Received: 13 October 2022 | Accepted: 8 November 2022

DOI: 10.1002/pamm.202200306

Graph Neural Network enhanced Finite Element modelling

Rutwik Gulakala[1], **Bernd Markert**[1], and **Marcus Stoffel**[1,*]

[1] Institute of General Mechanics (IAM), RWTH Aachen University, Aachen, Germany

In this study, we introduce a Graph network-enhanced Finite Element approach to accelerate Finite Element simulations. We utilize the discretized geometry from a Finite Element pre-processor to establish the graph and use the Graph Neural Network to solve the boundary value problem of the discretized domain. The advantage of graph neural networks is that they have a similar structure as compared to a discretized domain with nodes and elements. The underlying dynamics of the system are computed via a learned message-passing. The goal here is to enhance and accelerate the FEM simulations using the proposed GNN network by incorporating the underlying mechanics knowledge into the network to enhance the generalizing ability of the network on various loading and boundary conditions. All the proposed studies in the literature where graph networks are applied to Finite Element Methods use images as input and output. The advantage of the proposed model is that it takes inputs such as the nodal information, their corresponding edges, nodal coordinates and the boundary conditions for each particular node from a Finite Element pre-processor and computes the von-Mises stress at each node along with their edge connections as output that can be read by a Finite Element post-processor.

1 Introduction

Finite Element simulations have become an integral part of product development and are widely used in academics and industry. This numerical method can, depending on the complexity of the boundary value problem (BVP), lead to computationally extensive and complex simulations. Often their high computational cost is brought down by using reduced order models or surrogates, but at the expense of accuracy [1–6]. Crash simulations, for instance, are one of the more complicated problems solved using FE methods that have high computational costs and consume a lot of time [7].

Machine learning-based algorithms are gaining a lot of momentum for accelerating the computation time of structural simulations. Deep learning methods have demonstrated excellent results when utilized to speed up the physical simulations and learn the underlying physics without prior knowledge of the mechanical model [8–10]. In [11, 12], CNN-based models have been used to accelerate FE simulations without penalizing accuracy. Also, CNN-based models have been extensively used in solid mechanical simulations for inhomogeneous non-linear materials [13] and in continuum mechanical simulations [14].

The success in deep learning has emerged from finding the right structural inductive bias to apply to the right network architecture. The success of CNN models in modelling FE simulation results from their translation invariance, weight sharing and locality which are desirable inductive biases for these FE or continuum based models. That said, CNNs have a huge drawback concerning the computational domain. The input and output fields can only be defined on a structured rectangular domain. Any complex structures on an unstructured domain are not suitable to be modelled using a CNN. This is where Graph Neural Networks (GNNs) come in. They perform computations over the graph and are modelled based on relational inductive bias. Since they don't have any limitation of structure and are only governed by the set of entities and their relations, GNNs can be used to model complex and unstructured domains. In [15], Graph Network-based simulators (GNS) were used to simulate various physical phenomena and GNNs to predict fluid flow over a cylinder were proposed in [16, 17]. A Graph element network was proposed by [18] to model spatial processes with no prior graph structure such as images.

Images have been predominantly used in the literature as input and output to the graph networks to model the physical problem. In this study, we take the discretized BVP from an FE pre-processor with inputs such as the nodal information, their corresponding edges, nodal coordinates and the boundary conditions for each particular node from a Finite Element pre-processor and compute the von-Mises stress at each node along with their edge connections as output that can be read by a Finite Element post-processor.

2 Generation of data from FE simulations

To train the graph network, a rectangular plate with different dimensions and with a hole at various locations and sizes is considered. The plate is simulated using aluminium with elastoplastic material behaviour using the isotropic Johnson-Cook model with the following material properties [19] as shown in Table 1.

* Corresponding author: stoffel@iam.rwth-aachen.de

Density (g/cm3)	**Young's modulus (GPa)**	**Poisson's ratio (-)**	A (MPa)	B (MPa)	$\dot{\epsilon}_0$ ($^1/_s$)	C
2.7	70	0.33	520	477	5E-4	0.001

Table 1: Material parameters.

