Learning Structural Dynamics with Graph Neural Networks

Mingyuan Chi Department of Mathematics
Zürich, Switzerland
minchi@student.ethz.ch

***Abstract*—The integration of Finite Element Method (FEM) with Graph Neural Networks (GNN) is gaining traction in addressing complex real-world problems through neural networks. A significant challenge in this domain is managing boundary conditions within FEM frameworks. To advance research in this area, we introduce a pioneering Physical-Informed GNN architecture. This novel design is specifically tailored to accurately model Dirichlet boundary conditions using neural network methodologies. Furthermore, we delve into the relationship between neural networks and the Galerkin method. To this end, we have developed innovative neural network strategies for effectively modeling the Galerkin matrix. The efficacy of our proposed architecture is substantiated through a comprehensive series of experimental validations. This research not only addresses key challenges in FEM-GNN coupling but also opens new avenues for exploring the synergies between computational methods and neural network architectures in solving complex engineering problems.**

I. Introduction

Finite Element Method (FEM) is a predominant approach in Partial Differential Problem(PDE), particularly in structure mechanics problems, due to its precision and capability to handle non-Euclidean topologies. Despite its advantages, FEM is time-consuming and memory-intensive for large-sclae systems, often requiring linear or non-linear solving operatios. With the advancements in Deep Learning, Graph Neural Networks (GNN) have emerged as a promising solution for processing non-Euclidean data. Inspired by Physics-Informed Neural Networks (PINN) [15], the PI-GNN [5] model represents an innovative integration of FEM and GNN, aiming to efficiently process physical loss and non-Euclidean data with more flexibility than traditional grid data in PINN. However, to handle the Dirichlet boundary condition applied to the FEM, it intuitively adopts the static condensation formula to propagate the information from the Dirichlet boundary to the inner part of the solving domain. Which means that the Dirichlet boundary condition is decoupled in this model.

In this study, we focus on linear elasticity problems, exploring both the forward (solve loading force from displacement) and reverse (solve displacement from loading force) processes. Our approach includes adapting the static condensation formula in FEM to intuitively handle Dirichlet boundary conditions by propagating information from the boundary to the domain's interior.

Our key contributions are :

1) Introduction of Static Condensation Equivelant Architecture (SCEA) for modeling the Dirichlet boundary condition in the FEM problem.
2) Developmenet of Galerkin Equivelant Architecture (GEA), which takes several forms including the local pseudo linear, local pseudo bilinear, and global version.
3) Extensive experiementation to analyze the relation between the observation ratio and the precision of the model for various scenarios(invariant, boundary-variant, and force-variant), in which could we visualize the generalization competence for physical loss.
4) Proposal of a fast and differentialble assemble method representing the assemble step in FEM as sparse-dense tensor multiplication.

The source code for this project is publicly available and can be accessed at the following GitHub repository: https://github.com/walkerchi/ETHz-SP.

II. Background

A. Abbreviation

Abbreviation	Description
FEM	**F**inite **E**lement **M**ethod
PDE	**P**artial **D**ifferential **E**quation
DoF	**D**egree **o**f **F**reedom
DBC	**D**irichlet **B**oundary **C**ondition
NBC	**N**euman **B**oundary **C**ondition
NB	**N**on-**B**oundary
MLP	**M**ulti **L**ayer **P**erceptron
GNN	**G**raph **N**eural **N**etwork
GCN	**G**raph **C**onvolutional Neural **N**etwork
GAT	**G**raph **A**ttention Neural **N**etwork
SIGN	**S**calable **I**nception **G**raph Neural **N**etworks
PINN	**P**hysics **I**nformed **N**eural **N**etwork
SCEA	**S**tatic **C**ondenseing **E**quivelant **A**rchitecture
GEA	**G**alerkin **E**quivelant **A**rchitecture
MSE	**M**ean **S**quare **E**rror

TABLE I: Abbreviation Table

B. Symbol Table

C. Finite Element Method

Finite **E**lement **M**ethod is a commonly used method for solving **P**artial **D**ifferential Equation for it's scalability and accuracy. The canonical FEM process takes the steps as followed.

1) First, generate a $\mathcal{M}$ in a given PDE domain Ω

Symbol	Physical/Mathematical Meanning	Shape
d	dimension of the problems	$\mathbb{R}$
$\boldsymbol{\sigma}$	stress	$\mathbb{R}^{d\times d}$
$\boldsymbol{\varepsilon}$	strain, $\boldsymbol{\varepsilon}=\frac{1}{2}(\nabla u+\nabla u^\top)$	$\mathbb{R}^{d\times d}$
u	displacement	$\mathbb{R}^{d}$
$\boldsymbol{n}$	normal unit vector on the boundary	$\mathbb{R}^{d}$
Ω	the PDE domain	
Γ	boundary of domain	$\Gamma\subset\partial\Omega$
$\mathcal{M}$	mesh	$\{\mathcal{V},\mathcal{C}\}$
$\mathcal{V}$	vertices/nodes in the mesh	$\mathbb{N}^{\lvert\mathcal{V}\rvert}$
$\mathcal{C}$	elements/cells in the mesh	$\mathbb{N}^{\lvert\mathcal{C}\rvert\times h}$
h	number of basis for each element/cells	$\mathbb{N}$
$\mathcal{B}$	basis function $\mathbb{B}=\{b_i\}$	$\lvert\mathcal{V}\rvert\times(\mathbb{R}^d\to\mathbb{R})$
$\mathcal{E}$	connections in the graph	$\{(u,v)\lvert b_u(v)\neq 0\}$
K	Galerkin matrix	$\mathbb{R}_{\text{sparse}}^{\mathfrak{D}\times\mathfrak{D}}$
$\mathfrak{D},\mathfrak{D}_I,\mathfrak{D}_D$	Total/inner/DBC DoF	$\mathbb{N}$
$K_{II,ID,DD}$	Galerkin matrix within NB/NB & DBC/ DBC DoF	$\mathbb{R}_{\text{sparse}}^{\mathfrak{D}_I\times\mathfrak{D}_I}$ $\mathbb{R}^{\mathfrak{D}_I\times\mathfrak{D}_D}$ $\mathbb{R}^{\mathfrak{D}_D\times\mathfrak{D}_D}$
$\mathbb{C}$	stiffness tensor	$\mathbb{R}^{d\times d\times d\times d}$
$\mathcal{P}_{\mathcal{V}}$	assemble tensor for nodes/vertices	$\mathbb{R}_{\text{sparse}}^{\lvert\mathcal{C}\rvert\times h\times\lvert\mathcal{V}\rvert}$
$\mathcal{P}_{\mathcal{E}}$	assemble tensor for edges	$\mathbb{R}_{\text{sparse}}^{\mathcal{C}\times h\times h\times\lvert\mathcal{E}\rvert}$
A	adjacency matrix of the graph	$\mathbb{R}_{\text{sparse}}^{\lvert\mathcal{V}\rvert\times\lvert\mathcal{V}\rvert}$
D	degree matrix of a graph	$\mathbb{R}_{\text{diagonal}}^{\lvert\mathcal{V}\rvert\times\lvert\mathcal{V}\rvert}$
$\mathcal{N}_i$	neighbors of node i in a graph	$\mathbb{N}^{\lvert\mathcal{N}_i\rvert}$

TABLE II: Symbol Table

2) Second, turn the strong form PDE to weak form PDE. Take the poisson equation as an example $\Delta u=-f\Leftrightarrow\int_\Omega\nabla u\nabla v\mathrm{d}v=\int_\Omega fv\mathrm{d}v$[1].
3) Third, choose the basis function function $\mathcal{B}=\{b_i|b_i\in\mathbb{R}^d\to\mathbb{R}\}$ and quadrature rules $\mathcal{Q}=\{\xi_i,\phi_i|\xi_i\in\mathbb{R},\phi_i\in\mathbb{R}^d\}$ for each element in the mesh $\mathcal{M}$.
4) Fourth, compute the Galerkin matrix and load vector for each element. Take the poisson equation as an example $K_{ij}=\sum_k\xi_k\nabla b_i(\phi_k)\nabla b_j(\phi_k)|J|$, $F_i=\sum_k\xi_k b_i(\phi_k)f(\phi_k)|J|^2$.
5) Fifth, the galerkin discretilized weak form becomes $\int_\Omega\nabla u\nabla v\mathrm{d}v=\int_\Omega fv\mathrm{d}v\approx K_{ij}u_i=F_j$. By solving this linear system, we got the variable u for each basis.

