Surrogate Model for Structural Analysis Simulation using Graph Convolutional Network (Unrefereed Workshop Manuscript)

Amir Haderbache[1,a)] Koichi Shirahata[1,b)] Tsuguchika Tabaru[1,c)]
Hiroshi Okuda[2,d)]

Abstract: Deep learning surrogate models can replace simulation solver computations by a low-latency inference. However, current surrogate models have important drawbacks such as inefficient memory usage when mapping mesh into regular-grids or limited predictive capabilities to specific input conditions. We propose an efficient surrogate model based on graph convolutional neural network designed for any finite-element mesh and boundary conditions. The accuracy is guaranteed by a mapping of the FEA data into the GCN graph while scalability is achieved by using the principal component analysis on the stiffness matrix. Our technique achieves significant speed-up and maintain accuracy with 1e-03 precision compared with HPC simulations.

Keywords: Structural Analysis, Finite Element Method, Sparse Linear System, Newton-Raphson, Performance Analysis, Deep Learning, Graph Convolutional Network, Principal Component Analysis

1. Introduction

Computer-aided engineering (CAE) covers a broad range of applications for high performance computing (HPC) such as finite element analysis (FEA) or computational fluid dynamics (CFD). These simulations compute the dynamics of physical objects and output relevant information for designers and scientists. When dealing with large scale simulation models, the computation becomes very time-consuming as the number of iterations required for convergence is, in general, proportional to the problem size. In the post Moore's law era, multiple methods exist for improving the performance of scientific simulations. In addition to specialized/hybrid computer systems, novel algorithms or software tuning, the somewhat recent emergence of deep learning and its ability to train efficient predictive models has caught the attention of the HPC community. Using simulation data, supervised deep learning algorithms and GPU computation power, many researchers [1] [2] have developed efficient surrogate models which are able to predict simulation results at very low latency (compared with the traditional solvers) with yet, a certain trade-off in accuracy.

Besides the usual speed/accuracy trade-off, surrogate models raise important limitations, especially when problem size is large. In general, the simulation input mesh is mapped into a regular-grid and fed into a neural network for training. This "image-based" method is highly inefficient in terms of memory usage and computation as a large part of the regular-grid consists in void areas surrounding the actual model. In addition to performance deficiency, predictive capabilities of surrogate models are often limited to specific simulation conditions and thus, making a change in the input mesh or the boundary conditions often involves a drastic modification of the training data design.

We propose a general method for building accurate surrogate models for large structural analysis simulations. Instead of using input boundary conditions, we directly map the FEA linear system data into a graph convolutional neural network (GCN) for replacing the iterative solver computation. Relying on standard FEA data only, our method can be applied for any input mesh and boundary conditions. Moreover, the GCN local neighborhood convolution enables high training performance without waste of memory usage compared with "image-based" training. Also, we use the principal component analysis (PCA) to achieve scalability for larger problem size and to avoid the empirical limitation of GCN in propagating the local neighborhood information through large mesh.

We evaluated the predictive performance of our GCN-based surrogate model using different model sizes up to more than nine thousand nodes. For all cases, high accuracy is achieved with 1e-03 precision compared with the FEA simulation results. Using a nine thousand nodes model, our

1 Fujitsu Ltd., Kawasaki, Japan
2 The University of Tokyo, Tokyo, Japan
a) haderbache.amir@fujitsu.com
b) k.shirahata@fujitsu.com
c) tabaru@fujitsu.com
d) okuda@k.u-tokyo.ac.jp

trained GCN achieves more than 3x acceleration compared with FEA simulation. Such approach can help designers and engineers during design space exploration by accelerating the discovery of optimal parameters for structures.

The following is a summary of the paper contributions. (1) We designed a general method for building accurate surrogate models of FEA simulations using graph convolutional neural network. Instead of using specific boundary conditions, we map the standard stiffness matrix and forcing vector data into a graph for GCN supervised training. (2) We scaled the predictive capabilities of GCN-based surrogate models to large input graph by applying the PCA reduction technique to a custom stiffness matrix. (3) We evaluated the inference accuracy of our trained model using different problem size and achieved speed-up compared with traditional FEA simulations.

2. Background

In this section, we explain the motivations of graph-based surrogate models for predicting FEA simulations results.

2.1 Structural Analysis simulations

Structural Analysis (SA) simulation is a computer program used by designers and engineers for computing the effects of loads applied on physical structures. From an input mesh and boundary conditions, the program generates a linear system, $Ax = b$, using the finite element method. Then an iterative solver, such as the *Conjugate Gradient*, is called for computing an estimate of the solution x. In the case of SA simulation, the solution is the nodal displacement vector which represents how much the structure has been deformed from its initial position. In general, SA simulations require HPC platforms for optimized sparse matrix-vector multiplication (SpMV) and to achieve faster convergence.

2.2 Why surrogate models ?

Even though efficient preconditioners have been developed for solving large sparse linear system, such as *Algebraic Multigrid*, a further need for acceleration is still required for design applications. Often, engineers must iterate on a large input parameters space where each trial can be very time-consuming, especially when problem size is large. Because each simulation instance generates a valuable amount of data, it became legitimate for researchers to leverage data-driven algorithms, such as supervised deep learning, for building low latency prediction models. Such incentive has become stronger recently in HPC for two reasons. First, deep learning achieved state-of-the-art performance in several applications such as computer vision or natural language processing. Secondly because many specialized hardware devices (GPU,TPU) has been developed for efficient neural network training and inference.

