\mathbf{G}_{\mathbf{input}}^{n\Delta t}, \mathbf{h}^{(n-1)\Delta t}). \tag{8}$$

3.2.3. Edge Update and Node Update Functions

As Fig. 3 (b) shows, the GNN cell in *Piers* is designed based on MPNN [25] framework. In each message passing step, the edge attributes are first updated as

$$\mathbf{E}'^{n\Delta t}, \mathbf{h}'^{(n-1)\Delta t} = \mathbf{GRU}(\mathbf{E}^{n\Delta t}, \mathbf{V}_s^{n\Delta t}, \mathbf{V}_r^{n\Delta t}, \mathbf{h}^{(n-1)\Delta t}), \tag{9}$$

where GRU denotes the gated recurrent unit [27] that is used as the edge update function. $\mathbf{E}'^{n\Delta t}$ and $\mathbf{h}'^{(n-1)\Delta t}$ denote the updated edge attributes and edge hidden states after one message passing step. $\mathbf{V}_s^{n\Delta t}$ and $\mathbf{V}_r^{n\Delta t}$ denote the node attributes of each edge's sender node and receiver node. $\mathbf{h}^{(n-1)\Delta t}$ denotes the edge hidden states in the $(n-1)$ time step. It should be noted that the external forces $\mathbf{Ve}$ are not used in Eq. (9) but will be used in the following node updating. The updated edge attributes are used as messages and then aggregated in the receiver of each edge. They are concatenated with the receiver's node input $\mathbf{V}^{n\Delta t}$ and global attributes $\mathbf{u}$. The concatenated feature is fed into a node update function which is shown as

$$\mathbf{V}'^{n\Delta t} = \mathbf{MLP}(\mathbf{V}^{n\Delta t}, \sum \mathbf{E}'^{n\Delta t}, \mathbf{u}), \tag{10}$$

where MLP denotes the node update function, which is a multilayer perceptron (MLP) in this paper. $\sum \mathbf{E}'^{n\Delta t}$ denotes the aggregation [19] of updated edge attributes in each edge's receiver node. $\mathbf{V}'^{n\Delta t}$ denotes the updated node attribute after one message passing step. After k_{mp} steps message passing, the final $\mathbf{V}'^{n\Delta t}$ are fed into a decoder, and the target node output is computed as

$$\{(\ddot{d}_x, \ddot{d}_y, \ddot{d}_\alpha)_k^{n\Delta t}\}_{k=1,\ldots,n_n} = \mathbf{Decoder}(\mathbf{V}'^{n\Delta t}), \tag{11}$$

where Decoder is also a MLP in this GNN cell. Despite their naming as 'update functions,' it's important to clarify that the outputs of node/edge update function are not node/edge attributes for the next time step. Rather, the target output of the node update function includes nodes' acceleration along the positive x-axis, y-axis, and rotational acceleration. The edge update function aims to output the internal forces at the same time step. Then, new node displacements and edge positions in the next time step are computed using forward-Euler integration in Eq. (2) and Eq. (3). Finally, new attributes for both nodes and edges are calculated based on new node displacements and edge positions, which then serve as the input for the next time step.

3.2.4. Physics-informed Hidden States and Loss Function

In *Piers*, we incorporate the rest states of elements in a CDB as the initial hidden states of the corresponding edges. We refer to these as 'physics-informed hidden states' because they are derived from the physical properties of the elements within the CDBs. According to classical principles of structural dynamics, varying rest states lead to different internal forces, and consequently, unique dynamic behaviors. Recent studies on GNN-based simulators, such as the work by [13], also validate the importance of rest states in capturing the dynamics of physical systems. Therefore, we include these rest states in the training phase of our GNN-based simulator. Specifically, when simulating elastic or elastoplastic beam structures, we utilize the internal forces at time step zero as the elements' rest states. For the simulations involving plate and cloth structures, we employ the edges' mesh-space positions as their respective rest states.

As the term suggests, our 'physics-informed loss function' is designed to embed the inherent physical properties of the CDBs we are simulating as a form of prior knowledge. To be specific, we utilize the ground-truth final states of elements as the labels for edge attributes. The discrepancies between the model's edge output and these ground-truth labels are subsequently integrated into the loss function. Eq. (12) shows the physics-informed loss function. $\mathcal{L}$ consists of two parts: $\mathcal{L}_{\mathbf{acc}}$ and $\mathcal{L}_{\mathbf{force}}$. $\mathcal{L}_{\mathbf{acc}}$ denotes the mean square error (MSE) between the predicted and target node accelerations in all time steps, which are used in most of existing GNN based simulator [12, 18, 24, 13]. $\mathcal{L}_{\mathbf{force}}$ is an additional physics-informed constraint. It is the MSE between predicted and target element internal forces in the final time step. c_1 and c_2 are two hyper-parameters which are chosen as 1 and 0.01 after experiments.

$$
\begin{aligned}
\mathcal{L} &= c_1\mathcal{L}_{\mathbf{acc}} + c_2\mathcal{L}_{\mathbf{force}}, \\
\mathcal{L}_{\mathbf{acc}} &= \frac{1}{3n_nN}\sum_{i=\Delta t}^{N\Delta t}\sum_{j=1}^{n_n}\sum_{k=x,y,\alpha}(\ddot{d}^{i}_{jk\_p} - \ddot{d}^{i}_{jk\_t})^2, \\
\mathcal{L}_{\mathbf{force}} &= \frac{1}{6n_e}\sum_{i=1}^{n_e}\sum_{j=1}^{2}\sum_{k=x,y,\alpha}(F_{ijk\_p} - F_{ijk\_t})^2,
\end{aligned}
\tag{12}
$$

where $\ddot{d}^{i}_{jk\_p}$ denotes the standardized predicted acceleration of node j in direction k at time i, $\ddot{d}^{i}_{jk\_t}$ denotes the standardized target acceleration at time i, F_{ijk_p} denotes predicted standerlized internal forces of edge i in direction k to its receiver node (j=1) or sender node (j=2), F_{ijk_t} denotes the target standerlized internal forces.

3.2.5. Training and Inference of Piers

We build our *Piers* using Tensorflow 1.15, Sonnet 1.36, and Graph Nets library. All input data and target outputs are standardized with their average values and standard deviations. As Fig. 3 (a) shows, a sequence-to-sequence architecture is used in *Piers* to predict time-history responses. To accelerate the training process of *Piers* and improve its convergence, we adopt a teacher-forcing strategy during model training. Specifically, instead of using the outputs of previous GNN cells, we use the ground truth node input and edge input for each GNN cell. Once trained, *Piers* can be used for one-step prediction and multi-step prediction. The multi-step prediction is generated by the rollouts of *Piers*. Future node displacements P_x, P_y, P_α are not given but need to be predicted iteratively as Fig. 3 shows. Node boundary

conditions (b_x, b_y, b_α) and node external forces ($F_{ext\_x}, F_{ext\_y}, F_{ext\_\alpha}$) at each time step are known and used as the input for each GNN cell. Moreover, we introduce noise to both node and edge attributes consciously when training *Piers* to simulate more complex structures, such as plate and cloth in the following section. As has been validated in [24], the noises can force *Piers* to make robust and accurate predictions in the presence of variability and uncertainty. Models trained under such conditions have demonstrated increased robustness and accuracy in their predictions. The model training and inference processes are summarized in Algorithm 1.

Algorithm 1 Piers

Input: $\mathbf{G}_{\mathbf{input}}^{\Delta t}, ..., \mathbf{G}_{\mathbf{input}}^{N\Delta t}, \mathbf{h}^0$, Mode = Training or Inference
for $n = 1 \rightarrow N$ **do**
  for $j = 1 \rightarrow k_{mp}$ **do**
    $\mathbf{E}^{n\Delta t}, \mathbf{h}^{(n-1)\Delta t} = \mathbf{GRU}(\mathbf{E}^{n\Delta t}, \mathbf{V}_s^{n\Delta t}, \mathbf{V}_r^{n\Delta t}, \mathbf{h}^{(n-1)\Delta t})$
    $\mathbf{V}^{n\Delta t} = \mathbf{MLP}(\mathbf{V}^{n\Delta t}, \sum \mathbf{E}^{n\Delta t}, \mathbf{u})$
  end
  $\{(\ddot{d}_x, \ddot{d}_y, \ddot{d}_\alpha)_k^{n\Delta t}\}_{k=1,...,n_n} = \mathbf{Decoder}(\mathbf{V}^{n\Delta t})$
  if Mode = Inference
    Replace node displacements in $\mathbf{G}_{\mathbf{input}}^{(n+1)\Delta t}$ with
    $\{\mathbf{d}^{(n+1)\Delta t}\}_{k=1,...,n_n} = \Delta t^2 \ddot{\mathbf{d}}^{n\Delta t} + 2\mathbf{d}^{n\Delta t} - \mathbf{d}^{(n-1)\Delta t}$
  end
end
Output: $\{(\ddot{d}_x, \ddot{d}_y, \ddot{d}_\alpha)_k^{\Delta t}, ..., (\ddot{d}_x, \ddot{d}_y, \ddot{d}_\alpha)_k^{N\Delta t}\}_{k=1,...,n_n}$

4. Experimental Evaluation

First, we used *Piers* to learn the dynamics of a classic CDB: an 2D simply supported elastic beam. Then we use *Piers* to simulate the dynamic responses of an elastoplastic beam, validating its generability to structures with material nonlinearity. We also tested more complex structures. A 3D elastic plate with large deformation is simulated to validate *Piers*'s generability to geometry nonlinearity. Moreover, a plastic cloth (a waving flag) are simulated as well, validating *Piers*'s outstanding simulation capacity. Specifically, datasets for the elastic beam, elastoplastic beam, and elastic plate are generated by ourselves using several traditional FEM softwares. The dataset for cloth simulation is a public dataset proposed in [13].

