Exact Dirichlet Boundary Physics-informed Neural Network EPINN for Solid Mechanics

Jiaji Wang[1], Y.L. Mo[2], Bassam Izzuddin[3], Chul-Woo Kim[*,4]

Abstract: Physics-informed neural networks (PINNs) have been rapidly developed for solving partial differential equations. The **E**xact Dirichlet boundary condition **P**hysics-**i**nformed **N**eural **N**etwork (EPINN) is proposed to achieve efficient simulation of solid mechanics problems based on the principle of least work with notably reduced training time. There are five major building features in the EPINN framework. First, for the 1D solid mechanics problem, the neural networks are formulated to exactly replicate the shape function of linear or quadratic truss elements. Second, for 2D and 3D problems, the tensor decomposition was adopted to build the solution field without the need of generating the finite element mesh of complicated structures to reduce the number of trainable weights in the PINN framework. Third, the principle of least work was adopted to formulate the loss function. Fourth, the exact Dirichlet boundary condition (i.e., displacement boundary condition) was implemented. Finally, the meshless finite difference (MFD) was adopted to calculate gradient information efficiently. By minimizing the total energy of the system, the loss function is selected to be the same as the total work of the system, which is the total strain energy minus the external work done on the Neumann boundary conditions (i.e., force boundary conditions). The exact Dirichlet boundary condition was implemented as a hard constraint compared to the soft constraint (i.e., added as additional terms in the loss function), which exactly meets the requirement of the principle of least work. The EPINN framework is implemented in the Nvidia Modulus platform and GPU-based supercomputer and has achieved notably reduced training time compared to the conventional PINN framework for solid mechanics problems. Typical numerical examples are presented. The convergence of

1 Ph.D., Assistant professor, Department of Civil Engineering, The University of Hong Kong, Hong Kong, China. Email: cewang@hku.hk
2 Ph.D., John and Rebecca Moores Professor, Department of Civil and Environmental Engineering, University of Houston, US. Email: yilungmo@central.uh.edu
3 Ph.D., Professor, Department of Civil and Environmental Engineering, Imperial College London, London, UK. Email: b.izzuddin@imperial.ac.uk
4 Ph.D., Professor, Corresponding author, Department of Civil and Earth Resources Engineering, Kyoto University, Japan. Email: kim.chulwoo.5u@kyoto-u.ac.jp

EPINN is reported and the training time of EPINN is compared to conventional PINN architecture and finite element solvers. Compared to conventional PINN architecture, EPINN achieved a speedup of more than 13 times for 1D problems and more than 126 times for 3D problems. The simulation results show that EPINN can even reach the convergence speed of finite element software. In addition, the prospective implementations of the proposed EPINN framework in solid mechanics are proposed, including nonlinear time-dependent simulation and super-resolution network.

KEYWORDS: Physics-informed neural network (PINN); Exact Dirichlet boundary PINN (EPINN); Principle of least work; Solid mechanics; Finite element; Tensor decomposition; Meshless finite difference (MFD); Nvidia Modulus

1. INTRODUCTION

Over the past 50 years, there has been substantial development in simulating solid mechanics problems by solving the governing equations of partial differential equations (PDEs) using the finite element (FE) method [1]. Although significant progress has been achieved in the FE method for the forward problems of simulation in solid mechanics problems, the existing FE method may still face several challenges in solving inverse problems (i.e., model updating for material parameters [2]) or design optimization problems (i.e., optimum design of engineering structures) because of notable computational costs. Engineers and researchers may adopt FE models of varying levels of sophistication for mechanical performance assessment and design of structures, which may pose prohibitive procedural and time demands in computational structural optimization. In FE simulation, the gradient information of output fields with respect to design parameters is hard, if not impossible, to obtain. Therefore, any optimization of complicated structures according to the traditional approach would typically adopt gradient-free algorithms (e.g., genetic algorithm), which require thousands of FE simulations posing much greater computational demands compared to gradient-based algorithms (e.g., gradient descent algorithm). The optimization of structures may include hundreds of design parameters and gradient-free optimization using FE software can be infeasible in practice. In addition, traditional FE software mostly relies on Central Processing Units (CPUs),

while the utilization of Graphical Processing Units (GPUs) is seldom considered in conventional FE software. A CPU contains a few cores with substantial cache memory to complete fewer computational threads in parallel. In contrast, a GPU is composed of thousands of cores to complete thousands of threads in parallel. Compared to CPUs, GPUs are designed to subdivide complex problems into thousands of separate tasks and compute them in parallel, making them ideal for machine learning (ML) tasks. In general, the FE models may face challenges for inverse problems, optimization problems, and full utilization of GPUs and GPU-based supercomputers.

With the rapid development of artificial intelligence (AI) and GPU-based supercomputers, deep learning using deep neural networks (DNNs) has achieved success in many research fields, including but not limited to computer vision [3], natural language processing, self-driving cars, biological science [4], generative modeling [5], and recommendation systems. DNNs with trillions of trainable weights can be trained on state-of-the-art GPU-based supercomputers [5] and the infusing between AI and computational solid mechanics has been a heated research topic. Figure 1 shows the schematic plot of AI models from model-driven to data-driven algorithms for solid mechanics. As shown in Figure 1, the FE approach is model-driven and meets the governing equations. In comparison, conventional deep-learning methods including convolutional neural networks (CNN)[6], recurrent neural networks (RNN)[7], and Transformers [8] are data-driven. Physics-informed Neural Networks (PINNs) [9, 10] were proposed to solve forward problems and inverse problems of physics systems governed by partial differential equations (PDEs) by training neural networks on GPUs, which is similar to the FE model approach. In addition, Fourier Neural Operators (FNO) were also proposed to learn the mapping from the input field to the solution field based on big data generated by FE simulation with zero-shot super-resolution performance [11]. The development of PINN architecture is briefly reviewed below while a comprehensive literature review can be referred to Karniadakis et al. [1]. Based on the universal approximation theorem [12], E and Yu [13] proposed the Deep Ritz method for solving PDEs, including a fully-connected deep neural network with residual connections to serve as the trial function, and neural networks to obtain the total energy of the system based

on trial functions. The Deep Ritz method showed converged results for the Poisson equation in two dimensions and high dimension and transfer learning was also proposed and implemented. Raissi et al. [9] proposed PINNs to replicate the shape function and minimize the error of force equilibrium equations to simulate solid mechanics problems, including both forward and inverse problems. The PINNs proposed by Raissi et al. [9] include both continuous-time models and discrete-time models, and the objective function is to minimize the total error of force equilibrium equations. Haghighat et al. [14] proposed the PINN framework for surrogate modeling of solid mechanics including both a linear elasticity problem and a von Mises elastoplastic problem. The fully connected neural networks were used to input coordinates and predict the displacement field and stress field, and the loss function is formulated as the total error of governing equations of solid mechanics (i.e. strain compatibility equation, equilibrium equation, constitutive model, Dirichlet boundary conditions, and Neumann boundary conditions). The challenges of conventional PINN in simulation of solid mechanics problems may be categorized into (1) difficulty in exact imposition of boundary conditions; (2) lack of efficient architecture of neural networks to fit solid mechanics problems; (3) difficulty in optimum definition of loss function; (4) automated differentiation for gradient back-propagation may be inaccurate. Because of these challenges, current PINN architectures are mostly used when there are labeled data obtained from experimental study or FE analysis, while the successful training of PINN without labeled data are still very challenging.

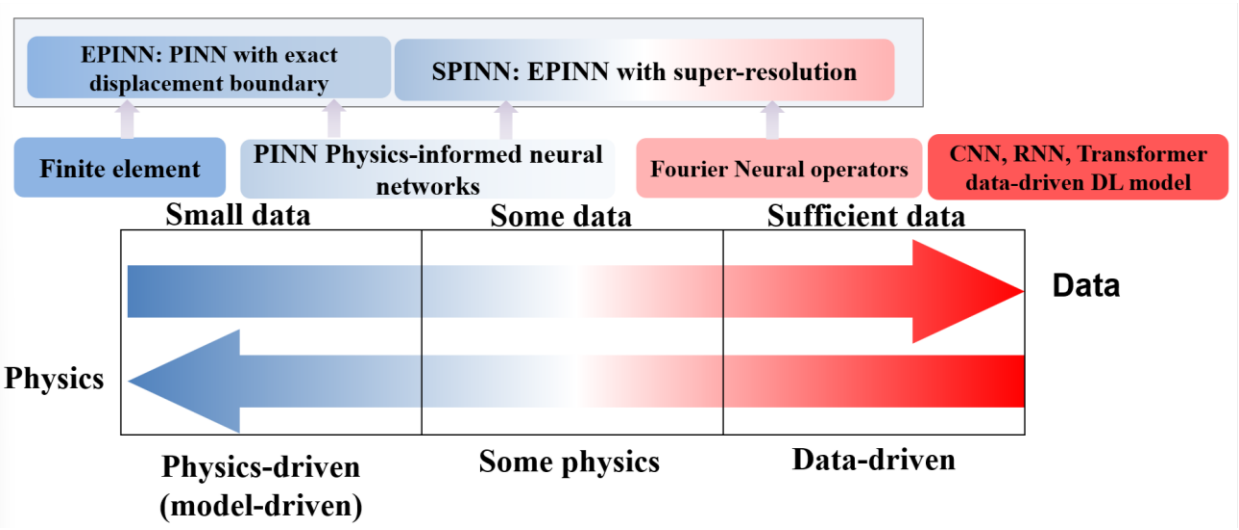


Figure 1. From model-driven to data-driven algorithms for solid mechanics

Note: EPINN is proposed and implemented in this study, while SPINN is proposed in Sect. 4.2.

For the boundary condition of PINN, Rao et al. [15] summarized that the conventional PINN framework considered residual loss components as soft constraints with Lagrange multipliers, which may not exactly meet the boundary conditions and may notably reduce convergence speed. In response to this issue, Rao et al. [15] proposed a PINN architecture to exactly imposing boundary conditions. The "hard" boundary condition enforcement was achieved by training and infusing three single neural networks: the boundary condition network, the distance function network, and the general solution network. Sukumar and Srivastava [16] also proposed a novel approach to exactly imposing boundary conditions in PINN architecture based on constructive solid geometry through approximate distance function (ADF). Based on ADF, a reasonable trial function can be obtained for the Dirichlet boundary condition and the Neumann boundary condition. However, the imposition of exact Neumann boundary conditions in solid mechanics PINN requires the neural network to predict displacement and stress simultaneously, thereby increasing the number of trainable weights in PINN architecture, and is inconsistent with the FE method. Furthermore, this is difficult to extend to nonlinear solid mechanics problems. In the EPINN architecture proposed in this study, the Dirichlet boundary condition is satisfied exactly by multiplying the displacement trial function with ADF, while the Neuman boundary condition is achieved by optimizing the loss function through the principle of least work.