2.3 Image-based deep learning

Unlike conventional HPC algorithms, surrogate models' performances (accuracy and generalization) rely on many design choices [3] such as training data, network architecture and hyperparameters. A naive approach for surrogate model training is to incorporate the unstructured mesh into a regular grid because convolutional neural network (CNN) are designed for regular tensor processing. However, such method has important issues. First, it generates a high number of additional points resulting in increased memory usage. Secondly, it removes important spatial information from the original structure such as distance between nodes and local neighborhood density. These are the reasons why geometric deep learning [4] has emerged and is attempting to generalize deep learning to non-Euclidean domains such as graphs.

2.4 Graph Neural Network

Graph neural network (GNN) is a deep learning method which operates directly on graphs and is used for different tasks such as node classification, link prediction or clustering. Depending on the graph type (oriented or not, static or dynamic, etc...), the learning task (node-level, edge-level or graph-level) or the training setting (supervised or unsupervised), GNN models can have different architectures and capabilities. In general, a GNN is built by combining three types of computational blocks (propagation, sampling, and pooling) which are stacked for computing expressive graph representation through the network. The propagation block aggregates both the features and topological information before propagating them between nodes via message passing. When the graph is large, a sampling block can be used to facilitate the propagation and, in a similar way to image-based deep learning, pooling blocks can extract high-level subgraphs or graphs when it is needed. The existence of several variant for each computational block [5] leads to different possible instances of GNN. In this paper, we focus on the mostly used propagation block for GNN models: the convolution operator.

2.5 Convolution on graphs

As convolutional neural networks (CNN) can construct highly expressive representations by extracting localized spatial features from images, several researches tried to generalize the convolution to graph domain. Indeed, CNN properties such as local connection, shared weights and the use of multiple layers are also important in solving problems on graphs. There are two main methods for applying convolutions on graphs: the spectral and the spatial approaches.

2.5.1 Spectral methods

The spectral methods apply the convolution in the spectral domain using the graph Fourier transform. First the graph signal is transformed to the spectral domain, then the convolution is computed. After convolution, the returned signal is transformed back using the inverse Fourier transform. Such method requires to compute the eigenvectors of the graph Laplacian which is compute expensive.

However, *ChebNet* [6][7] approximated the learnable filters using Chebyshev polynomials and thus defined an efficient K-localized spectral convolution which does not require the computation of eigenvectors anymore.

2.5.2 Spatial methods

The spatial methods apply the convolution directly in the graph domain by considering its topology. Such methods process an input node by considering an aggregation of the local neighborhood's features. One of the main challenges of spatial methods is to define the convolution on graphs containing a variable neighborhood density (nodes with different degrees). The *GraphSAGE* method [8] is one state-of-the-art implementation for spatial convolution which uses a sampling strategy for always aggregating the information from a fixed size set of neighbors.

2.5.3 Graph Convolutional Networks

The *Graph Convolutional Networks* (GCN) method [9] is the convolution operator we use in this paper. GCN can be considered as both a spectral and spatial convolutional method. On one hand, it operates directly on the graph domain, encoding both the local graph structure and nodes features. On the other hand, it is based on a first order localized spectral convolution and brings simplifications to *Chebnet*. The GCN convolution is described as:

$$H = \tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}XW \tag{1}$$

where X $\in \mathbf{R}^{N\times F}$ is the node features matrix, W $\in \mathbf{R}^{F\times F'}$ is the learnable weights matrix and H $\in \mathbf{R}^{N\times F'}$ is the output matrix representing the convolved features. The graph structure is represented by the adjacency matrix A $\in \mathbf{R}^{N\times N}$ which includes the nodes self-loop ($\tilde{A} = A + I_N$). It is worth noting that the adjacency matrix is normalized by the degree matrix D so that GCN can handle varying nodes degrees. In this paper notation, N is the number of nodes, F is the number of node features and F' is the new number of features obtained after convolution. As GCN demonstrated high scalability and accuracy performance for traditional nodes classification problems, we decided to rely on this method to train a graph-based surrogate model for structural analysis simulation.

3. Related Work

In this section, we present research that employed machine learning (ML) in scientific computing to achieve better trade-off between computational cost and accuracy. We separate the techniques which use ML for improving simulation algorithms from the surrogate models which replace entirely the governing equations and provide low latency predictive capabilities with yet a trade-off in the accuracy.

3.1 ML for improving simulation algorithms

Several works applied machine learning for improving scientific simulations. In Molecular Dynamics, neural networks (NN) [10] and gaussian processes [11] have been trained with DFT simulations to reconstruct Potential Energy Surfaces. In the field of Computational Fluid Dynamics, NN could predict turbulence velocity vector fields [12], flow around air foil [13] and Reynolds anisotropy tensors [14]. For Structural Analysis simulations, some works used DNN to infer the stress applied on concrete given the strain of mesoscale simulation [15] or even bypassed the high resolution computation of stress around fillet using coarse sharp corner parameters [16]. Such methods can speed-up traditional simulation techniques, but the obtained acceleration is often limited as the conventional numerical methods are still used. Instead of assisting a numerical method, our proposal replaces it by a surrogate model to make direct predictions.

3.2 Surrogate models based on neural networks

Surrogate models aim to predict instantaneous information that are too time-consuming to compute through conventional simulation methods. By training NN with domain-specific dataset, surrogate models have been used in several engineering problems. In quantum mechanics, [17] directly predicted molecules ground states from nuclear charges and atomic position instead of solving expensive Schrodinger's equation. Using NN in flow analysis, [18] predicted multiphase flow properties from impedance data, [19] adjusted the mass flow rates of jets for reducing the drag around aircraft cylinder body and [20] optimized the design of structure for CFD simulations. Like our target application, [21] used NN to predict the stress applied on materials from strain tensors in order to identify material characteristics. While sharing the same objectives, our proposal relies instead on graph processing for extracting knowledge not only from the features space but also directly from the topology.

