



HAL
open science

A neural network-based data-driven local modeling of spotwelded plates under impact

Afsal Pulikkathodi, Elisabeth Longatte-Lacazedieu, Ludovic Chamoin,
Juan-Pedro Berro Ramirez, Laurent Rota, Malek Zarroug

► **To cite this version:**

Afsal Pulikkathodi, Elisabeth Longatte-Lacazedieu, Ludovic Chamoin, Juan-Pedro Berro Ramirez, Laurent Rota, et al.. A neural network-based data-driven local modeling of spotwelded plates under impact. *Mechanics & Industry*, 2023, 24, pp.34. 10.1051/meca/2023029 . hal-04190457

HAL Id: hal-04190457

<https://hal.science/hal-04190457>

Submitted on 15 Sep 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A neural network-based data-driven local modeling of spotwelded plates under impact

Afsal Pulikkathodi^{1,*}, Elisabeth Lacazedieu^{1,2}, Ludovic Chamoin^{1,3}, Juan Pedro Berro Ramirez⁴, Laurent Rota⁵, and Malek Zarroug⁵

¹ Université Paris-Saclay, CentraleSupélec, ENS Paris-Saclay, CNRS, LMPS – Laboratoire de Mécanique Paris-Saclay, 91190 Gif-sur-Yvette, France

² EPF School of Engineering, 94230 Cachan, France

³ IUF, Institut Universitaire de France, Paris, France

⁴ Altair Engineering France, 92160 Antony, France

⁵ Stellantis, 78140 Velizy-Villacoublay, France

Received: 28 February 2023 / Accepted: 27 July 2023

Abstract. Solving large structural problems with multiple complex localized behaviors is extremely challenging. To address this difficulty, both intrusive and non-intrusive Domain Decomposition Methods (DDM) have been developed in the past, where the refined model (local) is solved separately in its own space and time scales. In this work, the Finite Element Method (FEM) at the local scale is replaced with a data-driven Reduced Order Model (ROM) to further decrease computational time. The reduced model aims to create a low-cost, accurate and efficient mapping from interface velocities to interface forces and enable the prediction of their time evolution. The present work proposes a modeling technique based on the Physics-Guided Architecture of Neural Networks (PGANNs), which incorporates physical variables other than input/output variables into the neural network architecture. We develop this approach on a 2D plate with a hole as well as a 3D case with spot-welded plates undergoing fast deformation, representing nonlinear elastoplasticity problems. Neural networks are trained using simulation data generated by explicit dynamic FEM solvers. The PGANN results are in good agreement with the FEM solutions for both test cases, including those in the training dataset as well as the unseen dataset, given the loading type is present in the training set.

Keywords: Artificial neural networks / data-driven modelling / local/global coupling / explicit dynamics / physics-guided architecture

1 Introduction

High-fidelity simulations of large structures containing many localized features such as spotwelds [1] or bolted joints are still a major scientific and industrial challenge. The mechanical behavior of these localized features has traditionally been addressed using simplified models to circumvent the complexities associated with numerical analysis procedures. For spotwelds, these models typically employ rigid or flexible beams with coincident nodes [2,3]. However, these simplified modeling approaches often fall short in accurately predicting failure. Therefore, in order to capture the complex and nonlinear behavior of these localized features accurately, refined 3D elements are necessary. Due to the space and time scale discrepancies between the global response of the structure and

the localized phenomena, such mesh refinements drastically increase the computational cost. This arises not only from the increase of the number of dofs but also from the requirement to reduce the time step to ensure the Courant–Friedrichs–Lewy (CFL) condition [4] of the explicit time integration process.

Numerous numerical methods devoted to multiscale computing have emerged over the last three decades to address this problem. They are mostly based on Domain Decomposition techniques such as the primal BDD method [5], dual FETI method [6], the mixed LATIN scheme [7], or the Arlequin framework [8]. These are well-suited to problems in which the refinement zones are fixed. In the case of rapidly evolving problems, non-intrusive domain decomposition techniques based on local/global approaches have proven to be effective (e.g., local plasticity [9,10], local crack propagation [11]). An overview of such techniques can be found in [12]. They allow the global mesh to remain unchanged while the local problem

* e-mail: afsal.pulikkathodi@ens-paris-saclay.fr

is refined in space and time. The non-intrusive local/global strategy relies on an iterative exchange of interface quantities between global and local computations. It was successfully applied to coupling of 2D and 3D models in thin composite panels with local stress concentration and debonding [13]. The same approach was extended in the context of explicit dynamics in [14,15]. Sub-modelling techniques are another non-intrusive option widely available in Finite Element Method (FEM) software and used in industry. However, they are referred to as “one-way coupling” because the data exchange occurs from global to local, and the global solution is not updated after the local solution has been modified.

The FEM simulations of local problems can be computationally expensive due to the requirement for refined space and time scales. To address the aforementioned challenge, it is proposed here to replace the original, time-consuming simulation by a reduced model in order to increase the computational performance. Model Order Reduction (MOR) [16] seems to be an appealing choice to overcome this issue. Projection-based MOR methods, which rely on linear transformations with some additional constraints, such as Proper Orthogonal Decomposition (POD) or Reduced-Basis technique [17], are widely used. However, they can hardly be used to model highly nonlinear phenomena and often require prior knowledge of the governing equations of the physics. Neural Networks (NNs) based on machine learning have proven to be highly effective for learning nonlinear manifolds. NNs often outperform physics-based models in many disciplines (e.g., materials science [18], applied physics [19], biomedical science [20], computational biology [21]) in terms of prediction accuracy and capability to capture the underlying nonlinear input-output relationship for complex systems. A main limitation of this technique is the correct selection of the network tuning parameters.

Training NNs requires a vast amount of data that must contain a rich input-output relationship, which is not available in the majority of engineering problems. To address this issue, researchers have developed a large variety of methods for integrating physical principles into NN models. Integrating physical principles into NN models provides several key advantages. It ensures physical consistency by aligning predictions with fundamental physics laws and constraints. This improves generalization, allowing accurate predictions with limited data. A detailed review of these methods applied in various fields can be found in [22]. They can be broadly classified into four main categories: (i) physics-guided loss function, (ii) physics-guided initialization, (iii) physics-guided architecture, and (iv) hybrid physics-Machine Learning (ML) models. The idea behind physics-guided loss function is to incorporate physical constraints into the loss function of NN models [23,24]. In physics-guided initialization, the physical or other contextual knowledge is used to help inform the initialization of the weights, so that the model training can be accelerated and may require fewer training samples [25]. In physics-guided architecture, the key idea is to incorporate physics-based guidance into architecture design making the NN more interpretable, this allows to include typically missing features into the NN

model [26,27]. In hybrid modelling the focus has been on augmenting ML models specifically; numerous approaches combine physics-based models with ML models where both are operating simultaneously [28].

In an explicit dynamic local/global coupling framework, the input and output of the local problem are the interface velocities and interface reaction forces, respectively. However, these quantities are highly noisy in nature, making it difficult to obtain direct input-output relationships. To address this fundamental challenge, we propose to use Physics-Guided Architecture of Neural Networks (PGANNs). The key idea behind this approach is to inject other physical variables of the whole local domain, such as displacement, stress, strain, plastic strain etc., at the local scale between the input and output layers of NNs in order to improve learning within the solution space. By including these additional physical variables, we provide the network with relevant domain-specific information that can potentially enhance optimization and training efficiency. It should be noted that the architecture of the proposed NN itself operates as a black box, as its internal mechanisms are not explicitly designed to resemble physics equations.

In this article, we focus on the injection of displacement as well as effective plastic strain as intermediate variables, as displayed in Figure 4. The neural network (NN) architecture begins by reconstructing the displacement of the local domain from the input. Next, an autoencoder extracts the most relevant features of displacement, known as Latent Vector (LV). Similarly, another autoencoder is employed to capture the significant features of effective plastic strain, forming its corresponding LV. The dynamics of these LVs is then modeled using a Long Short-Term Memory (LSTM) network. Finally, the predicted displacement at the next global time step is mapped to the interface reaction forces.

The paper is organised as follows: In Section 2, a summary of the non-intrusive local/global coupling reference problem in explicit dynamics is presented. Section 3 details the proposed PGANN to metamodel the local problem. In Section 4, the proposed method is tested on a numerical example of an elastoplastic plate with hole geometry discretized using 2D elements that has undergone rapid deformation. In Section 5, the proposed method is tested on a spotwelded plate geometry discretized using 3D elements. Finally, in Section 6, conclusions and outlooks of the proposed work are presented.

2 Problem formulation

In this study, we briefly present a method for addressing the non-intrusive local/global coupling in explicit dynamics [29]. To illustrate the method, we consider the problem configuration depicted in Figure 1. As shown in the figure, the overall domain, denoted as Ω , is partitioned into two subdomains: the local region, denoted as Ω_l , and the complementary region, denoted as Ω_c . The local region Ω_l may contain fine geometric features or large gradients, which requires a refined mesh in both space and time scales. Conversely, the complementary region Ω_c only requires a

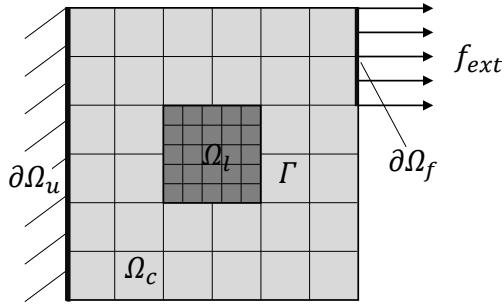


Fig. 1. Heterogeneous discretization in space of the reference problem.

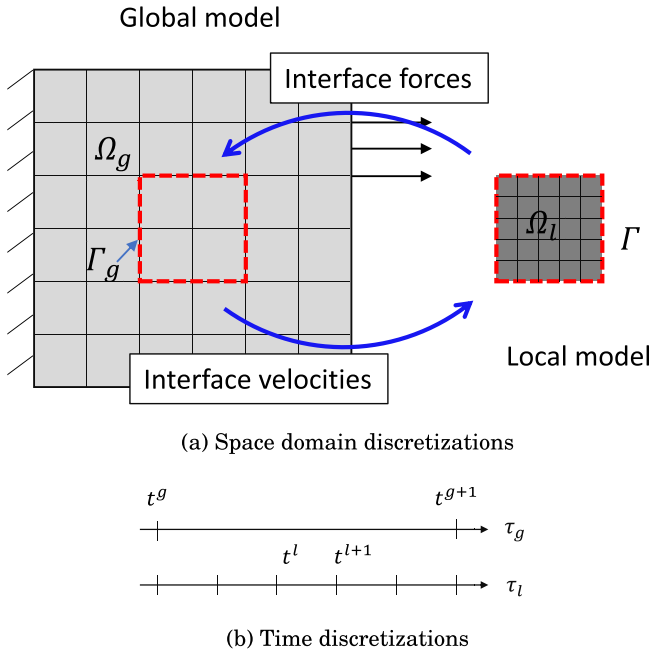


Fig. 2. An illustration of the non-intrusive local/global coupling method.

coarse mesh in both space and time. The interface between the two regions is denoted by Γ . In this problem, the displacement u_D is prescribed on the Dirichlet boundary $\partial\Omega_u$, the external force f_{ext} is applied on the Neumann boundary $\partial\Omega_f$, and the body force f_Ω may be applied in the domain Ω .