For the architectures of PINN, the research community proposed many network architectures [17] to improve the performance of PINN, including but not limited to the Fourier Network [18], Modified Fourier Network [19], Highway Fourier Network [20], Multi-scale Fourier Feature Network [21], Spatial-temporal Fourier Feature Network [21], Sinusoidal Representation Networks [22], Deep Galerkin Method (DGM) architecture [23] and Multiplicative Filter Network [24], which have been implemented in Nvidia Modulus platform [25]. In these architectures, it may be hard to define the concept equivalent to the mesh size in the FE method, and the number of trainable weights may be notably higher than the Degree of freedom (DOF)

in FE methods upon convergence. In addition, because the loss function of PINN is typically a summation of the error of various governing equations with various units, it may be challenging to balance each term in the loss function. Various normalization methods to balance the gradient of each loss term was proposed, including SoftAdapt [26], Relative Loss Balancing with Random Lookback (ReLoBRaLo) [27], and GradNorm [28]. However, these methods cannot guarantee the convergence of solid mechanics problems and the physical meaning of these methods in PINN are not clear. Recently, the infusing between PINN architecture and the FE method was proposed to achieve efficient architecture and loss function which are consistent with the FE method. Saha et al. [29] and Zhang et al. [30] proposed Hierarchical Deep Learning Neural Network (HiDeNN), where the weights and bias of deep neural networks are implemented based on spatial discretization and element mesh of the FE approach. As proved by Saha et al. [29] and Zhang et al. [30], HiDeNN can exactly replicate the spatial discretization of the FE method. HiDeNN was developed to achieve the construction of DNNs in the same manner as the FE software, which takes in the nodal coordinates as input and produces a shape function in the form of DNN, whose weights are exactly derived from nodal positions. The number of trainable parameters can be reduced to the same as the number of DOFs in traditional FE software at the same mesh size. Recently, a reduced-order Hierarchical Deep learning Neural Network based on Tensor Decomposition (HiDeNN-TD) was also proposed [31], which infuses the HiDeNN with TD methods and achieved convergence for solid mechanics problems with high accuracy and notably lower trainable weights. The TD method and principle of least work are adopted in the EPINN framework. In PINN architecture, the gradient of the solution field with respect to coordinates and time is mostly obtained using back-propagation and automated differentiation of neural networks. However, the automated differentiation may obtain notably high gradient results at stress localization regions, which may reduce the speed of convergence for solid mechanics problems. In this study, the meshless finite difference (MFD) is adopted in EPINN framework for efficient simulation of solid mechanics problems. In general, EPINN is proposed in this study to solve solid mechanics problems even without the need of additional labeled data of the solution field from FE simulation or experimental study.

In this study, the following technical development in this study is summarized:

(1) First proved the shape function of truss element can be reformulated by Convolutional neural networks.

(2) Adopted tensor decomposition to further reduce the number of trainable weights in EPINN to be even lower than the DOF of finite element method at the same mesh size, thereby notably reducing the training cost and improving the convergence.

(3) Developed efficient loss function following the total action of the mechanical system, which is explainable and avoids the problem in conventional PINN methods, there is no need to balance the loss terms of PDE loss, Dirichlet boundary loss, and Neumann boundary loss.

(4) In the EPINN framework, there is no need to meet the Neumann boundary condition exactly, thereby notably reducing the difficulty in training PINNs. The Dirichlet boundary condition can be exactly met, which may avoid the influence of error from the Dirichlet boundary to influence the training process.

2. MODEL ARCHITECTURE OF EPINN

In this section, the model architecture of EPINN is illustrated in detail with a comparison to existing PINN models. Figure 2 shows the schematic plot of EPINN. As shown in Figure 2, EPINN formulates neural networks in each direction (x,y,z) to exactly replicate the shape function of linear or quadratic truss element (Sect. 2.1) and use tensor decomposition (TD) to construct the 2D or 3D displacement field. For the forward problem, all trainable parameters in the EPINN framework are nodal displacements in this spatial discretization layer (denoted in blue color in Figure 2). TD can build the solution field without the need of generating mesh conforming to the complicated structure (Sect. 2.2). The exact displacement boundary condition was implemented (Sect. 2.4) based on ADF (approximate distance function). The meshless finite difference (MFD) was adopted to calculate the gradient of the displacement field with respect to coordinates (i.e. updating strain field) efficiently (Sect. 2.5). For the linear elastic problem or hyperelastic problem, the stress field can be obtained from strain field directly. For nonlinear problems, pretrained deep learning constitutive models can be adopted to achieve stress updating, including but not

limited to Temporal Convolutional Network (TCN) [32], Long-Short Term Memory (LSTM), Gated Recurrent Unit (GRU) and Sequence to Sequence models (Seq2Seq) [33]. MFD was used to formulate the strain solution layer and calculate the total strain energy of the system. By minimizing the total energy of the system, the solution can be achieved. All the PINN models reported and compared in this study are implemented using the Nvidia Modulus platform (version 22.07) [25] using Intel Xeon Platinum 8368 CPU with Nvidia A100 40GB SXM GPU accelerators provided by Osaka University SQUID (Supercomputer for Quest to Unsolved Interdisciplinary Datascience). The Nvidia Modulus platform is an open-source platform developed based on PyTorch and Sympy and can be used for Nvidia GPU accelerators. Single precision was used for the training process of all PINN models in this study.

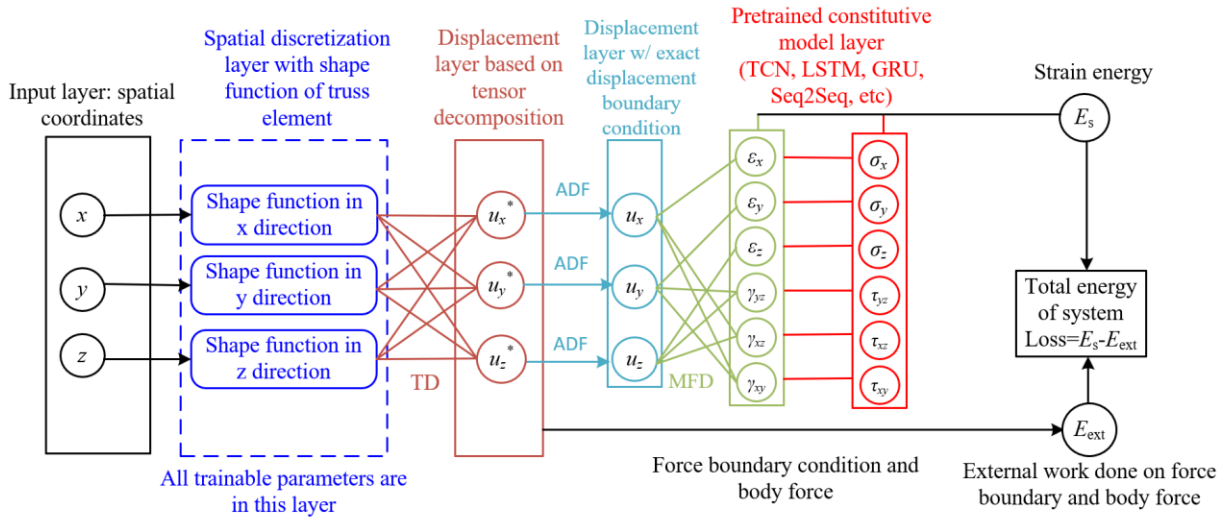


Figure 2. Flowchart of EPINN framework for solving static solid mechanics problems based on the principle of least work (Note: TD denotes tensor decomposition, ADF denotes approximate distance function, MFD denotes meshless finite difference)

2.1 Reformulating shape function of 1D truss element with neural networks

For the 1D problem of solid mechanics, the shape function of truss element (linear or quadratic) can be exactly reformulated by a few layers of neural networks, where the weights and biases can be calculated based on node coordinates of 1D truss elements as proved by Zhang et al. [30]. EPINN framework adopted

this method for constructing the shape function in the 1D case with convolutional architecture (which is slightly different from the original architecture proposed by Zhang et al. [30]), while tensor decomposition can be used to construct the solution field of displacement in 2D and 3D solid mechanics problems. Figure 3 shows the schematic plot of the shape function of the truss element reformulated by neural networks. As shown in Figure 3(a), the shape function of linear 1D truss element is formulated as follows:

$$N_I(x)=\begin{cases}\dfrac{x-x_{I-1}}{x_I-x_{I-1}}, & x_{I-1}\le x\le x_I\\ \dfrac{x_{I+1}-x}{x_{I+1}-x_I}, & x_I\le x\le x_{I+1}\\ 0, & \text{elsewhere}\end{cases} \tag{1}$$

$$\begin{aligned}&\mathcal{N}_I\left(x;x_{I-1},x_I,x_{I+1}\right)\\&=\text{Relu}\left(\frac{-1}{x_I-x_{I-1}}\text{Relu}\left(-x+x_I\right)+1\right)+\text{Relu}\left(\frac{-1}{x_{I+1}-x_I}\text{Relu}\left(x-x_I\right)+1\right)-1,\end{aligned} \tag{2}$$

$$\begin{aligned}u_I(x)&=\mathcal{N}_I\left(x;x_{I-1},x_I,x_{I+1}\right)u_I\\&=\left(\text{Relu}\left(\frac{-1}{x_I-x_{I-1}}\text{Relu}\left(-x+x_I\right)+1\right)-0.5\right)u_I+\left(\text{Relu}\left(\frac{-1}{x_{I+1}-x_I}\text{Relu}\left(x-x_I\right)+1\right)-0.5\right)u_I\end{aligned} \tag{3}$$

where u_I denotes the nodal solution at node number I, $N_I(x)$ denotes the shape function at node number I, and x_{I-1}, $x_{I,}$ and x_{I+1} are node coordinates. Relu denotes Rectified Linear Unit (ReLU) activation function. When the node coordinates are fixed, the only trainable weights are the nodal displacement u_I.