We prioritized simulating beam structures over rod structures in this study due to the more simplistic representation of rod structures. Rod structures are often better described as rigid body systems due to their simpler internal forces. Specifically, rod elements primarily transmit axial forces, and exhibit a uniform internal stress. This means that the entire internal force distribution of a rod structure can be succinctly captured with a set of discrete values. Upon discretization, such a structure can be interpreted as an ensemble of mass points governed by axial interactions, closely mirroring a rigid body system's behavior. Conversely, the internal stress within a beam presents as a continuously varying field, dictated by its position. This makes beams fundamentally more intricate, as they don't just bear axial forces but also shear forces and moments. To put it into perspective, rod elements can be viewed as a subset of beam elements. Given that *Piers* has demonstrated its efficacy in simulating the nuanced dynamics of beam structures, it implicitly suggests its capability to simulate the simpler rod structures. Hence, our decision was to predominantly focus on beam structures in this paper.

We compare *Piers* with three baseline models in this paper. Baseline I is a GNN-based framework proposed both in GNS [24] and MESHGRAPHNETS [13]. This model adopts an encoder-decoder architecture with a MPNN as the processor. MLPs are chosen as node and edge update functions in the MPNN. Moreover, we designed two ablation experiments to validate the contribution of the physics-informed loss function and internal forces as hidden states in *Piers*. *Piers* with zero hidden states is used as baseline II. *Piers* without the physics-informed term in loss function is used as baseline III. It should be noted that we don't have Baseline III when using *Piers* to simulate the elastic plate and cloth. Because physics-informed loss functions are not used while simulating the two CDBs, which will be illustrated in the two subsections for plate and cloth simulation.

For each CDB, we use the same dataset to train *Piers* and baseline models. Determining the optimal node density for each CDB involves balancing the competing demands of simulation accuracy and computational efficiency. Employing a denser mesh in the discretization of a CDB offers two key advantages for its simulation in *Piers*. Firstly, higher node density minimizes the simulation error incurred when using a finite set of nodes to represent the continuous behaviors of a CDB. Secondly, a denser mesh provides a richer data set for model training, further enhancing the model's predictive power. However, these benefits come at a cost: a denser mesh significantly increases the computational workload during model training due to the augmented number of nodes and edges. Thus, for the simulation of the four distinct CDBs, moderate node densities are chosen to achieve the desired level of accuracy with acceptable training time. The node input of Baseline I is chosen according to the node input of GNS [24]. Except for the node attributes used in the other three models, it also includes node velocities and accelerations in the previous time step. The input of those models and their hyperparameters are introduced in the Appendix. The learning rate for all models decays exponentially from 1×10^{-3} to 1×10^{-5}. After model training, the well-trained *Piers* and baseline models are used to predict the dynamic responses of four CDBs under external forces unseen in training sets. We use root mean square error (RMSE) as the evaluation metric for response prediction. The RMSE between ground truth and predicted displacements is computed using:

$$\mathbf{RMSE} = \sqrt{\frac{1}{3n_n N}\sum_{i=\Delta t}^{N\Delta t}\sum_{j=1}^{n_n}\sum_{k=x,y,\alpha}(d^i_{jk\_p} - d^i_{jk\_t})^2}, \tag{13}$$

where $d^i_{jk\_p}$ and $d^i_{jk\_t}$ denote the predicted and target displacement of node j in direction k at time i.

4.1. Elastic Beam

A simply supported elastic beam is a fundamental structure in structural dynamics. Fig. 4 shows the vibration of the beam under external forces. The external forces include horizontal external node forces F_{ext_x}, vertical external node forces F_{ext_y}, and external bending moment M_{ext}. The boundary conditions of the beam are $b_{x1} = b_{y1} = b_{y2} = 1$, which denotes the displacements of node 1 along the x-axis, the displacement of node 1 along the y-axis, and the displacement of node 11 along the y-axis are fixed. Please refer to Table 1 for the material and geometric properties of the beam.

After discretization, the beam is represented with $n_n = 11$ nodes and $n_e = 20$ unidirectional edges. We recognize that the graph utilized to depict the beam is relatively simplistic,

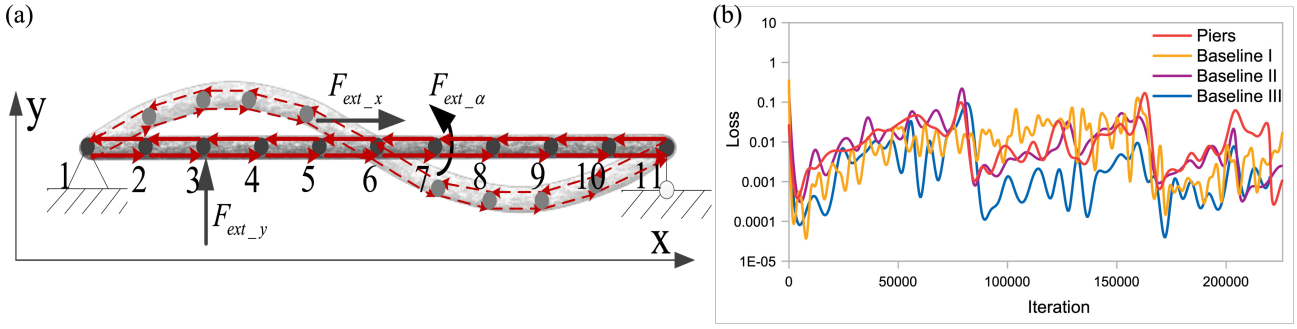


Figure 4: (a)Vibration of a simply supported beam under external forces. The beam is described with 11 nodes and 20 unidirectional edges. (b) Learning curves of *Piers* and Baseline models for simulating the elastic beam.

Table 1: Material and geometric properties of the simulated beam.

Properties	Value
Elastic modulus (Pa)	2.6×10^9
Hardening parameter (Pa)	1×10^9
Initial yield stress (Pa)	2×10^5
Damping ratio	0.05
Mass density (kg/m^3)	5000
Sectional area (m^2)	6×10^{-2}
Section moment of inertia (m^4)	0.625
Length (m)	70
Acceleration of gravity (m/s^2)	9.806

Note: Elastic modulus is the property of the elastic beam; Hardening parameter and initial yield stress are properties of the elastoplastic beam; Other properties are shared by the two beams.

consisting of linearly arranged nodes at uniform intervals. Nonetheless, the utility of using a graph extends beyond its ability to capture complex or irregular forms; it also provides a versatile medium for representing high-order edge attributes. While a high-dimensional matrix could theoretically be used to encapsulate the node attributes in such a straightforward arrangement. Then, a convolutional neural network can be used to process the matrix. However, the challenge lies in concurrently embedding edge attributes—especially those represented as vectors—into the same matrix. For this reason, we maintain that a graph representation is optimal, even for this seemingly simple beam structure.

The dynamic responses of the beam under external node forces are simulated with vector form intrinsic finite element (VFIFE) [28, 29, 30], a type of FEM that proficient in solving geometric nonlinearity and material nonlinearity issues in structural dynamics. The time step Δt is 0.001s in the simulation. To simulate the beam's dynamic properties comprehensively, we use Gaussian white noises as external forces. Because such forces have flat-shape spectra in the frequency domain. Every natural frequency of the beam can be excited, and the vibration responses can describe the beams' dynamic properties comprehensively. To simplify the dynamic responses, we apply external force on one node in one direction when generating one trajectory. The standard deviation of external forces varies in different trajectories to generate responses with diverse amplitudes.

Table 2: One-step and multi-step displacement prediction error for the elastic beam.