The istropic elasto-plastic Jonhson-Cook model is given by

$$\sigma^0 = [A + B(\hat{\epsilon}^{pl})^n](1 - \hat{\theta}^m). \qquad \bar{\sigma} = \sigma^0(\bar{\epsilon}^{pl}, \theta)R(\dot{\bar{\epsilon}}^{pl}). \tag{1a}$$

$$\dot{\bar{\epsilon}}^{pl} = \dot{\epsilon}_0 \exp\frac{1}{c}(R - 1). \qquad \bar{\sigma} = [A + B(\hat{\epsilon}^{pl})^n][1 + C \ln\frac{\dot{\bar{\epsilon}}^{pl}}{\dot{\epsilon}_0}](1 - \hat{\theta}^m). \tag{1b}$$

Where $\bar{\sigma}$ is the yield stress at nonzero strain rate, $\hat{\theta}$ is the nondimensional temperature, $\dot{\bar{\epsilon}}^{pl}$ is the equivalent plastic strain rate, $\dot{\epsilon}_0$ and C are material parameters measured from experiments, $\sigma^0(\bar{\epsilon}^{pl}, \theta)$ is the static yield stress and $R(\dot{\bar{\epsilon}}^{pl})$ is the ratio of the yield stress at nonzero strain rate to the static yield stress.

The dimensions of the plate are varied between 0.2 to 0.4 m with equal and unequal side lengths. Dirichlet boundary conditions are applied on the left end of the plate where all its degrees of freedom are constrained and a displacement is applied on the right end of the plate. The displacement is varied from 0.01 to 0.1 m with an increment of 0.1 m in each case. A hole is modelled inside the plate at various locations and sizes. This resulted in 6 different cases which constitute 9 plates for each case and 10 different loads. Fig. 1 shows the 6 cases of plate with holes.

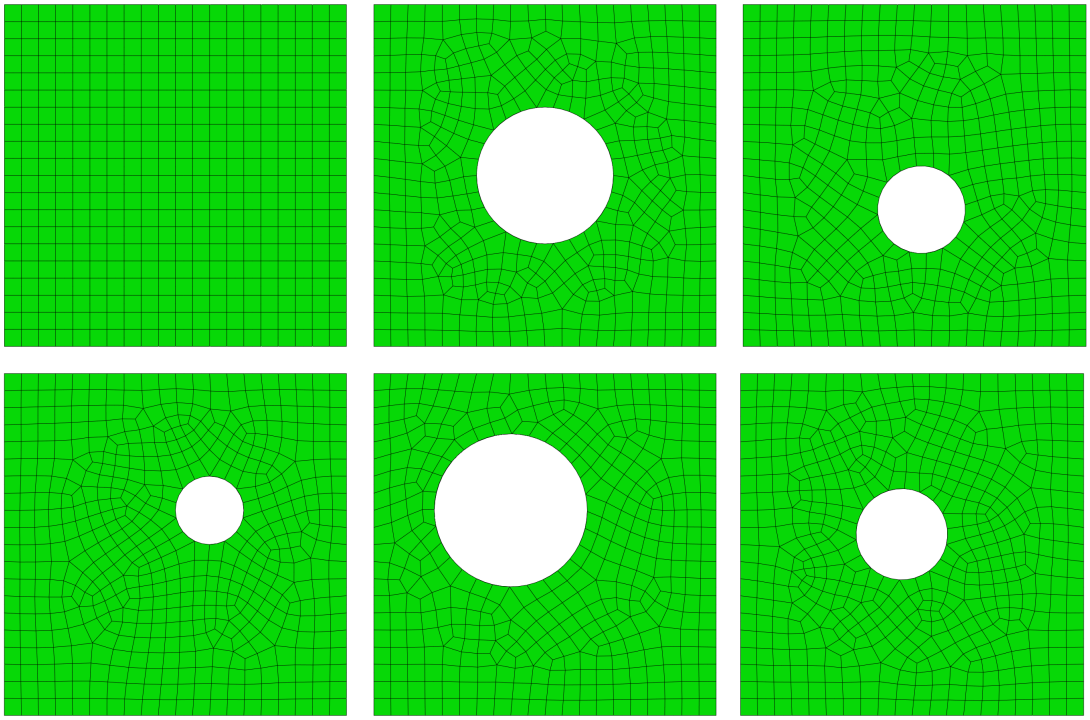


Fig. 1: Pre-processor images of the 6 cases considered for data generation.