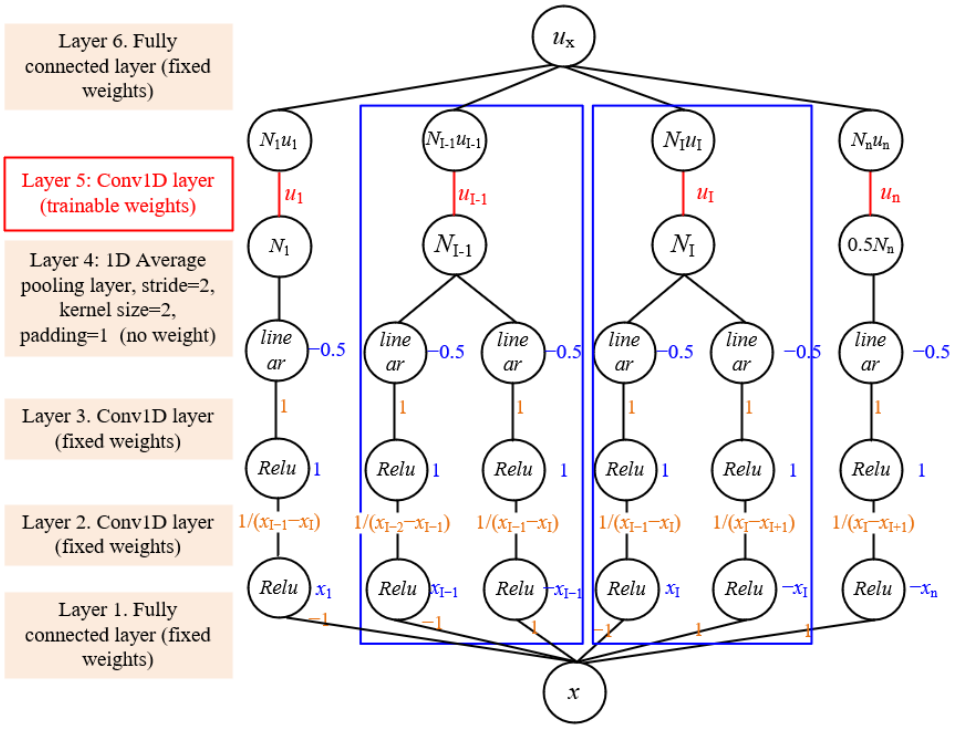
Figure 3 shows the assembly of the shape function in the 1D truss element. Zhang et al. [30] adopted MLP (multi-layer perception) in the HiDeNN model. In this study, it is found that the shape function of the 1D truss element can be reformulated by convolutional neural networks (CNNs). Figure 3(a) shows the neural network architecture to exactly reformulate the shape function of linear truss elements adopting convolutional architecture. As shown in Figure 3(a), the input layer is the x coordinate at an arbitrary point in the solution domain and it is fed into a fully connected layer and two subsequent Conv1D (1D convolutional) layers. Subsequently, Layer No. 4 is a 1D average pooling layer with a stride of 2, kernel size of 2, and padding of 1, which will obtain shape function at various nodes based on Eq. (2). After that, a Conv1D layer is used to obtain the displacement field inside each element as per Eq. (3). Finally, the

element displacements are assembled to obtain the total displacement field of the truss structure. In this architecture, the only trainable parameter is the solution of displacement fields at nodes in Layer No. 5 in Figure 3(a), while all other layers have fixed weights and biases when the coordinates of mesh nodes are fixed. Therefore, the number of trainable parameters in EPINN can be notably reduced to the mesh number in the 1D case compared to conventional PINN models [17-24]. Figure 3(b) shows the neural network architecture to exactly reformulate the shape function of the quadratic truss element. Compared to the linear truss element, the quadratic truss element has an additional shape function at the internal node $N^2_{\mathrm{I}+1/2}(x)$, which can be obtained by multiplication of the linear truss element shape function as follows:

$$N^2_{\mathrm{I}+1/2}(x) = \begin{cases} \dfrac{\left(x - x_{\mathrm{I}}\right)\left(x - x_{\mathrm{I}+1}\right)}{\left(x_{\mathrm{I}+1/2} - x_{\mathrm{I}}\right)\left(x_{\mathrm{I}+1/2} - x_{\mathrm{I}}\right)} & x_I \le x \le x_{I+1} \\ 0 \qquad\qquad \text{elsewhere} \end{cases} \tag{4}$$

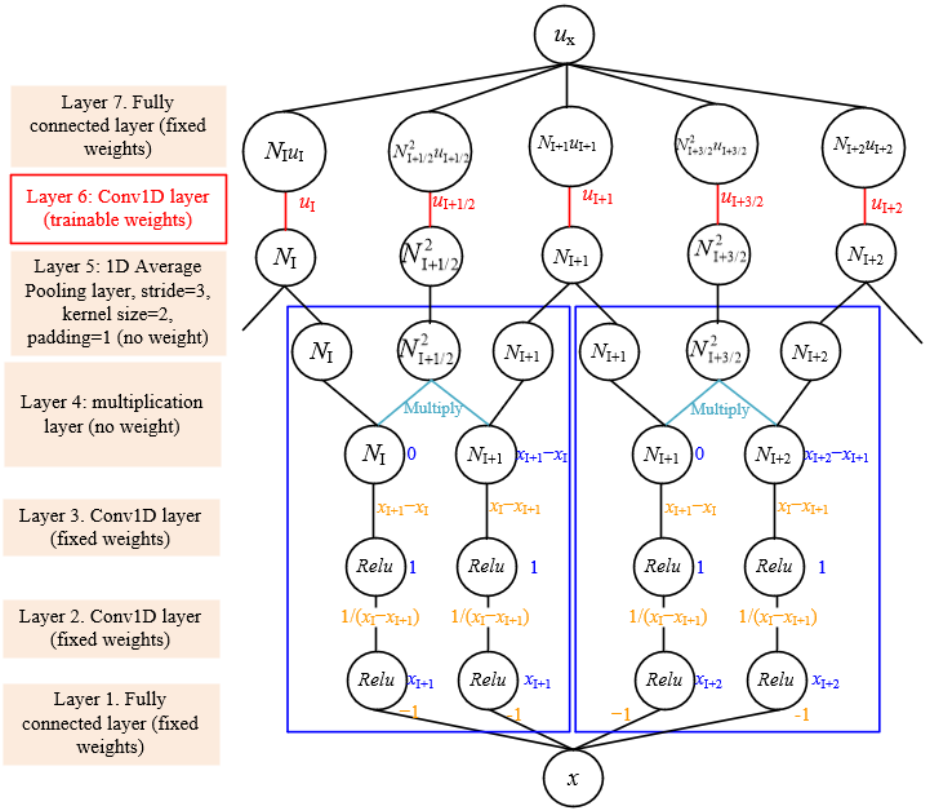
$$= N_{\mathrm{I}}(x) \cdot N_{\mathrm{I}+1}(x) \cdot \frac{\left(x_{\mathrm{I}+1} - x_{\mathrm{I}}\right)^2}{\left(x_{\mathrm{I}+1/2} - x_{\mathrm{I}}\right)\left(x_{\mathrm{I}+1/2} - x_{\mathrm{I}}\right)}$$

As shown in Figure 3(b), the input layer is the *x* coordinate of the node and it is fed into a fully connected layer and two subsequent Conv1D layers. Subsequently, Layer No. 4 is a multiplication layer, where the linear shape function N_{I} and $N_{\mathrm{I}+1}$ are multiplied to obtain the quadratic shape function at the interior point $N^2_{\mathrm{I}+1/2}$ as shown in Eq. (4). Layer No. 5 is a 1D average pooling layer with a stride of 3, kernel size of 2, and padding of 1, which will obtain shape function at various nodes. After that, a Conv1D layer (Layer No. 6) is used to obtain the displacement field inside each element. All trainable weights are in Layer No. 6, which are equivalent to the nodal displacements in FE methods. Finally, the element displacements are assembled to the total displacement field of the truss structure. As shown in Figure 3(b), the shape function of the quadratic truss element can be obtained by multiplication of the linear shape function of the truss element, which can be achieved by the PyTorch and Nvidia Modulus platform [25] by multiplication of tensors to obtain the quadratic shape function of truss elements. Similarly, the shape function of higher-order truss elements (with an order higher than 2) can be formulated in terms of neural networks by multiplication of linear truss elements.



220

221 (a) Neural networks that exactly reformulate the shape function of linear truss elements



222

(b) Neural networks that exactly reformulate the shape function of the quadratic truss element

Figure 3. Shape function of truss element reformulated by neural networks

(Note: orange font denotes fixed weights, blue font denotes fixed bias, red font denotes trainable weights, Relu denotes Rectified Linear Unit, blue box denotes neural networks to reformulate shape function inside the single element, linear denotes no activation function is used in this layer, Conv1D denotes one-dimensional convolutional layer)

2.2 Tensor decomposition for 2D and 3D problems

For 2D and 3D static solid mechanics problems, Zhang et al. [31] proposed to use the tensor decomposition (TD) method in the PINN framework to further reduce the dimension of unknowns. Figure 4 shows the schematic plot of TD represented in terms of neural networks. TD was also known as canonical tensor decomposition [34], which decomposes a tensor as a summation of rank-one tensors. For the 3D case, the governing equations of TD are formulated as the following equations:

$$u_{\mathrm{TD}}(\mathbf{x}) = u_{\mathrm{TD}}(x, y, z) = \sum_{q=1}^{Q} X^{(q)}(x) Y^{(q)}(y) Z^{(q)}(z) \tag{5}$$

$$X^{(q)}(x) = \sum_{I=1}^{n_1} N_I(x) \beta_I^{(q)} \tag{6}$$

$$Y^{(q)}(y) = \sum_{J=1}^{n_2} N_J(y) \gamma_J^{(q)} \tag{7}$$

$$Z^{(q)}(z) = \sum_{K=1}^{n_3} N_K(z) \theta_K^{(q)} \tag{8}$$

where Q denotes the total number of modes (typically ranging from 10 to 100 in numerical study); n_1, n_2, and n_3 denote the number of nodes in the x, y, and z directions, respectively. $\beta_I^{(q)}$, $\gamma_J^{(q)}$ and $\theta_K^{(q)}$ denote the nodal displacement weights in the x, y, and z directions in mode number q.