Prediction Mode	1-step ($\times 10^{-4}$)	Multi-step rollouts ($\times 10^{-4}$)						
		100-step	200-step	300-step	400-step	500-step	600-step	700-step
Piers	0.36	12.59	24.60	45.31	68.30	123.31	237.19	693.62
Baseline I	1.32	144.60	1166.86	5862.48	15561.67	28560.99	43100.23	58612.25
	(72.46%)	(91.29%)	(97.89%)	(99.23%)	(99.56%)	(99.57%)	(99.45%)	(98.82%)
Baseline II	0.26	32.93	115.22	625.47	3652.60	12691.45	31302.65	62381.63
	(-37.81%)	(61.76%)	(78.65%)	(92.76%)	(98.13%)	(99.03%)	(99.24%)	(98.89%)
Baseline III	0.47	15.58	48.88	100.90	168.82	244.85	444.78	1421.58
	(22.01%)	(19.17%)	(49.67%)	(55.09%)	(59.54%)	(49.64%)	(46.67%)	(51.21%)

Note: Numbers in brackets denote the relative decrease of RMSE from *Piers* to baselines.

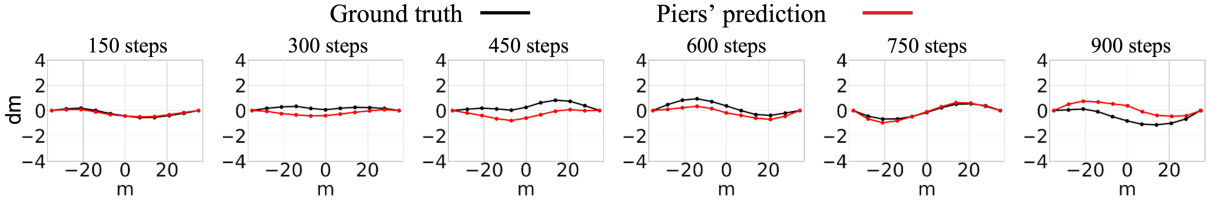


Figure 5: Ground truth and *Piers* predicted deformation of the elastic beam at different time steps. This figure shows the predicted deformations are visually close to the ground truth within 750 steps.

We generate 660, 33, and 33 trajectories for training, validation, and testing, respectively. Moreover, different time-history external forces are used in different datasets. Each trajectory is a sequence of graph-shaped data with 2002 time steps. A moving window (1000 steps in length) is applied to each trajectory to generate multiple samples. Therefore, each sample contains 1000 graph-shaped data $\mathbf{G_{input}} =< \mathbf{u}, \mathbf{V}, \mathbf{E} >^{n\Delta t}$ $(n = 1, ..., 1000)$ and one set of initial hidden states $\mathbf{h}^0$ for all edges. The labels of each sample are node accelerations at each time step and elements' internal forces in the sample's final time step. We train *Piers* and 3 baseline models for 230000 iterations. The learning rate is exponentially decayed from 1×10^{-3} to 1×10^{-5} and optimized with Adam optimizer. Fig. 4(b) shows the learning curves of the four models. The training loss of all models decrease with the increase of iterations. The apparent superiority of Baseline III over *Piers* in terms of loss after the same training iterations can be attributed to the loss function in Baseline III. Specifically, its loss function omits a term that assesses the prediction error concerning internal forces. Consequently, while it might initially seem as though Baseline III performs better, this is somewhat deceptive. Its lower loss doesn't necessarily translate to a more accurate or comprehensive model. The following experimental results in testing sets also validate that Baseline III makes larger displacements prediction error than *Piers*.

We summarize *Piers'* and baseline models' prediction errors for the elastic beam in Table 2. We found *Piers* achieves the least prediction error for most cases. The large prediction error of Baseline I shows that it fails to learn the dynamics of CDBs. The other three models all achieve less multi-step prediction error than baseline I, because they incorporate RNN as the edge update function. An interesting finding is *Piers* doesn't got the least error in 1-step prediction. The reason should be that *Piers* is designed for multi-step prediction instead of 1-step prediction. Baseline II got the smallest error in 1-step prediction probably because of over-fitting. The worse performance of Baseline II over *Piers* in multi-step prediction also

supports our point our points of view. The prediction errors of Baseline II and III are more vulnerable to the increase of time steps than *Piers*, which validates the functionalities of using initial internal forces as hidden states and the physics-informed loss function. We also draw the ground truth and *Piers'* predicted deformation of the elastic beam in Fig. 5. The predicted vibration are visually similar to the ground truth vibration within 750 steps. While noticeable discrepancies between the predicted and ground truth values can be observed at 300, 450, and 600 steps in Fig. 5, the overall contours of the predicted beam deformation closely resemble the ground truth at each step. This confirms that *Piers* has successfully grasped the dynamics of elastic beams. However, there remains potential to enhance *Piers'* predictive accuracy, especially across extended rollout steps. This will be further elaborated in Section 5.3.

4.2. Elastoplastic Beam

Different from elastic structures, the dynamics of elastoplastic structures are subject to their previous movement trajectories. Because their deformation becomes plastic once their stress exceeds the yield stress. In this paper, we use *Piers* to simulate an elastoplastic beam, whose mechanical and geometrical properties are summarized in Table 1. All of those properties, except the elastic modulus, are the same as the previously simulated elastic beam. The dataset generation process and the size of the training samples are consistent with those used for the elastic beam in Section 4.1. Each training sample spans a length of 1000 steps along the time axis. This elastoplastic beam is made up of bilinear elastoplastic material with a hysteresis curve shown in Fig. 6. As the load path in Fig. 6 shows, the stress increases linearly with the increase of strain within the elastic period (line segments 1). When stress exceeds the initial yield stress σ_Y^0, the material will enter the hardening period (line segments 2), and the yield stress will increase. When the strain decrease, the material enters the unloading period, and the stress decreases linearly (line segments 3). However, there will be a plastic deformation (strain) even after stress decrease to zero because of the accumulation of previous plastic deformation. E_e and E_p denote the elasticity modulus in the elastic stage and plastic stage, respectively. E_p can be computed using $E_p = \frac{E_e H}{E_e + H}$, where H denotes the hardening parameter. The σ_Y^0, E_e, and H of the simulated elastoplastic beam are $2.0 \times 10^5 (Pa)$, $2.6 \times 10^9 (Pa)$, and $1.0 \times 10^9 (Pa)$ respectively. The update of yield stress in a plastic stage is shown as

$$\begin{aligned} \Delta\epsilon_e &= \frac{\Delta\epsilon}{1 + E_e/H} \\ \sigma_Y^n &= \sigma_Y^{n-1} + H\Delta\epsilon_e, \end{aligned} \tag{14}$$

where σ_Y^n and σ_Y^{n-1} denote the yield stress at step n and $n-1$. Apparently, the stiffness of a material is dependent on its load history. And our proposed *Piers* is supposed to capture interactions as it incorporates a GRU as the edge update function.

We train *Piers* and 3 baseline models for 350000 iterations. The learning rate and optimizer in this experiment are the same with the previous experiment. Fig. 6(b) shows the learning curves of the four models. Although the overall training loss shows a downtrend, training loss in this experiment go through much more violent fluctuation with the increase of iterations when compared with learning curves in Fig. 4(b). It shows the difficulty of learning the dynamics of an elastoplastic structure. As Table 3 shows, *Piers* makes the least error in 1-step and multi-step rollout trajectory prediction. The displacement prediction errors are slightly different from those in Table 2, where *Piers* performs worse than Baseline II model in 1-step

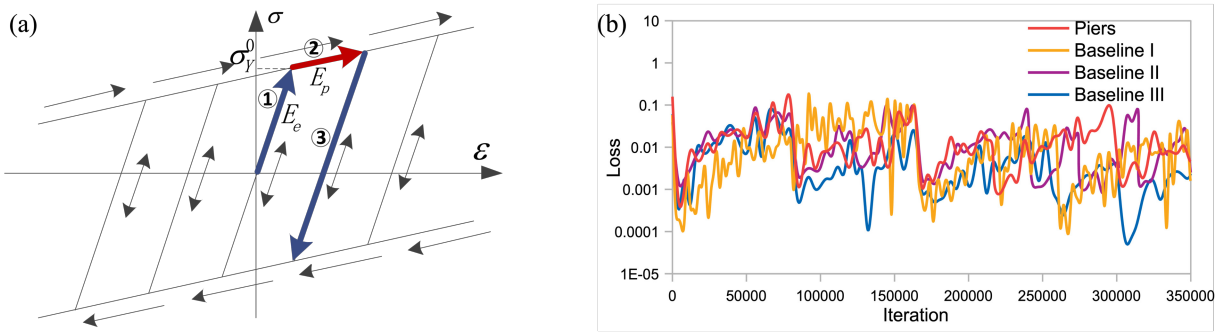


Figure 6: (a)Bilinear hysteresis of the elastoplastic beam. The two-way line segments denote elastic stages, and the one-way line segments denote plastic stages. (b) Learning curves of *Piers* and Baseline models for simulating the elastoplastic beam.

Table 3: One-step and multi-step displacement prediction error for the elastoplastic beam.