Using a displacement-based finite element method with the heterogeneous spatial discretization described in Figure 2a (left), the problem can be expressed as follows.

$$\begin{aligned} \mathbf{M} \ddot{\mathbf{U}} &= \mathbf{F}^{ext} - \mathbf{F}^{int} & \text{over } & \Omega_c \cup \Omega_l \times [t_0, t_{end}] \\ \mathbf{U} &= \bar{\mathbf{U}} & \text{along } & \partial\Omega_u \times [t_0, t_{end}] \\ \{\mathbf{U}, \dot{\mathbf{U}}\} &= \{\mathbf{U}_0, \mathbf{V}_0\} & \text{over } & \Omega_c \cup \Omega_l|_{t_0} \end{aligned} \quad (1)$$

where \mathbf{M} is the lumped mass matrix, \mathbf{F}^{ext} and \mathbf{F}^{int} are, respectively, the external and internal force vectors, and $\Omega_c \cup \Omega_l$ represents the union of two different homogeneous finite element discretizations that span the entire domain. \mathbf{U} represents the vector containing the nodal displacement

($\ddot{\mathbf{U}}$ and $\dot{\mathbf{U}}$ are the associated acceleration and velocity respectively). The initial and final times are denoted t_0 and t_{end} .

The central difference technique is used to implement an explicit time integration scheme. The time integration scheme is not unconditionally stable, so the CFL condition governs the selection of the time step size. The critical time step Δt_{cr} is given by:

$$\Delta t_{cr} = \frac{2}{\omega_{max}} \quad (2)$$

where ω_{max} is the maximum eigenfrequency of the problem. The value of ω_{max} is inversely proportional to the smallest element size. As a result, the time integration step is reduced because of the refined space mesh.

In a non-intrusive local/global coupling framework, the global model extends over the whole structure with a global mesh and never changes (Fig. 2a). The local analysis is carried out with a more refined mesh where the boundary conditions are derived from the global problem. Two different time steps Δt_g and Δt_l are applied in the two partitions Ω_c and Ω_l (Fig. 2b). Separate explicit dynamics analyses of the two meshes are performed concurrently, allowing the models to run with their own time increment. The continuity of velocities between the two models is ensured along the interface Γ through Lagrange multipliers [9]. Substituting the local model into the global model is achieved by iteratively exchanging the velocities and forces at the interface Γ , as displayed in Figure 2a. The global domain, shown with a coarse mesh on the left, is initially solved at a global time scale τ_g . The velocity at the interface is then transferred to solve the local problem as a Dirichlet problem at local time scale τ_l . The global problem is then re-solved by applying the reaction forces from the local problem at the interface. This process is repeated until the difference in reaction forces reaches a predefined tolerance $\bar{\epsilon}$. It was shown in [30] that for explicit dynamic problems, the global computation may be performed only once per global time step, while a repeated solution is needed only for the local problems. This finding serves as a motivation for our development of a NN-based ROM for the local problem, which significantly reduces the overall iteration cost.

The non-intrusive local/global coupling strategy is not implemented in this article; it is a work in progress to integrate NN-based ROM with explicit FEM solvers. In this context, our present goal is to develop a reduced model of the local problem based on NN, which can predict the interface forces at time $t + \Delta t_g$, given the boundary conditions of the local problem at previous global time steps t . More precisely, as shown in Figure 3, given the interface velocities and other model parameters at time instants such as $t_g^n, t_g^{n-1}, t_g^{n-2}, t_g^{n-3}$, the metamodel should be able to accurately predict the interface forces at t_g^{n+1} . It is worth noticing that the number of past time steps considered is arbitrary, and a greater amount of historical information generally improves the predictive performance of the model. However, it is important to consider the limitations of storage space associated with long-term data retention.

Algorithm 1: Local/Global Coupling in Explicit Dynamic Simulation**Data:** Initial conditions, time step, total simulation time, \bar{e} **while** $t_g < t_{end}$ **do**

// Solve the global problem

$$\mathbf{M}_g \ddot{\mathbf{U}}_g = \mathbf{F}_g^{\text{ext}} - \mathbf{F}_g^{\text{int}};$$

while $e > \bar{e}$ **do**// Local computation using velocities extracted at Γ from
the global problem

$$\mathbf{M}_l \ddot{\mathbf{U}}_l = \mathbf{F}_l^{\text{ext}} - \mathbf{F}_l^{\text{int}};$$

// Compute residual and global correction acceleration at
the interface Γ

$$\mathbf{r}_\Gamma = \mathbf{r}_{l,\Gamma} - \mathbf{r}_{c,\Gamma} = (\mathbf{F}_{l,\Gamma}^{\text{ext}} - \mathbf{M}_{l,\Gamma} \ddot{\mathbf{U}}_{l,\Gamma} - \mathbf{F}_{l,\Gamma}^{\text{int}}) - (\mathbf{F}_{c,\Gamma}^{\text{ext}} - \mathbf{M}_{c,\Gamma} \ddot{\mathbf{U}}_{c,\Gamma} - \mathbf{F}_{c,\Gamma}^{\text{int}})$$

$$\mathbf{M}_{g,\Gamma} \ddot{\mathbf{U}}_{g,\Gamma}^{\text{corr}} = \mathbf{r}_\Gamma;$$

// update global acceleration at the interface Γ

$$\ddot{\mathbf{U}}_{g,\Gamma} = \ddot{\mathbf{U}}_{g,\Gamma} + \ddot{\mathbf{U}}_{g,\Gamma}^{\text{corr}}$$

Compute e ;**end**

// Global stabilization and update acceleration [30]

// Update time

$$t_g = t_g + \Delta t_g;$$

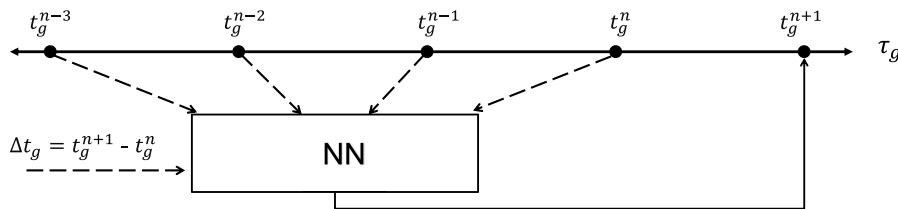
end

Fig. 3. Structure of the reduced model: the NN inputs are interface velocity and interface nodal position at time instances $t_g^n, t_g^{n-1}, t_g^{n-2}, t_g^{n-3}$ in global time scale denoted using dashed arrows, while the output is the interface forces at time t_g^{n+1} denoted using solid arrow.

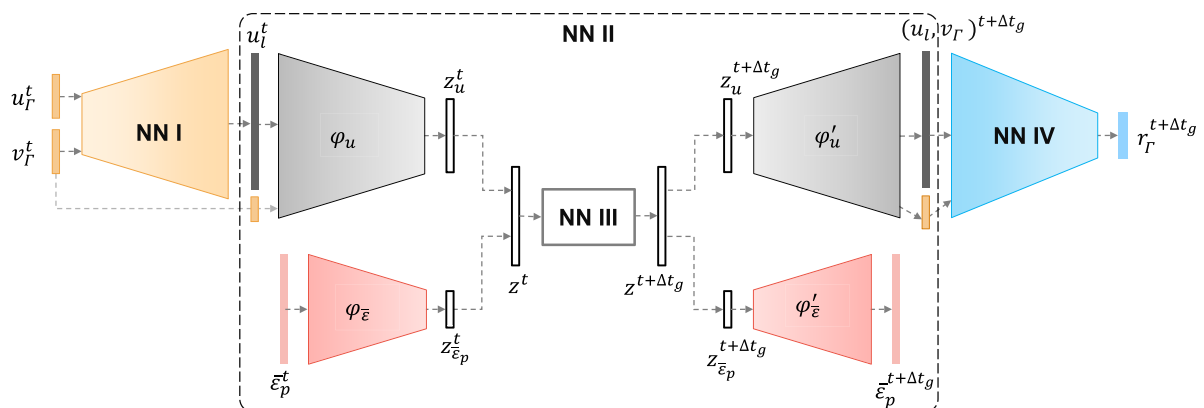


Fig. 4. A schematic for the architecture of the PGANN framework.

3 Proposed physics-guided NN architecture (PGANN)

3.1 Structure of PGANN

In this section, we shall elaborate on the proposed PGANN model. Due to the lack of a robust input-output relationship, it is challenging to train a NN using only the velocity and reaction forces of the interface. This is primarily due to their noisy nature and dependence on other model parameters, such as material parameters and geometric parameters. In order to address this issue, a new layer containing information about the displacement u_l of the local problem is inserted between the input and output layers.

Plastic deformation of materials is a complex process that exhibits history-dependent behavior. Recent studies in [31,32] have investigated the application of (Recurrent Neural Networks) RNNs in modeling path-dependent plasticity models. These RNNs, which are an extension of traditional NNs, have shown promising results in predicting complex history-dependent plasticity. However, their architecture is more complex, requiring a large amount of historical data for accurate predictions. It was shown in [33] that traditional NN architectures can also capture path-dependent behavior effectively, provided that a set of internal variables representing the material history is appropriately included in the training process. Inspired by this work, we adopt a similar methodology in this study to avoid the need for an extensive history of data while still achieving accurate predictions. In metal plasticity, the history of plastic deformation is often characterized by using a scalar quantity called the effective plastic strain, which is given by

$$\bar{\varepsilon}_p = \int \|\dot{\varepsilon}_p\| dt \quad (3)$$

where $\dot{\varepsilon}_p$ is the plastic strain rate. The examples presented in this study use the effective plastic strain $\bar{\varepsilon}_p$ as the internal variable.

An important aspect of our method is the preprocessing step, in which the actual FEM grid is interpolated to a Cartesian grid to represent the local displacement. Thus, the interpolated displacement can be treated analogous to a colour image, and the three components of displacement (x , y , and z) are treated analogous to the three colour channels, allowing the use of convolutional NN architectures. As a result, the memory and computation costs depend primarily on the size of the interpolated grid rather than the size of the actual FEM grid. A schematic of the training pipeline for the PGANN model is presented in Figure 4. It consists of four separately trained NNs. In the following paragraphs of this section, we shall explain the procedures for training all four networks separately.

The first NN (**NN I**) of the architecture is trained to capture the nonlinear relationship between the input variables, specifically the velocities v_{Γ}^t and displacements u_{Γ}^t of the interface nodes, at a specific time t , and the corresponding displacement of the local model u_l^t . It is important to highlight that while this study focuses on

these specific input variables, the approach is not limited exclusively to this set of variables. Other model parameters, such as material parameters or geometric parameters, can also be included as input variables, further enhancing the flexibility and applicability of the methodology. The input layers are concatenated to form a dense layer. A few dense layers and convolutional layers are then added between the concatenated layer and the output layer u_l^t . The number of these added layers and their properties, such as the number of filters used and filter size, are the main hyper-parameters to be optimized in this model; this is detailed in Section 4.2.1.

The second NN, referred to as (**NN II**), utilizes autoencoders to encode the local solution into a reduced manifold known as a latent vector z^t . Autoencoders are trained by setting the target layers equal to the input layers, where the size of the middle layer (latent vector) is much smaller than that of the input/output layers. The main objective is to extract the most dominant features of the system, while effectively filtering out the noise generated during explicit dynamics simulation. This reduced manifold enhances the training performance of the subsequent RNN model (**NN III**), which is responsible for predicting the system time evolution. Within (**NN II**), two autoencoders are employed and trained separately. The first autoencoder reduces the displacement u_l^t to its corresponding latent vector z_u^t using its encoder component φ_u , while the decoder component φ'_u converts the latent vector z_u^t back to the original displacement u_l^t . Similarly, the second autoencoder reduces the $\bar{\varepsilon}_{p,l}^t$ to its corresponding latent vector $z_{\bar{\varepsilon}_p}^t$ using an encoder component $\varphi_{\bar{\varepsilon}_p}$, and a decoder component $\varphi'_{\bar{\varepsilon}_p}$ converts the latent vector back to the original $\bar{\varepsilon}_{p,l}^t$. During the online phase, when the trained model is utilized, the first explicit cycle initializes the $\bar{\varepsilon}_{p,l}$ with zero, assuming no plastic deformation has occurred. The predicted $\bar{\varepsilon}_{p,l}^{t+\Delta t_g}$ is then used as input for the subsequent cycle. Since we are not interested in observing $\bar{\varepsilon}_{p,l}$ during the online phase, only its latent vector $z_{\bar{\varepsilon}_p}^{t+\Delta t_g}$ needs to be stored at the end of each cycle. This approach avoids the need for evaluating $\varphi_{\bar{\varepsilon}_p}$ and $\varphi'_{\bar{\varepsilon}_p}$, thereby reducing the computational cost involved.