A Finite Element solver, ABAQUS, is used to generate the data required for the Graph networks and the whole process is automated using Python scripting. Since the finite element solver doesn't output the results in an open-source format, an ODB2VTK interface is used to convert the ODB files into VTK format that is open-source and is imported into Python for training the Graph network. From the discussed input combinations, a total of 540 samples are generated.

3 Graph network

Graph neural networks (GNNs) perform computations over a graph, based on the idea that entities exist and exhibit certain relations. The nodes and edges of the FE domain are modelled as the nodes of the graph and their relationship. Graph networks has been widely used in social sciences [20], natural sciences (physical systems [15, 21], protien-protien interaction [22] and drug discovery [23]), knowledge graphs [24] and NP-hard algorithms [25].

Graphs can be usually classified as Directed/Undirected, Homogeneous/Heterogeneous and Static/Dynamic. In this study, we use an undirected, homogeneous and static graph architecture. An Encoder-Decoder architecture is chosen to encode the information into a latent vector and reconstruct the output of the succeeding state.

The discretized data from the pre-processor is passed into the encoder of the graph. The input is embedded with material data and nodal features such as boundary conditions and spatial coordinates. The future state of the nodal feature vector h is computed by considering the influence of its neighbours. This is done using an aggregation function. In the current study, we use a transformer convolution layer as an aggregation function and a multi-layer perceptron (MLP) as a message-passing function. The aggregation of the node feature vector, subsequent message passing and update are mathematically described using the following equations.

$$G = (h_i, e_{ij}). \tag{2a}$$

$$\alpha_{ij} = \text{softmax}_j\left(\frac{(W_\alpha^k h_i^k)^T (W_\beta^k h_j^k)}{\sqrt{d}}\right). \tag{2b}$$

$$h_i^{k+1} = U^k(h_i^k, \alpha_{ij}^k(h_j^k)). \tag{2c}$$

$$h_i^{k+1} = MLP_\theta^k(h_i^k, \alpha_{ij}^k(h_j^k)). \tag{2d}$$

Where h_i is the node feature vector of the node being updated, h_j is the node vector of the neighbouring node, e_{ij} is the edge tensor, U is the message-passing or the state update function, α is the aggregation function and $W_{\alpha,\beta}$ are trainable linear weights.

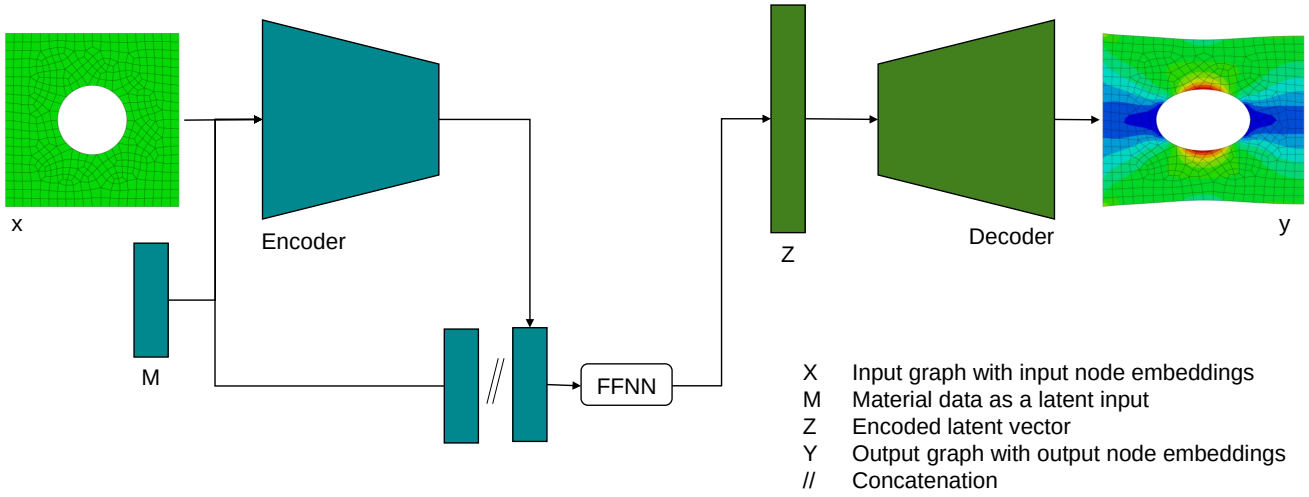


Fig. 2: Distribution of data over different classes.