Prediction Mode	1-step ($\times 10^{-4}$)	Multi-step rollouts ($\times 10^{-4}$)						
		100-step	200-step	300-step	400-step	500-step	600-step	700-step
Piers	0.79	18.84	50.93	159.41	453.36	864.51	1429.44	2795.62
Baseline I	0.87 (9.93%)	48.62 (61.26%)	264 (80.71%)	994.81 (83.98%)	2890.87 (84.32%)	6126.75 (85.89%)	10154.33 (85.92%)	15121.15 (81.51%)
Baseline II	1.01 (22.32%)	67.52 (72.10%)	712.31 (92.85%)	2618.42 (93.91%)	6460.01 (92.98%)	12226.27 (92.93%)	19558.76 (92.69%)	27847.66 (89.96%)
Baseline III	0.92 (14.88%)	25.13 (25.05%)	79.85 (36.22%)	261.99 (39.15%)	1186.75 (61.80%)	3307.29 (73.86%)	6931.07 (79.38%)	12208.17 (77.10%)

Note: Numbers in brackets denote the relative decrease of RMSE from *Piers* to baselines.

prediction. The reason is that internal forces in previous time steps influence the movement of an elastoplastic beam. *Piers* can accurately learn such influence through incorporating previous edge hidden state in updating the edge states. We also draw Fig. 7 to compare the *Piers* predicted and ground truth deformation of elastoplastic beam during consecutive time steps. They are visually close within 500 steps but the prediction error gets larger when the time step increases.

It is important to clarify that in this experiment, the unloading and reloading patterns were intentionally designed to be linear and relatively simple. This experiment serves to affirm that *Piers* is capable of handling elastoplastic CDBs with path-dependent properties within a limited scope. However, we acknowledge that *Piers*, in its present form, may fall in accurately predicting the behavior of elastoplastic CDBs when subjected to random unloading and reloading patterns that were unseen in the training data. A potential avenue for overcoming this limitation could involve embedding domain-specific prior knowledge about the CDB's constitutive model directly into *Piers*. This approach, as evidenced by related works [31], could enhance the model's predictive accuracy for such complex scenarios. This is an area we intend to explore in our future research.

4.3. Elastic Plate

We use ANSYS to generate the dynamic responses of a 3D elastic plate under dynamic external forces. The material and geometric parameters of the plate are summarized in Table 4. One edge of the plate is fixed and the rest part is free. External forces from arbitrary directions

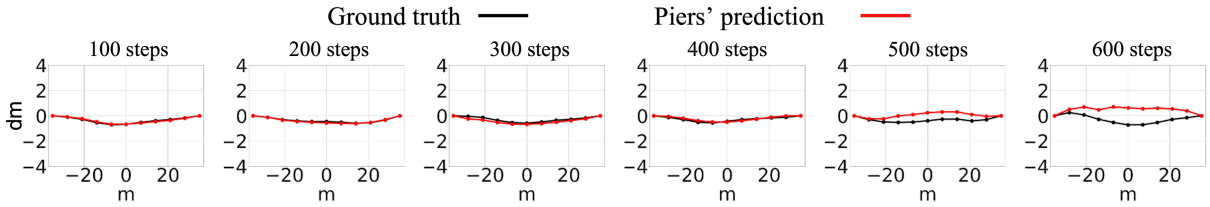


Figure 7: Ground truth and *Piers* predicted deformation of the elastoplastic beam at different time steps. This figure shows the predicted deformations are visually close to the ground truth within 500 steps.

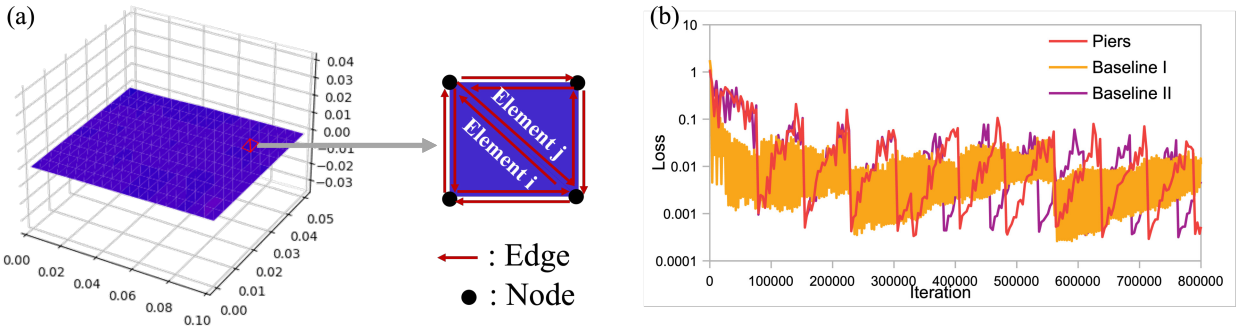


Figure 8: (a)Graph representation for the plate. (b) Learning curve of *Piers* and Baseline models while simulating the plate.

are applied to each node of the plate simultaneously. These external node forces have arbitrary components in x, y, and z axis directions. Similar to the previous simulation for beams, amplitudes of the plate's node external forces are generated with Gaussian distribution in order to cover all vibration patterns. The plate is meshed into 400 triangular elements and then represented with a graph. Different from the beam element, each triangular element has three vertexes. Fig. 8 shows how to generate the graph representation for two triangular elements. The two elements are represented with 4 nodes connected with 10 directed edges. The time-history displacement, external forces, and von-Mises stress at each node are generated with ANSYS and used to generate node and edge attributes in the graph representation.

Table 4: Material and geometric properties of the simulated plate.

Properties	Value
ANSYS element type	Shell 181
Elastic modulus (Pa)	2.1×10^9
Damping ratio	1
Poisson's ratio	0.3
Mass density (kg/m^3)	7850
Thickness (m)	0.002
Width (m)	0.05
Length (m)	0.1
Acceleration of gravity (m/s^2)	9.806

In this dataset, node attribute is $\mathbf{V}^{n\Delta t} = \{(\dot{d}^{(n-1)\Delta t}, \ddot{d}^{(n-1)\Delta t}, d^{n\Delta t}, n_{type}, \mathbf{Ve}_k^{n\Delta t})_k\}_{k=1,\ldots,n_n}$, where $\dot{d}^{(n-1)\Delta t}$ and $\ddot{d}^{(n-1)\Delta t}$ denote the velocity and acceleration in the last time step. $d^{n\Delta t}$

Table 5: One-step and multi-step displacement prediction error for the elastic plate.

Prediction Mode	1-step ($\times 10^{-3}$)	Multi-step rollouts ($\times 10^{-3}$)						
		50-step	100-step	150-step	200-step	250-step	300-step	350-step
Piers	5.20	16.98	30.22	63.13	352.7	2814.62	11625.40	31799.12
Baseline I	2.60 (-99.90%)	46.08 (63.16%)	910.12 (96.68%)	4126.14 (98.47%)	12702.18 (97.22%)	29054.85 (90.31%)	54674.76 (78.74%)	90579.85 (64.89%)
Baseline II	5.25 (0.99%)	60.69 (72.03%)	248.54 (87.84 %)	497.24 (87.30%)	2496.35 (85.87%)	8057.84 (65.07%)	18338.23 (36.61%)	33973.60 (6.40%)

Note: Numbers in brackets denote the relative decrease of RMSE from *Piers* to baselines.

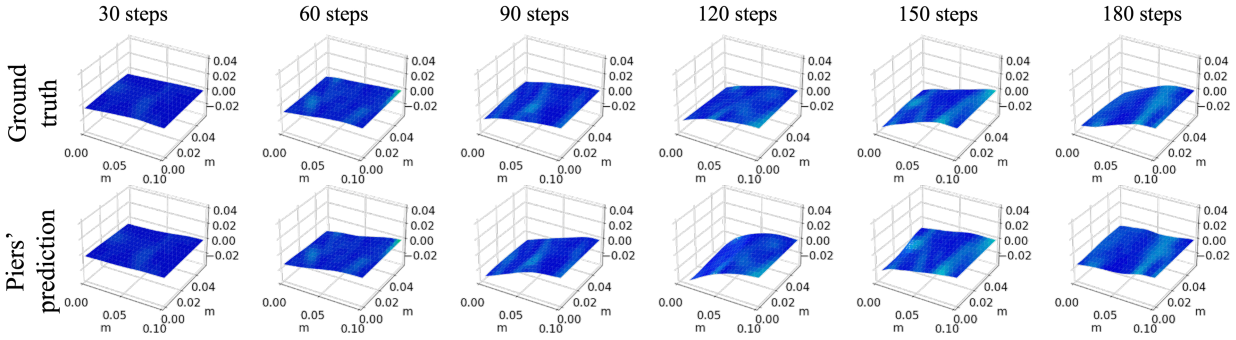


Figure 9: Ground truth and *Piers* predicted deformation of the elastic plate at different time steps. The color map shows the von-Mises stress in the elastic pate when it vibrates under external forces. This figure shows the predicted deformations are visually close to the ground truth within 150 steps. The predicted von-Mises stress are nearly identical with the ground-truth for all time steps.

denotes the displacement at current time step. n_{type} denotes the embedding of node type and $\mathbf{Ve}_k^{n\Delta t}$ denotes external forces at the current time step. Its edge attribute $\mathbf{E}^{n\Delta t} = \{(u_{ij}, |u_{ij}|, x_{ij}, |x_{ij}|)_k^{n\Delta t}\}_{k=1,\ldots,n_e}$, where u_{ij} and x_{ij} denotes the vector of edge in mesh space and world space. To generate training samples for *Piers*, we apply a moving window (20 steps in length) to each trajectory. Referring to [13], we add nodes' von-Mises stress as the additional target output of nodes except for accelerations. Physics-informed loss functions are not used in this experiment because internal forces are transmitted through meshed triangle elements instead of along the edges of elements. And it still remains unsolved how to assign internal forces in each triangle element to its three edges properly in a GNN-based simulator. We'll analyze this task in our future work. Moreover, physics-informed edge hidden states are still used in this experiment. Instead of using the initial internal forces of elements, we use edge features in mesh space $(u_{ij}, |u_{ij}|)$ as edge hidden states in the simulation of the plate. Please refer to Table A2 in Appendix for the summarization of input graph and network architecture for plate simulation.