In a canonical FEM process. We first generate a mesh $\mathcal{M}$ in a given PDE domain $\mathcal{D}$.

D. Linear Elasiticity

As for continuum mechanics, the relationship between stress and strain is modeled by a 4-th order tensor according to the following Equation 1

$$\boldsymbol{\sigma}_{ij}=\mathbb{C}_{ijkl}\boldsymbol{\varepsilon}_{kl} \tag{1}$$

Assume the most simple model, Linear Elasticity, which could be formulated as Equation 2

$$\boldsymbol{\sigma}_{ij}=\lambda\delta_{ij}\boldsymbol{\varepsilon}_{kk}+2\mu\boldsymbol{\varepsilon}_{ij} \tag{2}$$

[1] u is called "trial space", v is called "test space"
[2] J is the jacobian matrix, which define the transformation between the standard element to the global deformed element

E. Static Condensing

Normally there are three boundary conditions for PDE equation

1) **D**irichlet **B**oundary **C**ondition : $u=g$ on Γ_D
2) **N**euman **B**oundary **C**ondition : $\frac{\partial u}{\partial\boldsymbol{n}}=h$ on Γ_N

The NBC can be easily coupled in the weak form. As for the DBC, static condensing is usually used to solve the DBC. Considering the Dirichlet Boundary ΓD, we could partite the Galerkin matrix into 2×2 block and load vector into 2 blocks shown in Equation 3

$$\begin{bmatrix}K_{II}&K_{ID}\\K_{ID}^\top&K_{DD}\end{bmatrix}\begin{bmatrix}u_I\\u_D\end{bmatrix}=\begin{bmatrix}F_I\\F_D\end{bmatrix} \tag{3}$$

Where, K_{II} is the galerkin matrix for non-boundary DoF of shape $\mathbb{R}_{\text{sparse}}^{\mathfrak{D}_I\times\mathfrak{D}_I}$. And K_{ID} is the commute galerkin matrix for non-boundary DoF and DBC DoF, which is of shape $\mathbb{R}_{\text{sparse}}^{\mathfrak{D}_I\times\mathfrak{D}_D}$. The K_{DD} is the galerkin matrix for DBC Dof of shape $\mathbb{R}_{\text{sparse}}^{\mathfrak{D}_D\times\mathfrak{D}_D}$. In the DBC, the unkown and known parts are notated in Figure 1, where orange parts are unknown parts and blue parts are known parts. In this way, we could conclude that the purpose of static condensatsion is calculate u_I,F_D using K,u_D,F_I

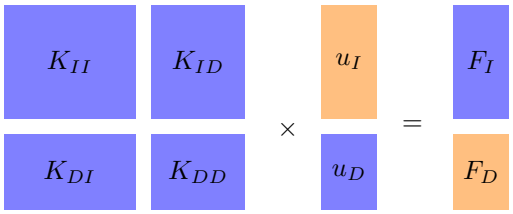

Fig. 1: Static Condenstation: blue parts are known, while orange parts are unkown

According to Equation 3, we could obtain two essenstial formulars for static condensing in Equation 4

$$\begin{cases}K_{II}u_I=F_I-K_{ID}u_D\\K_{ID}^\top u_I+K_{DD}u_D=F_D\end{cases} \tag{4}$$

F. Graph Neural Network

Graph **N**eural **N**etwork is a popular deep learning tool to handel non-euclidean data like social network [21] and particle interaction [14]. And GNN is well researched for it's application on large scale graph with highly parallelism and hardware optimization [1].

Chebyshev Spectral Convolutional Neural **Net**work [2] first introduce spectral approch to Graph Neural Network. Meanwhile, **G**raph **C**onvoultional **N**etwork [11] is the first order approximation of ChebNet, which takes a simple form as Equation 5

$$H^{l+1}=L\sigma(WH^l+b) \tag{5}$$

where L is the laplacian matrix which could be computed by $L=D^{-\frac{1}{2}}\hat{A}D^{-\frac{1}{2}}$. To prevent the self-information from lost during the propogation, adjacency matrix with self loop $\hat{A}$ is adapted here. Furthermore, to ensure the consistent of

information scale for all nodes, the propogation weight is normalized by the square root inverse of degree of two nodes connected by a edge.

Since Attention [19] obtain remarkable achievement in **N**ature **L**anguage **P**rocessing and **C**omputer **V**ision, it's also applied to GNN [20], which is Graph Attention Network. The aggregation of GAT takes the form as Equation 6. Where $\mathbf{W}$ is the learnable weight, $\mathcal{N}_i \cup \{i\}$ denotes all the neighbors of vertice i including itself. The compute the aggregation weight, a softmax is used as in Equation 7

$$H^{l+1} = \sigma\left(\sum_{j \in \mathcal{N}_i \cup \{i\}} \alpha_{ij} \mathbf{W} h_j\right) \tag{6}$$

$$\alpha_{ij} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^\top [\mathbf{W}h_i \| \mathbf{W}h_j]))}{\sum_{k \in \mathcal{N}_i \cup \{i\}} \exp(\text{LeakyReLU}(\mathbf{a}^\top [\mathbf{W}h_i \| \mathbf{W}h_k]))} \tag{7}$$

Inspired by U-Net [16], Graph U-Net [6] is proposed to learn the multi-scale information of the graph. As for the pooling part, topk pooling is normally adopted, which takes the form as Equation 8. Where $\mathbf{i}$ is the selected vertice index, $\mathbf{p}$ is projection vector

$$\mathbf{i} = \text{top}_k(\mathbf{p}^\top H) \tag{8}$$

Oversmoothing [17] is a command curse caused by the message-passing framework. Since multiple propagations will make far-away node features indistinguishable. However, **S**calable **I**nception **G**raph **N**eural Networks [3] keep multi-propagation node features and concatenate them as the latent feature. The architecture of SIGN is shown in Figure 2 and Equation 9.

$$H = \text{MLP}(\|_{i=0}^{n} \text{MLP}_i(L^i X)) \tag{9}$$

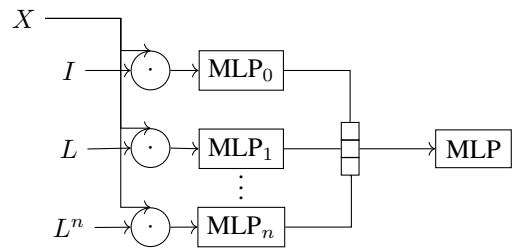

Fig. 2: SIGN overview: The node feature X is propogated by multiple times $\{I \cdot X, L \cdot X, \cdots, L^n \cdot X\}$. And these features are concatenated to maintain the information in each stage.

G. GNN for FEM

Since the mesh from FEM is non-euclidean, some researchers tried to use GNN to achieve the same goal as what FEM does.

Some study use the element as node to build the graph [4, 9], while others use the vertices in the mesh to build the graph [5, 12, 18]. We call the former graph the 'Element Graph', the latter one 'Vertice Graph'.

As for the element graph, study [9] directly encoder the boundary condition into the feature of the element, using a encoder-decoder GNN structure to simulate FEM. And study [4] assume K_{ID} is a dense matrix, therefore, they use dense connection between the boundary elements and the interior elements.

For the vertice graph, study [5] use the naive method to tackle the condensing, and directly process the load according to Equation 4. To learn the latent physical rule, they use **P**hysical **I**nformed **N**eural **N**etwork to train the model.