The third NN (**NN III**) is a RNN that learns the temporal evolution of the latent variable. More specifically, the goal is to predict the latent vector at time $t + \Delta t_g$ based on the latent vector from previous global time steps. The input of this network consists of a sequence of latent vectors (z_u and $z_{\bar{\varepsilon}_p}$) from past global time steps, while the output is a sequence of latent vectors in future local time steps. From this sequence, the latent vector corresponding to Δt_g is selected. It is important to note that here the purpose of the RNN is not to learn the path dependency of elastoplastic materials. We employ an encoder-decoder (also known as Sequence-to-Sequence models) type LSTM architecture. This means that the model will not output a vector sequence directly. Instead, the model will consist of two sub-models: the encoder and the decoder. The encoder reads and summarizes the input sequence information, referred to as the internal state vectors. At each time step, the encoder receives an input element from the sequence

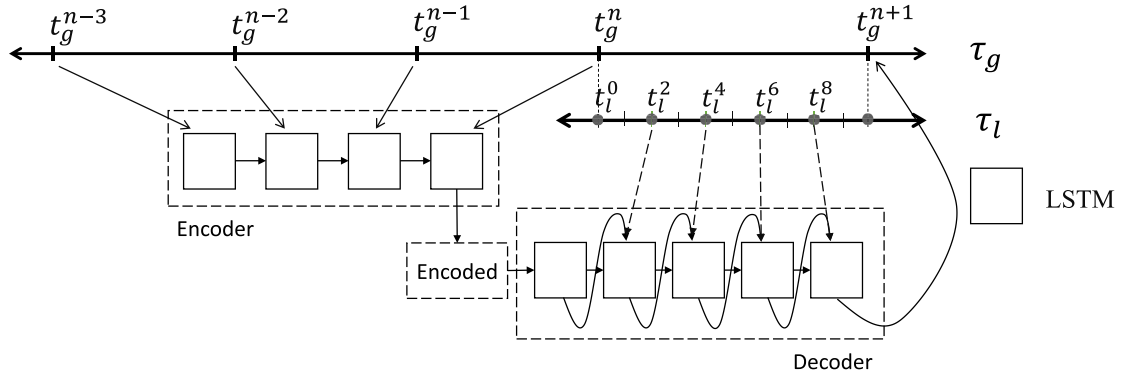


Fig. 5. Example of proposed RNN architecture used. The network is trained by taking the latent vectors from the past four global time steps and the following five latent vectors of local scale as output.

and updates its internal state based on the current input and the previous hidden state. The LSTM memory cells maintain a memory of past inputs, which enables them to retain information over longer sequences. The outputs of the encoder are discarded and only the internal states are kept. The decoder reads the final state of the encoder and makes a one-step prediction for each element of the output sequence (Fig. 5). We enforce that the input of the decoder at each time step is the output from the previous time step, which helps to train the decoder faster. The fact that the output sequence corresponds to local time steps allows prediction for variable global time step Δt_g using simple interpolation. The shapes of input and output are specific to the case, therefore the implementation will be detailed in Section 4.2.4.

The final NN, (NN IV), is responsible for mapping the predicted local nodal displacement $u_l^{t+\Delta t_g}$, the predicted interface velocity $v_\Gamma^{t+\Delta t_g}$ at time $t + \Delta t_g$, and its corresponding interface reaction forces $r_\Gamma^{t+\Delta t_g}$. The architecture used is similar to that of the encoder part φ_u of the autoencoder. First the multi-dimensional displacement layer is transformed to a dense layer using convolutional layers, and merged with other model parameter layers to output the interface forces.

3.2 Training configuration and metrics

The quantitative performance of each NN framework is evaluated using a metric called Normalized Mean Absolute Error (NMAE). NMAE is calculated for each component k between the actual value and the predicted value by the NN, from the initial time to the final time. The overall NMAE is obtained by summing the NMAEs of each component. The NMAE is defined as:

$$\text{NMAE} = \sum_{k=1}^R \left(\frac{\sum_{i=1}^N |y_k^{(i)} - \tilde{y}_k^{(i)}|}{\sum_{i=1}^N y_k^{(i)}} \right) \quad (4)$$

where R is the total number of components of the dynamical system, N is the total number of time steps in the evolution of the dynamical system, y is the true solution and \tilde{y} is the solution predicted by the NN.

Throughout this study, we employ the mean squared error (MSE) as the loss function, Rectified Linear Unit (ReLU) activation function for all hidden layers and linear activation function for the output layer in all NNs. The ReLU activation function σ can be defined as:

$$\sigma(a) = \max(0, a) \quad (5)$$

where a is the input of the node.

The NN training has been performed in the Python environment using TensorFlow. We used the TensorFlow 2.0 API `tf.distribute.MultiWorkerMirroredStrategy` to distribute the training across multiple machines (up to 4 machines) with single NVIDIA Quadro RTX 4000 GPU cards. Source codes are available at <https://github.com/afsalpt1/ROM-PGANN>. Using the TensorFlow API `tf.data.Dataset`, efficient input pipelines are written for the data generated in Section 4.1. The pipeline allows for easy access to the data to distribute the training across multiple machines. We set the training batch size to 128 to ensure better generalization of each batch. To train the NN, the Adam optimization algorithm is used. In the following section, the proposed method is implemented on a plate with hole example.

4 A first validation example: plate with a hole

In the present section we consider an elementary problem to test and validate the PGANN architecture for meta-modelling of a highly nonlinear local problem. Let us consider a rectangular 2D domain with a hole in the centre, as shown in Figure 6. The region surrounding the hole is taken as the local domain Ω_l , where plastic deformation is expected during the loading process, and the remaining part refers to Ω_c where only elastic deformation occurs.

The elastic behavior is defined by the Young modulus $E = 210$ GPa, the Poisson ratio $\nu = 0.3$, and the material density $\rho = 7,800$ kg/m³. The isotropic elastoplastic material is described using the Johnson-Cook material model. Neglecting the effect of temperature or strain rate,

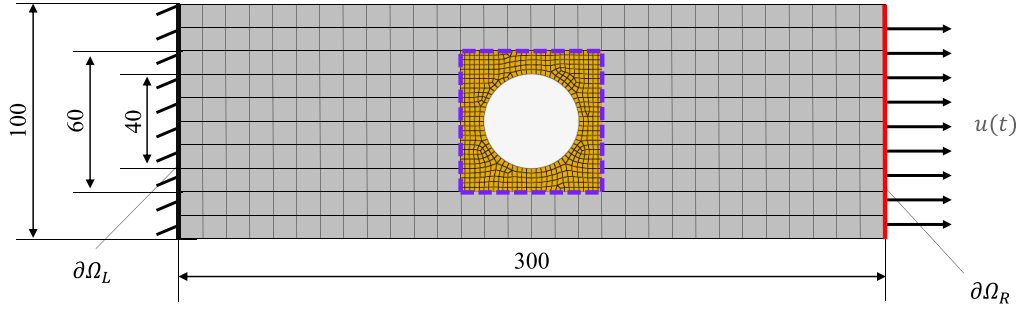


Fig. 6. Model of a plate with hole. The local domain, shown in yellow, is a refined region near a geometrical detail (a hole).

the expression for stress is given by:

$$\sigma = a + b\varepsilon_p^n \quad (6)$$

where ε_p is the plastic strain. The Johnson Cook parameter a , hardening modulus b , and the hardening exponent n are defined as

$$a = \sigma_y, \quad b = \frac{\sigma_u}{n\varepsilon_u^{(n-1)}}, \quad n = \frac{\sigma_u \varepsilon_u}{\sigma_u - \sigma_y} \quad (7)$$

where σ_y is the yield stress, σ_u is the engineering Ultimate Tensile Stress (UTS), and ε_u is the engineering strain at UTS. For simulation purposes, the following values are considered: $a = 0.792$ GPa, $b = 0.51$ GPa, and $n = 0.26$.

The dimensions of the computational domain shown in Figure 6 are $H_x = 300$ mm, $H_y = 100$ mm. The diameter of the hole is $d = 40$ mm, and the length of the local domain is $H_l = 60$ mm. The structure is clamped on the left boundary $\partial\Omega_L$, and the displacements are imposed on the right boundary $\partial\Omega_R$.

4.1 Data generation and post-processing

The data required for training the NNs are obtained by solving the reference model shown in Figure 6 using a nonlinear explicit FEM dynamic solver. The simulation spans from an initial time $t_0 = 0$ ms to a final time $t_{\text{end}} = 2.5$ ms. To discretize the time domain, a constant time step $\Delta t_l = 2.3 \times 10^{-4}$ ms is employed, calculated based on (2). Consequently, each simulation comprises approximately 10,800 time steps. For simplicity it is assumed here that the ratio $\Delta t_g / \Delta t_l$ is equal to 10.

The first dataset comprises five principal loading directions, namely translational displacements (s_x, s_y, s_z) and rotational angles (θ_x, θ_y). Each load direction undergoes eight simulations with varying magnitudes, including four positive and four negative magnitudes, resulting in a total of 40 simulations. The second dataset is created to enrich the first dataset by including additional loading directions. Unlike the single-direction loading in the first set, the second dataset involves simultaneous monotonic loading in two directions, as outlined in Table 1. This results in a total of 24 distinct loading directions. For each combination of load directions, four simulations are performed, encompassing two positive and two negative magnitudes, thereby generating additional 96 simulations.

Consequently, the second dataset comprises a total of 136 simulations. Further information regarding the magnitude and direction of loading can be found in Appendix A. While alternative approaches like Design of Experiment (DoE) methodologies could be employed to efficiently generate optimal data parameters, it is important to note that the simplified load cases used in this article primarily serve to present the core concept of our strategy. In reality, the load cases are much more complex and can be derived from the local/global boundaries of a coupled model.

In this particular example, snapshots are collected at each local time step. At the interface Γ , velocities ($v_\Gamma = (v_{\Gamma,x}, v_{\Gamma,y}, v_{\Gamma,z})$), rotational velocities ($\omega_\Gamma = (\omega_{\Gamma,x}, \omega_{\Gamma,y}, \omega_{\Gamma,z})$), and reaction forces ($r^\Gamma = (r_{\Gamma,x}, r_{\Gamma,y}, r_{\Gamma,z})$) are recorded. Additionally, the nodal displacement ($u_l = (u_x, u_y, u_z)$) of the local domain, and the elemental effective plastic strain ($\bar{\varepsilon}_p$) of the local domain is collected. The model contains a total of 120 interface nodes; thus, the interface quantities v, ω , or r are a vector of size 120×3 (x, y and z components). The displacement in the actual FEM grid is interpolated to a Cartesian regular grid of size 40×40 , as shown in Figure 7. To improve the training performance, all data are standardized on a scale of $[0, 1]$ using a MinMax scale. The network is trained using 80% of the total data, while the remaining portion is divided equally between the cross-validation set (10%) and the test set (10%).