We train *Piers* and 2 baseline models for 800000 iterations. The learning rate is exponentially decayed from 1×10^{-4} to 1×10^{-6} and optimized with Adam optimizer. As shown in Fig. 8(b), the training loss of all models decrease with the increase of iterations. As Table 5 shows, *Piers* achieves less prediction error than all baseline models in all multi-step rollouts. However, the prediction error of *Piers* in 1-step prediction is larger than Baseline I. Similar to our discussion to Table 2, the reasonable explainantion is that overfitting happends in the 1-step prediction task for Baseline I. Moreover, prediction error of Baseline I and II diverges much faster than *Piers* with the increase of rollout steps. Fig. 9 shows that the overall pre-

Table 6: One-step and multi-step displacement prediction error for the cloth.

Prediction Mode	1-step ($\times 10^{-3}$)	Multi-step rollouts ($\times 10^{-3}$)							
		50-step	100-step	150-step	200-step	250-step	300-step	350-step	400-step
Piers	28.81	259.75	257.08	229.46	213.90	202.37	194.09	188.57	184.55
Baseline I	32.46 (11.25%)	239.02 (-8.68%)	254.79 (-0.90%)	231.93 (1.06%)	221.88 (3.59%)	212.33 (4.69%)	205.82 (5.70%)	201.50 (6.41%)	198.56 (7.05%)
Baseline II	18.00 (-60.01%)	733.06 (64.57%)	868.39 (70.40 %)	888.90 (74.19%)	850.62 (74.85%)	828.90 (75.59%)	822.11 (76.39%)	824.41 (77.13%)	830.58 (77.78%)

Note: Numbers in brackets denote the relative decrease of RMSE from *Piers* to baselines.

dicted deformation of *Piers* is simillar to the ground truth deformation within 150 steps. The von-Mises stress prediction performance is even better than the performance in displacement prediction. Fig. 9 shows that the predicted von-Mises stresses are nearly same with the ground truth within even 180 steps, demonstrating the impressive simulation performance of *Piers*.

4.4. Plastic Cloth

We use the dataset FLAGSIMPLE in [13] to evaluate *Piers* in simulating the dynamics of a plastic body, a flag waving in the wind. In FLAGSIMPLE, the flag is meshed into 3028 triangular elements and their trajectories under different wind forces are collected. The same procedure in simulating the 3D plate is used to generate graph representation for the flag because both of them use triangular elements. Nevertheless, their node attribute $\mathbf{V}^{n\Delta t} = \{(\dot{d}_x, \dot{d}_y, \dot{d}_z, n_{type})_k^{n\Delta t}\}_{k=1,\dots,n_n}$, which is slightly different from the node attribute used in the previous 3D plate. The edge attribute $\mathbf{E}^{n\Delta t}$ is the same as the edge attribute in the plate. To generate training samples for *Piers*, we apply a moving window (20 steps in length) to each trajectory. Similar to the plate in the last subsection, physics-informed loss functions are not used in this experiment. Nevertheless, physics-informed edge hidden states are kept in this experiment. Similar to the plate simulation in the last section, we use edge features in mesh space as the hidden states for edges. Please refer to Table A3 in Appendix for the summarization of input graph and network architecture for cloth simulation.

We train *Piers* and 2 baseline models for 340000 iterations. The learning rate is exponentially decayed from 1×10^{-3} to 1×10^{-5} and optimized with Adam optimizer. Fig. 10 shows the learning curves of the four models and all training loss decrease with the increase of iterations. The prediction error for the cloth under wind forces is shown in Table 6. Its multi-step prediction error fluctuates around 0.2 and is smaller than two baseline models in most cases. Fig. 11 compare the *Piers* predicted and ground truth deformation under consecutive time steps. The shape and deformation of the cloth in *Piers*'s prediction matches well with the ground truth within 300 steps. It illustrate that *Piers* can simulate the dynamic of plastic CDBs.

5. Discussion

5.1. Selection of Hyperparameters

In this section, we explore the optimal selection of two critical hyperparameters: window length and message passing steps. Initially, we train *Piers* on an elastic beam simulation with varying window lengths, ranging from 100 to 1500. The error from 100 to 700 step rollout predictions are shown in Fig. 12 (a). As the window length increases, the prediction

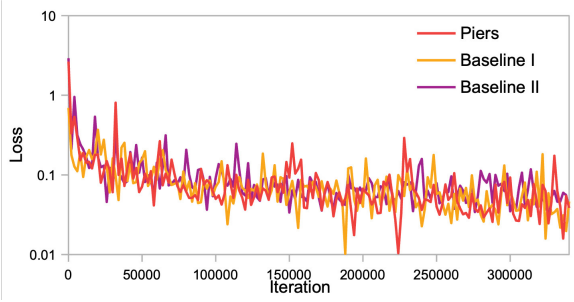


Figure 10: Learning curves of *Piers* and Baseline models for simulating the cloth.

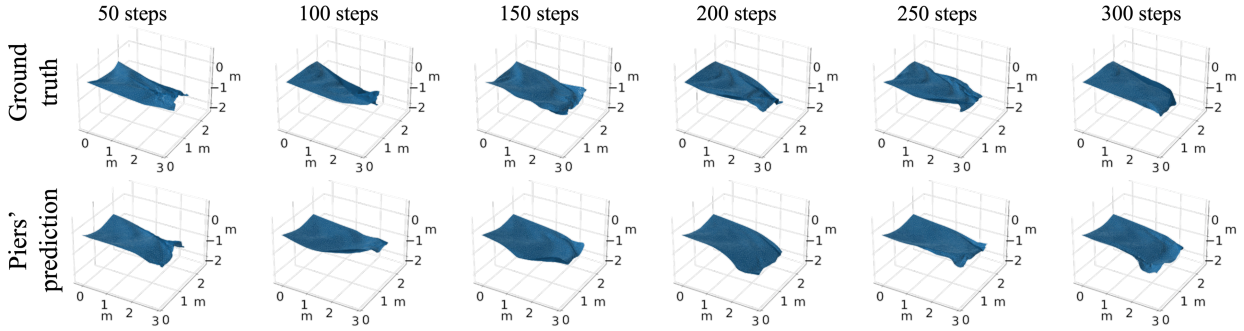


Figure 11: Ground truth and *Piers* predicted deformation of a cloth under wind forces at consecutive time steps. This figure shows the predicted deformations are visually close to the ground truth within 300 steps.

error generally demonstrates a declining trend. This is attributed to the fact that a larger window length necessitates the calculation of error over a longer prediction range in the loss function, thereby compelling *Piers* to minimize its prediction. However, such a trend stops when the window length exceeds 1000 steps. Notably, the errors associated with a 1500 window length can occasionally exceed that of a 1000 window length. This results from the increased complexity of training a RNN using training samples with increased sequence length. Given that *Piers* utilizes an RNN framework, its capacity to capture long-term temporal correlations may be compromised, resulting in diminished learning performance when the input data extends excessively in temporal length. Additionally, large window lengths significantly prolong the training time. Consequently, we opt for a window length of 1000 steps per training sample, as it strikes a balanced compromise between prediction accuracy and computational efficiency.

We use the elastic plate to study the influence of the message-passing step in final learning performance. We vary the message passing steps in *Piers* from 10 to 30, while keeping the number of training iterations constant at 800,000 across the same training set. The rollout prediction errors on the same testing set are presented in Fig. 12(b). Intriguingly, as the number of message-passing steps increases, the prediction error of *Piers* initially rises before subsequently declining. This can be attributed to the fact that a larger message-passing step enhances node connectivity, thereby facilitating more rapid propagation of vibrations from one node to another. However, an excessively high number of message-passing steps can render the network too deep to be trained effectively, leading to a deterioration in prediction accuracy. To achieve the optimal balance between prediction error and model trainability when simulating the elastic plate, we elect to utilize 20 message passing steps.