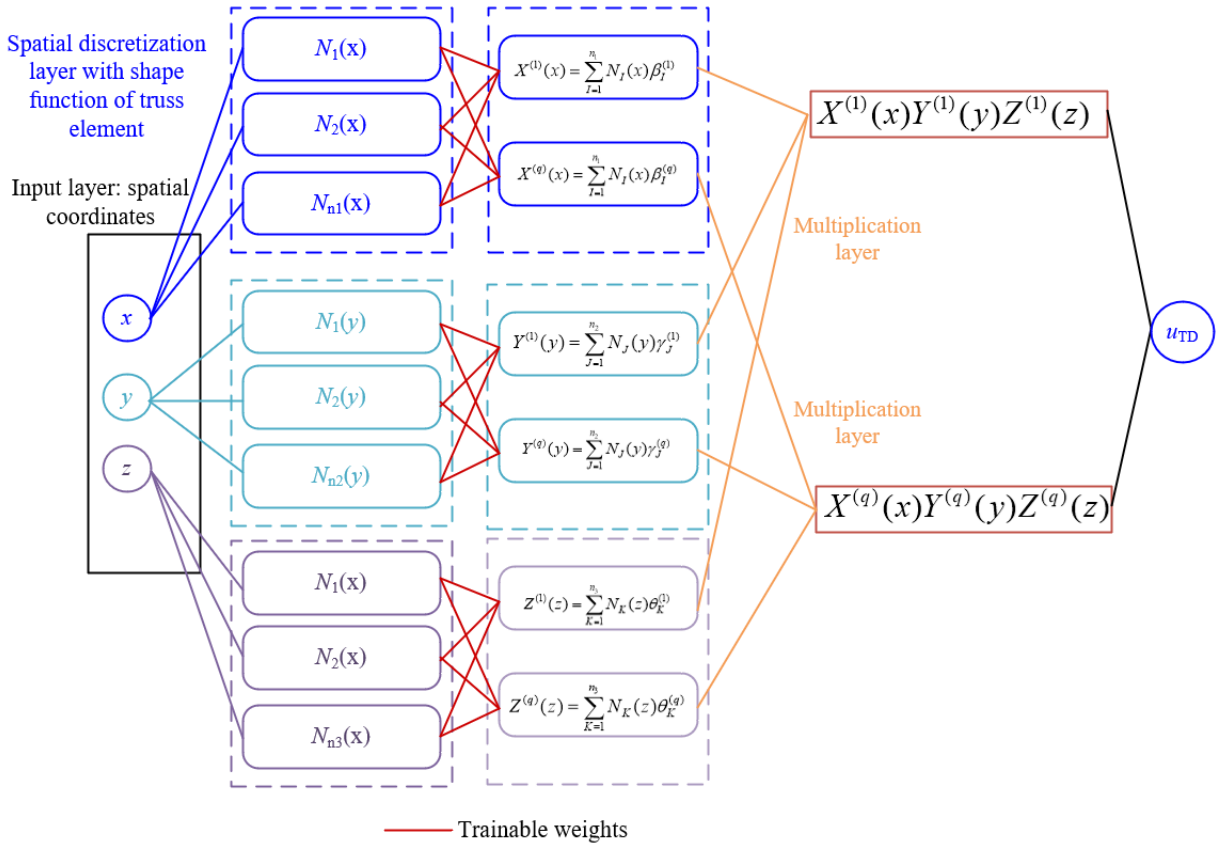


Figure 4. Schematic plot of tensor decomposition formulated by neural networks

There are two motivations for adopting TD in the EPINN framework. First, the number of trainable parameters for 3D solid mechanics problems can be reduced to $Q\times(n_1+n_2+n_3)$, which may be notably lower than the number of DOFs in FE methods (i.e., $n_1\times n_2\times n_3$). Second, TD is efficient for the simulation of complicated shapes in solid mechanics, where the mesh conforming to the shape of the complicated solid body is not needed. TD can be used to generate the solution field of the box containing the simulation region, which notably reduces the difficulty in model generation.

2.3 Principle of least work for the loss function

Consider the solid mechanics problems where the labeled data of the solution field are unavailable (i.e. forward problem), the basic objective in the conventional PINN approach was to minimize the total loss as a sum of boundary loss and the PDE loss as follows (2D plane stress case was illustrated here for simplicity) as shown below. For conventional PINNs, the loss function based on weighted residuals is given as follows:

$$
\begin{aligned}
\mathcal{L} = & \lambda_1 \int_{\partial\Omega_D} \left(\left| u_x - u_x^* \right|_{\Gamma_D} + \left| u_y - u_y^* \right|_{\Gamma_D} \right) d\partial\Omega_D \\
& + \lambda_2 \int_{\partial\Omega_N} \left| \sigma_{xx} cos^2\theta + \sigma_{yy} sin^2\theta + 2\tau_{xy} sin\theta cos\theta - \sigma^* \right| d\partial\Omega_N \\
& + \lambda_2 \int_{\partial\Omega_N} \left| \left(\sigma_{yy} - \sigma_{xx} \right) sin\theta cos\theta + \tau_{xy} \left(cos^2\theta - sin^2\theta \right) - \tau^* \right| d\partial\Omega_N \\
& + \lambda_3 \iint_{\Omega} \left(\left| \sigma_{xx,x} + \sigma_{xy,y} + f_x^* \right| + \left| \sigma_{xy,x} + \sigma_{yy,y} + f_y^* \right| \right) d\Omega \\
& + \iint_{\Omega} \left(\left| (\lambda + 2\mu)\varepsilon_{xx} + \lambda\varepsilon_{yy} - \sigma_{xx} \right| + \left| (\lambda + 2\mu)\varepsilon_{yy} + \lambda\varepsilon_{xx} - \sigma_{yy} \right| + \left| \mu\gamma_{xy} - \tau_{xy} \right| \right) d\Omega
\end{aligned}
\tag{9}
$$

where $\partial\Omega_D$ is the Dirichlet boundary condition (i.e. displacement boundary) of domain Ω, $\partial\Omega_N$ is the Neumann boundary (i.e. force boundary) of domain Ω, u_x^* and u_y^* are the displacement components in x and y direction at displacement boundary condition, σ^* and τ_{xy}^* are the normal stress and shear stress at the force boundary, λ and μ are Lame first and Lame second parameters of elastic material, and λ_1, λ_2, and λ_3 are three coefficients (similar to the Lagrange multiplier in FE methods) to balance the loss term with different units.

The major issues of conventional PINN loss function in Eq. (9) are as follows. First, the loss function consists of a constitutive equation, compatibility equation, and boundary loss, which have different units and needs to be scaled by coefficients λ_1, λ_2, and λ_3 to formulate a reasonable loss function. Although the coefficients λ_1, λ_2, and λ_3 can be updated based on loss balancing approaches such as SoftAdapt [26], ReLoBRaLo [27], and GradNorm [28], there is a lack of physical meaning for the coefficients λ_1, λ_2 and λ_3 in conventional loss balancing approaches [26-28], and the training process involving loss balancing approaches may slow down the convergence of the PINN. Second, because the PDE of static solid mechanics problems are elliptical PDE, the error of boundary condition will propagate to the whole solution field at infinite speed. Therefore, the existence of a boundary condition error may notably hinder the convergence of PINN in solving static solid mechanics problems. Although some approaches were proposed to generate a solution field to meet the Dirichlet boundary condition and Neumann boundary condition simultaneously [15], it requires the PINN to predict the displacement field and stress field

simultaneously, which may increase the number of trainable parameters and may be difficult to extend to nonlinear constitutive models [32, 33].

In this study, the principle of least work is selected as the loss function for static solid mechanics problems, and the loss function is formulated as follows:

$$\mathcal{L}\left(\mathbf{u};\mathbf{f},\bar{\mathbf{t}}\right)=\frac{1}{2}\int_{\Omega}\sigma\left(\mathbf{u}\right):\varepsilon\left(\mathbf{u}\right)d\Omega-\left(\int_{\Omega}\mathbf{u}\cdot\mathbf{f}d\Omega+\int_{\partial\Omega_{\mathrm{N}}}\mathbf{u}\cdot\bar{\mathbf{t}}d\partial\Omega_{\mathrm{N}}\right) \tag{10}$$

$$\mathbf{u}=\mathbf{u}^{*}_{\partial\Omega_{\mathrm{D}}}+\mathbf{u}_{TD}\cdot ADF^{\mathrm{p}} \tag{11}$$

$$\mathbf{u}_{solution}=\mathrm{argmin}\mathcal{L}\left(\mathbf{u};\mathbf{f},\bar{\mathbf{t}}\right)\ \text{ for }\mathbf{u}\left(x,y,z\right)=\mathbf{u}^{*}_{\partial\Omega_{\mathrm{D}}}\ \ \forall\left(x,y,z\right)\in\partial\Omega_{\mathrm{D}} \tag{12}$$

where $\mathbf{u}$ is the displacement field, σ and ε are the stress and strain tensors, respectively, $\mathbf{f}$ is the body force and $\bar{\mathbf{t}}$ is the external traction applied to the force boundary. *ADF* denotes the Approximate Distance Function from any point to the Dirichlet boundary of the simulation region [16], *p* is a positive real number and is fixed as 1.0 in this study. $\mathbf{u}_{TD}$ is the trial displacement field constructed following Eq. (5) in Sect. 2.2, and $\mathbf{u}^{*}_{\partial\Omega_{\mathrm{D}}}$ denotes the displacement field at the Dirichlet boundary condition.

EPINN adopts the loss function as the total energy of the system. Based on the principle of least work, EPINN can efficiently solve solid mechanics problems by minimizing the total work of the system. The exact displacement boundary condition was required by the principle of least work. The Dirichlet boundary condition is satisfied throughout the training process by introducing Eq. (11), and the Neumann boundary condition and force equilibrium equations can be satisfied after the training process converges. Compared to the Deep Ritz method [13], HiDeNN [29, 30], and HiDeNN-TD [31], the exact Dirichlet boundary condition is enforced in the EPINN framework, which conforms to the requirement of the principle of least work. In addition, when calculating the numerical integration, the number and coordinates of integration points may differ from the mesh size. It is recommended that the number of integration points should exceed the number of mesh size in the EPINN framework to evaluate the integration result efficiently.

2.4 Exact Dirichlet boundary condition

The exact Dirichlet boundary condition is achieved by implementing the following ADF, which was

proposed [16] as an alternative to the signed distance function (SDF). ADF is 0 on the Dirichlet boundaries and the first-order derivative along the normal direction of the boundary equals 1. For complicated boundary shapes consisting of various surfaces, it may be too hard to compute the derivative of SDF with respect to coordinates and the derivative may be stiff, which may hinder the convergence of PINN. In comparison, ADF is a second-order smooth function with respect to the coordinate, which is favorable for the stochastic gradient descent method to minimize the loss function in Eq. (10). When there are multiple Dirichlet boundaries, the following equation is used to obtain the ADF of the whole system to achieve an exact Dirichlet boundary condition. This approach ensures the displacement field is exactly met on the Dirichlet boundary.

$$ADF = \phi(\phi_1, ..., \phi_n) := \frac{1}{\sqrt[m]{\frac{1}{(\phi_1)^m} + \frac{1}{(\phi_2)^m} + ... + \frac{1}{(\phi_n)^m}}} \tag{13}$$

where ϕ_i denotes the approximate distance from the point (x,y,z) to boundary number i. On the displacement boundary, $\phi=0$. m is a positive number and is selected as 2.0 in this study.