3.3 Surrogate models based on graph networks

Recently, graph-based surrogate models have been used to represent and process unstructured data in a more natural way. In biology, [22] proposed a GCN method for protein interface prediction by representing the residues as nodes. In particle systems, [23] used GNN to predict particle trajectories from learned interaction graphs. In a broader scope, [24] developed a general framework for particle-based simulations by combining an "encode-process-decode" architecture with a learning of dynamics based on message-passing. More recently, [25] has extended this framework to mesh-based simulations and presented a speed-up of two orders of magnitude compared with traditional simulations. Focusing on the learning of graph isometric transformation invariant features, [26] enhanced the GCN layer and demonstrated efficient scalability and prediction accuracy on nonlinear heat diffusion problems compared with conventional FEA simulations. In these works, the surrogate models are trained to predict future physical states from the previous one by learning the transient evolution of the node features. Such method has the main disadvantage to accumulate the prediction error through the iterations because previous predictions become the inputs for next inferences. Moreover, when choosing problem specific data to represent node features such as the input temperature field, it becomes difficult

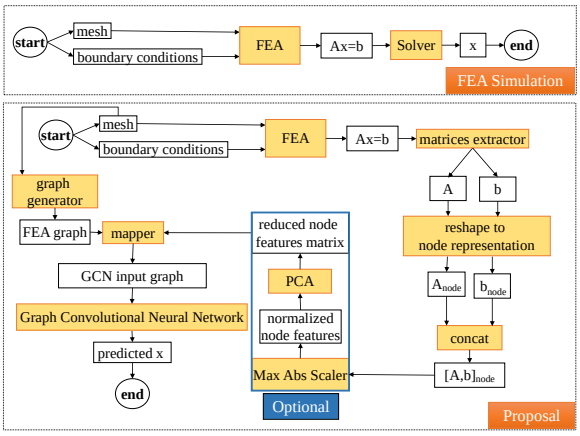

Fig. 1 Difference between the conventional FEA simulation and our proposal which replaces the solver computation bottleneck by a GCN inference.

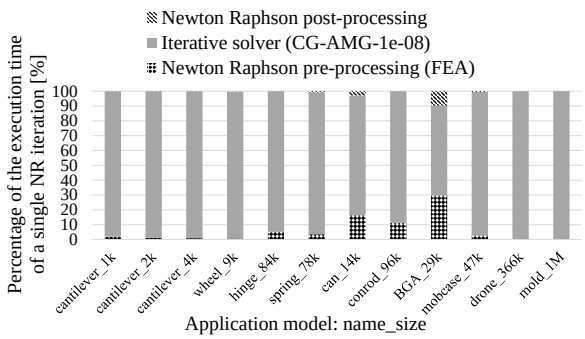

Fig. 2 FrontISTR simulation: CPU time breakdown of a single Newton-Raphson iteration for several application models.

to generalize to other types of simulation input conditions. Our proposal deals with such limitations by processing directly a nodal representation of the FEA system using GCN.

4. Proposal

We explain in this section the design choices and the implementation details of our proposed GCN-based surrogate model of FEA structural analysis simulations.

4.1 Structural Analysis simulations bottleneck

As explained in Section 2 and illustrated in the top frame of Figure 1, a Structural Analysis simulation is composed of two main computational blocks: the FEA and the linear solver. Figure 2 shows the CPU time breakdown of a single Newton-Raphson (NR) iteration for several application models. Each NR iteration is composed of one FEA call and one linear solver call. Readers can imagine that these breakdown results are repeated for multiple NR iterations throughout the entire nonlinear SA simulations. Independently of the problem size, which is indicated after the application model name, Figure 2 reveals that the major part of the computation is spent by the linear solver. The details of the experimental environment are described in Section 5. For this breakdown measurement, we used the CG solver with AMG preconditioner, a solver tolerance of 1^{-08} and a single OMP thread for all models except for the drone and mold which use 36 threads. Because the FEA computation is negligible compared to the solver one, we propose a surrogate model for replacing only the solver block. Instead of using an end-to-end black box, we keep the FEA computation and leverage its output data for training our AI model in predicting the solver estimation at lower latency. This global mechanism is described in Figure 1 and will be explained in the following sections.

4.2 Linear System as input for Surrogate Model

Using the linear system data "$Ax = b$" generated by the FEA as the input of our surrogate model has several advantages. First it standardizes the training data for all types of simulations input mesh and boundary conditions. Surrogate models are often trained with specific conditions dataset which makes difficult the generalization to other types of analysis. As the stiffness matrix A and the forcing vector b are constructed in a similar way whatever the simulation input data is, our proposal can be used for any types of FEA-based simulations such as stress, flow or heat conduction analysis. Another merit of using input A and b for predicting x is the confidence we have in the ability of our predictive model for solving the inverse problem $x = A^{-1}b$ and providing satisfying accuracy. At the inference phase, our proposal is designed to predict the linear solution from the FEA output within a chosen NR iteration. Because the FEA data is generated by the simulation itself, our mechanism avoids a typical error accumulation as it is often the case when a surrogate model is fed by its own previous prediction.