4.2 Training and verification of PGANN

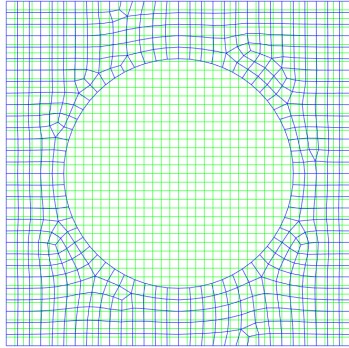
In this section, we explore the individual training and verification of all four sub-networks of the proposed PGANNs. First we outline a systematic approach for determining the optimal hyper-parameters of a NN. The hyper-parameters we focus on include: the number of layers, the number of neurons or filters in dense or convolutional layers, the choice of activation functions, the type of loss function, the learning rate, and the batch size, among others. Then to evaluate the accuracy of the trained model, aside from the dedicated test dataset comprising 10% of the overall data, four additional test cases are generated, which are neither included in the first dataset nor in the second dataset. To avoid confusion, we will refer to the former as the test dataset and the latter as test cases. Table 2 provides details regarding the direction and magnitude of the load applied on the boundary $\partial\Omega_R$ for each test case.

Table 1. Training datasets.

Dataset	Loading direction	Remark
I	$s_x, s_y, s_z, \theta_x, \theta_y$	Loading in single direction
II	$s_x, s_y, s_z, \theta_x, \theta_y, [s_x \& s_y], [s_x \& s_z], [s_x \& \theta_x], [s_x \& \theta_y], [s_y \& s_z], [s_y \& \theta_x], [s_y \& \theta_y], [s_z \& \theta_x], [s_z \& \theta_y], [\theta_x \& \theta_y]$	Single & bidirectional loading

Table 2. Test cases used for verification of PGANN architectures for 2D example.

	Dataset	Translational loading (mm)			Rotational loading (rad)		Remark
		s_x	s_y	s_z	θ_x	θ_y	
Case 1	I	16					Interpolation of magnitude
Case 2	I	32					Extrapolation of magnitude
Case 3	I	21			1.5		Interpolation of magnitude and direction
Case 4	II	21			1.5		Enriched training dataset

**Fig. 7.** Grid used to feed displacement to NN (green), and actual grid used by FEM solver (blue).

The performance of the first three test cases is evaluated using the NNs trained on first dataset. The first two test cases involve loading of similar direction that were present in the training data, but with different, unseen magnitudes. The first test case has a magnitude within the range of the training data, while the second test case has a larger magnitude. The third and fourth test cases have the same loading magnitude and direction and correspond to combination loading. The third test case is evaluated on model trained using the first training set, while the fourth test case is evaluated on model trained using the second training set.

4.2.1 NN I

Figure 8 illustrates the initial proposed reference architecture for NN I. It comprises three input layers: v_{Γ}^t , ω_{Γ}^t , and displacement of interface nodes u_{Γ}^t . After concatenating input layers, a dense layer of size 1,600 is added to help reshape to a 3D layer of shape (5,5,64). Subsequently, three deconvolution layers (Conv2DTranspose) with 64, 32 and 16 filters, respectively, are added to achieve an output shape of (40,40,3). For each deconvolutional layer, filters have a kernel size of 2 and a stride of size 2 is used.

When working with NNs, two crucial hyper-parameters to tune are the learning rate and batch size. The learning rate determines the step size at each iteration during the optimization process, while the batch size refers to the number of training examples processed in a single forward and backward pass. If the learning rate is set too high, the optimization process may oscillate or fail to converge. On the other hand, if the learning rate is too low, the training process may become extremely slow. A larger batch size can provide computational efficiency, as more examples are processed in parallel, but it may lead to sub-optimal generalization. Smaller batch sizes, on the other hand, can introduce more noise and result in slower convergence. Figure 9a shows the NMAE plot against the learning rate of Adam optimizer using 512 batch size, it becomes evident that a value of 5×10^{-5} provides an ideal learning rate. Similarly, the Figure 9b shows the NMAE plot against the batch size for a learning rate of 5×10^{-5} ; it indicates that a batch size of 128 is optimal for this scenario.

Next, we focus on optimizing the number of layers used in the training network. This involves incorporating one deconvolutional layer with same shape as its preceding layer at positions A, B, or C as depicted in Figure 10. The results are tabulated in Table 3. The results indicate that adding a single layer at positions A and B leads to the lowest NMAE values. The effect of incorporating multiple layers at position A and B is shown in Figure 10, which indicates that adding eight such layers is optimal. The inclusion of dense layers after each input layer does not seem to have a significant impact on the training process.

After 1,000 epochs, the NMAEs of the optimized (NN I) on the first and second test datasets (10% of total data) are 6.52×10^{-4} and 4.8×10^{-3} , respectively. Furthermore, we investigated the robustness of the trained NN I on unseen data using the test cases presented in Table 2. Figure 11 displays the displacement predicted by NN I and the corresponding reference displacement (FEM) at $t = 2.5$ ms for each test case. Even though the test cases were unseen data during the training, it was

Table 3. Effect of added layers at positions **A**, **B** and **C**.

Added layer	NMAE on test dataset	Training time (in minutes for 500 epochs)
Position A	3.53×10^{-3}	98
Position B	3.873×10^{-3}	102
Position C	4.59×10^{-3}	115
Position A & B	3.51×10^{-3}	111
Reference network	4.54×10^{-3}	92

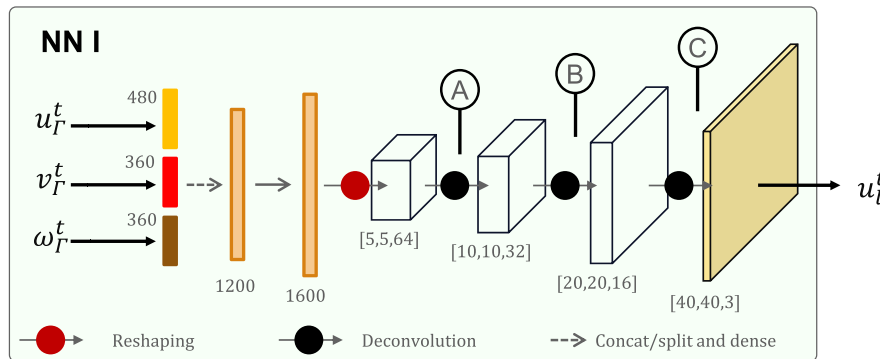


Fig. 8. A schematic for the architecture of **NN I**.

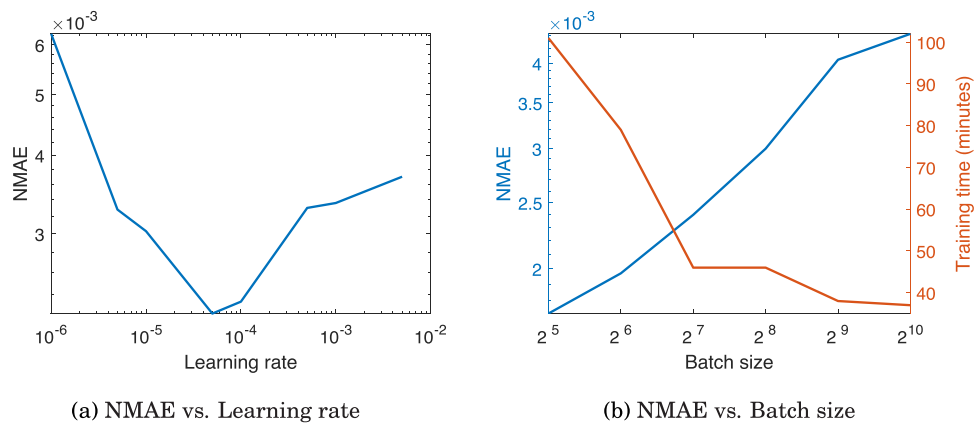


Fig. 9. Effect of learning rate and batch size on the learning.

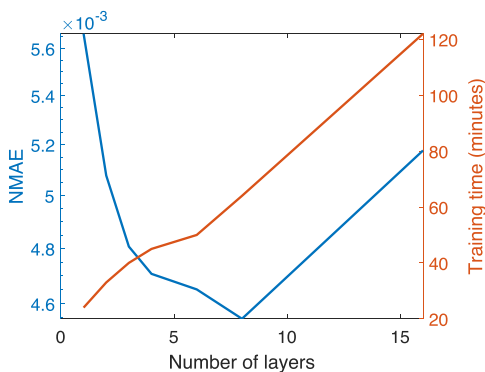


Fig. 10. Effect of NMAE and training time on the number of added layers at positions **A**&**B**.

observed that the predicted solution and FEM solution were in excellent agreement for the first two test cases in which the same direction of loading cases were seen during training. Based on the results of the first two test cases in [Figure 11](#), **NN I** achieved good performance not only on loadings within the range it has been trained on, but also on loadings far outside the range it has been trained on. Based on the results of test cases 3 and 4, it can be deduced that the predicted solution using **NN I** is in good agreement with FEM solution only if similar directions of loading are present in the training set. From the aforementioned results, we can also conclude that the trained **NN I** model can generalize well to unseen values of loadings given the loading direction is present in the training set.


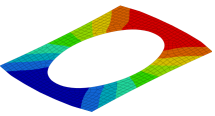
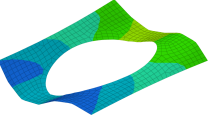
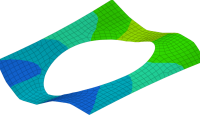
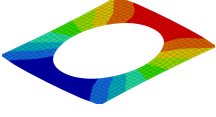
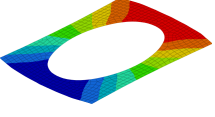
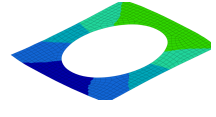
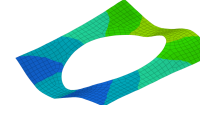
	Test case 1	Test case 2	Test case 3	Test case 4
NMAE	6.55×10^{-4}	7.26×10^{-4}	6.30×10^{-2}	2.62×10^{-3}
FEM				
NN I				

Fig. 11. Comparison between the reconstructed displacement using NN I of the PGANN framework vs. baseline model (FEM) for various test cases. The displacements shown are at the final time of each solution, whereas the NMAE value indicates the mean of all time steps.