III. Methodology

A. Mesh to Graph

For this work, we choose vertice graph rather than element graph. Since element graph is closer to **F**inite **V**olume **M**ethod and it's not equivelant to **F**inite **E**lement **M**ethod from physical view.

We use the Algorithm 1 to generate the graph connections $\mathcal{E}$ from the given elements $\mathcal{C}$. It's easily implemented in Pytorch [13] and could easily paralleled on GPU. Line 4 could use `torch.vmap(torch.meshgrid)` to parallel it. And for Line 5, we could use `torch.sparse_coo_tensor.coalesce` to remove the duplicated edges.

Algorithm 1 Mesh To Graph

1: **function** MESH2GRAPH($\mathcal{C}$)
2: **Input:** $\mathcal{C}$ ▷ Elements
3: **Output:** $\mathcal{E}$ ▷ Edges of the graph
4: $\mathcal{E} \leftarrow \{(v_i, v_j) \mid v_i, v_j \in \mathcal{C}_k\}$
5: $\mathcal{E} \leftarrow$ coo coalesce($\mathcal{E}$)
6: **return** $\mathcal{E}$
7: **end function**

According to the Algorithm 1. Each element is considered as a fully connected subgraph. The process of converting from mesh $\mathcal{M}$ to graph $\mathcal{G}$ could be visualized as Figure 3

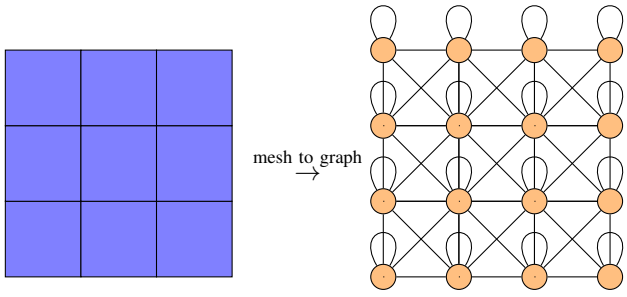

Fig. 3: Mesh To Graph: Each element is considered as a fully connected subgraph

B. Fast Assemble

The assembly process is also a time-consuming part of FEM. Traditionally, `for` loop is used to do the assembly. However, it's not paralleled and of low efficiency. In this subsection, we proposed a fast python-scope assembling method

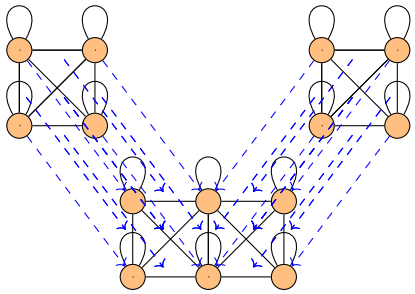

Fig. 4: Edge Assemble: The assemble process could be considered as a sparse matrix multiplication.

called 'Fast Assemble', which is fast and easy to parallel, thus unlocking the potential of GPU and NPU.

Fast Edge Assemble The Pseudo code of Fast Edge Assemble is presented below as Algorithm 2. It's output is of shape $\mathbb{R}^{\mathfrak{D}\times\mathfrak{D}}$. The `coo` in Line 46 denotes the **Coo**rdinate List. And `csr` represents **C**ompressed **S**parse **R**ow sparse matrix storage format, which is fast for multiplication and indexing.

The assemble process for Galerkin matrix could be viewed as a sparse matrix multiplication (non-symmetry) shown in Figure 4

Algorithm 2 Fast Edge Assemble

1: **function** BUILD EDGE PROJECTION($\mathcal{E}$,$\mathcal{C}$)
2: **Input:** $\mathcal{E},\mathcal{C}$ ▷ edges, elements
3: **Output:** $\mathcal{P}_\mathcal{E}$ ▷ Edge assemble matrix
4: $\mathcal{E}_{\text{id}} \leftarrow coo([1,\cdots,|\mathcal{E}|],\mathcal{E}.u,\mathcal{E}.v).tocsr()$
5: $\mathcal{E}_{\text{id,local}} \leftarrow \mathcal{E}_{\text{id}}(\mathcal{C}_{ki},\mathcal{C}_{kj})$
6: $\mathcal{P}_\mathcal{E} \leftarrow coo(\mathbf{1},\mathcal{E}_{\text{id,local}},[1,\cdots,h^2|\mathcal{C}|])$
7: **return** $\mathcal{P}_\mathcal{E}$
8: **end function**
9: **function** FAST EDGE ASSEMBLE($\mathcal{C},\hat{K}_{\text{local}}$)
10: **Input:** $\mathcal{C},\hat{K}_{\text{local}}$ ▷ Elements, local Galerkin matirx
11: **Output:** K ▷ Global Galerkin matrix
12: $\mathcal{E} \leftarrow$ MESH2GRAPH($\mathcal{C}$)
13: $\mathcal{P}_\mathcal{E} \leftarrow$ BUILD EDGE PROJECTION($\mathcal{E},\mathcal{C}$)
14: $\hat{K}^{nkl}_{\text{global}} \leftarrow \mathcal{P}^{nhij}_\mathcal{E}\hat{K}^{hklij}_{\text{local}}$
15: $K \leftarrow block\_coo(\hat{K}^{nkl}_{\text{global}},\mathcal{E})$
16: **return** K
17: **end function**

Fast Node Assemble There is also an algorithm for 'Fast Node Assemble', the result of which is $\mathbb{R}^{\mathfrak{D}}$. The similar pseudo code is shown in Algorithm 3

If the topology of mesh stays consistent, then the computational cost for both Fast Edge Assemble and Fast Node Assemble is only a sparse matrix multiplication. Therefore, the algorithm is fast and easy to parallel and can be directly implemented on GPU using Pytorch [13].

C. Static Condenseing Equivelant Architecture

Suppose a naive situation 1, that for DBC vertice, all dimension are constraints. For example, in the FEM problem, all three dimension of the boundary node are fixed with certain displacement.

Algorithm 3 Fast Node Assemble

function BUILD NODE PROJECTION($\mathcal{C}$)
Input: $\mathcal{E},\mathcal{C}$ ▷ edges, elements
Output: $\mathcal{P}_\mathcal{V}$ ▷ Edge assemble matrix
$\mathcal{P}_\mathcal{V} \leftarrow coo(\mathbf{1},\mathcal{C},[1,\cdots,h^2|\mathcal{C}|])$
return $\mathcal{P}_\mathcal{V}$
end function
function FAST EDGE ASSEMBLE($\mathcal{C},\hat{F}_{\text{local}}$)
Input: $\mathcal{C},\hat{F}_{\text{local}}$ ▷ Elements, local load vector
Output: K ▷ Global load vector
$\mathcal{P}_\mathcal{E} \leftarrow$ BUILD NODE PROJECTION($\mathcal{C}$)
$\hat{F}^{nk}_{\text{global}} \leftarrow \mathcal{P}^{nhi}_\mathcal{E}\hat{F}^{hki}_{\text{local}}$
$F \leftarrow block\_coo(\hat{F}^{nkl}_{\text{global}},\mathcal{E})$
return F
end function

Assumption 1. *The DBC is applied to all dimensions of a vertice*

In Equation 4, we substitute the first formula with GNN which is formulated in Equation 10

$$\begin{aligned}\text{GNN}_{\theta_1}(A_{II},f'_I,x) &\approx K^{-1}_{II}(x_I)(f)\\ \text{B-GNN}_{\theta_2}(A_{IB},u_B,x_B) &\approx -K_{IB}u_B\end{aligned} \tag{10}$$

The backbone GNN, denoted as GNN_{θ_1} and parameterized by θ_1, is designed to solve the linear system. In contrast, the Bi-partite GNN, B-GNN_{θ_2}, with parameters θ_2, models the propagation from boundary displacement to inner loading force. The architecture of B-GNN_{θ_2}, as described by Equation 11, comprises two Multi-Layer Perceptrons (MLPs). Figure 5 provides a detailed visualization of this process.