4.3 Understanding the FEA data

During the simulation, each NR iteration relies on the FEA to generate a new stiffness matrix A and forcing vector b. The stiffness matrix encodes the topology of the input mesh, the relationship between the nodes as well as the material properties. On the other hand, the forcing vector encodes the applied boundary conditions (external forces f) on the structure. Figure 3 illustrates an example of A and b generated from a simple 3D-mesh composed of a single 3-nodes triangle element. In such case, the matrix A is like the local stiffness matrix corresponding to that element. Given the number of nodes n (equals 3 here) and the number of degree of freedom dof (equals 3 also because we work in 3D), such matrix can be split into n^2 blocks of "$n * dof$" com-

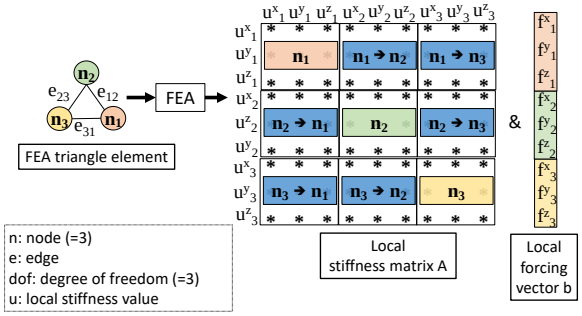

Fig. 3 Example of linear system data generated by FEA from a simple 3-nodes triangle element in 3D space.

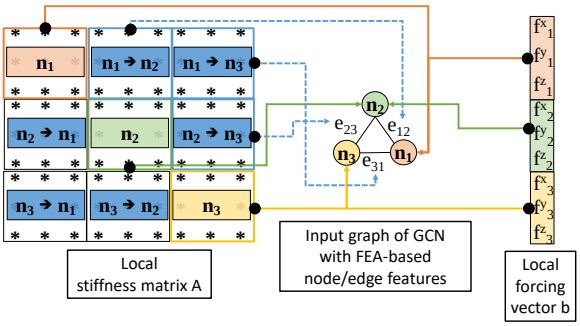

Fig. 4 The mapping from the FEA data to the input graph of GCN which defines the FEA-based node/edge features.

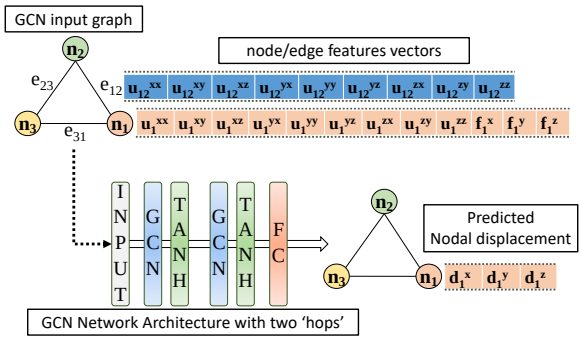

Fig. 5 GCN graph with FEA-based nodes/edges features as input of GCN network for predicting SA simulation results.

ponents. The diagonal blocks represent the local stiffness data of each individual node in the mesh, while the symmetric off-diagonal blocks represent the stiffness relationship between each pair of nodes. Because FEA mesh are not full-connected graphs, a large number of the off-diagonal blocks are filled in with zeros, explaining why the stiffness matrix is sparse. The definition of the forcing vector is much simpler: it contains the 3D components (x,y,z) of the external boundary conditions for each individual node. Besides the matrix sizes, the construction of A and b for larger mesh and different types of elements is similar.

4.4 Mapping the FEA data to the GCN graph

In this paper, we generate the GCN input graph identical to the FEA mesh. For learning, we must associate to each node and edges an input features vector. Choosing the appropriate set of features is an important step of the surrogate model design. However, there is no standard rules which help to make this choice as it often depends on the targeted task. This paper proposes a mapping which is a standard and a natural choice for building a surrogate model for FEA simulations. From the definition of matrix A and vector b, we slice the FEA matrix data into a set of blocks which are either related to a node or an edge of the GCN graph. As shown in Figure 4, the stiffness matrix diagonal blocks, along with the forcing vector components, are assigned to theirs corresponding nodes and become the "node features". Instead, the stiffness matrix off-diagonal blocks are allocated to theirs corresponding edges and become the "edge features". Such mapping translates the stiffness data from the regular matrix to a graph representation and makes possible the GCN training.

4.5 GCN: Training Data and Architecture

The Figure 5 describes an example of the node and edge features vectors generated by our mapping. When generating the training data, we associate each GCN input graph with a label graph which encodes the FEA simulation results. In this label graph, each node is associated with a 3D features vector representing the nodal displacement values computed by the simulation solver. Our GCN model is fed by these pairs of graphs and trained in a supervised manner to predict the displacement field over the output graph. The architecture of the GCN model is composed of three types of layer. First the GCN layer applies the convolution on each node and outputs a new set of features as explained in Section 2.5.3. After each GCN layer, we use a hyperbolic tangent as we want to learn a nonlinear function which can outputs real numbers. Finally, we predict the 3D displacement vectors by using a full-connected layer without additional activation. The number of "GCN-tanh" blocks inside the neural network is an important hyperparameter which controls how far the node features are propagated to distant neighbours. For example, the model in Figure 5 propagates each local node features "two hops" away because two blocks are used. Section 5 explains in more details the impact of this hyperparameter on the model performance.

5. Experimental Evaluation

In this section, we explain the experiments we conducted to evaluate the performance of our proposed surrogate model. We also discuss the scalability limitation when using the raw FEA data and describe the improvements we obtained when incorporating the PCA in our method.

5.1 Experimental Environment

All experiments were conducted on a dual-socket machine with two 2.1 GHz 18-core (36-thread) Intel Xeon E5-2695 v4 processors and 8 16GiB DDR4 memory modules (128 GiB total). The machine is equipped with an Intel NVMe SSD 750 Series with a storage capacity of 1.2 TiB and two Nvidia Tesla P100 GPUs with a memory capacity of 16 GiB. The installed operating system is CentOS 7.2 with Linux kernel v3.10. For FEA structural analysis simulation, we use FrontISTR v5.2 built with Intel compilers v18.0 and linked with Intel MKL and Lapack external libraries. For GCN training and inference, we use PyTorch Geometric v1.7 installed with Python 3.7.9 and Cuda 10.2.