2.5 Meshless finite difference for efficient gradient information

Conventional PINN architecture mostly adopts automated differentiation to obtain the gradient of displacement with respect to coordinates to formulate the strain field. However, the strain localization in some solid mechanics problems may induce notably high gradient information and may hinder the convergence of conventional PINN models. Recently, meshless finite differentiation (MFD) was developed in the Nvidia Modulus platform (v22.07) [25] and numerical examples show that MFD can achieve a speedup of more than 50% based on conventional PINN architecture [25] on typical mechanics problems. EPINN implemented MFD in the framework as an alternative to automated differentiation. The motivations are summarized below. First, in solid mechanics problems, the stress localization effect may occur, while automated differentiation might induce notably high derivatives in the PINN framework, while MFD may mitigate the influence of stress localization by pre-definition of increments in each direction. Second, because the shape function of the 1D truss element requires the mesh size to be input as a parameter, EPINN

has the concept of mesh size in each direction, which is notably different from conventional PINN models without an explicit definition of the mesh size. Therefore, when MFD is adopted, the spatial increment of MFD is selected to be equal to the smallest mesh size in three directions in this study.

2.6 Comparison between EPINN with existing PINN architecture

The advantages of the EPINN model compared to the recurrent networks may be summarized as:

(1) Improved training speed due to reduced number of trainable weights: In conventional PINN architecture, the number of trainable weights required to converge may be very high (exceeding 1 million trainable weights for many problems), which may notably hinder the solution of solid mechanics problems. In EPINN, based on tensor decomposition and shape function of the truss element, the number of trainable weights can be notably reduced and training speeds are notably reduced accordingly.

(2) Extension to nonlinear problems: The EPINN model resembles the FE approach because only the displacement field is obtained from neural networks, while the strain fields are obtained from MFD of the displacement field, and the stress field is the output of the strain field. Because deep-learning-based constitutive models [32] are rapidly developing, the pre-trained deep-learning-based constitutive models [32] can be infused into EPINN architecture to achieve nonlinear simulation of solid mechanics problems.

3. Performance of EPINN Framework for Solid Mechanics

In this section, three typical cases are illustrated for solving a 1D truss problem, a 2D plane stress problem, and a 3D solid mechanics problem for the application of EPINN without the need for labeled data (which is the case for the forward problem). This study successfully implemented EPINN with Nvidia Modulus [25] deep-learning platform, which is open-source deep-learning software installed on the Osaka University supercomputer SQUID. All models are trained using Intel Xeon Platinum 8368 CPU with a single Nvidia A100 40GB SXM GPU accelerator at Osaka University SQUID (Supercomputer for Quest to Unsolved Interdisciplinary Datascience). The reference PINN architecture adopted a fully-connected neural network with 6 layers and a hidden layer size of 512 is used without skip connections. For PINN in

the reference group, the loss function adopts the conventional PINN approach. The initial learning rate is 0.01 and the training precision was completed on TF32 precision [17] for both EPINN and PINN approaches. The reference FE simulation results in Sect. 3.2 and Sect. 3.3 was obtained using a single Intel i7-10870H CPU using the double precision FE solver without GPU acceleration.

3.1 One-dimensional truss under complicated body force

Consider a 1D elastic truss problem reported by Zhang et al. [30] with an elastic modulus of 175, area of 1, and length of 10 under complicated body force $b(x)$ and fixed at both ends as shown in Figure 5(a):

$$b(x) = -\frac{4\pi^2(x-2.5)^2 - 2\pi}{e^{\pi(x-2.5)^2}} - \frac{8\pi^2(x-7.5)^2 - 4\pi}{e^{\pi(x-7.5)^2}} \tag{14}$$

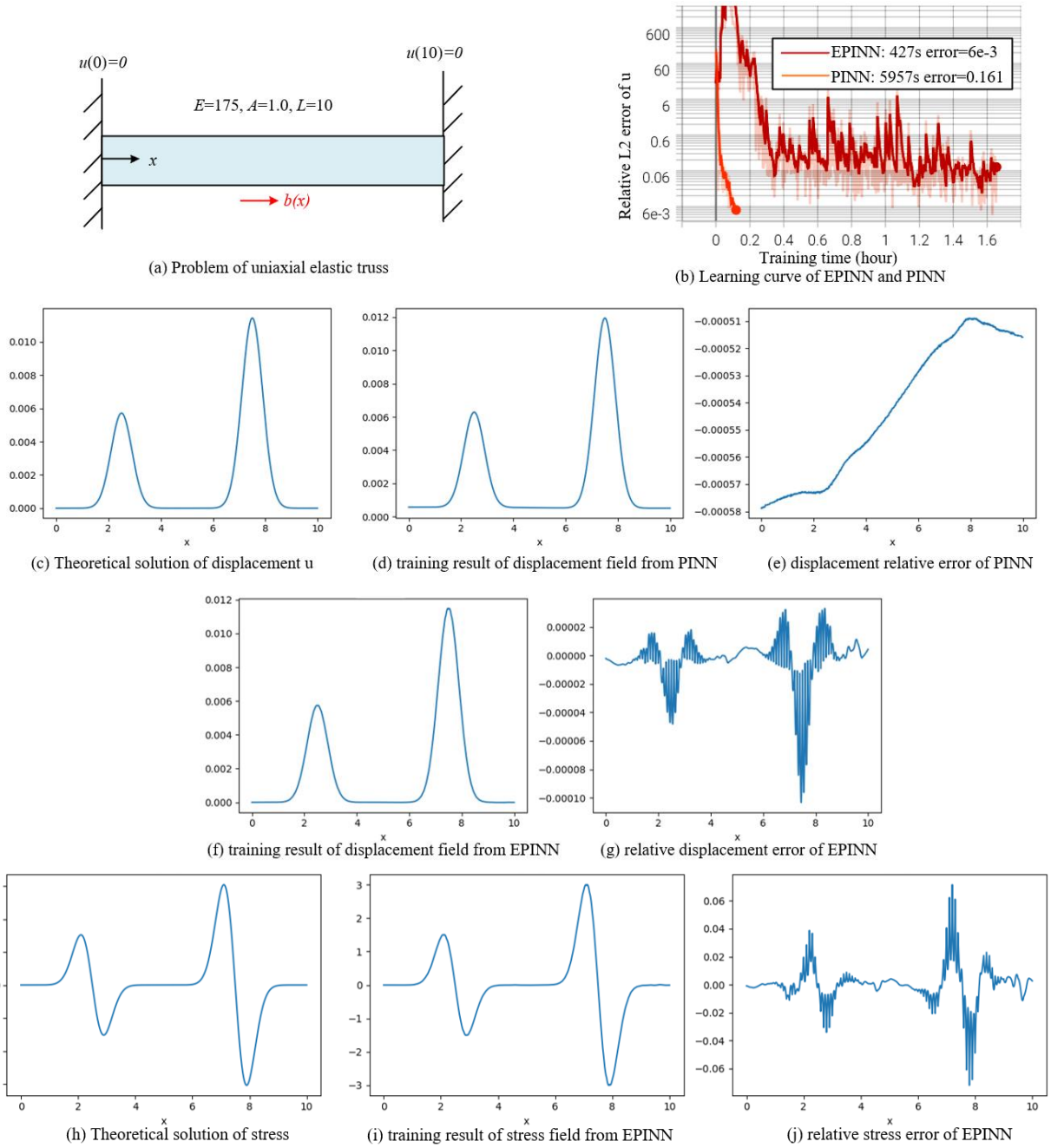
The theoretical solution to this 1D problem is derived by Zhang et al. [30] as follows:

$$u(x) = \frac{1}{AE}\left(e^{-\pi(x-2.5)^2} - e^{-6.25\pi}\right) + \frac{2}{AE}\left(e^{-\pi(x-7.5)^2} - e^{-56.25\pi}\right) - \frac{e^{-6.25\pi} - e^{-56.25\pi}}{10AE}x \tag{15}$$

$$\sigma_{xx} = E\frac{du}{dx} = \frac{2}{A}\left(-\pi e^{-\pi(x-2.5)^2}(x-2.5)\right) + \frac{4}{A}\left(-\pi e^{-\pi(x-7.5)^2}(x-7.5)\right) - \frac{e^{-6.25\pi} - e^{-56.25\pi}}{10A} \tag{16}$$

For PINN architecture, maximum training steps are selected as 2,000,000 steps, and the learning rate decays at a decay rate of 0.95 for every 1,500 steps. For EPINN architecture, the maximum number of training steps is selected as 50,000 steps, and the learning rate decays at a decay rate of 0.95 for every 500 steps. The mesh size was selected as 0.1 and the total number of mesh is 100 for EPINN. The shape function of the quadratic truss element was adopted in EPINN. The comparison of the learning curve (i.e., relative L2 error between theoretical solution and PINN solution) is shown in Figure 5(b). Based on training results, the EPINN achieved convergence after 427 seconds of training with a displacement relative L2 error of 6e-3, while conventional PINN with fully-connected architecture requires a significant training time of 5957 seconds to reach a displacement relative L2 error of 0.161. Therefore, EPINN achieved a speedup of more than 13 times compared to conventional PINN for this truss problem. Figure 5(c-g) further compares the simulation results of the axial displacement field obtained from PINN and EPINN. Conventional PINN may observe a boundary condition error with non-zero displacement at both ends, while the EPINN result did not show this issue due to the exact Dirichlet boundary condition in the EPINN framework. Figure 5(h-

j) compares the stress field predicted by EPINN, which also converged with a slightly larger relative error compared to the displacement field. Based on the comparison in Figure 5, EPINN achieved an efficient solution of a 1D truss under complicated body force.