$$\text{B-GNN}_{\theta_2}(x) = \text{MLP}_2(A_{DI}\text{MLP}_1(x)) \tag{11}$$

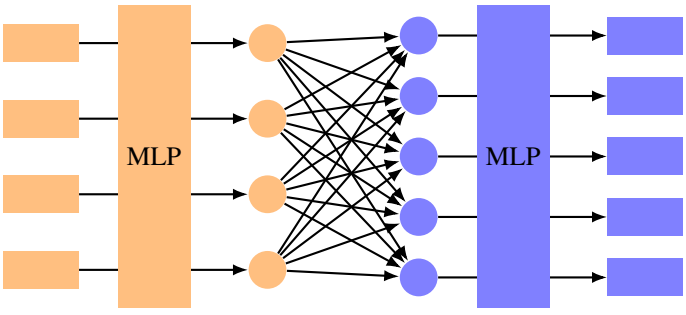

Fig. 5: Bi-Partite Graph Neural Network: Orange circles represent boundary nodes, while blue circles denote inner nodes. Rectangles indicate the node features. Boundary node features are processed by a Multi-Layer Perceptron (MLP) before aggregation at the inner nodes, followed by another MLP to map the aggregated features.

By substituting the mutual Galerkin K_{IB} and inverse of inner Galerkin K^{-1}_{II} with GNN. We could obtain the static condensation as Equation 12

$$u_I = K_{II}^{-1}(f_I - K_{IB}u_B)$$
$$\downarrow$$
$$u_I = \text{GNN}_{\theta_1}(A_{II}, f_I + \text{B-GNN}_{\theta_2}(A_{BI}, u_B), x_I) \tag{12}$$

To visualize the architecture more intuitively, we draw the pipeline in Figure 6.

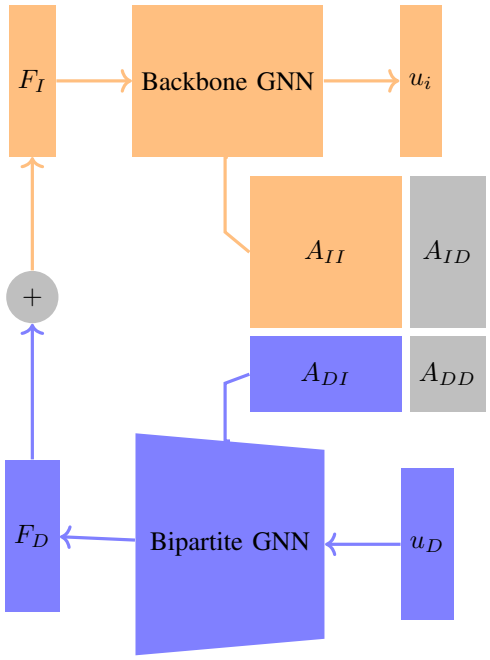

Fig. 6: SCEA: The Static Condense Equivelant Architecture overview. The Bipartite GNN is a kind of GNN that propagate message on a bi-partite graph. The Backbone GNN could be any canonical GNN. For example, GCN and GAT.

The Bipartite GNN is built specially for propagating information from DBC nodes to inner nodes. It takes the form of Equation 13

$$F_D \approx \text{MLP}_I(A_{ID}\text{MLP}_D(u_D)) \tag{13}$$

D. Physical Loss

To illustrate the physical loss in detail, we take the Linear Elasticity PDE here.

In order to optimize the parameters in GNN to predict a better displacement u.

Strong Form Loss The most intuitive loss is the strong form loss. Which is directly the left and right side residual of the linear system equation. Normally, **M**ean **S**quare **E**rror is usually used. Therefore, we formulate the strong form loss as following Equation 14

$$\mathcal{L}_{\text{phy strong}} = \|Ku - f\|_2 \tag{14}$$

Weak Form Loss Apart from the linear system, we could also get a equation at the weak form equation. Therefore, we could also formulate the another loss as Equation 15

$$\mathcal{L}_{\text{phy weak}} = \left\| \mathcal{P}_{\mathcal{V}} \left(\int_\Omega \begin{bmatrix} \frac{\partial u}{\partial x} & 0 \\ 0 & \frac{\partial u}{\partial y} \\ \frac{\partial u}{\partial y} & \frac{\partial u}{\partial x} \end{bmatrix}_{eid} D_{eij} B_{ebjd} |J|_e - f_{ebd} N_b |J|_e \mathrm{d}v \right) \right\|_2 \tag{15}$$

where D and B are denoted using vigoit notation. The matrix $\mathcal{P}_{\mathcal{V}}$ is referred to as the vertice assemble matrix, which is a transformation mapping from $\mathbb{R}^{|\mathcal{C}|\times b \to |\mathcal{V}|}$. The gradient $(\nabla u)_{ed}$ is calculated by multiplying the gradient of the shape functions $(\nabla N)_{ebd} u_{ebd}$ with the nodal values u_{ebd}. Here, e stands for element, b for basis, d for dimension, and i, j are used for indices in the context of reduction.

E. Galerkin Equivelant Architecture

As for another problem, we want to model the forward process which is $K, u \to f$ rather than $K, f \to u$. We could either predict the local Galerkin matrix to get the global Galerkin matrix by Fast Edge Assemble 2 or predict the global Galerkin matrix directly.

Local Pseudo Linear GEA Assume we could predict the local Galerkin matrix directly from the vertice coordinates by a simple neural network $\text{MLP}_\theta(x) \approx \hat{K}_{\text{local}}$, then we could propose the first Local Pseudo Linear GEA as Equation 19.

$$K = \text{bsr\_matrix}(\mathcal{P}_{\mathcal{E}} \hat{K}_{\text{local}}) \tag{16}$$
$$\downarrow \tag{17}$$
$$K = \text{bsr\_matrix}(\mathcal{P}_{\mathcal{E}} \text{MLP}_\theta(x)) \tag{18}$$

Local Pseudo Bilinear GEA Since the Local Pseudo Linear GEA cannot maintain the bilinear feature of the vigoit notation. Therefore, we proposed another approach: Local Pseudo Bilinear GEA with assumption $\text{MLP}_{\theta_1}(x) \approx B$ and $\theta_{2,ij} + \theta_{2,ji} - \theta_{2,ii} \approx D$. Then we could also obtain the prediction for global Galerkin 16.

$$K = \text{bsr\_matrix}(\mathcal{P}_{\mathcal{E}} B^\top D B) \tag{19}$$
$$\downarrow \tag{20}$$
$$K = \text{bsr\_matrix}(\mathcal{P}_{\mathcal{E}} \text{MLP}_{\theta_1}(x)^\top \tag{21}$$
$$(\theta_2 + \theta_2^\top - \text{diag}(\theta)) \text{MLP}_{\theta_1}(x)) \tag{22}$$

Global GEA We could also predict the global Galerkin matrix directly using an Edge-GNN 23.

$$K_e = \text{MLP}([x_u || x_v]) \tag{23}$$

For all these methods, the loss function is defined as Equation 24

$$\mathcal{L} = \|Ku - f\| \tag{24}$$

Method	GNN	iterative method	direct method
Memory Consumption	$\mathcal{O}(e)$	$\mathcal{O}(e)$	$\mathcal{O}(n^{\frac{3}{2}})$
Time Complexity	$\mathcal{O}(e)$	$\mathcal{O}(le)$	$\mathcal{O}(n^{\frac{3}{2}})$

TABLE III: Approaximated Complexity for each method

F. Complexity Analysis

There are many methods to solve the sparse linear system $Au = f$. We could mainly categorized them into two parts: the iterative methods such as Conjugated Gradient, Bi-Conjugated Gradient Stable, etc. and direct methods Cholesky or LU decomposition.