5.2 Application models and Training phase

To evaluate the performance of our proposal, we started by generating different sizes of FrontISTR cantilever models going from 28 to 1035 nodes as shown in Figure 7. After that, we evaluated the limitation of GCN propagation and the benefits of the PCA representation by experimenting on larger cantilever versions of size 2238 and 4473 nodes as

problem size	epochs	training time
28	50	15.9 sec
60	50	16.01 sec
130	50	16.45 sec
260	500	165.76 sec
516	500	195 sec
1035	1000	760.61 sec
2238	1000	2430 sec
4473	1000	11 hours
9162	1000	1.35 days
14119	1000	3 days

Table 1 Problem size (number of nodes), epochs and training time. The GCN surrogate model is trained using PCA dataset, one GCN layer and a single GPU.

shown in Figure 9. Finally, we evaluated our proposal on two additional models, a wheel model (9k nodes) from the ABC dataset and a plastic can model (14k nodes) from the FrontISTR tutorial. For the cantilever models, the simulation keeps the leftmost area fixed and applies an external load on the rightmost area so that the structure is bent down. The wheel model is pushed from top to down while the plastic can is pulled out from right to left. For each application models, we generate 200 training samples by varying the external load intensity in some range depending on the structure dimension. Then we use these samples to train a GCN model as illustrated in Figure 5. The training time depends on the problem size and the number of epochs that we used for obtaining accurate enough prediction results. We summarized such information in the Table 1.

5.3 Node features matrix representations

As explained in Section 4, we generate the graph features from the FEA data A and b. However, because the GCN layer cannot handle edge features processing, we implemented a "node features only" version of the Figure 4 mapping by concatenating the edge features with the corresponding node features. In such case, the obtained node features vector incorporates both the targeted node stiffness data (diagonal block) and the stiffness relationship with all the other nodes (the off-diagonal blocks of the corresponding "row"). As illustrated in the top-middle image of Figure 6, we reshaped the n $3*3$ blocks into a single 1D vector and obtain a node features matrix that we call 'original data'. In this representation, each node (row) has 9n+3 features (columns). The "original data" column of Table 2 shows the evolution of the features number with increased problem sizes. As the dimension of stiffness matrix grows with O(9n) complexity, it becomes not practical to use the 'original data' representation for GCN training (in such case, the GPU memory becomes the bottleneck). To overcome this problem, our first approach is to consider only the nonzero values of the (sparse) stiffness matrix A.

5.4 The nonzero representation

When we remove the zeros from the 'original data', we obtain the 'nonzero' representation shown in the top-right image of Figure 6. In this case, the new number of node features "nz" is equal to the maximum number of nonzero

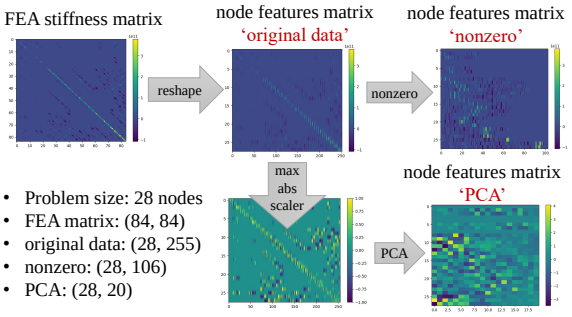

Fig. 6 Three possible representations of the GCN node features matrix generated from FEA data.

problem size	original data	nonzero data	PCA data
28	255	106	20
60	543	127	42
130	1173	120	89
260	2343	138	173
516	4647	138	341
1035	9318	156	692
2238	20145	156	1511
4473	40260	—	2998
9162	82461	—	6546
14119	127071	—	9527

Table 2 Comparison of the node features vector size for each data representations. Data type is 64-bit floating point.

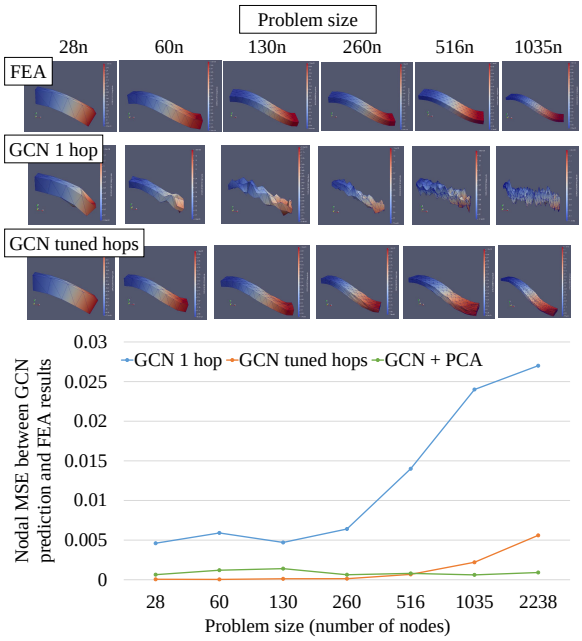

Fig. 7 Using nonzero data: comparison between FEA (baseline), GCN 1 hop and GCN with tuned hops (propagation).

elements that a single node can have. Because all nodes must have the same number of features, we use zero-padding to artificially extend the size of the smaller vectors up to "nz". As shown in the 'nonzero data' column of Table 2, this method drastically reduces the number of node features which does not proportionally increase with the problem size anymore. Indeed, the number "nz" is generated by the node which has the highest degree (more connections in the graph means more nonzero values in the matrix) and not by the graph size.