The encoder which constitutes the aggregation function and message-passing encodes the input into a latent vector which is concatenated with material data as shown in Fig. 2. The latent vector is then passed through a learned FFNN that has information about the dynamics of the problem and the resultant latent vector is decoded in the output node feature vector using the decoder. The decoder constitutes an MLP network that learns to reconstruct the latent vector into an output node feature vector.

4 Network training and hyperparameter tuning

Out of 540 samples of training data, 500 samples are used for training the proposed network. The network was trained for 1000 epochs. The network hyperparameters such as the number of transformer convolution layers, Number of units in latent vector and number of layers in the MLP network in the decoder have to be optimal to obtain efficient model performance. To achieve this, Hyperband search algorithm [26] is employed. The hyperparameter sampling domain and the selected hyperparameters are shown in Table 2

Hyperparameters	**Paremeterset**	**Selected Hyperparameters**
Transformer Convolution layers	[1,2,...,8]	3
Latent units	[4,8,16,...,128]	16
Number of MLP layers	[1,2,...,6]	5

Table 2: Hyperparameter tuning.

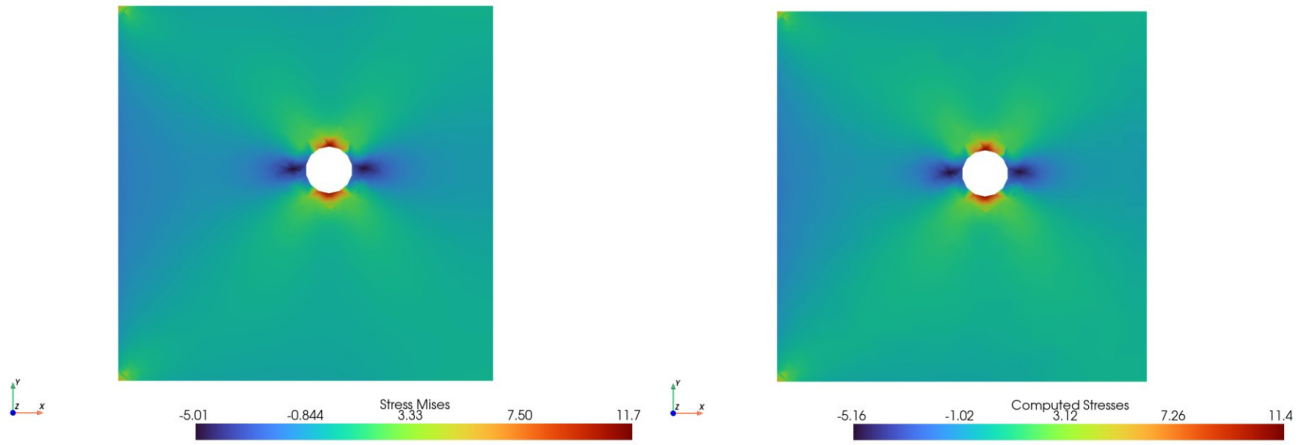
The network is optimized and trained using the Adam optimizer and mean squared error is used as the loss metric. A learning rate of 3E-3 is used with an exponential decay over epochs starting from the 200th epoch. This is done to eliminate the local minima that were being observed after 200 epochs into the training.

5 Results and Discussion

The network computed results are then embedded into the VTK file with a key named "Computed Stresses" to facilitate the comparison of the results in a FE post-processing environment. Here a comparison is drawn between the von-Mises stress distribution from the FE simulation with the Graph Network computed von-Mises stress distribution. A computational gain of 12.7% has been observed by using the Graph Network, also considering the computational effort for training the network. The results section is subdivided to discuss the performance of the graph on extrapolation results.

5.1 From the trained limit

The graph network is evaluated on the test set and an MSE error of $5.89E-2$ is achieved for the whole test batch, which roughly translated to an error of around $1.7E9-3.18E9$ and $2.15E10-2.9E10$ GPa for different input geometry and loading cases. The results from the Graph network show an excellent correlation to the FE simulations with a very small error in the magnitude. This shows that the Graph Networks can efficiently replicate FEM simulations, preserving the elemental integrity and overall structure.