Assume a sparse matrix with e non-zero entries and of size $n \times n$, where typically $e \gg n$. For the operation of matrix-vector multiplication Au, both time and memory complexities are $O(e)$. These complexities are detailed in Table III, where different methods are compared. Iterative methods, which dynamically assess convergence, introduce an iteration count l. Commonly, the maximum number of iterations is capped at $10,000$, and the convergence threshold for the residual is set to 10^{-5}.

IV. Experiments

A. Setup

Baselines

- GCN [11]: Since GCN suffers from over-smoothing, we make GCN to be 3 layers in depth with 64 hidden size. The activation function is defaulted to ReLU [8].
- GAT [20]: The GAT also suffers from an over-smoothing problem. We fix the depth, hidden size, and activation function of GAT the same as GCN. The number of head for multi-head attention is fixed at 4.
- GraphUNet [6] : The topk pooling will sample 50% of vertices. The ensure the information could reach the other side on the mesh. The depth of the GraphUNet is fixed at 3.
- SIGN [3] : SIGN is a decoupled method, it will keep the information for each propagation step. Therefore, it won't suffer from the over-smoothing. Each MLP is assumed to be 3 layers with a hidden size 64 and activation function ReLU. The number of propagation n in Equation 9 and Figure 2 is set to be 8. In this way, the information will reach the same distance as GraphUNet.
- NodeEdgeGNN: Sometimes, relative position is preferred instead of normalized position. Therefore, we need a artificial GNN that only process the edge feature. So we propose a GNN called NodeEdgeGNN. Inspired by [14], we build basic block of graph convolution as Equation 25.

$$
\begin{aligned}
H_{\mathcal{E}}^{l+1} &\leftarrow \sigma\left(W_{\mathcal{E}}\left([H_{\mathcal{V},u}^{l} \| H_{\mathcal{V},v}^{l} \| H_{\mathcal{E}}^{l}]\right) + b_{\mathcal{E}}\right) \\
H_{\mathcal{V},i}^{l+1} &\leftarrow \sigma\left(W_{\mathcal{V}} \frac{1}{|\mathcal{N}_i \cup \{i\}|} \sum_{j \in \mathcal{N}_i} H_{\mathcal{E},ij}^{l+1} + b_{\mathcal{V}}\right)
\end{aligned} \tag{25}
$$

The depth, hidden size and activation function are set the same as GCN.

- MLP :Like the neural operator, without topology information, we could also implement the prediction task using MLP.

Hardware

The experiments are carried on a laptop with i5-11300H CPU with 16GB RAM and a Nvidia MX450 GPU with 2GB RAM.

Dataset

The mesh $\mathcal{M}$ is generated using Gmsh [7], a fast mesh generating library in C++. Each element $\mathcal{C}_k$ is supposed to be a triangle with three bases $h = 3$. The basis function for the triangle element is shown in Equation 26.

$$
\begin{aligned}
b_1 &= 1 - x - y \\
b_2 &= x \\
b_3 &= y
\end{aligned} \tag{26}
$$

where x, y here is the local coordinate, which is considered be constrained by $x, y \in [0, 1] \cap x + y \leq 1$.

We applied different boundary conditions and load conditions for mesh, which is shown from Figure 7 to Figure 9. The models are expected to train on the condition in Figure 7, and test on all these three Figures 789.

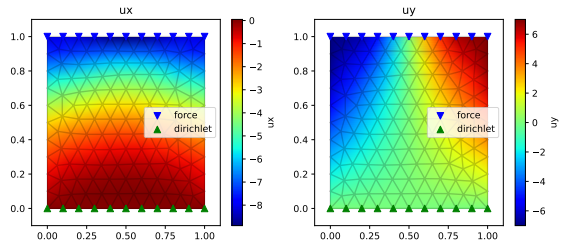

Fig. 7: Rectangle shape with triangle mesh. The upper boundary is applied with a downward force $f = sin(2\pi x)$, the bottom boundary are fixed in all two dimensions

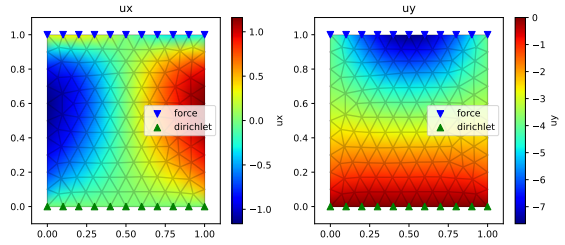

Fig. 8: Rectangle shape with triangle mesh. The upper boundary is applied with a downward force $f = sin(\pi x)$, the bottom boundary are fixed in all two dimensions

Loss

To combine the physical loss with data loss, we proposed three methods, which are

- equal : the total loss is the direct sum of data loss and physical loss.

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \mathcal{L}_{\text{phy}} \tag{27}$$

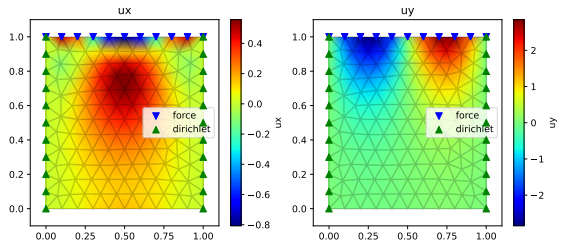

Fig. 9: Rectangle shape with triangle mesh. The upper boundary is applied with a downward force $f = sin(2\pi x)$, the left and right boundaries are fixed with all two dimensions.

- weight : the total loss is the weighted sum of data loss and physical loss.

$$\mathcal{L} = \mathcal{L}_{\text{data}} + \lambda L_{\text{phy}} \tag{28}$$

  where $\lambda = 0.01$
- auto weight [10]: the data loss is the parameters $(\theta_{\text{data}}, \theta_{\text{phy}})$ controlled weighted sum of data loss and physical loss.

$$\mathcal{L} = \frac{\mathcal{L}_{\text{data}}}{2\theta_{\text{data}}^2} + \frac{\mathcal{L}_{\text{phy}}}{2\theta_{\text{phy}}^2} + \log(1+\theta_{\text{data}}^2) + \log(1+\theta_{\text{phy}}^2) \tag{29}$$

B. Observed Data Variant

In this experiment, when the `condense type` is `none`, it means that all the DBC is added to the training set. As for `static`. Then part of the static condensation will be executed, it takes form as Equation 30. When the `nn` is adopted, then the exact SCEA will be applied according to Equation 12

$$u \approx \text{GNN}_\theta(A_{II}, f_I - K_{ID}u_D) \tag{30}$$

The `train ratio` means how many portions of data except DBC are observable. When `train ratio` becomes 1.0, then all the data are known. On the contrary, when `train ratio` is 0.0, only DBC are known.

Invariant Test

In the invariant test, the dataset utilized for testing is identical to the one used for training.

When the `training ratio` is set to 0.0, it implies exclusive reliance on the physical loss for `condense type` settings of either `static` or `nn`. This configuration allows for an assessment of the loss function's effectiveness in adhering to physical constraints. Conversely, for a `condense type` of `none`, Dirichlet Boundary Conditions (DBC) are employed, forming a data-driven training set.

There are some interesting observations.

- When the `train ratio` is set to 0.0, SIGN with either `static` or `nn` demonstrates optimal performance, indicating its capability to infer latent physical rules effectively in the absence of observed data.
- At a `train ratio` of 0.0, the performance of `static` and `nn` condense types is nearly identical. This suggests that the B-GNN is adept at modeling the propagation of boundary conditions.