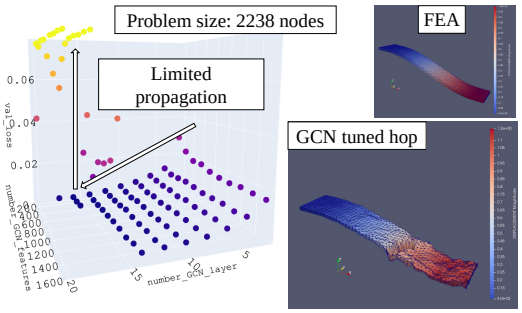

Fig. 8 GCN propagation limitation for larger graph.

5.5 GCN propagation

From the cantilever simulations, we generate a training dataset based on the nonzero representation. For each problem size, we train a GCN model and evaluate its prediction accuracy using an input Dirichlet intensity of -1cm. The set of images in Figure 7 shows for each problem size the difference between the FEA baseline (top row), the GCN prediction using a single "gcn-tanh" block (middle-row) and the GCN prediction using a tuned number of blocks (bottom-row). By looking at the "GCN 1 hop" error curve, we see that using a single hop is not enough for obtaining a good accuracy as the nodal mean square error increases with the problem size. If we tune the number of hops, as well as the GCN number of output features, we can improve the prediction accuracy, at least for small problem sizes. However, the "GCN tuned hops" error curve starts to increase from the 1035 nodes graph and shows the limits of GCN propagation by stacking multiple layers. Figure 8 illustrates this limitation with a 2238 nodes model. As the graph size increases, we must stack more GCN layers to propagate the information to more distant hops. However, we observed an empirical threshold after 15 layers from which the validation loss increases abruptly. Because of such limit, it becomes difficult to achieve good prediction accuracy with larger graph. To overcome this limitation, we proposed another data representation for the node feature vectors which incorporates more global information instead of local ones.

5.6 PCA representation

When it comes to large input graph, GCN-based surrogate models using FEA data must deal with two main challenges: scalability and propagation of information. As the nonzero representation can only solve the first problem, we propose a better representation to deal with the propagation limitation. As shown in Figure 6, we process the FEA 'original data' on a different path and generate a new representation that we call 'PCA'. First, we use the max absolute scaler formula:

$$\tilde{x}_i = x_i / absolute(x_{max}) \quad (2)$$

for normalizing each feature vector in the [-1,1] range without destroying any sparsity of the original data (the zeros are not modified). Such normalization makes all features share the same standard deviation which is a requirement

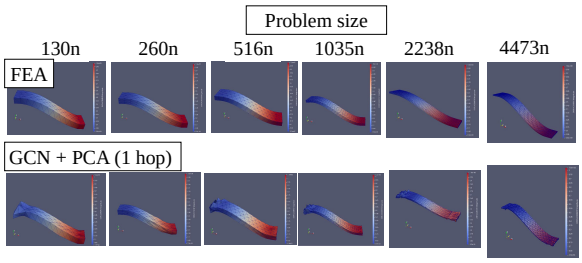

Fig. 9 Using PCA data: comparison between FEA (baseline) and GCN prediction. MSE is always below 0.001

for computing a relevant projection of the dataset into a smaller dimension space. To do so, we apply the PCA on our scaled FEA data and generate a reduced set of feature vectors which has the merit of preserving a large part of the original dataset variance. For example, Table 2 third column shows that a features matrix, coming from a 9162 nodes graph, can be reduced from 82k to a 6k features while 95% of the original information are retained. This percentage value is an input parameter that we use to indicates the PCA how many eigenvectors (dominant modes) it should extract for computing the new set of feature vectors. Because the extraction prioritizes eigenvectors with the largest eigenvalues, the PCA computed features incorporate a large part of the original dataset variance. The lowest error curve of Figure 7 and the images of Figure 9 show the performance of GCN model predictions which have been trained using a PCA-reduced dataset and only one GCN layer in the neural network. Because PCA computes global feature vectors before the training, a propagation using stacked GCN layers is not required anymore which leads to an easier and simpler neural network architecture design. Using a single GCN layer, better prediction accuracy can be obtained with larger models compared with previous data representation.

5.7 Acceleration of design optimization

In this work, we aim to accelerate design optimization for engineers and designers. In the design workflow, several instances of a simulation are run, using a fixed structure mesh but with different boundary conditions and/or material properties so that the designers can understand the behaviour of the structure. Because different conditions or material properties do not affect the stiffness matrix data, the PCA in our proposal must be computed only once for the targeted structure. By eliminating the need for new PCA computation on the stiffness matrix (only normalization for the new forcing vector is required), our proposed surrogate model can be used to replace the solver computation for each NR iteration and thus can provide a significant speed-up.

5.8 Speed-up of our proposal

The Figure 10 shows a comparison of the execution time of a single NR iteration when it is computed by the CG-AMG solver using 36 threads versus when the results is directly inferred by our proposal using GPU. The proposal elapsed time incorporates the loading of stored PCA results, the

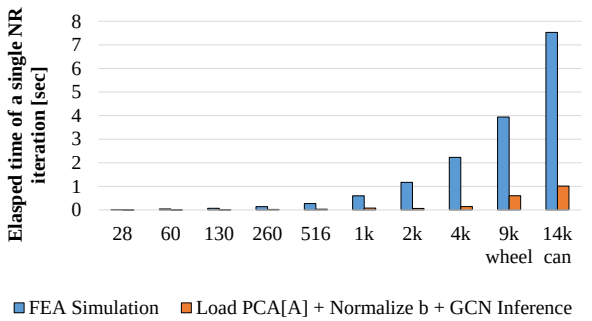

Fig. 10 GCN speed-up using different problem size models.