(a) von-Mises stress distribution from FE simulation.

(b) von-Mises stress distribution from the proposed Graph network.

Fig. 3: Comparision of actual vs network generated von-Mises stress for case IV.

Fig. 4 shows the epoch vs loss plot of the trained graph network. The spikes in the loss that start around 200 epochs are successfully suppressed by the learning rate decay and are much less significant as a result.

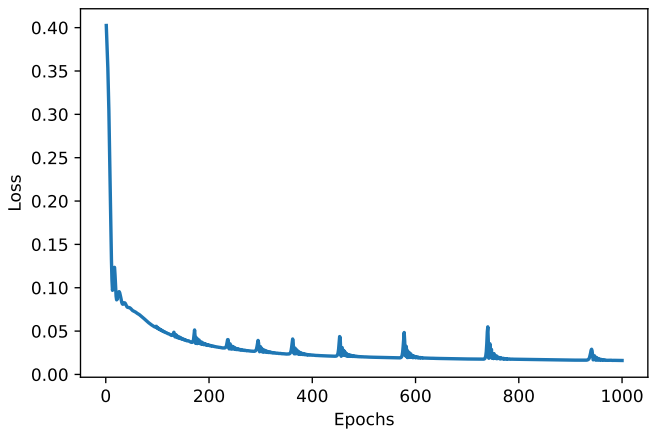
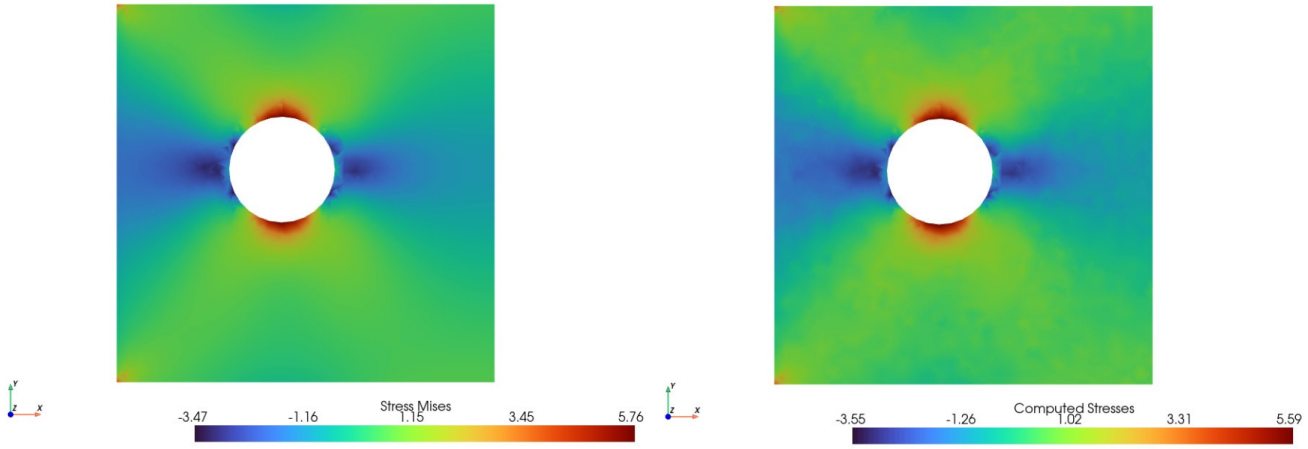


Fig. 4: Loss vs Epochs from training.

5.2 From the extrapolation

The extrapolation ability of the proposed network is tested on two test samples as shown in Figs. 5 and 6. The geometry and the loading condition are taken out of the training set and the samples are evaluated using the trained Graph Network. In Fig. 5, although it looks identical, some artefacts can be observed in the stress distribution over the entire domain. This example is taken with a square plate of side length 0.5 m (where the maximum side length of the training set is 0.4 m) and a displacement of 0.125 m applied at the right end of the plate (maximum displacement applied in training is 0.1 m). The network although is able to predict the stress distribution and magnitude very well, it is not as accurate as that of the test samples with parameters in the bounds of the training range.