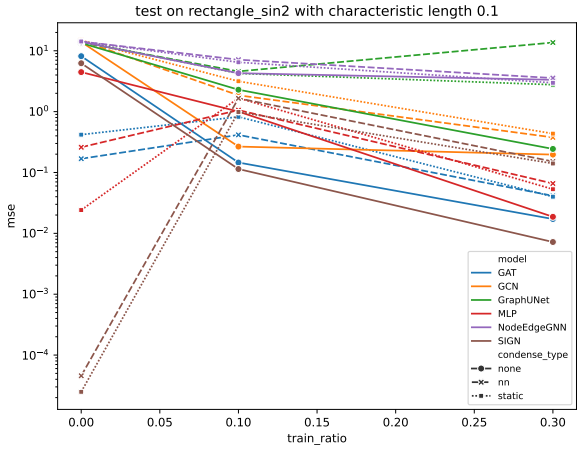

Fig. 10: Invariant Test(with load condition $f = sin(2\pi x)$ and bottom boundary fixed and **strong** physical loss(Equation 14) with equal total loss $\mathcal{L} = \mathcal{L}_{\text{data}} + \lambda L_{\text{phy}}$ is adopted)

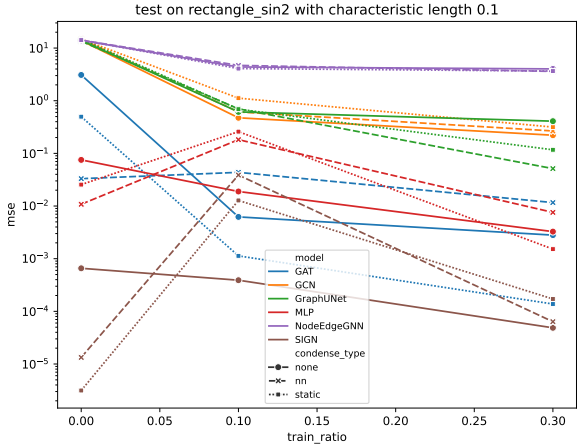

Fig. 11: Invariant Test(with load condition $f = sin(2\pi x)$ and bottom boundary fixed and **strong** physical loss(Equation 14) with equal total loss $\mathcal{L} = \frac{\mathcal{L}_{\text{data}}}{2\theta_{\text{data}}^2} + \frac{\mathcal{L}_{\text{phy}}}{2\theta_{\text{phy}}^2} + \log(1+\theta_{\text{data}}^2) + \log(1+\theta_{\text{phy}}^2)$ is adopted)

Frequency Variant Test

In the frequency variant test, distinct datasets are used for training and testing. The model is trained using the dataset depicted in Figure 7 and evaluated on the dataset illustrated in Figure 8. The outcomes of this test are presented in Figure **??**.

Boundary Variant Test

In boundary variant test, the training dataset and testing dataset are different. The model is trained on dataset shown in Figure 7 and tested on dataset shown in Figure 9. The result is shown in Figure 13.

For the frequency variant and boundary variant tests, we present only the results using the auto weight loss, as defined in Equation 29.

Based on these tests, we can draw the following conclusions:

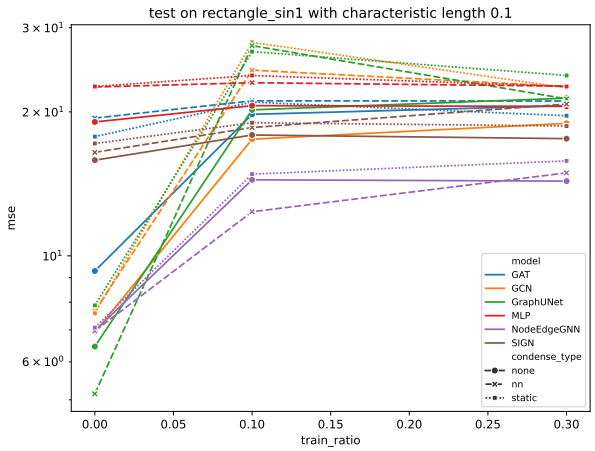

Fig. 12: Frequency Variant Test(with load condition $f = sin(\pi x)$ and bottom boundary fixed and **strong** physical loss(Equation 14) with auto weight total loss(Equation 29) is adopted)

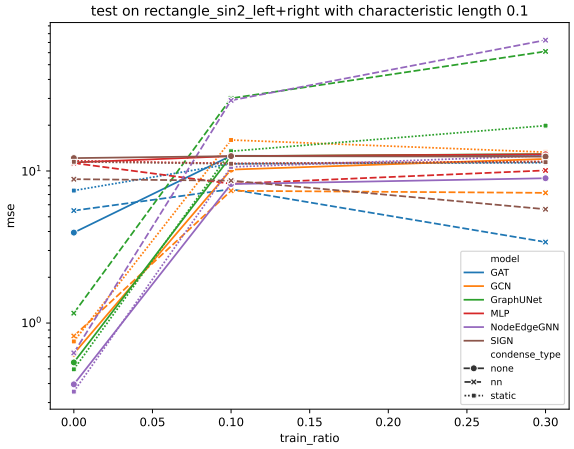

Fig. 13: Boundary Variant Test(with load condition $f = sin(2\pi x)$ and left+right boundary fixed and **strong** physical loss(Equation 14) with auto weight total loss(Equation 29) is adopted)

- Performance in the frequency variant test surpasses that in the boundary variant test.
- A notable disparity exists between the physical loss and data loss, indicating that achieving generalization in models trained with physical loss is challenging.

C. Case Study

In this subsection, we visualize the prediction result from SCEA. Specifically, we adopt the GNN SIGN trained only on the condition of the bottom Dirichlet boundary and top loading force of $f(x) = sin(2x)$. The model is trained for 1000 epochs with a learning rate of 0.01. The target loss function is strong form loss as shown in Equation 14.

We first test the prediction result on the same boundary and loading condition as the training dataset as shown in Figure 14. As can be seen, the result is pretty close to the ground truth.

To better investigate if the model has learned the frequency information from the loading function, we perform the frequency variant test, which changes the test loading frequency from 2 to 1, meaning the test mesh is somewhat different from the training mesh. The result is shown in Figure **??**. The result is not as good as expected, meaning the model seldom learns anything from the loading information.

On the other hand, to study the influence of the Dirichlet boundary condition topology, we also change the boundary condition of the dataset, which is shown in Figure 16. The result is also not as good as expected. Since one can hardly find any common from ground truth and prediction. It means that the model also cannot learn the boundary information from the one-shot training.

D. Generalization

To thoroughly evaluate the generalization capabilities of models trained with physical loss, we generated a comprehensive collection of random datasets. These datasets are primarily categorized into two topological types: triangles and quadrilaterals.

The triangle datasets were created by introducing a slight deviation to a standard unit triangle. This process is methodically detailed in Equation 31. Each triangle is slightly altered to introduce variability, yet maintaining a basic triangular structure. On the other hand, the generation of quadrilateral datasets follows a similar approach but adheres to the formula outlined in Equation 32. These quadrilaterals vary in shape and size, offering a diverse range of geometries for analysis. Representative examples from these datasets are illustrated in Figures 17 and 18, showcasing the range of variability within each category.

$$\mathcal{V} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix} + \mathcal{N}(0, 0.4) \tag{31}$$

$$\mathcal{V} = \begin{bmatrix} 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} + \mathcal{N}(0, 0.4) \tag{32}$$

In total, 840 distinct datasets were produced, each characterized by unique boundary conditions and loading forces. This extensive collection is designed to rigorously test the model's ability to adapt and learn from varied geometrical configurations and stress scenarios.

The experimental setup involves training the model on these datasets for a duration of 200 epochs, with a set learning rate of 0.005. This training regimen is aimed at thoroughly exposing the model to a wide spectrum of data, thereby challenging its learning and generalization mechanisms.