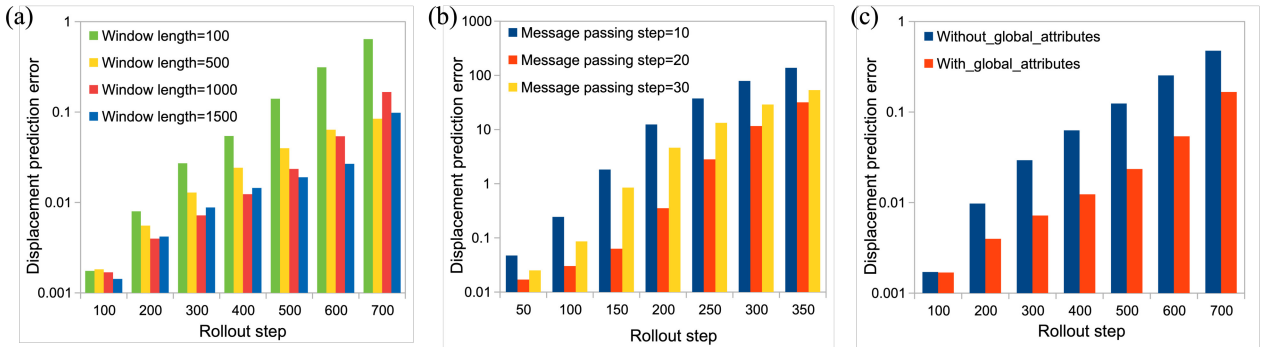


Figure 12: (a) Displacement prediction error of *Piers* trained with samples in different window lengths. The length of training samples varies from 100 to 1500. Prediction errors under 100-700 rollout steps are summarized. (b) Displacement prediction error of *Piers* with different message passing steps. The message passing step varies from 10 to 30. Prediction errors under 50-350 rollout steps are summarized.(c) Displacement prediction error of *Piers* with and without global attributes.

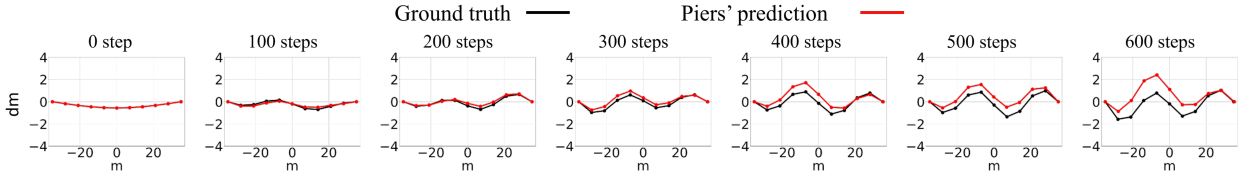


Figure 13: Ground truth and *Piers* predicted deformation at different time steps. Sine forces are applied at node 2.

We delved into the contribution of including constant global attributes in the dataset. For comparative purposes, we trained *Piers* on the elastic beam dataset, excluding the global attributes, while maintaining all other hyperparameters unchanged. The trained model is used for rollout prediction. As depicted in Figure 12(c), the inclusion of global attributes markedly improved performance. Notably, the rollout prediction errors across various time steps consistently diminished upon integrating constant global attributes into the input.

5.2. Generalizability to Unseen Load Conditions

To test the generalization ability of *Piers*, we applied sine force $F_{ext} = F_{am}sin(20\pi t)$ to the elastic beam, where F_{am} denotes the amplitude of the sine force. Fig. 13 shows that our *Piers* can still predict the beam's vibration accurately even though sine forces are not used in model training. The predicted deformation of the elastic beam shows the sine mode shape clearly, which matches the ground deformation very well. Moreover, Fig. 14 shows the *Piers* predicted deformation and ground truth beam deformation under sine forces. Because sine forces are unseen load conditions for *Piers* during model training, the predicted error grows with the increase of time steps. However, the predicted dynamic responses and the ground truth are visually close in frequency, phase, and amplitude in the first 300 steps. Although the predictions deviate from target displacements faster than tests in Fig. 15(a), it is still impressive to find that deformations predicted by *Piers* are very close to the ground truth at the first hundreds of time steps.

5.3. Node-wise Time-history Displacements Comparison

We compare the node-wise time-history displacements predicted by the four models with the ground truth in Fig. 15. Displacements predicted by *Piers* are the closest to the ground

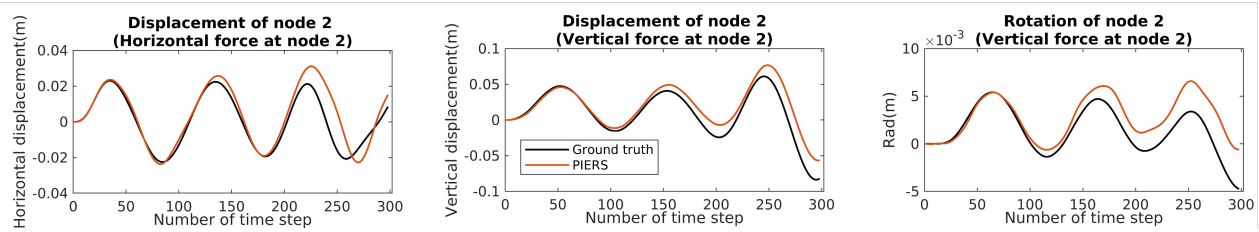


Figure 14: Node 2's displacements predicted by *Piers* and ground truth. Sine forces are applied at node 2.

truth visually. Although the prediction of *Piers* finally deviates from the ground truth with the increase of time steps, the predicted node-wise displacements show very similar vibration patterns (e.g., frequency, amplitude, and phase) with the ground truth within hundreds steps. Meanwhile, the responses predicted by Baseline I deviate from the ground truth at the first several steps. Predictions of Baseline II and III also deviate from the ground truth earlier than *Piers*.

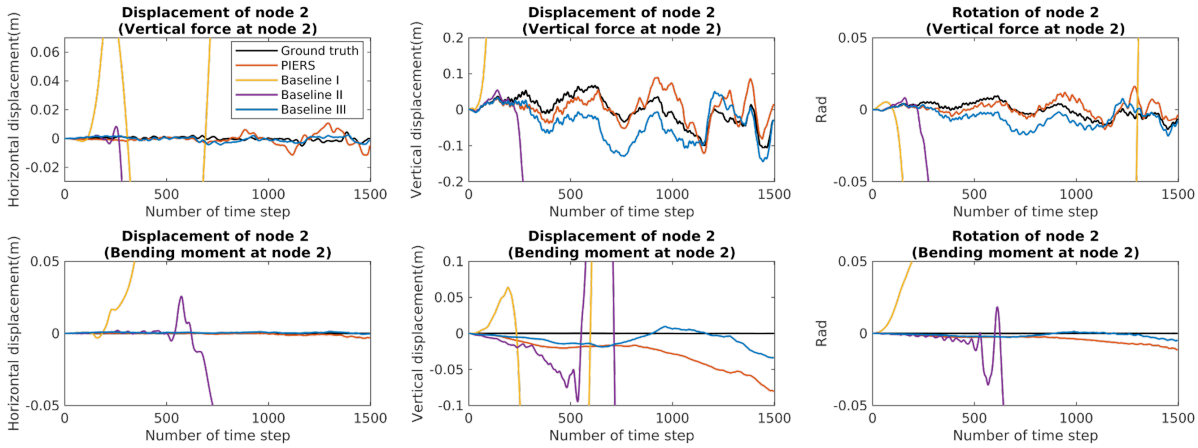
However, Fig. 15 also underscores the room for improvement in *Piers'* prediction accuracy. The current version of *Piers* can only achieve high-fidelity predictions over a limited number of time steps. While the vibrational patterns predicted by *Piers* align closely with the ground truth when the rollout step increases, the predictive precision tends to wane simultaneously. A paramount focus for our subsequent work is to enhance the displacement accuracy during multi-step rollouts.