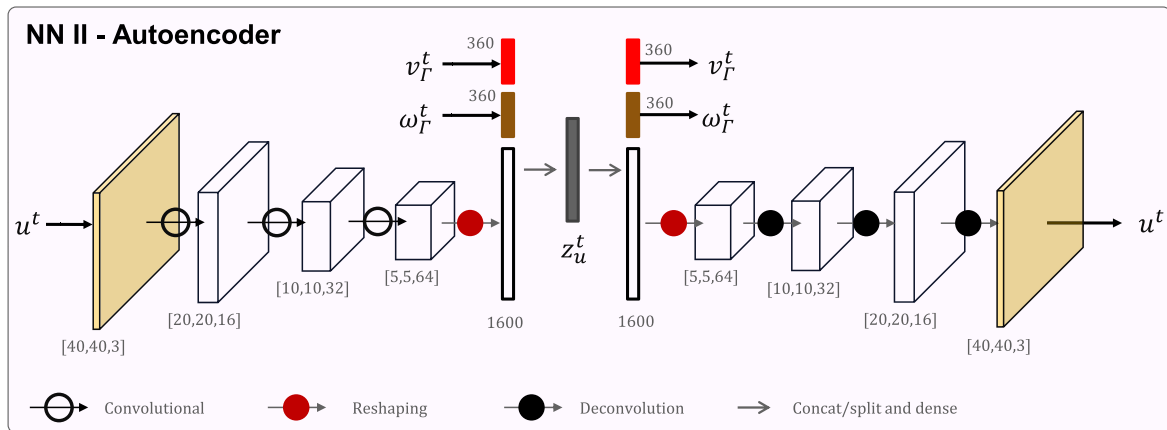


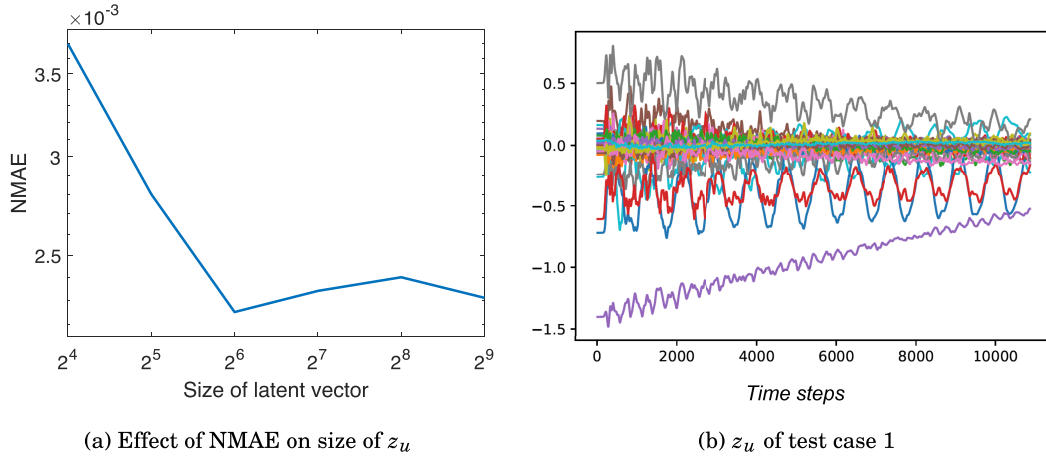
Fig. 12. Schematics of the architecture of NN II: Converting displacement input tensor to vector via convolutional layers and merging with other inputs, obtaining latent vector through dense layer, and generating outputs via dense and convolutional layers.

4.2.2 NN II : reduction of u_l

The architecture utilized to train the first autoencoder is shown in Figure 12. The encoder component (φ_u) has the following structure. It begins with three input layers: v_r^t , ω_r^t , and u_l^t . Following the input layer u_l^t , three convolutional layers (Conv2D) with 16, 32, and 64 filters and 2 strides are incorporated to achieve an output shape of $[5,5,64]$. This output is then reshaped to form a dense layer with a size of 1,600. For the input layers v_r^t and ω_r^t , several dense layers are added. Subsequently, these layers are concatenated with the dense layer derived from the input u_l^t to form the latent vector z_u^t . The decoder component (φ'_u), has the same structure as the encoder but in reverse order.

In a similar manner to the previous section of training NN I, various hyper-parameters such as learning rate, batch size, number of convolutional layers are optimized.

Using a learning rate of 5×10^{-5} and a batch size of 256 yielded optimal results for both datasets. By comparing the error and its corresponding training cost, we concluded that it is best not to add layers at positions A, B, and C. Additionally, we found that including 2 dense layers after v_r^t and ω_r^t resulted in the best performance. One of the main hyper-parameters to optimize in autoencoder is the size of latent vector z_u^t . Figure 13a shows the performance with respect to the size of the latent vector after 100 epochs. It can be observed that the model has the best performance when using 64 latent variables. We choose a latent vector of size 50 as an optimal value as there is no significant decrease in NMAE with increase in size of z_u^t . Figure 13b displays the latent vector of test case 1 with size 50. After 1,000 epochs, the NMAEs of the optimized model on the first and second test datasets are 6.38×10^{-4} and 4.38×10^{-3} respectively.


Fig. 13. Training of NN II.

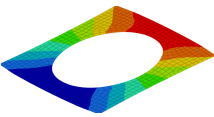
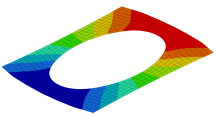
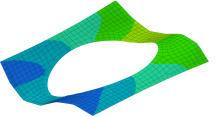
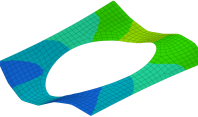
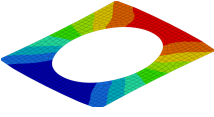
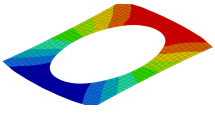
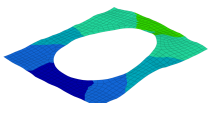
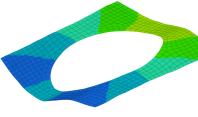
	Test case 1	Test case 2	Test case 3	Test case 4
NMAE	6.98×10^{-4}	7.64×10^{-4}	9.48×10^{-2}	1.69×10^{-3}
FEM				
$\varphi'_u(\varphi_u)$				

Fig. 14. Comparison between the reconstructed displacement using NN II of PGANN framework vs. reference model (FEM) for various test cases.

Figure 14 shows the performance of the network and the result of reconstructed displacement using 50 latent variables. From the first two test cases, it can be observed that, although the input is scaled by a large magnitude, the network is able to reconstruct local displacement accurately just using 50 latent variables. However, the reconstruction is poor in test case 3. Notably, in test case 4, where a combination loading similar to that in the training set is present, the reconstruction performance improves.

The use of skip connections in an autoencoder architecture can have a significant impact on its performance [34]. Skip connections allow for the flow of information from the input layer directly to the output layer, bypassing the bottleneck or compressed representation in the middle layers. This allows for the preservation of important information that may be lost during the compression process. Here it was observed that the use of skip connections did not result in a significant reduction of NMAEs.

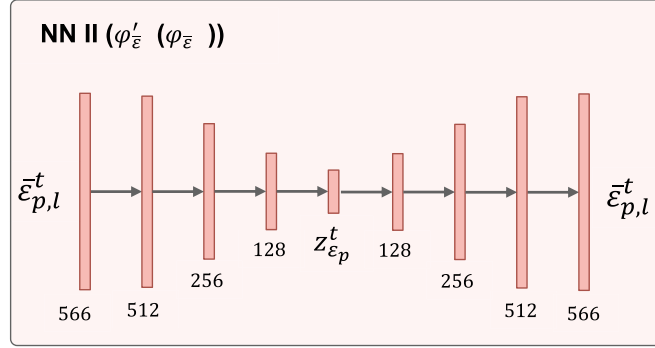
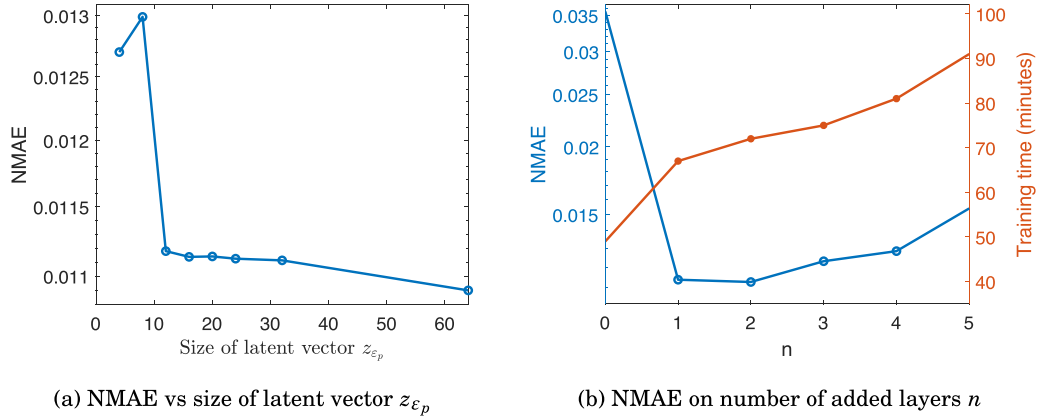
4.2.3 NN II : reduction of $\bar{\varepsilon}_p$

The reference architecture of the proposed autoencoder, designed for reducing the dimension of effective plastic strain of local problem $(\bar{\varepsilon}_{p,l})$, consists of two main components: the encoder $(\varphi_{\bar{\varepsilon}_p})$ and the decoder $(\varphi'_{\bar{\varepsilon}_p})$, as shown in Figure 15. The $\varphi_{\bar{\varepsilon}_p}$ takes $\bar{\varepsilon}_{p,l}$ as input data and maps it to a lower-dimensional latent vector $z_{\bar{\varepsilon}_p}^t$. The input layer has a size of 300, corresponding to each element of the local mesh. In this case, we opted to use only dense layers instead of convolutional layers, given the relatively small size of the input layer. The $\varphi'_{\bar{\varepsilon}_p}$ which mirrors the architecture of $\varphi_{\bar{\varepsilon}_p}$ aims to reconstruct $\bar{\varepsilon}_{p,l}$ from $z_{\bar{\varepsilon}_p}^t$.

The most crucial hyper-parameter to optimize in this model is the size of the latent vector $z_{\bar{\varepsilon}_p}^t$. Figure 16a displays the NMAE in relation to the size of the latent vector after 500 epochs with batch size 256 and learning rate 5×10^{-5} . Notably, the best performance is achieved

Table 4. NMAE on NN II training of $\bar{\varepsilon}_{p,l}$ for various test cases.

	Test case 1	Test case 2	Test case 3	Test case 4
NMAE	6.1×10^{-3}	7.98×10^{-3}	7.2×10^{-2}	3.4×10^{-2}

**Fig. 15.** Schematic representation of the architecture of NN II: An autoencoder for converting the $\bar{\varepsilon}_{p,l}$ input vector to a latent vector through dense layers, and its reconstruction.**Fig. 16.** Training of NN II.

when employing a latent vector of size 16. Additionally, the impact of adding n layers between two layers of the reference architecture, each having the same size as the layer before, is shown in Figure 16b. It is obvious that $n = 1$ is the ideal value. After 2,000 epochs, the NMAE of the optimized model on the first and second test datasets are 6.6×10^{-3} and 4.1×10^{-3} respectively. Table 4 shows the performance of the network and the result of reconstructed $\bar{\varepsilon}_{p,l}$ using 12 latent variables.

4.2.4 NN III

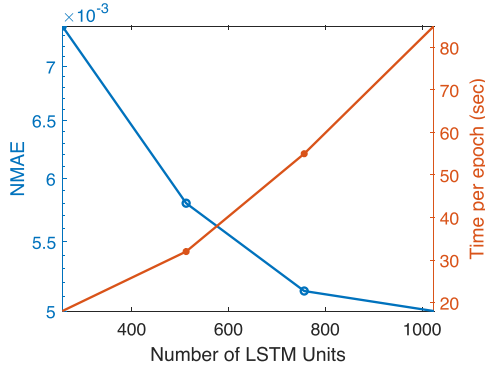
In this example, the goal of NN III is to predict the LV at t_g^{n+1} given LVs at t_g^{n-3} , t_g^{n-2} , t_g^{n-1} and t_g^n . The encoder-decoder LSTM architecture takes two sequential inputs: the latent vector z_u and the latent vector $z_{\bar{\varepsilon}_p}$ from past

instances of the local time scale. During the online phase, as the LVs are only available in the global time scale, a cubic interpolation method is employed to interpolate the LVs into the local time scale. This interpolation approach also enables the handling of varying time steps for the global problem during the online stage. In this specific implementation, the LSTM is fed with the previous 30 time steps, resulting in input shapes of [31,50] for z_u and [31,16] for $z_{\bar{\varepsilon}_p}$, respectively.

The data required to train NN III are generated using the encoder (φ_u and $\varphi_{\bar{\varepsilon}_p}$) components of the trained NN II. The first dataset is created using the FEM simulation with load applied in a single direction. Each simulation consists of 10,800 time steps. To create a single input-output data pair, a sequence of 61 time steps is selected and first 31 are allotted to the input and the remaining is

Table 5. NMAE on NN III training for various test cases.