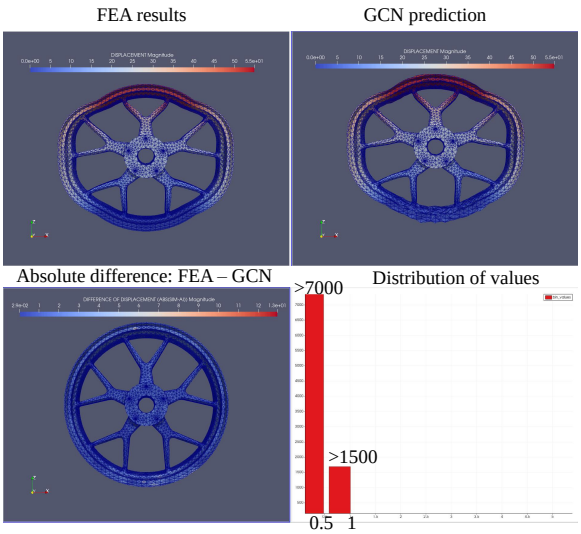

Fig. 11 The ABC Wheel 9k nodes model: FEA vs. GCN

normalization of the new vector b for the NR iteration and the GCN inference itself. The proposal time does not incorporate the PCA computation time (which for example takes approximatively one hour for the 9k wheel model), as it is computed only once and reused for different simulation input conditions. Also, the GCN inference time does not incorporate the training time which is described in Section 5.2. The obtained average speed-up for all the models that we analysed is around 7 times faster than traditional NR iteration. In the case of nonlinear analysis, where many NR iterations are required, such speed-up can be accumulated and thus it can lead to drastic acceleration of the overall simulation time. The Figures 11 and 12 show a comparison of the structure deformations computed by the FEA versus the GCN prediction for the Wheel and Can models. The image and bar graph of the nodal difference values shows that GCN prediction is accurate on a large part of the simulation model.

6. Conclusion

Our proposed surrogate model can accelerate any simulation based on finite element method such as structural analysis or computational fluid dynamics. We mapped the FEA data into the input graph of GCN and created a standard way to train an AI prediction model which works for any input mesh and any boundary conditions. Using the PCA to represent our feature vectors, we overcome the propagation limitation of GCN and achieved scalability with larger graphs. We applied our method to structural analysis models (up to 14k nodes) and achieved both a significant speed-up of NR iteration computation (an average of x7 speed-up for all models) and high accuracy results with 1e-03 precision. In future work, we plan to demonstrate similar results with larger model sizes using distributed GCN training.

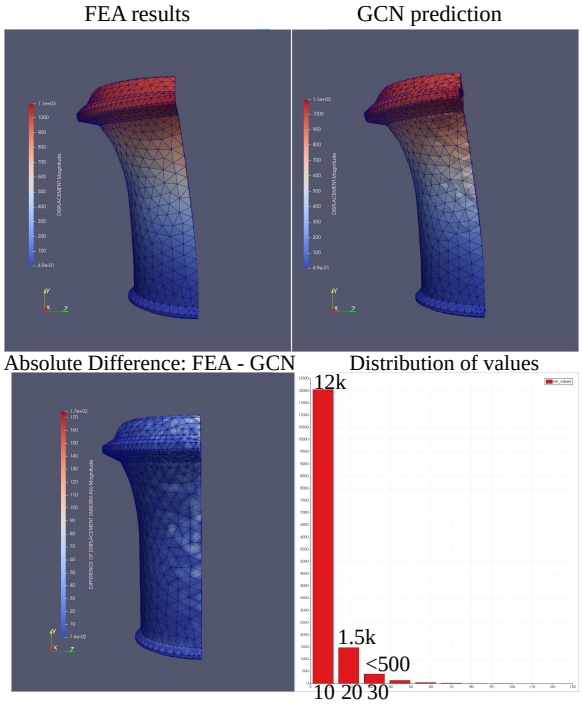

Fig. 12 The FrontISTR Can 14k nodes model: FEA vs. GCN

References

[1] Adie, J., Juntao, Y., Zhang, X. and See, S.: Deep Learning for Computational Science and Engineering.

[2] Frank, M., Drikakis, D. and Charissis, V.: Machine-Learning Methods for Computational Science and Engineering, *Computation*, Vol. 8, No. 1 (online), DOI: 10.3390/computation8010015 (2020).

[3] Battaglia, P. W., Hamrick, J. B., Bapst, V., Sanchez-Gonzalez, A., Zambaldi, V. F., Malinowski, M., Tacchetti, A., Raposo, D., Santoro, A., Faulkner, R., Gülçehre, Ç., Song, H. F., Ballard, A. J., Gilmer, J., Dahl, G. E., Vaswani, A., Allen, K. R., Nash, C., Langston, V., Dyer, C., Heess, N., Wierstra, D., Kohli, P., Botvinick, M., Vinyals, O., Li, Y. and Pascanu, R.: Relational inductive biases, deep learning, and graph networks, *CoRR*, Vol. abs/1806.01261 (online), available from ⟨http://arxiv.org/abs/1806.01261⟩ (2018).

[4] Bronstein, M. M., Bruna, J., LeCun, Y., Szlam, A. and Vandergheynst, P.: Geometric deep learning: going beyond Euclidean data, *CoRR*, Vol. abs/1611.08097 (online), available from ⟨http://arxiv.org/abs/1611.08097⟩ (2016).

[5] Zhou, J., Cui, G., Zhang, Z., Yang, C., Liu, Z. and Sun, M.: Graph Neural Networks: A Review of Methods and Applications, *CoRR*, Vol. abs/1812.08434 (online), available from ⟨http://arxiv.org/abs/1812.08434⟩ (2018).