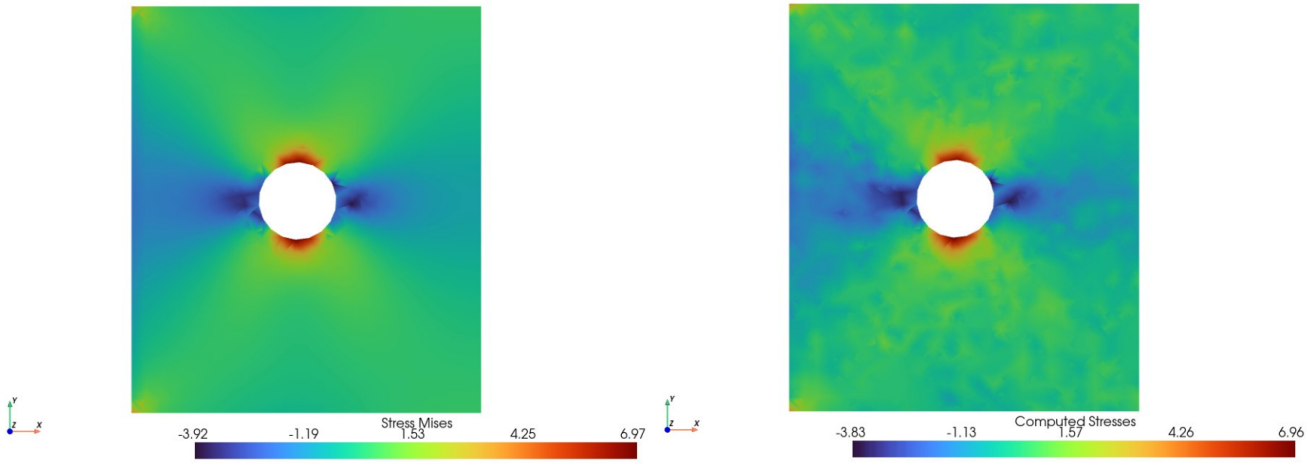
In Fig. 6, it can be observed that the computed stress distribution obtained from the Graph network in no way resembles the actual von-Mises stress distribution obtained from FE simulations. Here we encounter the bottleneck of the proposed architecture. This example is taken with a rectangular plate of side length $0.65, 0.45$ (where the maximum side length of the training set is 0.4 m) and a displacement of 0.15 m applied at the right end of the plate (maximum displacement applied in training is 0.1 m). The network was able to perform satisfactorily until the test samples are well within 30% of excess of the training set. When the geometry and load cases exceed this limit, the network fails to give a meaningful output.



(a) von-Mises stress distribution from FE simulation.

(b) von-Mises stress distribution from the proposed Graph network.

Fig. 5: Comparision of actual vs network generated von-Mises stress for case II.



(a) von-Mises stress distribution from FE simulation.

(b) von-Mises stress distribution from the proposed Graph network.

Fig. 6: Comparision of actual vs network generated von-Mises stress for case VI.

Although graph networks have the ability to extrapolate, it is limited, in this case by the lack of physical knowledge in the network. Also, even if the geometry is slightly enlarged, the domain discretization changes by adding much more nodes and the adjacency matrix becomes much larger. When this happens in a static graph, after a certain limit, the network no longer has the ability to predict meaningful information, although the node connectivity and overall structure are preserved. These

limitations can be overcome by incorporating physical information by using a physics-informed graph network and also using dynamics graphs.

6 Conclusion

In the current study, we deploy a Graph network enhanced Finite Element model, to compute the final deformations and von-Mises stress distribution that can be read by an FE postprocessor from a discretized FE Preprocessor input. The proposed architecture is based on an undirected, static and homogeneous graph structure. A data-driven approach is employed to train the model and exploit the relational inductive bias of the Graph Neural Networks. A plate with a hole with various dimensions, positions and boundary conditions is simulated using an FE-Solver which is used for training the model. The trained graph networks show a strong correlation with FE simulations and has the ability to extrapolate the data outside of training bounds to an extent. The model's extrapolation ability deteriorates when the data exceeds 30% over the training bounds and cannot be used for any meaningful predictions. The next steps will be to overcome the limitations of the current architecture by proposing a hybrid, data+physics-driven model, with a dynamic graph architecture. That said, the proposed architecture has a very low error relative to that of FEM (2.78%) and can generate results that can be read by an FE Postprocessor in which the novelty of the study lies.