The outcomes of this training are depicted in Figure 19. A cursory examination of this figure reveals a notable struggle in the model's ability to consistently learn and replicate physical patterns across the diverse range of datasets. This observation is further substantiated by the recorded loss metrics, as shown in Figure 20. Here, a significant discrepancy is observed between the data loss and physical loss metrics. Such a disparity

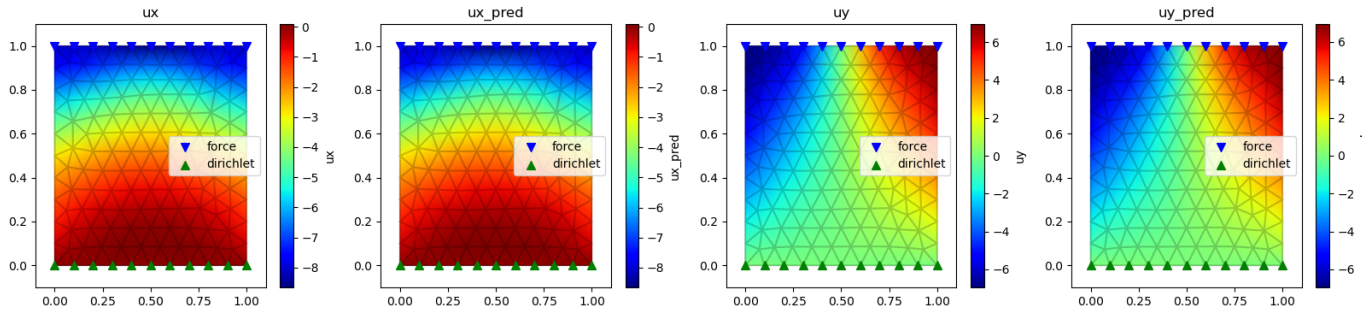

Fig. 14: model trained with loading function $f(x) = sin(2x)$ and bottom Dirichlet boundary using physical loss is tested on loading function of $f(x) = sin(2x)$ and same boundary condition

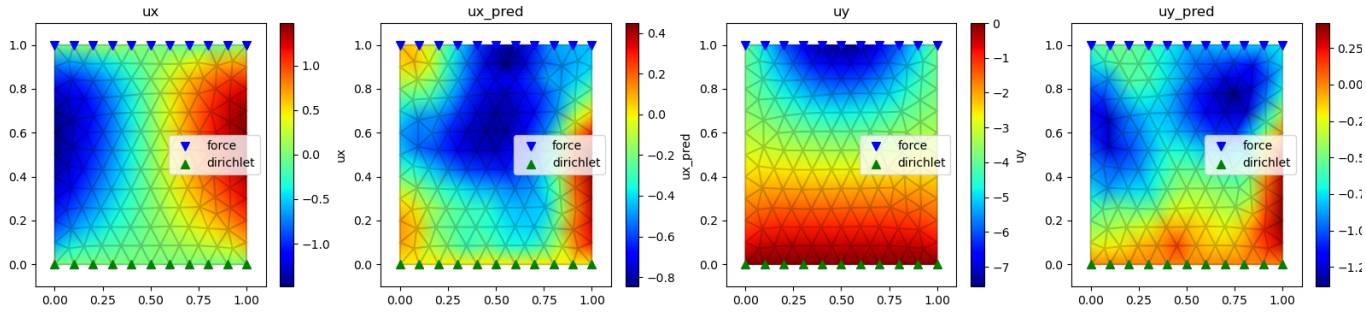

Fig. 15: model trained with loading function $f(x) = sin(2x)$ and bottom Dirichlet boundary using physical loss is tested on loading function of $f(x) = sin(x)$ and same boundary condition

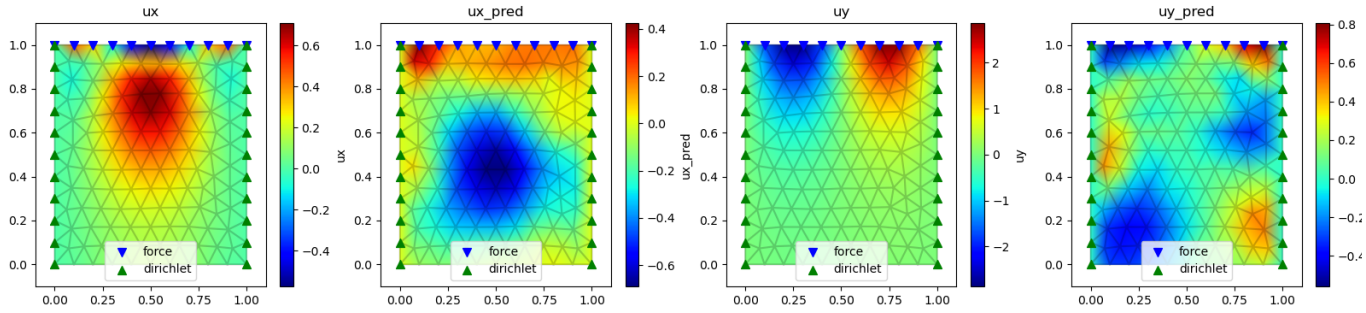

Fig. 16: model trained with loading function $f(x) = sin(2x)$ and bottom Dirichlet boundary using physical loss is tested on loading function of $f(x) = sin(2x)$ with left and right Dirichlet boundary

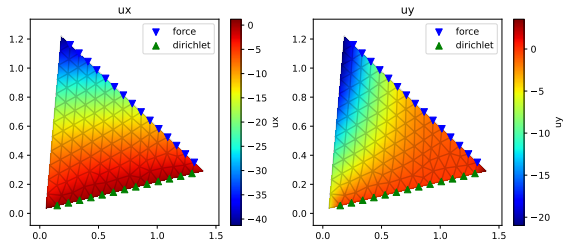

Fig. 17: Randomly generated triangle dataset

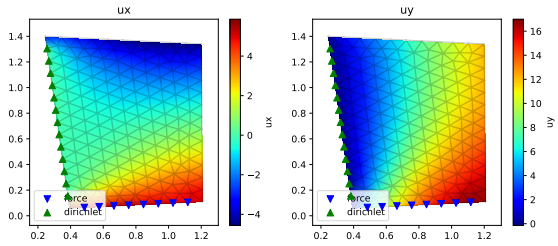

Fig. 18: Randomly generated triangle dataset

could be attributed to the fundamental differences in their calculation methods. The physical loss computation involves the gradient of the shape function, focusing on the variation of geometric properties, whereas the data loss calculation is

solely concerned with the discrepancies in function values. This divergence in approach may underlie the challenges faced by the model in achieving a high degree of generalization across varied datasets.

E. Galerkin Prediction

In our study, we focus on the forward problem, specifically targeting the direct prediction of the Galerkin matrix. To this end, we employ the Galerkin Equivalent Architecture (GEA) method, which encompasses several sub-methods.

For our experiments, we utilize the Pratt bridge truss dataset, which is composed of trusses. The structure of these trusses is depicted in Figure 21.

The models undergo extensive training for 20,000 epochs using the Adam optimizer. To enhance the learning process, we implement a cosine scheduler that progressively reduces the learning rate with each epoch. The Multi-Layer Perceptron (MLP) used in our models is configured with four layers and includes residual connections, enhancing its capacity for learning complex patterns.

Among the various approaches, the most intuitive is the pseudo bilinear method, as described in Equation 19. The efficacy of this method is evaluated by comparing the predicted Galerkin matrix with the ground truth, as shown in Figure 22.

In addition to the Local Pseudo Bilinear Galerkin Equivalent Architecture, we also explore other variants. The results of another such variant are illustrated in Figure 23.

Furthermore, we extend our analysis to the global Galerkin prediction, as outlined in Equation 23. The comparison of predictions with the actual Galerkin matrices is visually represented in Figure 24.

Upon examining Figures 22, 23, and 24, it becomes evident that, despite the challenges inherent in predicting the Galerkin matrix as indicated by the loss function in Equation 24, the Global Galerkin Equivalent Architecture demonstrates superior performance compared to the local variants.

V. Conclusion

In this study, we delve into two principal problems in the realm of structural analysis. The first is the reverse problem, focused on deducing the displacement of a structure given the applied loading force. The second, known as the forward problem, involves predicting the loading force from known displacements. To address the reverse problem, we introduce the Static Condensation Equivalent Architecture (SCEA). This innovative approach integrates the concept of static condensation into Graph Neural Network (GNN) frameworks, offering a novel perspective in computational mechanics.