(a) Problem of uniaxial elastic truss

(b) Learning curve of EPINN and PINN

(c) Theoretical solution of displacement u

(d) training result of displacement field from PINN

(e) displacement relative error of PINN

(f) training result of displacement field from EPINN

(g) relative displacement error of EPINN

(h) Theoretical solution of stress

(i) training result of stress field from EPINN

(j) relative stress error of EPINN

Figure 5. Comparison of EPINN and PINN for simulating 1D truss problem

3.2 Plane stress panel under eccentric tension

Rao et al. [15] reported a benchmark FE model and PINN architecture for the plane-stress panel. Figure 6(a) shows the FE simulation results using ABAQUS software as well as the boundary conditions. The panel is elastic material with an elastic modulus of 10 and a Poisson ratio of 0.2. The panel length and height are both 1.0 and the bottom line was fixed. The left half of the top surface was subject to vertical tension displacement u_y of 0.1 while the horizontal displacement u_y is 0. All other lines are stress-free. ABAQUS implicit solver was used to obtain the solution at 160,000 reduced-integration plane stress elements at a mesh size of 1/400 and the large displacement option was turned on. For conventional PINN with a fully connected network, maximum training steps are selected as 20,000 steps and the learning rate is decreasing by a decay rate of 0.95 for every 2,000 steps. For EPINN architecture, the maximum training steps are selected as 30,000 steps and the learning rate is decreasing by a decay rate of 0.95 for every 500 steps. EPINN in this section adopts a mesh size of 1/50, mode number Q of 10 for tensor decomposition, and first-order shape function of the truss element. Figure 6(b) shows the time history L2 relative error of displacement u_x field from PINN and EPINN models compared to ABAQUS output. The ABAQUS solver converged after 265 seconds of simulation time (using a single Intel i7-10870H CPU). In comparison, the EPINN solution also converged to ABAQUS simulation results after a training time of 253s (using a single Nvidia A100 GPU accelerator), which is even faster than the ABAQUS model. In comparison, conventional PINN architecture did not converge after 5957s of training. Figure 6(c-e) shows the comparison between EPINN training results and FE simulation outcome of the displacement u_x field. Figure 6(f-h) shows the comparison between EPINN training results and FE simulation outcome of the displacement u_y field. Based on the comparison, the EPINN model with the mesh size of 1/50 well captures the displacement field of FE simulation, which was obtained at a finer mesh size of 1/400. Therefore, the applicability and accuracy of EPINN for 2D plane stress cases are examined based on this case study.

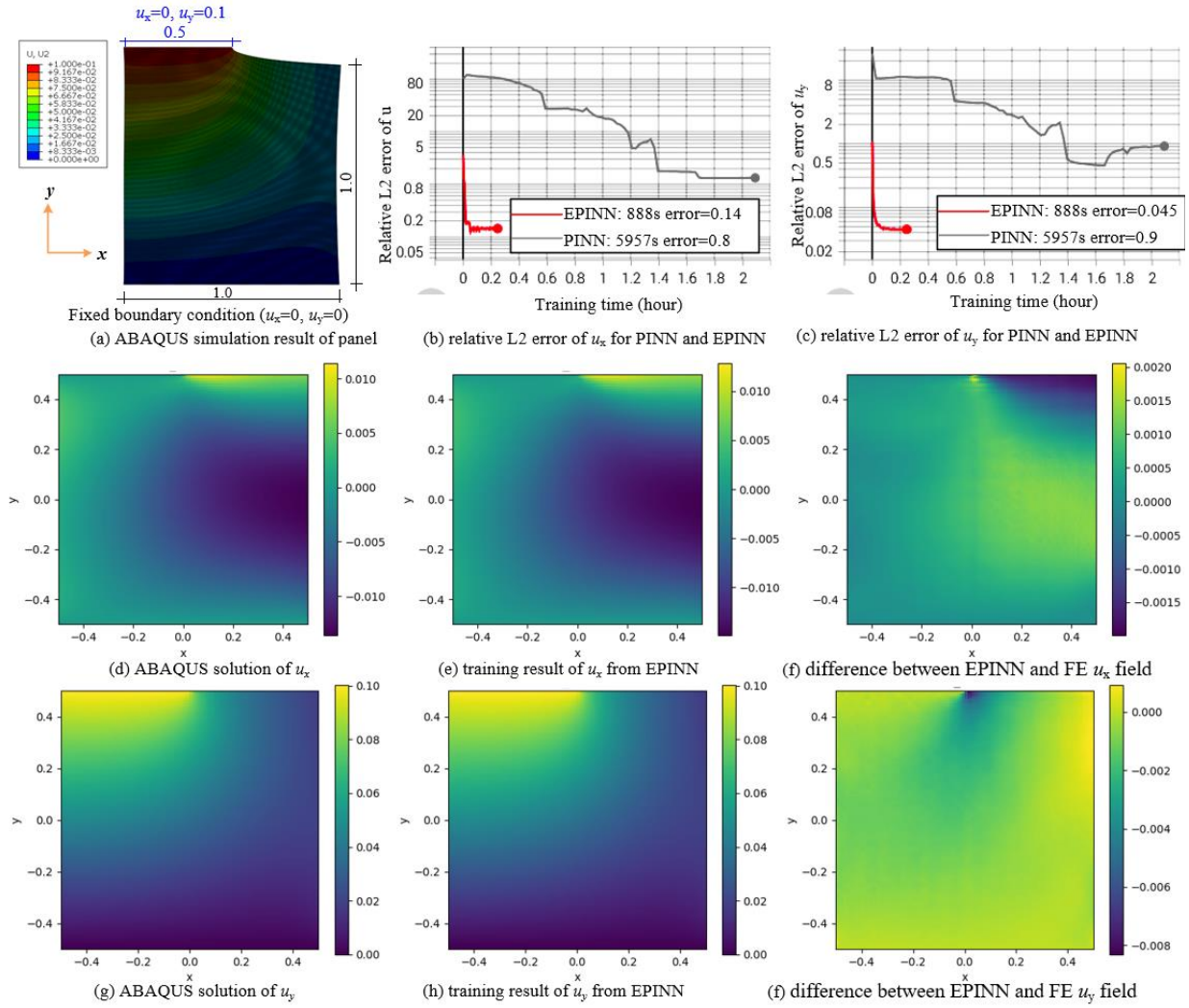
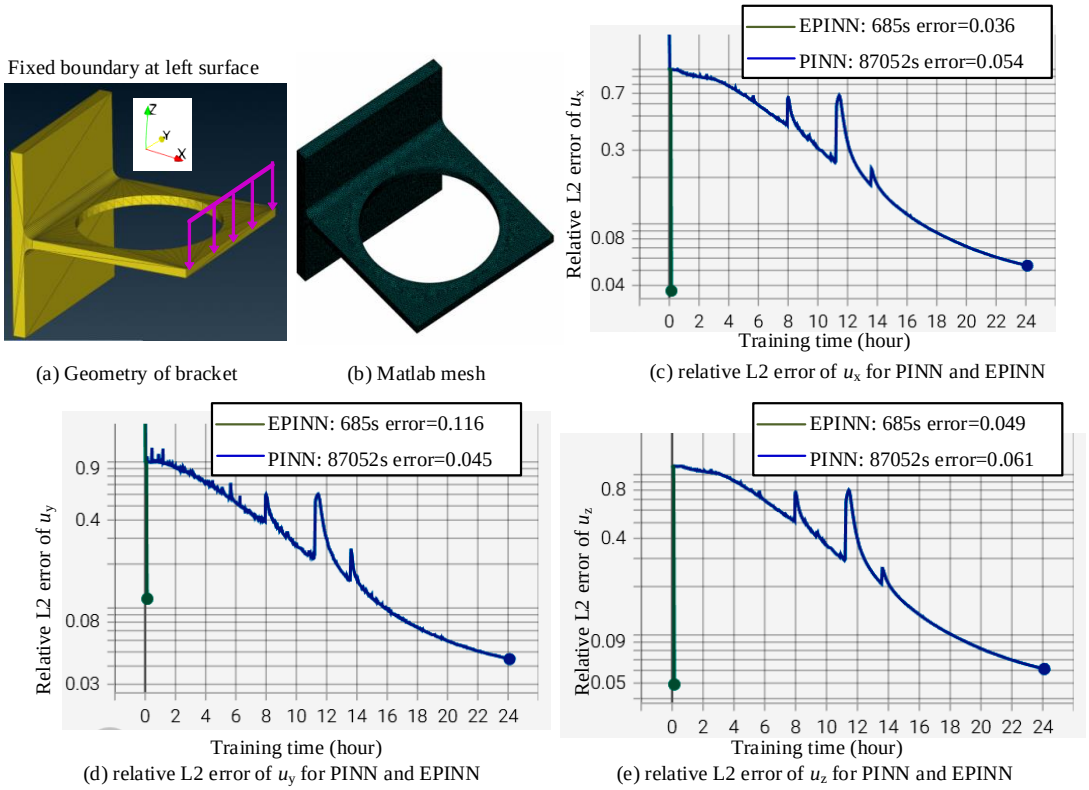


Figure 6. Comparison of EPINN and PINN for simulating plane stress panel

3.3 Three-dimensional bracket

In this section, a three-dimensional problem is illustrated to show the performance of EPINN in solid mechanics. Figure 7(a) shows the schematic plot of the bracket. The bracket has the back face fixed and the shear stress is applied to the front surface in the negative z direction. The traction force induces shear stress of 0.4 MPa and the rest of the bracket was stress-free boundaries. The height, weight, and length of the bracket are equal to 1.0 m. The elastic modulus is 100GPa and the Poisson ratio is 0.3. Figure 7(b) shows the finite element mesh in MATLAB using quadratic tetrahedral elements with a mesh size of 0.003. A

total of 1,157,983 nodes are generated in the MATLAB FE model, and the linear solution of this reference FE model costs a total of 1140s on CPU. For conventional PINN with a fully connected network (6 layers, 512 neurons each layer, no skip connection), maximum training steps are selected as 2,000,000 steps and the learning rate is reduced by a decay rate of 0.95 every 15,000 steps. For EPINN architecture, the maximum training steps are selected as 10,000 steps and the initial learning rate is set as 0.001. A total of 25 meshes are used for each direction (i.e. mesh size of 0.04) and the number of modes is set to 20 for EPINN. The shape function of the linear truss element was used for EPINN. Figure 7(c-e) shows the relative L2 error of displacement components of PINN and EPINN compared to MATLAB FE results. As shown in Figure 7(c-e), EPINN rapidly converges to the MATLAB FE model within 685s. In comparison, conventional PINN with fully-connected architecture converged to a similar level of accuracy after 87052s of training. Therefore, EPINN achieved a speedup of 127 times compared to conventional PINN in this 3D solid example.