6. Conclusion

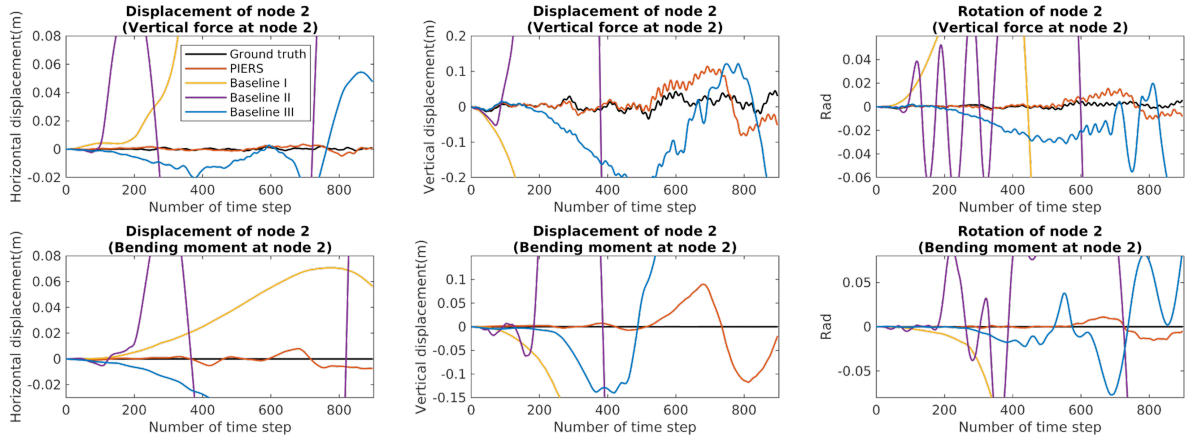
In this paper, we proposed *Piers*, the first unified GNN-based simulator to predict the dynamic responses of CDBs. The proposed model can learn the relationship between arbitrary node external forces and structural response by incorporating GRU for computing the message and a physics-informed loss function for model training. Its prediction results in elastic, elastoplastic, and plastic CDBs show that the trained *Piers* can accurately predict the dynamic response CDBs with different material using an unified framework. Moreover, it can be generalized to load conditions unseen in training sets. Though the predicted response finally deviates from the ground truth with the increase of time steps, the impressive match between the predicted and ground-truth responses at the first hundreds of time steps proves the capability of GNN in simulating the dynamics of a CDB. Based on the work in this paper, we will further improve its prediction accuracy and simulate more complex structures in our future work.

Acknowledgments

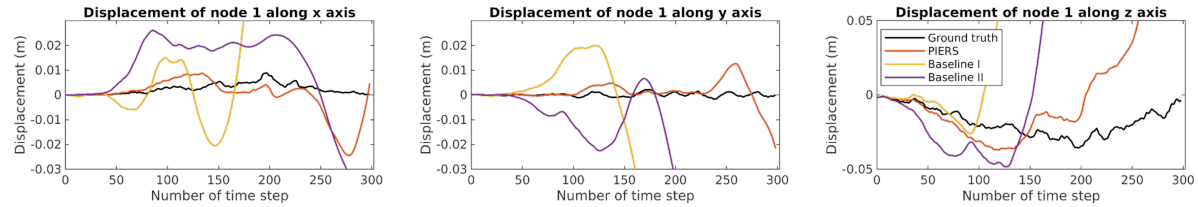
The authors are grateful for the financial supports from the Innovation and Technology Commission of Hong Kong through Smart Railway Technology and Applications (No. K-BBY1), the Research Grants Council of Hong Kong through the Theme-based Research Scheme (T22-502/18-R), and the Research Institute for Artificial Intelligence of Things, the Hong Kong Polytechnic University. The findings and opinions expressed in this paper are from the authors alone and are not necessarily the views of the sponsors.



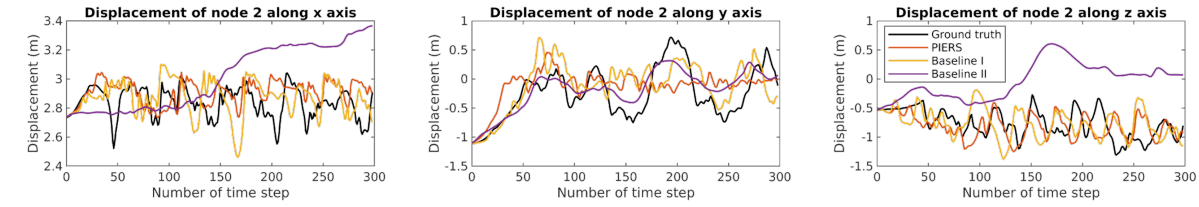
(a) Displacements of node 2 in the elastic beam



(b) Displacements of node 2 in the elastoplastic beam



(c) Displacements of node 1 in the elastic plate



(d) Displacements of node 2 in the cloth

Figure 15: Node-wise displacements predicted by *Piers* and baseline models for the four CDBs (Best viewed in color). Compared with baseline models, displacements predicted by *Piers* have more similar vibration patterns (e.g., frequency, amplitude, and phase) with the ground truth.

References

[1] W. Gawronski, Advanced structural dynamics and active control of structures, Springer Science & Business Media, 2004.

[2] Y. C. Kurama, S. Sritharan, R. B. Fleischman, J. I. Restrepo, R. S. Henry, N. M. Cleland, S. Ghosh, P. Bonelli, Seismic-resistant precast concrete structures: state of the art, Journal of Structural Engineering 144 (4) (2018) 03118001.

[3] A. Nealen, M. Müller, R. Keiser, E. Boxerman, M. Carlson, Physically based deformable models in computer graphics, in: Computer graphics forum, Vol. 25, Wiley Online Library, 2006, pp. 809–836.

[4] F. Cheli, G. Diana, Advanced dynamics of mechanical systems, Springer, 2015.

[5] A. K. Chopra, Dynamics of structures, Pearson Education India, 2007.

[6] O. C. Zienkiewicz, R. L. Taylor, J. Z. Zhu, The finite element method: its basis and fundamentals, Elsevier, 2005.

[7] C. Coetzee, Calibration of the discrete element method, Powder Technology 310 (2017) 104–142.

[8] M. H. Aliabadi, The boundary element method, volume 2: applications in solids and structures, Vol. 2, John Wiley & Sons, 2002.

[9] B. Ummenhofer, L. Prantl, N. Thuerey, V. Koltun, Lagrangian fluid simulation with continuous convolutions, in: International Conference on Learning Representations, 2019.

[10] N. Thuerey, K. Weißenow, L. Prantl, X. Hu, Deep learning methods for reynolds-averaged navier–stokes simulations of airfoil flows, AIAA Journal 58 (1) (2020) 25–36.

[11] T. Takahashi, J. Liang, Y.-L. Qiao, M. C. Lin, Differentiable fluids with solid coupling for learning and control, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 35, 2021, pp. 6138–6146.

[12] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, K. Kavukcuoglu, Interaction networks for learning about objects, relations and physics, arXiv preprint arXiv:1612.00222 (2016).

[13] T. Pfaff, M. Fortunato, A. Sanchez-Gonzalez, P. W. Battaglia, Learning mesh-based simulation with graph networks, in: International Conference on Learning Representations, 2021.

[14] Y. Gao, G. Strang, Geometric nonlinearity: Potential energy, complementary energy, and the gap function, Quarterly of Applied Mathematics 47 (3) (1989) 487–504.

[15] Y. Li, J. Wu, R. Tedrake, J. B. Tenenbaum, A. Torralba, Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids, arXiv preprint arXiv:1810.01566 (2018).

[16] H. Peng, H. Wang, B. Du, M. Z. A. Bhuiyan, H. Ma, J. Liu, L. Wang, Z. Yang, L. Du, S. Wang, et al., Spatial temporal incidence dynamic graph neural networks for traffic flow forecasting, Information Sciences 521 (2020) 277–290.

[17] Y. Seo, M. Defferrard, P. Vandergheynst, X. Bresson, Structured sequence modeling with graph convolutional recurrent networks, in: Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I 25, Springer, 2018, pp. 362–373.

[18] A. Sanchez-Gonzalez, N. Heess, J. T. Springenberg, J. Merel, M. Riedmiller, R. Hadsell, P. Battaglia, Graph networks as learnable physics engines for inference and control, in: International Conference on Machine Learning, PMLR, 2018, pp. 4470–4479.

[19] P. W. Battaglia, J. B. Hamrick, V. Bapst, A. Sanchez-Gonzalez, V. Zambaldi, M. Malinowski, A. Tacchetti, D. Raposo, A. Santoro, R. Faulkner, et al., Relational inductive biases, deep learning, and graph networks, arXiv preprint arXiv:1806.01261 (2018).

[20] T. Wang, R. Liao, J. Ba, S. Fidler, Nervenet: Learning structured policy with graph neural networks, in: International Conference on Learning Representations, 2018.

[21] S. Wiewel, M. Becher, N. Thuerey, Latent space physics: Towards learning the temporal evolution of fluid flow, in: Computer graphics forum, Vol. 38, Wiley Online Library, 2019, pp. 71–82.

[22] F. de Avila Belbute-Peres, T. D. Economon, J. Zico Kolter, Combining differentiable pde solvers and graph neural networks for fluid flow prediction, arXiv e-prints (2020) arXiv–2007.

[23] S. Garg, M. Pant, Meshfree methods: A comprehensive review of applications, International Journal of Computational Methods 15 (04) (2018) 1830001.

[24] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, P. Battaglia, Learning to simulate complex physics with graph networks, in: International Conference on Machine Learning, PMLR, 2020, pp. 8459–8468.

[25] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, G. E. Dahl, Neural message passing for quantum chemistry, in: International conference on machine learning, PMLR, 2017, pp. 1263–1272.

[26] T. Wu, T. Maruyama, Q. Zhao, G. Wetzstein, J. Leskovec, Learning controllable adaptive simulation for multi-resolution physics, arXiv preprint arXiv:2305.01122 (2023).