Acknowledgements The authors gratefully acknowledge the financial support provided by Deutsche Forschungsgemeinschaft Priority Programme: SPP 2353 (DFG Grant No. STO 469/16-1). Open access funding enabled and organized by Projekt DEAL.

References

[1] E. B. Rudnyi and J. G. Korvink, Model order reduction for large scale engineering models developed in ansys, in: Applied Parallel Computing. State of the Art in Scientific Computing, edited by J. Dongarra, K. Madsen, and J. Waśniewski (Springer Berlin Heidelberg, Berlin, Heidelberg, 2006), pp. 349–356.
[2] S. Wang, Y. Wang, Y. Zi, and Z. He, Journal of Sound and Vibration **359**, 116–135 (2015).
[3] S. Wulfinghoff, F. Cavaliere, and S. Reese, Computer Methods in Applied Mechanics and Engineering **330**, 149–179 (2018).
[4] S. van Ophem, O. Atak, E. Deckers, and W. Desmet, Computer Methods in Applied Mechanics and Engineering **325**, 240–264 (2017).
[5] J. Kůdela and R. Matousek, Soft Computing(07) (2022).
[6] J. G. Hoffer, B. C. Geiger, P. Ofner, and R. Kern, Applied Sciences **11**(20) (2021).
[7] F. Duddeck, Structural and Multidisciplinary Optimization **35**(01), 375–389 (2008).
[8] M. Stoffel, R. Gulakala, F. Bamer, and B. Markert, Computer Methods in Applied Mechanics and Engineering **364**, 112989 (2020).
[9] M. Stoffel, F. Bamer, and B. Markert, Mechanics Research Communications **95**, 85–88 (2019).
[10] S. B. Tandale, B. Markert, and M. Stoffel, Computer Methods in Applied Mechanics and Engineering **401PB** (2022).
[11] A. Koeppe, F. Bamer, and B. Markert, Computer Methods in Applied Mechanics and Engineering **366**, 113088 (2020).
[12] Q. Tan, Z. Pan, L. Gao, and D. Manocha, IEEE Robotics and Automation Letters **5**(2), 2325–2332 (2020).
[13] J. Mianroodi, N. Siboni, and D. Raabe, npj Computational Materials **7**(07), 99 (2021).
[14] S. Ye, Z. Zhang, X. Song, Y. Wang, Y. Chen, and C. Huang, Scientific Reports **10**(03) (2020).
[15] A. Sanchez-Gonzalez, J. Godwin, T. Pfaff, R. Ying, J. Leskovec, and P. W. Battaglia, Learning to simulate complex physics with graph networks, 2020.
[16] M. Lino, C. Cantwell, A. A. Bharath, and S. Fotiadis, Simulating continuum mechanics with multi-scale graph neural networks, 2021.
[17] F. de Avila Belbute-Peres, T. D. Economon, and J. Z. Kolter, Combining differentiable pde solvers and graph neural networks for fluid flow prediction, 2020.
[18] F. Alet, A. K. Jeewajee, M. Bauza, A. Rodriguez, T. Lozano-Perez, and L. P. Kaelbling, Graph element networks: adaptive, structured computation and memory, 2019.
[19] E. Flores-Johnson, L. Shen, I. Guiamatsia, and G. Nguyen, Composites Science and Technology **96**(05), 13–22 (2014).
[20] Y. Wu, D. Lian, Y. Xu, L. Wu, and E. Chen, Proceedings of the AAAI Conference on Artificial Intelligence **34**(04), 1054–1061 (2020).
[21] P. W. Battaglia, R. Pascanu, M. Lai, D. Rezende, and K. Kavukcuoglu, Interaction networks for learning about objects, relations and physics, 2016.
[22] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, Advances in neural information processing systems **30** (2017).
[23] M. Sun, S. Zhao, C. Gilvary, O. Elemento, J. Zhou, and F. Wang, Briefings in Bioinformatics **21**(3), 919–935 (2019).
[24] T. Hamaguchi, H. Oiwa, M. Shimbo, and Y. Matsumoto, arXiv preprint arXiv:1706.05674 (2017).
[25] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, Advances in neural information processing systems **30** (2017).
[26] L. Li, K. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, The Journal of Machine Learning Research **18**(1), 6765–6816 (2017).