	Test case 1	Test case 2	Test case 3	Test case 4
NMAE	1.22×10^{-3}	1.03×10^{-3}	1.27×10^{-1}	4.99×10^{-2}

**Fig. 17.** Optimization of the number of LSTM units in NN III

allotted to the output. By selecting various combinations, a total of 9,000 input-output pairs are generated from one simulation, resulting a 360,000 input-output pair from 40 simulations. A second dataset is also created with enriched loading cases, yielding 1,800,000 input-output pairs.

The number of LSTM units used in both the encoder and decoder parts of the NN III is one of the most crucial hyper-parameters to optimize in this network. Because both input and output have the same shape, here we use the same number of LSTM units in both the encoder and decoder. This design allows for a seamless flow of information and promotes consistency between the encoding and decoding processes. Figure 17 shows the relationship between the number of LSTM units and two key metrics, namely the NMAE and training time for the first dataset over 500 epochs. To achieve a balance between computational time and NMAE, the optimal number of LSTM units is 512. After 1,000 epochs, the NMAE of the optimized NN III (with 512 LSTM units) on the first and second test datasets (10% of total data) is 4.56×10^{-7} and 1.10×10^{-6} , respectively.

In Figure 18, the predictions of the latent vector z_u using NN III are displayed. The NN model is trained with 1,000 epochs on the second dataset, and randomly selected data from the test dataset of the second dataset is used for plotting. The plot shows 5 out of 50 latent variables that are randomly selected. The prediction using 512 LSTM units in the NN model is reasonably accurate. However, the prediction with 756 LSTM units matches well with the ground truth, as shown in Figure 18c. It is important to note that the predictions of the latent variables with 128 and 1024 LSTM units are not displayed in the plot. This is because the predicted latent variable has a poor match with the ground truth for 128 LSTM units and an excellent match for 1024 LSTM units.

Figure 18 displays the prediction of the z_u using NN III trained with 1,000 epochs on second dataset with 256,

512, and 756 LSTM units of randomly selected data from the test dataset of the second dataset. Randomly selected 5 out of 50 latent variables are displayed in the plot. Using 512 LSTM units, the prediction is reasonably accurate. The prediction with 756 LSTM units matches well with ground truth, see Figure 18c. Note that the predictions of LV with 128 and 1024 LSTM units are not shown since the predicted LV has a poor and excellent match with the ground truth, respectively. Similarly, Figure 19 showcases the predictions of the variable z_{ε_p} using NN III with the same configuration as before. Out of the 12 latent variables, four are chosen for plotting purposes that are varying with time. It is evident from the figure that using 756 LSTM units yields the best results.

We analyze the robustness of the trained NN III of 756 LSTM units on unseen data with inputs different from the two training datasets on which it was trained. We use the same test configurations as in Section 4.2.1. Table 5 summarizes the performance of the NN III for unseen test cases. The predicted evolution of LV given its past is shown in Figure 20. From Figure 20a-20b, it can be clearly seen that the evolution of LV with time can be perfectly predicted using the proposed NN III even though data with such higher magnitudes is not available in the training set.

4.2.5 NN IV training and evaluation

The reference NN (first guess) model used to train NN IV is displayed in Figure 21. In this architecture, the input displacement layer u_i^t of size $[40 \times 3]$ is encoded to a dense layer of size 1,600 using convolutional layers. The encoded layer is combined with the other input layers v_{Γ}^t and ω_t^{Γ} . The number of dense layers added between the output layer r_{Γ}^t and combined layer is one of the hyper-parameters we optimize in this section. After 1,000 iterations, the NMAE of the optimized NN IV on the first and second test datasets is 2.4×10^{-3} and 1.02×10^{-2} , respectively.

The trained model is verified on the same test configurations as in Section 4.2.1. The performance of the trained NN IV for various test configurations is tabulated in Table 6. Figure 22 shows the comparison of the reconstructed r_{Γ}^t using NN IV and the actual r_{Γ}^t from FEM. It is obvious from the reconstruction with first two test cases that the trained NN IV can capture the variations in the force very well for unseen large magnitudes. However, from the third test case the reconstruction using NN IV is poor when unseen direction of loading is present. It is inferred from last two test cases that the reconstruction can be improved by adding combination loading to the training set.

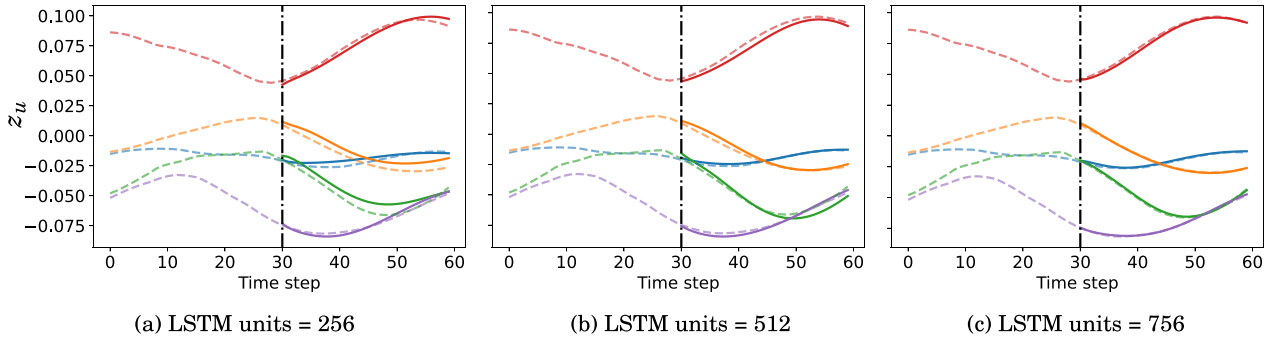


Fig. 18. Prediction of the latent vector z_u using NN III with 256, 512, and 756 LSTM units: given the LV of past time steps (0–30), the NN III predicts the LVs of next 30 time steps, with dashed and solid lines indicating predicted and ground truth, respectively.

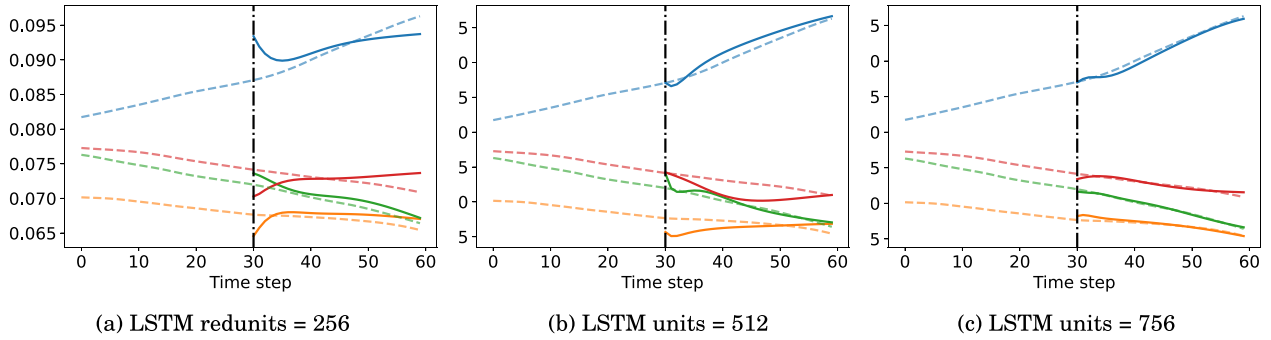


Fig. 19. Prediction of the latent vector z_p using NN III with 256, 512, and 756 LSTM units. Dashed lines represent the predicted latent vectors, while solid lines represent the ground truth.

5 Application to spotwelded plates

In this section, we extend the previously introduced techniques to an industrial application in three dimensions. The car body-in-white (BIW) is assembled by spotwelding numerous sheet metals together. Due to heat treatment during the spotwelding process, these welds might have complex properties. In order to accurately model these localized nonlinear behaviors, refined 3D elements are required near the spotwelded region. Due to the high computational cost, these refinements are avoided in a full vehicle crash simulation, and simplified models (1D elements) are traditionally used. Having numerous spotwelds in a car structure, replacing the fine 3D FEM model with data-driven ROM in localized spotwelded zones can significantly reduce the computational cost.

5.1 Data generation

Let us consider a spotweld domain as shown in Figure 23. The edges of bottom plates are clamped and the loading is applied on the edges of the top plate for 1.5 ms. The integration time step is set to a constant value $\Delta t_l = 5.0 \times 10^{-5}$ ms, calculated based on (2).

The training dataset was constructed using a total of 60 simulation cases, using various loading magnitudes and directions. The minimum and maximum magnitudes of loading applied in each direction are listed in Table 7.

These extreme values denote the approximate maximum and minimum load capacity of the material, respectively, before failure occurs. The first 22 simulations correspond to instances in which the loading is applied in a single direction, while the remaining simulation cases correspond to loading in two different directions simultaneously.

In this study, snapshots of the simulation were created by saving the global interface velocities, denoted as v_{Γ_g} , the global interface forces, denoted as r_{Γ_g} , and the displacement of the local domain, denoted as u_l , at selected time steps. Each simulation contains approximately 3,000 time steps. Unlike the previous example presented in Section 4, the quantities at the interface nodes are chosen at the global interface nodes. The model contains a total of 24 global interface nodes. Furthermore, the local displacement u_l was interpolated to a regular three-dimensional cartesian grid of size $31 \times 31 \times 8$. As in the previous example, the data is partitioned such that 80% of the total dataset is used for training the network, while the remaining 20% is divided between cross-validation and test datasets, with each set comprising 10% of the total data.

5.2 Training and verification of PGANN

In this section, we detail the process of individually training each section of the proposed PGANN, as well as its verification on unseen data. In the context of a 3D

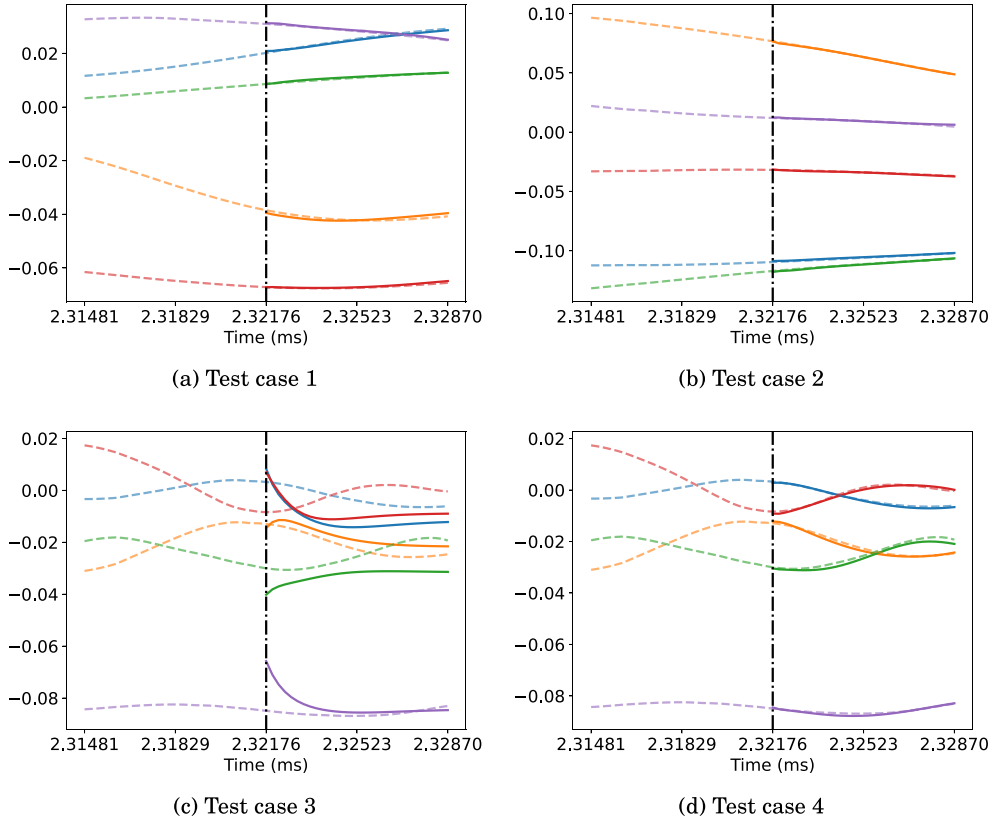


Fig. 20. Prediction of the latent vector $z_{\bar{\epsilon}_p}$ using NN III for various test cases. The dashed and solid lines indicate predicted and reference values, respectively.