[27] K. Cho, B. Van Merriënboer, D. Bahdanau, Y. Bengio, On the properties of neural machine translation: Encoder-decoder approaches, arXiv preprint arXiv:1409.1259 (2014).

[28] E. C. Ting, C. Shih, Y.-K. Wang, Fundamentals of a vector form intrinsic finite element: Part i. basic procedure and a plane frame element, Journal of Mechanics 20 (2) (2004) 113–122.

[29] E. C. Ting, C. Shih, Y.-K. Wang, Fundamentals of a vector form intrinsic finite element: Part ii. plane solid elements, Journal of Mechanics 20 (2) (2004) 123–132.

[30] Y. Duan, S. Wang, J. Yau, Vector form intrinsic finite element method for analysis of train–bridge interaction problems considering the coach-coupler effect, International Journal of Structural Stability and Dynamics 19 (02) (2019) 1950014.

[31] N. N. Vlassis, W. Sun, Sobolev training of thermodynamic-informed neural networks for interpretable elasto-plasticity models with level set hardening, Computer Methods in Applied Mechanics and Engineering 377 (2021) 113695.

Appendix A. Architectures of *Piers* and baseline Models

Table A1 shows the input features and size of neural networks for beam simulation. Baseline I has the same architecture with the model in GNS and MESHGRAPHNETS. To ensure the fair competition, edge input of Baseline I is the same with that of *Piers*. We choose node input of Baseline I based on that in GNS. The two most recent velocities and the previous acceleration of each node are included in the node attributes for baseline I except for all the node input used in *Piers*. The encoder and decoder in Baseline I are MLPs and their sizes are listed in Table A1. $\mathbf{Vd}_k^{n\Delta t}, \mathbf{Ve}_k^{n\Delta t}, \mathbf{E}^{n\Delta t}, \mathbf{u}$ have been introduced in Section 3.3. $(\dot{d}_x, \dot{d}_y, \dot{d}_\alpha)_k^{(n-2)\Delta t}$ and $(\dot{d}_x, \dot{d}_y, \dot{d}_\alpha)_k^{(n-1)\Delta t}$ denote the velocities of node k at time $(n-2)\Delta t$ and $(n-1)\Delta t$. $(\ddot{d}_x, \ddot{d}_y, \ddot{d}_\alpha)_k^{(n-1)\Delta t}$ denote the accelerations of node k at time $(n-1)\Delta t$.

Table A2 and shows the input features and size of neural networks for plate simulation. Table A3 shows the input features and size of neural networks for cloth simulation. The used $\mathbf{V}^{n\Delta t}$ and $\mathbf{E}^{n\Delta t}$ have been introduced in Section 4.3 and Section 4.4. Moreover, similar to the training noises in [24], we also add noises to node and edge attributes when training *Piers* and baseline models for the plate and cloth simulation.

Table A1: Input and architecture of *Piers* and baseline models for beam simulation.

Models	Input				Architecture of models
	Node (k = 1,...,n_n)	Edge (k = 1,...,n_e)	Hidden	Global	
Baseline I	$\mathbf{Vd}_k^{n\Delta t}, \mathbf{Ve}_k^{n\Delta t}$, $(\dot{d}_x, \dot{d}_y, \dot{d}_\alpha)_k^{(n-2)\Delta t}$, $(\dot{d}_x, \dot{d}_y, \dot{d}_\alpha)_k^{(n-1)\Delta t}$, $(\ddot{d}_x, \ddot{d}_y, \ddot{d}_\alpha)_k^{(n-1)\Delta t}$	$\mathbf{E}^{n\Delta t}$	/	$\mathbf{u}$	Edge encoder: MLP (128*128*128) Node encoder: MLP (128*128*128) Edge update function: MLP (128*128*128) Node update function: MLP (128*128*128) Node decoder: MLP (128*128*3) Message passing step: 10
Baseline II	$\mathbf{Vd}_k^{n\Delta t}, \mathbf{Ve}_k^{n\Delta t}$, $(d_x, d_y, d_\alpha)_k^{(n-1)\Delta t}$	$\mathbf{E}^{n\Delta t}$	/	$\mathbf{u}$	Edge update function: GRU (hidden size=6) Node update function: MLP (128*128*9) Node decoder: MLP (128*128*3)
Baseline III	$\mathbf{Vd}_k^{n\Delta t}, \mathbf{Ve}_k^{n\Delta t}$, $(d_x, d_y, d_\alpha)_k^{(n-1)\Delta t}$	$\mathbf{E}^{n\Delta t}$	$\mathbf{h}^{n\Delta t}$	$\mathbf{u}$	Edge update function: GRU (hidden size=6) Node update function: MLP (128*128*9) Node decoder: MLP (128*128*3)
Piers	$\mathbf{Vd}_k^{n\Delta t}, \mathbf{Ve}_k^{n\Delta t}$, $(d_x, d_y, d_\alpha)_k^{(n-1)\Delta t}$	$\mathbf{E}^{n\Delta t}$	$\mathbf{h}^{n\Delta t}$	$\mathbf{u}$	Edge update function: GRU (hidden size=6) Node update function: MLP (128*128*9) Node decoder: MLP (128*128*3)

Note: Baseline I has the same architecture with the GNS and MESHGRAPHNETS; Baseline II is *Piers* without internal forces as hidden states; Baseline III is *Piers* without physics-informed loss function; The MLP with size 128*128*128 has two hidden layers with 128 neurons. its output size is 128

Table A2: Input and architecture of *Piers* and baseline models for plate simulation.

Models	Input				Architecture of models
	Node (k = 1,...,n_n)	Edge (k = 1,...,n_e)	Hidden	Global	
Baseline I	$\dot{d}_k^{(n-1)\Delta t}, \ddot{d}_k^{(n-1)\Delta t}$, $d_k^{n\Delta t}, n_{type}, \mathbf{Ve}_k^{n\Delta t}$	$\mathbf{E}^{n\Delta t}$	/	/	Edge encoder: MLP (128*128*128) Node encoder: MLP (128*128*128) Edge update function: MLP (128*128*128) Node update function: MLP (128*128*128) Edeg decoder: MLP (128*128*3) Message passing step: 20
Baseline II	$\dot{d}_k^{(n-1)\Delta t}, \ddot{d}_k^{(n-1)\Delta t}$, $d_k^{n\Delta t}, n_{type}, \mathbf{Ve}_k^{n\Delta t}$	$(x_{ij}, \lvert x_{ij}\rvert)_k^{n\Delta t}$	/	/	Edge update function: GRU (hidden size=6) Node update function: MLP (128*128*9) Node decoder: MLP (128*128*3) Message passing step: 20
Piers	$\dot{d}_k^{(n-1)\Delta t}, \ddot{d}_k^{(n-1)\Delta t}$, $d_k^{n\Delta t}, n_{type}, \mathbf{Ve}_k^{n\Delta t}$	$(x_{ij}, \lvert x_{ij}\rvert)_k^{n\Delta t}$	$(u_{ij}, \lvert u_{ij}\rvert)_k^{n\Delta t}$	/	Edge update function: GRU (hidden size=6) Node update function: MLP (128*128*9) Node decoder: MLP (128*128*3) Message passing step: 20

Note: Baseline I has the same architecture with the GNS and MESHGRAPHNETS; Baseline II is *Piers* without edge hidden states

Table A3: Input and architecture of *Piers* and baseline models for cloth simulation.

Models	Input				Architecture of models
	Node (k = 1,...,n_n)	Edge (k = 1,...,n_e)	Hidden	Global	
Baseline I	$(\dot{d_x}, \dot{d_y}, \dot{d_z}, n_{type})_k^{n\Delta t}$	$\mathbf{E}^{n\Delta t}$	/	/	Edge encoder: MLP (128*128*128) Node encoder: MLP (128*128*128) Edge update function: MLP (128*128*128) Node update function: MLP (128*128*128) Edeg decoder: MLP (128*128*3) Message passing step: 15
Baseline II	$(\dot{d_x}, \dot{d_y}, \dot{d_z}, n_{type})_k^{n\Delta t}$	$(x_{ij}, \lvert x_{ij}\rvert)_k^{n\Delta t}$	/	/	Edge update function: GRU (hidden size=6) Node update function: MLP (128*128*9) Node decoder: MLP (128*128*3) Message passing step: 5
Piers	$(\dot{d_x}, \dot{d_y}, \dot{d_z}, n_{type})_k^{n\Delta t}$	$(x_{ij}, \lvert x_{ij}\rvert)_k^{n\Delta t}$	$(u_{ij}, \lvert u_{ij}\rvert)_k^{n\Delta t}$	/	Edge update function: GRU (hidden size=6) Node update function: MLP (128*128*9) Node decoder: MLP (128*128*3) Message passing step: 5

Note: Baseline I has the same architecture with the GNS and MESHGRAPHNETS; Baseline II is *Piers* without edge hidden states