[6] Hammond, D. K., Vandergheynst, P. and Gribonval, R.: Wavelets on Graphs via Spectral Graph Theory (2009).

[7] Defferrard, M., Bresson, X. and Vandergheynst, P.: Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering, *CoRR*, Vol. abs/1606.09375 (online), available from ⟨http://arxiv.org/abs/1606.09375⟩ (2016).

[8] Hamilton, W. L., Ying, R. and Leskovec, J.: Inductive Representation Learning on Large Graphs, *CoRR*, Vol. abs/1706.02216 (online), available from ⟨http://arxiv.org/abs/1706.02216⟩ (2017).

[9] Kipf, T. N. and Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks,

CoRR, Vol. abs/1609.02907 (online), available from ⟨http://arxiv.org/abs/1609.02907⟩ (2016).

[10] Behler, J. and Parrinello, M.: Generalized Neural-Network Representation of High-Dimensional Potential-Energy Surfaces, *Phys. Rev. Lett.*, Vol. 98, p. 146401 (online), DOI: 10.1103/PhysRevLett.98.146401 (2007).

[11] Chmiela, S., Tkatchenko, A., Sauceda, H. E., Poltavsky, I., Schtt, K. T. and Mller, K.-R.: Machine learning of accurate energy-conserving molecular force fields, *Science Advances*, Vol. 3, No. 5, p. e1603015 (online), DOI: 10.1126/sciadv.1603015 (2017).

[12] Giralt, F., Arenas, A., J, F.-G., Rallo, R. and Kopp, G.: The Simulation and Interpretation of Turbulence with a Cognitive Neural System, *Physics of Fluids*, Vol. 12, p. 1826 (online), DOI: 10.1063/1.870430 (2000).

[13] Zhang, W., Zhu, L., Kou, J. and Liu, Y.: Machine learning methods for turbulence modeling in subsonic flows over airfoils (2018).

[14] Ling, J., Kurzawski, A. and Templeton, J.: Reynolds averaged turbulence modelling using deep neural networks with embedded invariance, *Journal of Fluid Mechanics*, Vol. 807, pp. 155–166 (online), DOI: 10.1017/jfm.2016.615 (2016).

[15] Unger, J. F. and Knke, C.: Neural networks as material models within a multiscale approach, *Computers and Structures*, Vol. 87, No. 19, pp. 1177–1186 (online), DOI: https://doi.org/10.1016/j.compstruc.2008.12.003 (2009).

[16] Yamaguchi, T. and Okuda, H.: Prediction of stress concentration at fillets using a neural network for efficient finite element analysis, *Mechanical Engineering Letters*, Vol. 6, pp. 20–00318–20–00318 (online), DOI: 10.1299/mel.20-00318 (2020).

[17] Carleo, G. and Troyer, M.: Solving the quantum many-body problem with artificial neural networks, *Science*, Vol. 355, No. 6325, p. 602606 (online), DOI: 10.1126/science.aag2302 (2017).

[18] Mi, Y., Ishii, M. and Tsoukalas, L.: Flow regime identification methodology with neural networks and two-phase flow models, *Nuclear Engineering and Design*, Vol. 204, No. 1, pp. 87–100 (online), DOI: https://doi.org/10.1016/S0029-5493(00)00325-3 (2001).

[19] Rabault, J., Kuchta, M., Jensen, A., Rglade, U. and Cerardi, N.: Artificial neural networks trained through deep reinforcement learning discover control strategies for active flow control, *Journal of Fluid Mechanics*, Vol. 865, p. 281302 (online), DOI: 10.1017/jfm.2019.62 (2019).

[20] Poloni, C., Giurgevich, A., Onesti, L. and Pediroda, V.: Hybridization of a multi-objective genetic algorithm, a neural network and a classical optimizer for a complex design problem in fluid dynamics, *Computer Methods in Applied Mechanics and Engineering*, Vol. 186, No. 2, pp. 403–420 (online), DOI: https://doi.org/10.1016/S0045-7825(99)00394-1 (2000).

[21] Waszczyszyn, Z. and Ziemiaski, L.: Neural networks in mechanics of structures and materials new results and prospects of applications, *Computers and Structures*, Vol. 79, No. 22, pp. 2261–2276 (online), DOI: https://doi.org/10.1016/S0045-7949(01)00083-9 (2001).

[22] Fout, A., Byrd, J., Shariat, B. and Ben-Hur, A.: Protein Interface Prediction using Graph Convolutional Networks, *Advances in Neural Information Processing Systems* (Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S. and Garnett, R., eds.), Vol. 30, Curran Associates, Inc., (online), available from ⟨https://proceedings.neurips.cc/paper/2017/file/f507783927f2ec2737ba40afbd17efb5-Paper.pdf⟩ (2017).

[23] Kipf, T., Fetaya, E., Wang, K.-C., Welling, M. and Zemel, R.: Neural Relational Inference for Interacting Systems (2018).

[24] Sanchez-Gonzalez, A., Godwin, J., Pfaff, T., Ying, R., Leskovec, J. and Battaglia, P. W.: Learning to Simulate Complex Physics with Graph Networks (2020).

[25] Pfaff, T., Fortunato, M., Sanchez-Gonzalez, A. and Battaglia, P. W.: Learning Mesh-Based Simulation with Graph Networks, *CoRR*, Vol. abs/2010.03409 (online), available from ⟨https://arxiv.org/abs/2010.03409⟩ (2020).

[26] Horie, M., Morita, N., Hishinuma, T., Ihara, Y. and Mitsume, N.: Isometric Transformation Invariant and Equivariant Graph Convolutional Networks, *International Conference on Learning Representations*, (online), available from ⟨https://openreview.net/forum?id=FX0vR39SJ5q⟩ (2021).