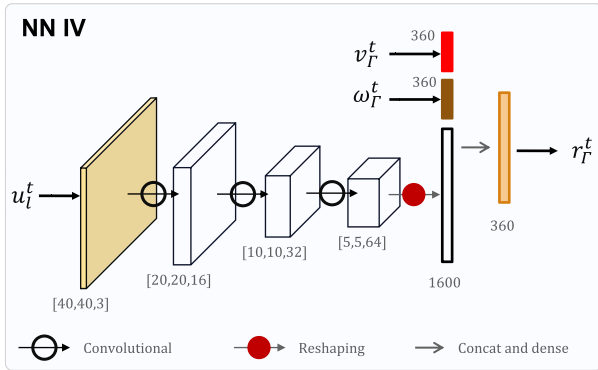


Fig. 21. Schematics of the NN IV architecture, converting input u_i^t tensor to vector via convolutional layers and merging with other inputs, and obtaining the output t_r^t through dense layers.

structural problem, the input layer of the PGANN comprises of two components: v_{Γ_g} and u_{Γ_g} , and the output is r_{Γ_g} . Unlike the previous example, the information on plasticity is not fed into the LSTM, this is mainly for reducing the complexity of the architecture. The training strategy employed for the current models is similar to the one outlined in Section 4. However, in contrast to the model discussed in the previous example, the current model uses higher dimensional convolutional layers,

specifically Conv3D and Conv3DTranspose. The NMAE of each component of the trained PGANN after 2,000 epochs on the test cases is presented in Table 8.

Similar to the previous example, each hyper-parameter is optimized. The most critical hyper-parameter to optimize in this study is the size of the latent vector. The NMAE of the NN II after 600 epochs with respect to the size of the latent vector is shown in Figure 24a. It is observed that the NN II demonstrates optimal performance when using a latent vector of 50 variables. A representative latent vector of size 50 from one of the training cases is presented in Figure 24b. It is obvious that the majority of the components of the latent vector are time-dependent.

To evaluate the effectiveness of the trained model, four additional test cases were generated with randomly selected magnitudes and direction that were not present in the training data set. The direction and magnitude of the applied displacement on the edges of the top boundary are reported in Table 9. Note that unlike the previous example, the magnitudes of the test cases have been limited to the range of magnitudes used in the training dataset to prevent failure of material.

In this study, we investigate the robustness of trained PGANN on unseen data using the test cases presented in Table 9. To evaluate the performance of the PGANNs, we compare the displacement predicted by the NN (NN I and NN II) with the reference displacement obtained

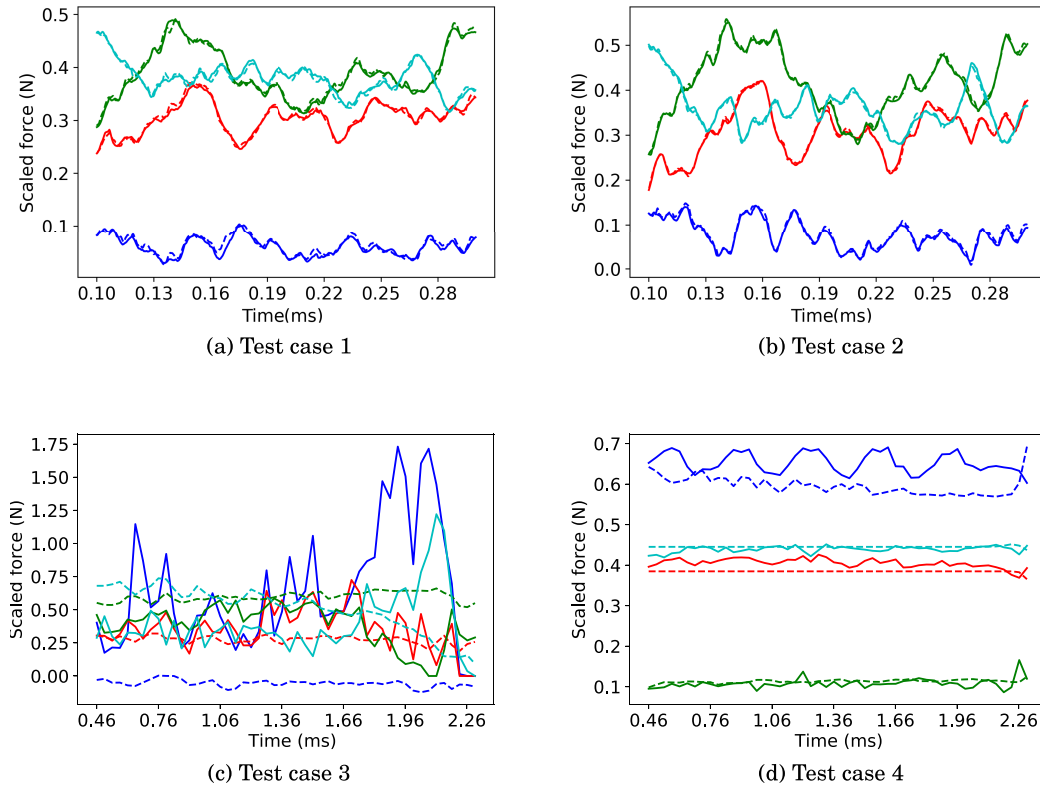


Fig. 22. Prediction of interface forces using NN IV for various test cases: displaying only 4 out of 360 variables with dashed and solid lines indicating predicted and reference values, respectively.

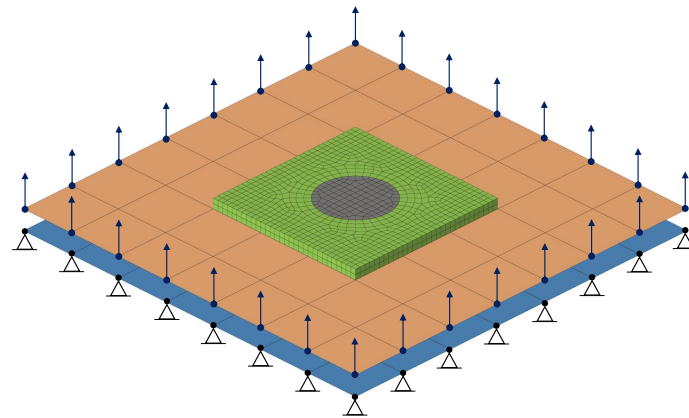


Fig. 23. Model of a spotwelded plate.

from FEM. The results of this comparison is presented in Figure 25, where the displacement at the final time step $t = 1.5$ ms for each test case is displayed. The NMAEs corresponding to each case are shown in the bracket. Additionally, we also evaluate the prediction of the LVs using NN III, which was trained with 756 LSTM units. The results of this evaluation are displayed in Figure 26.

Furthermore, we compare the reconstructed $r_{\Gamma_g}^t$ using NN IV and the actual $r_{\Gamma_g}^t$ from FEM, as shown in Figure 27.

From the results of these evaluations, it can be observed that the solution from the Neural Network is in good agreement with its reference solution. This suggests that the PGANNs trained in this study are robust and can provide accurate predictions on unseen data.

Table 6. Prediction error on NN IV training for various test cases.

	Test case 1	Test case 2	Test case 3	Test case 4
NMAE	1.05×10^{-2}	9.77×10^{-3}	1.03	2.10×10^{-2}

Table 7. Parameters of training data

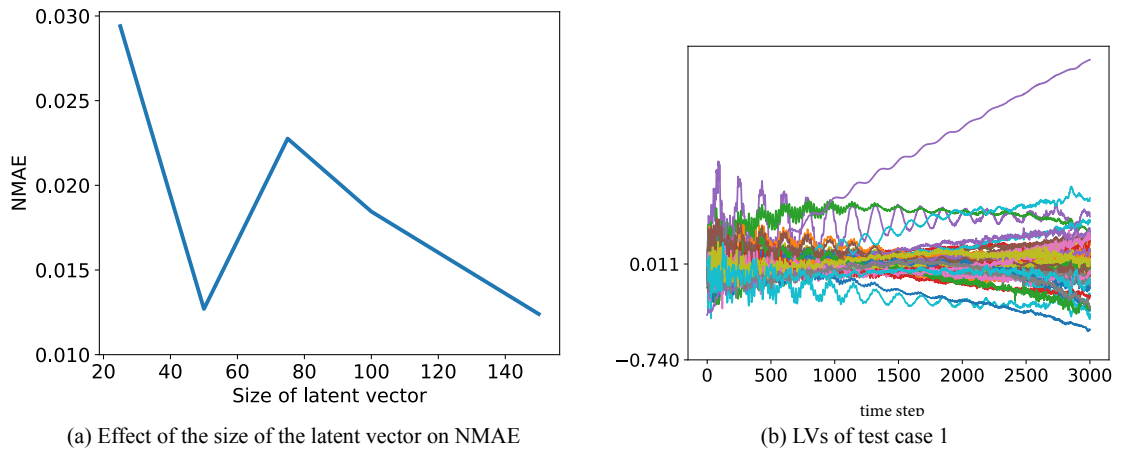
	Translational loading (mm)			Rotational loading (rad)		
	s_x	s_y	s_z	θ_x	θ_y	θ_z
min	-0.5	-0.5	0	-0.1	-0.1	-0.4
max	0.5	0.5	2.6	0.1	0.1	0.4

Table 8. NMAE of test dataset.

	NN I	NN II	NN III	NN IV
NMAE	5.12×10^{-3}	2.85×10^{-3}	3.45×10^{-2}	5.35×10^{-3}

Table 9. Test cases used for verification of PGANN architectures for 3D example.

Test case	Translational loading (mm)			Rotational loading (rad)		
	s_x	s_y	s_z	θ_x	θ_y	θ_z
1	-	0.125	-	0.025	-	-
2	-	-	0.650	-	0.025	-
3	-0.12	-	-	-	-	-0.10
4	-	-0.12	-	-0.03	-	-

**Fig. 24.** Training of NN II

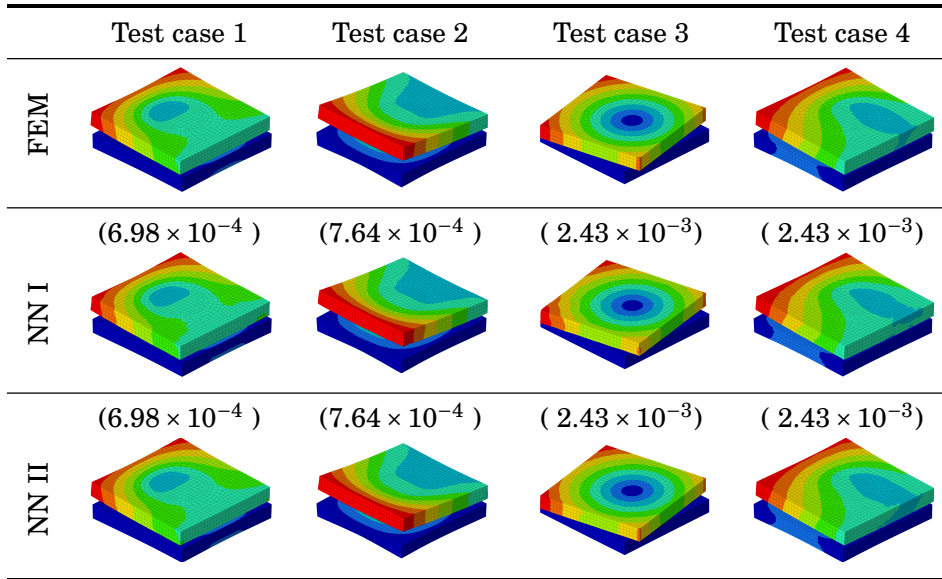


Fig. 25. Comparison between the reconstructed displacement using NN I and NN II of PGANN framework vs. baseline model (FEM) for various test cases. The displacements shown are at the final time, whereas the NMAE values shown in brackets indicate the mean of all time steps.