For the forward problem, we propose the Galerkin Equivalent Architecture (GEA). This architecture is not a singular solution but rather a suite of implementations, each tailored to capture the complex interactions in structural systems under varying conditions.

Our experimental findings shed light on the underlying connections between Graph Neural Networks (GNNs) and the Finite Element Method (FEM). These insights not only demonstrate the potential of GNNs in accurately modeling structural responses but also pave the way for future explorations in this interdisciplinary field. The SCEA and GEA, with their respective focuses, contribute significantly to our understanding of the intricate relationship between machine learning algorithms and traditional finite element analysis, opening new avenues for research and development in structural engineering and computational mechanics.

References

[1] Sergi Abadal, Akshay Jain, Robert Guirado, Jorge López-Alonso, and Eduard Alarcón. Computing graph neural networks: A survey from algorithms to accelerators. *ACM Comput. Surv.*, 54(9), oct 2021.

[2] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29, 2016.

[3] Fabrizio Frasca, Emanuele Rossi, Davide Eynard, Ben Chamberlain, Michael Bronstein, and Federico Monti. Sign: Scalable inception graph neural networks. *arXiv preprint arXiv:2004.11198*, 2020.

[4] Xingyu Fu, Fengfeng Zhou, Dheeraj Peddireddy, Zhengyang Kang, Martin Byung-Guk Jun, and Vaneet Aggarwal. An finite element analysis surrogate model with boundary oriented graph embedding approach for rapid design. *Journal of Computational Design and Engineering*, 10(3):1026–1046, 03 2023.

[5] Han Gao, Matthew J Zahr, and Jian-Xun Wang. Physics-informed graph neural galerkin networks: A unified framework for solving pde-governed forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 390:114502, 2022.

[6] Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

[7] Geuzaine, Christophe and Remacle, Jean-Francois. Gmsh.

[8] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323. JMLR Workshop and Conference Proceedings, 2011.

[9] Chunhao Jiang and Nian-Zhong Chen. Graph neural networks (gnns) based accelerated numerical simulation. *Engineering Applications of Artificial Intelligence*, 123:106370, 2023.

[10] Alex Kendall, Yarin Gal, and Roberto Cipolla. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7482–7491, 2018.

[11] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

[12] Yangfan Li, Jun Liu, Wenyu Liang, and Zhuangjian Liu. Towards optimal design of dielectric elastomer actuators using a graph neural network encoder. *IEEE Robotics and Automation Letters*, 8(10):6339–6346, 2023.

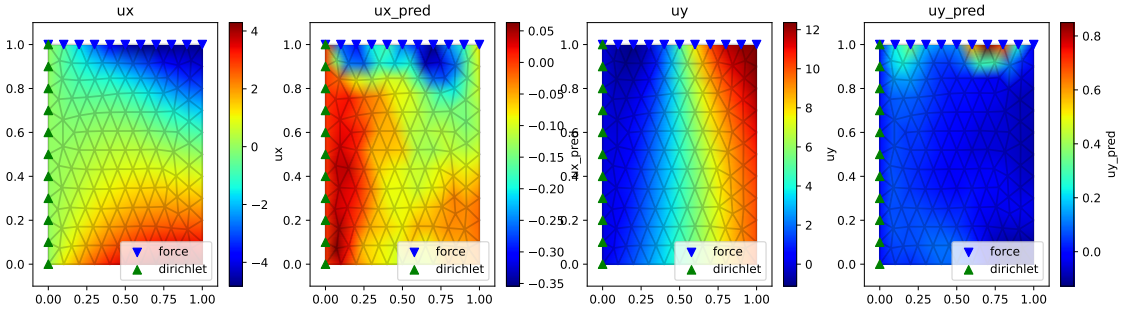

Fig. 19: The prediction result from the model trained on a bunch of different meshes

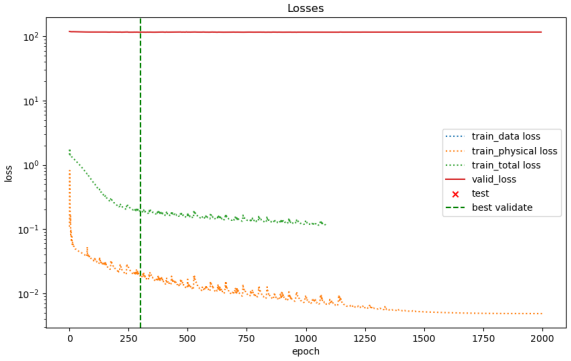

Fig. 20: The recording loss for multiple graphs training

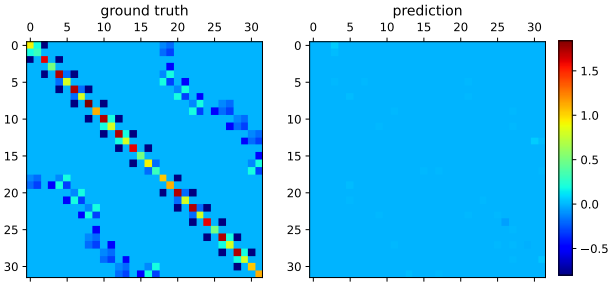

Fig. 22: Pseudo Bilinear Local Galerkin Equivalent Architecture trained result, with right the prediction Galerkin matrix while the left is the ground truth Galerkin matrix

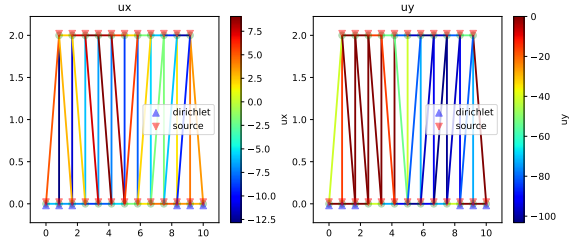

Fig. 21: Pratt bridge truss mesh

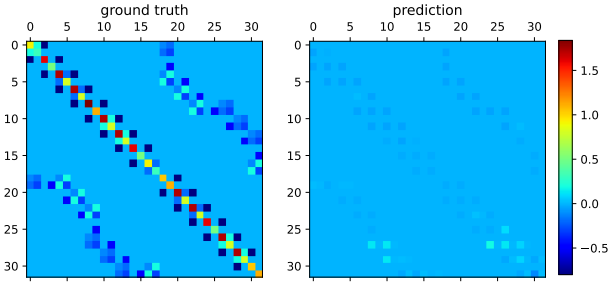

Fig. 23: Pseudo Linear Local Galerkin Equivalent Architecture trained result, with right the prediction Galerkin matrix while the left is the ground truth Galerkin matrix

[13] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[14] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

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

[16] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*, pages 234–241. Springer, 2015.

[17] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993*, 2023.

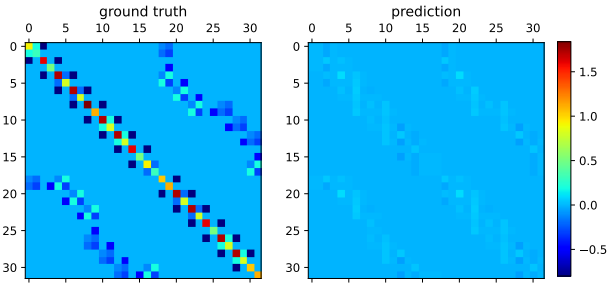

Fig. 24: Global Galerkin Equivalent Architecture with positional encoding frequency 8. The right is the predicted Galerkin matrix while the left is the ground truth Galerkin matrix

[18] Ling-Han Song, Chen Wang, Jian-Sheng Fan, and Hong-Ming Lu. Elastic structural analysis based on graph neural network without labeled data. *Computer-Aided Civil and Infrastructure Engineering*, 38(10):1307–1323, 2023.

[19] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[20] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

[21] Shiwen Wu, Fei Sun, Wentao Zhang, Xu Xie, and Bin Cui. Graph neural networks in recommender systems: A survey. *ACM Comput. Surv.*, 55(5), dec 2022.