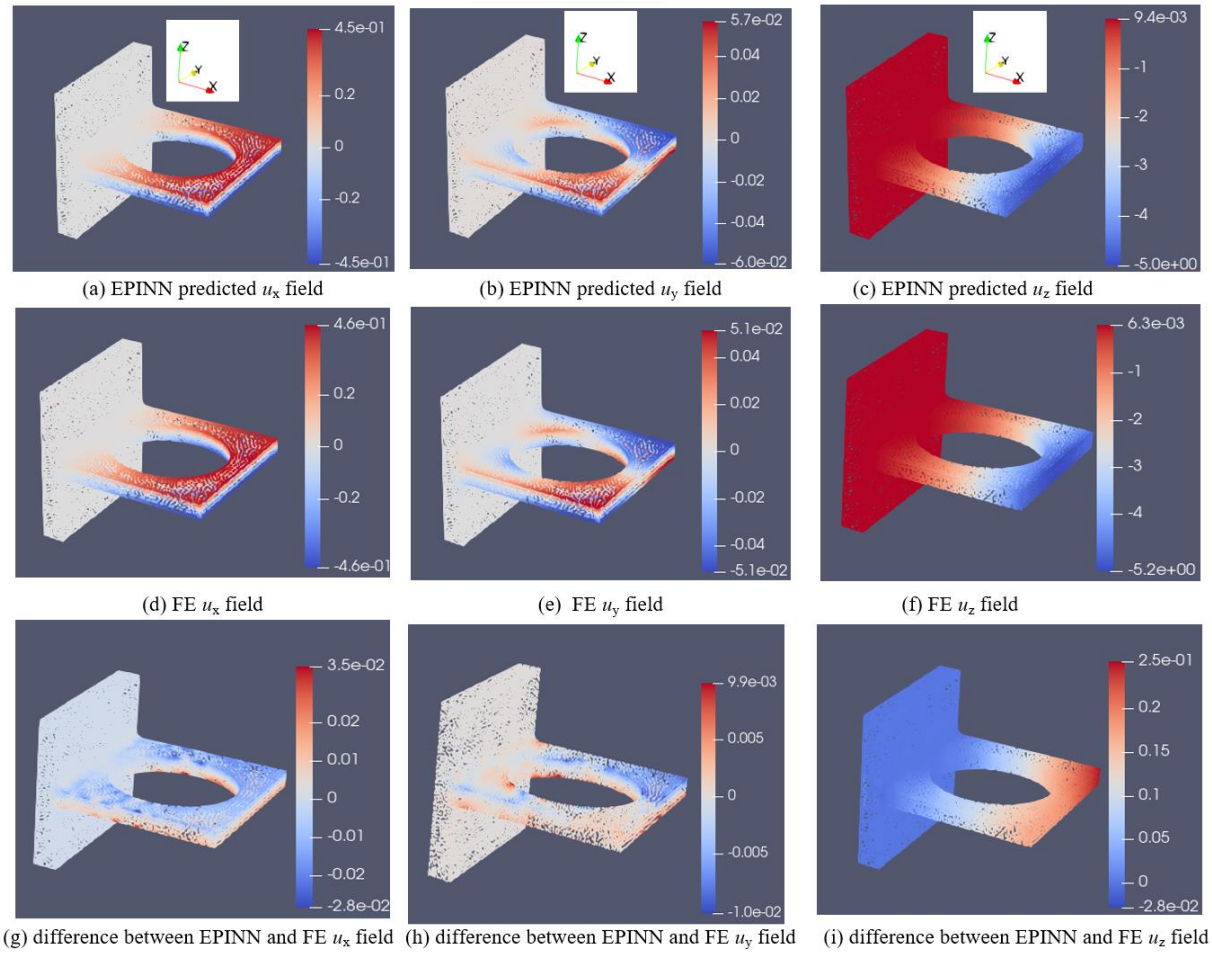


(a) Geometry of bracket (b) Matlab mesh (c) relative L2 error of u_x for PINN and EPINN

(d) relative L2 error of u_y for PINN and EPINN (e) relative L2 error of u_z for PINN and EPINN

Figure 7. Comparison of EPINN and PINN for simulating 3D bracket under uniform force boundary

Figure 8 shows the comparison of displacement simulation outcomes on the 3D point cloud. Figure 8(a-c) shows the EPINN simulation results of the displacement field in the x, y, and z directions. Figure 8(d-e) shows the FE analysis results. Figure 8(f-h) shows the relative error defined as EPINN simulation results minus the FE analysis results. As shown in Figure 8, EPINN well captured the simulation results of FE results even with the mesh size of 0.04. In this study, EPINN trained on Nvidia A100 GPU achieved faster convergence compared to FE analysis and achieved more than 100 times speedup compared to the conventional PINN method. In general, EPINN rapidly converges to the solution field of FE simulation results with similar computational time for the forward problem. Compared to FE analysis, the gradient of the solution field with respect to input parameters can be obtained in real-time, which is favorable for inverse problems and design problems.



(a) EPINN predicted u_x field (b) EPINN predicted u_y field (c) EPINN predicted u_z field

(d) FE u_x field (e) FE u_y field (f) FE u_z field

(g) difference between EPINN and FE u_x field (h) difference between EPINN and FE u_y field (i) difference between EPINN and FE u_z field

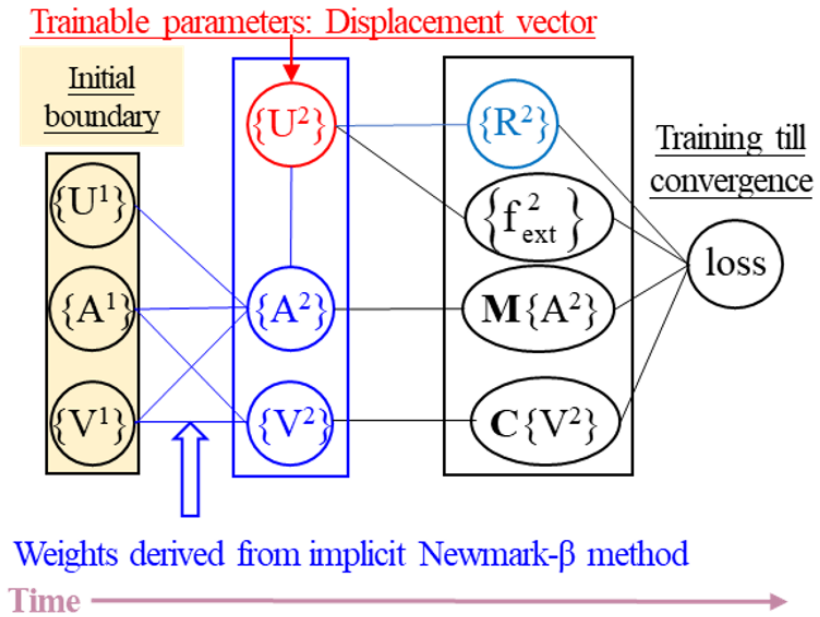
Figure 8. Comparison of EPINN and PINN for simulating 3D bracket under uniform force boundary (in

422 units of mm)

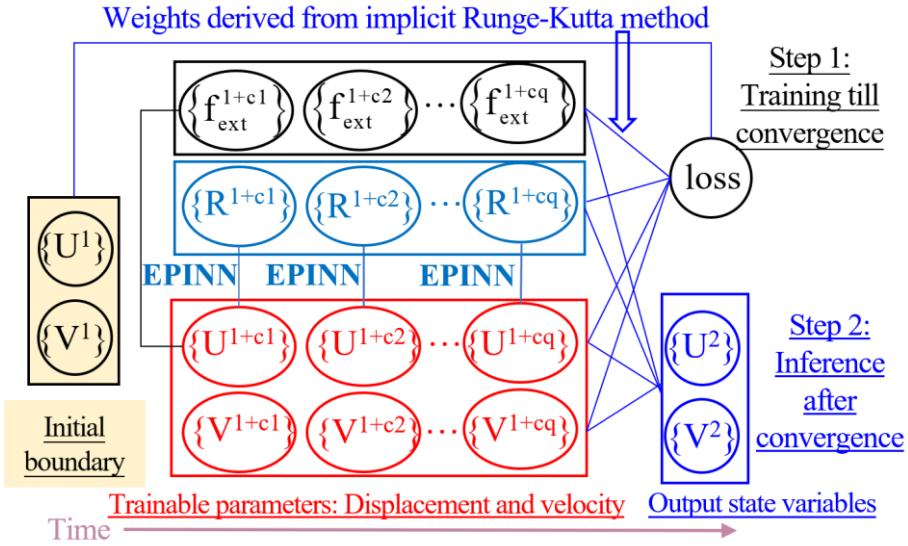
423 # 4. DISCUSSION AND EXTENSIONS

424 ## 4.1 Extension to nonlinear mechanical systems

425 The principle of least work is adopted in EPINN for solving static solid mechanics problems. For
426 dynamic problems, EPINN can also be further extended to solve the nonlinear static analysis and nonlinear
427 dynamic analysis problems with efficient access to gradient information. In dynamic analysis, FE software
428 based on implicit solvers generally adopts the implicit Newmark-β method with second-order accuracy,
429 while high-order (typically fourth-order) implicit Runge-Kutta methods are also adopted in some FE
430 software. Figure 9(a) shows the proposed EPINN to reformulate the implicit Newmark-β method with
431 second-order accuracy. The input layer contains the initial boundary condition, and the trainable parameters
432 are displacement vectors $\{U^2\}$ to be solved in Step 2. The input layer and $\{U^2\}$ will be fed to neural networks
433 to obtain the acceleration $\{A^2\}$ and velocity $\{V^2\}$. The weights of the neural network will be derived from
434 the Newmark-β method. Subsequently, the internal force $\{R^2\}$ will be updated based on $\{U^2\}$ based on the
435 EPINN framework. Because this step is a static problem, it can be solved by training EPINN. The loss
436 function is formulated as the total error of the force equilibrium equation, which will be minimized to solve
437 displacement $\{U^2\}$. Figure 9(b) shows the proposed EPINN architecture to reformulate the alternative high-
438 order implicit Runge-Kutta method with arbitrary q stages [9], which can increase stable time increments
439 and exploit the parallel computing capability of GPUs by parallel internal force updating at q stages. The
440 trainable parameters are displacement and velocity at q stages. The internal force vectors at q stages will be
441 updated based on displacement vectors in parallel using GPUs. The loss function will be formulated
442 following the implicit Runge-Kutta method [35] and the optimization function can be used to solve the
443 displacement and velocity at q stages. Finally, the implicit Runge-Kutta method will infer the displacement
444 $\{U^2\}$ and velocity $\{V^2\}$ at the end of the multi-stage step. For both methods, the gradient of output fields
445 with respect to major input parameters (such as dimensions of components or external loads) can be
446 obtained using the automated backpropagation of the neural network efficiently.



(a) EPINN to reformulate implicit Newmark-β method



(b) EPINN to reformulate implicit Runge-Kutta method with *q* stages

Figure 9. Proposed EPINN architecture to reformulate implicit time integration methods for dynamic analysis

In addition, causal training [37] has also been proposed in the PINN research field. Causal training is another method to ensure the continuous-time PINN models obey causality. This method discretized the time domain and defined the temporal residual loss. The temporal residual loss ensures the training process will first train the PINN for initial short time period, and the training of future time will only start upon convergence of previous time steps. Therefore, the future development includes extending EPINN to

include the causal training technique to extend to nonlinear problems.

4.2 Extension to Super-resolution EPINN (SPINN)

In the computational mechanics research field, the super-resolution networks [8, 36] were also rapidly adopted in turbulence super-resolution. In addition, the Fourier Neural Operator (FNO) also achieves zero-shot super-resolution performance [11]. This study proposes to infuse EPINN with a super-resolution network to achieve high-fidelity PINN with an acceptable computational cost. The motivation is that EPINN has a mesh size of *x*, *y*, and *z* direction. The mesh size needs to be fixed before training. After training is complete, when finer results of the solution field are needed, a super-resolution network can be adopted based on the following steps:

(1) Develop solid mechanics datasets including FE analysis data, test data, and EPINN simulation data.

(2) Pre-train super-resolution networks [8, 36] based on solid mechanics dataset.

(3) For specific solid mechanics problems, use EPINN to solve solid mechanics problems with low-resolution grid size.

(4) Based on the solution of EPINN at a low-resolution grid, use a pre-trained super-resolution network to infer a high-resolution solution field.

5. CONCLUSIONS

To achieve efficient simulation of solid mechanics problems with reduced computational cost, the EPINN framework is proposed in this study, which adopts the exact Dirichlet boundary condition and principle of least work for the simulation of solid mechanics problems. There is no requirement for additional labeled data of the solution field and notably reduces the training cost compared to recurrent networks. Even when no labeled data of the solution field are input, the EPINN architecture can directly solve the solid mechanics problems by minimizing the PDE loss. The future extensions of the EPINN model into nonlinear dynamic simulation and super-resolution networks are also discussed. (5) Because EPINN is a special version of PINN, it can be extended to parametric simulation, structural health monitoring

problems, and design optimization problems with higher flexibility compared to conventional finite element methods. The major conclusions are summarized as follows:

(1) The EPINN framework adopts the principle of least work and exactly meets the Dirichlet boundary condition throughout the training process, while the Neumann boundary condition and governing equations (PDEs) are met when the training converges. Based on the shape function of the 1D truss element and tensor decomposition theory, the number of trainable weights can be reduced notably compared to conventional PINN architecture. The number of trainable weights can be even smaller than the DOFs in the FE method at the same spatial resolution.

(2) For the 1D truss problem, EPINN achieved a speedup of 13 times compared to conventional PINN architecture adopting a fully connected neural network in the case study with notably reduced error.

(3) For the 2D plane stress problem, EPINN rapidly converged to ABAQUS FE simulation results with a similar solution time compared to ABAQUS implicit solver while conventional PINN did not converge for the same problem.

(4) For the 3D plane stress problem, EPINN efficiently converged to FE simulation results with lower training time than the FE solver. EPINN achieved a speedup of 127 times compared to the conventional PINN model with a reasonable level of accuracy.

CONFLICT OF INTEREST

The authors declare no conflict of interest.

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the financial support provided by the JSPS (Japan Society for the Promotion of Science) postdoctoral fellowship and Grant-in-Aid for JSPS Fellows. The authors gratefully acknowledge the support provided by Osaka University Supercomputer SQUID (Supercomputer for Quest to Unsolved Interdisciplinary Datascience).

DATA AVAILABILITY STATEMENT

All data, models, or codes that support the findings of this study are available from the corresponding author upon reasonable request, including the datasets, codes for implemented models, and pre-trained models developed in this research.

REFERENCES

[1] G.E. Karniadakis, I.G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, L. Yang, Physics-informed machine learning, Nature Reviews Physics, 3 (2021) 422-434.

[2] J.-J. Wang, C. Liu, X. Nie, J.-S. Fan, Y.-J. Zhu, Nonlinear model updating algorithm for biaxial reinforced concrete constitutive models of shear walls, Journal of Building Engineering, (2021) 103215.

[3] J.-J. Wang, Y.-F. Liu, X. Nie, Y.L. Mo, Deep convolutional neural networks for semantic segmentation of cracks, Struct. Control. Health Monit., (2021) 1-18.

[4] A. Yazdani, L. Lu, M. Raissi, G.E. Karniadakis, Systems biology informed deep learning for inferring parameters and hidden dynamics, PLoS Comput Biol, 16 (2020) e1007575.

[5] R. Rombach, A. Blattmann, D. Lorenz, P. Esser, B. Ommer, High-resolution Image Synthesis with Latent Diffusion Models, in: CVPR 2022, 2022.

[6] K. Simoyan, A. Zisserman, Very Deep Convolutional Networks for Large-scale Image Recognition, in: Proc. International Conference on Learning Representations (ICLR 2015), San Diego, CA, 2015.

[7] S. Bai, J.Z. Kolter, V. Koltun, An empirical evaluation of generic convolutional and recurrent networks for sequence modeling, in, 2018.

[8] N.S. Ashish Vaswani, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, Illia Polosukhin, Attention is all you need, in: Proc. Advances in Neural Information Processing Systems 30 (NIPS 2017), Long Beach, CA, 2017.

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

[10] N. Long, R. Maziar, P. Seshaiyer, Efficient Physics Informed Neural Networks Coupled with Domain Decomposition Methods for Solving Coupled Multi-physics Problems, Advances in Computational Modeling and Simulation, (2022) 41-53.

[11] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, A. Anandkumar, Fourier Nerual Operator for Parametric Partial Differential Equations, in: International Conference on Learning Representations (ICLR), 2021.

[12] K. Hornik, M. Stinchcombe, H. White, Multilayer feedforward networks are universal approximators, Neural Networks, 2 (1989) 359-366.

[13] W. E, B. Yu, The Deep Ritz Method: A Deep learning-based numerical algorithm for solving variational problems, Communications in Mathematics and Statistics, 6 (2018) 1-12.

[14] E. Haghighat, M. Raissi, A. Moure, H. Gomez, R. Juanes, A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics, Computer Methods in Applied Mechanics and Engineering, 379 (2021).

[15] C. Rao, H. Sun, Y. Liu, Physics informed deep learning for computational elastodynamics without labeled data, Journal of Engineering Mechanics, 147 (2021).

[16] N. Sukumar, A. Srivastava, Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks, Computer Methods in Applied Mechanics and Engineering, 389 (2022).

[17] S.N. Oliver Hennigh, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, Sanjay Choudhry NVIDIA SimNet™: An AI-Accelerated Multi-Physics Simulation Framework, in: Proc. International Conference on Computational Science, 2021, 447-461.

[18] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F.A. Hamprecht, Y. Bengio, A. Courville, On the spectral bias of neural networks, in: 36th International Conference on Machine Learning, California, 2019.

[19] S. Wang, Y. Teng, P. Perdikaris, Understanding and mitigating gradient flow pathologies in physics-informed neural networks, SIAM Journal on Scientific Computing, 43 (2021) 3055-3081.

[20] R.K. Srivastava, K. Greff, J. Schmidhuber, Training very deep networks, in: Advances in neural information processing systems 2015.

[21] S. Wang, H. Wang, P. Perdikaris, On the eigenvector bias of fourier feature networks: From regression to solving multi-scale pdes with physics-informed neural networks., Computer Methods in Applied Mechanics and Engineering, 384 (2021).
[22] V. Sitzmann, J.N.P. Martel, A.W. Bergman, D.B. Lindell, G. Wetzstein, Implicit neural representations with periodic activation functions, in: Proc. 34th Conference on Neural Information Processing Systems (NeurIPS 2020, Vancouver, Canada, 2020.
[23] J. Sirignanoa, K. Spiliopoulosb, DGM: A deep learning algorithm for solving partial differential equations, Journal of Computational Physics, 375 (2018) 1339-1364.
[24] R. Fathony, A.K. Sahu, D. Willmott, J.Z. Kolter, Multiplicative filter networks, in: International Conference on Learning Representations (ICLR) Vienna, Austria, 2021.
[25] Nvidia, https://docs.nvidia.com/deeplearning/modulus/index.html, in, 2022.
[26] A.A. Heydari, C.A. Thompson, A. Mehmood, SoftAdapt: Techniques for Adaptive Loss Weighting of Neural Networks with Multi-Part Loss Functions, Preprint, (2019).
[27] R. Bischof, M. Kraus, Multi-Objective Loss Balancing for Physics-Informed Deep Learning, Preprint, (2021).
[28] Z. Chen, V. Badrinarayanan, C.-Y. Lee, A. Rabinovich, GradNorm: Gradient Normalization for Adaptive Loss Balancing in Deep Multitask Networks, in: Proc. 35 th International Conference on Machine Learning, Stockholm, Sweden, 2018.
[29] S. Saha, Z.T. Gan, L. Cheng, J.Y. Gao, O.L. Kafka, X.Y. Xie, H.Y. Li, M. Tajdari, H.A. Kim, W.K. Liu, Hierarchical Deep Learning Neural Network (HiDeNN): An artificial intelligence (AI) framework for computational science and engineering, Computer Methods in Applied Mechanics and Engineering, 373 (2021) 28.
[30] L. Zhang, L. Cheng, H.Y. Li, J.Y. Gao, C. Yu, R. Domel, Y. Yang, S.Q. Tang, W.K. Liu, Hierarchical deep-learning neural networks: finite elements and beyond, Computational Mechanics, 67 (2021) 207-230.
[31] L. Zhang, Y. Lu, S. Tang, W.K. Liu, HiDeNN-TD: Reduced-order hierarchical deep learning neural networks, Computer Methods in Applied Mechanics and Engineering, 389 (2022).
[32] J.-J. Wang, C. Wang, J.-S. Fan, Y.L. Mo, A deep learning framework for constitutive modeling based on Temporal Convolutional Network, Journal of Computational Physics, 449 (2022).
[33] C. Wang, L.-y. Xu, J.-s. Fan, A general deep learning framework for history-dependent response prediction based on UA-Seq2Seq model, Computer Methods in Applied Mechanics and Engineering, 372 (2020) 113357.
[34] T.G. Kolda, B.W. Bader, Tensor decompositions and applications, SIAM Review, 51 (2009) 455-500.
[35] J.C. Butcher, Numerical Methods for Ordinary Differential Equations Third Edition, Wiley, 2016.
[36] S. Anwar, S. Khan, N. Barnes, A deep journey into super-resolution: A survey, ACM Computing Surveys, 53 (2021) 1-34.
[37] Wang, Sifan, Sankaran, Shyam, and Perdikaris, Paris. Respecting causality is all you need for training physics-informed neural networks. arXiv preprint arXiv:2203.07404, 2022.