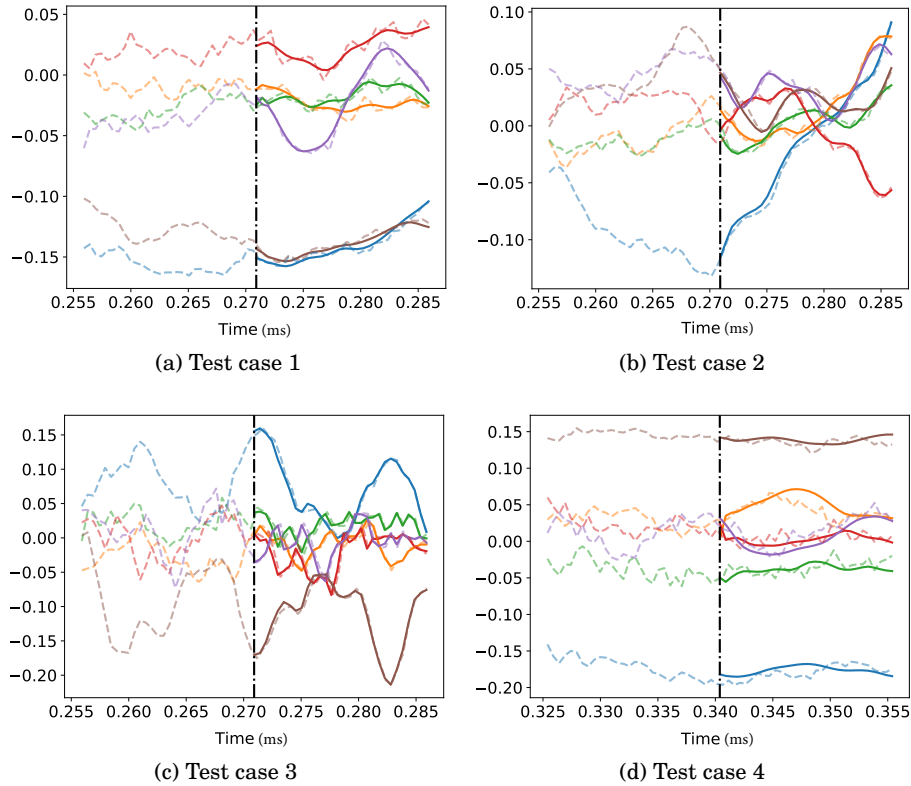


Fig. 26. Prediction of the latent vector $z_{\bar{p}}$ using NN III for various test cases. The solid and dashed lines indicating predicted and reference values, respectively.

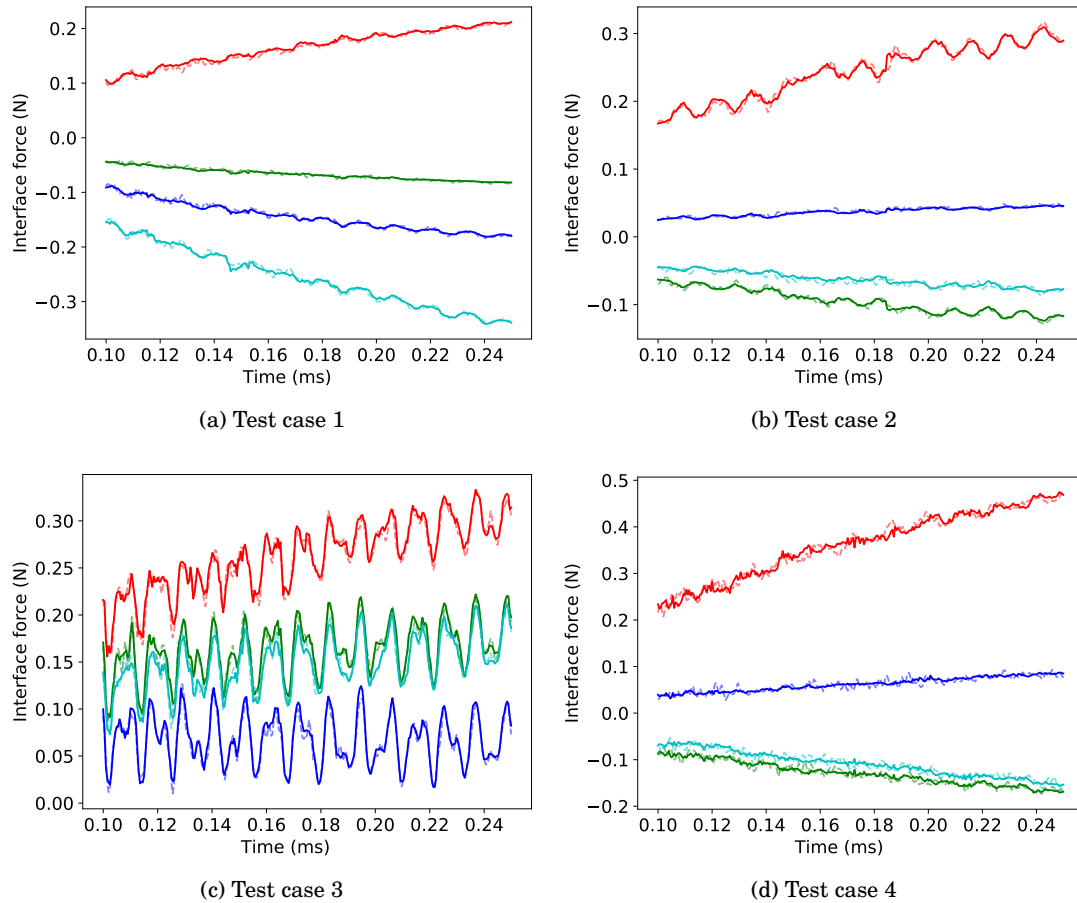


Fig. 27. Prediction of interface forces using NN IV for various test cases: displaying only 4 out of 360 variables in different colours with dashed and solid lines indicating predicted and reference values, respectively

6 Conclusion

The application of PGANNs to the metamodeling of local nonlinear structures was shown in this work. We developed this approach on a 2D plate with a hole as well as a 3D case with spot-welded plates undergoing fast deformation, representing nonlinear elastoplasticity problems. In the proposed network architecture, we introduced a pair of displacement and effective plastic strain layers in between input and output. We compared the results obtained from PGANN architecture to those of traditional FEM methods to demonstrate its ability to generate physically consistent and generalizable solutions; the network yields promising results even for unseen magnitude of data. Despite the success exhibited by the PGANN approach, we have found that it faces challenges when dealing with unseen type of loading when the similar loading direction is not present in the training set. The network architecture trained on single load cases is less accurate on problems with combination loading

types. We found that the reconstruction can be improved by adding more complex loading data to the training set.

Future benchmarking with other model order reduction tools such as POD will provide valuable insights into the effectiveness and limitations of the PGANN approach. Looking ahead, future research will be needed to further improve the accuracy and performance of the PGANN approach, as well as to assess its industrial applicability. To achieve this, we plan to use design of experiment tools to generate parameters required for generating data. Another objective is to enrich the training set with various model parameters such as material and geometric parameters, including plate thickness and diameter of spotweld. Additionally, non-intrusive coupling of NN-based reduced model with explicit FEM solver is a work in progress. We also plan to investigate the use of Physics-Informed Neural Networks to impose basic principles of physics on the metamodel, thereby improving its physical consistency and alleviating training data requirement.

Appendix A

Table A.1. Datasets used to train/cross-validate/test the plate with a hole example.

		Direction	Magnitude of monotonically varying load during ($t_0 \rightarrow t_{\text{end}}$) s : Translational Displacement (mm), θ : rotation angle (rad)
Dataset 2	Dataset 1	s_x	(0 \rightarrow 10), (0 \rightarrow 15), (0 \rightarrow 20), (0 \rightarrow 25), (0 \rightarrow -5), (0 \rightarrow -10), (0 \rightarrow -15), (0 \rightarrow -20)
		s_y	(0 \rightarrow 8), (0 \rightarrow 16), (0 \rightarrow 24), (0 \rightarrow 32), (0 \rightarrow -8), (0 \rightarrow -16), (0 \rightarrow -24), (0 \rightarrow -32)
		s_z	(0 \rightarrow 8), (0 \rightarrow 16), (0 \rightarrow 24), (0 \rightarrow 32), (0 \rightarrow -8), (0 \rightarrow -16), (0 \rightarrow -24), (0 \rightarrow -32)
		θ_x	(0 \rightarrow 0.5), (0 \rightarrow 1.0), (0 \rightarrow 1.5), (0 \rightarrow 2.0), (0 \rightarrow -0.5), (0 \rightarrow -1.0), (0 \rightarrow -1.5), (0 \rightarrow -2.0)
		θ_y	(0 \rightarrow 2), (0 \rightarrow 4), (0 \rightarrow 6), (0 \rightarrow 8), (0 \rightarrow -2), (0 \rightarrow -4), (0 \rightarrow -6), (0 \rightarrow -8)
		$[s_x \ \& \ s_y]$	$[(0 \rightarrow 16) \ \& \ (0 \rightarrow 20)], [(0 \rightarrow 16) \ \& \ (0 \rightarrow -20)],$ $[(0 \rightarrow -16) \ \& \ (0 \rightarrow 20)], [(0 \rightarrow -16) \ \& \ (0 \rightarrow -20)]$
		$[s_x \ \& \ s_z]$	$[(0 \rightarrow 16) \ \& \ (0 \rightarrow 20)], [(0 \rightarrow 16) \ \& \ (0 \rightarrow -20)],$ $[(0 \rightarrow -16) \ \& \ (0 \rightarrow 20)], [(0 \rightarrow -16) \ \& \ (0 \rightarrow -20)]$
		$[s_x \ \& \ \theta_x]$	$[(0 \rightarrow 16) \ \& \ (0 \rightarrow 1.5)], [(0 \rightarrow 16) \ \& \ (0 \rightarrow -1.5)],$ $[(0 \rightarrow -16) \ \& \ (0 \rightarrow 1.5)], [(0 \rightarrow -16) \ \& \ (0 \rightarrow -1.5)]$
		$[s_x \ \& \ \theta_y]$	$[(0 \rightarrow 16) \ \& \ (0 \rightarrow 5.0)], [(0 \rightarrow 16) \ \& \ (0 \rightarrow -5.0)],$ $[(0 \rightarrow -16) \ \& \ (0 \rightarrow 5.0)], [(0 \rightarrow -16) \ \& \ (0 \rightarrow -5.0)]$
		$[s_y \ \& \ s_z]$	$[(0 \rightarrow 16) \ \& \ (0 \rightarrow 16)], [(0 \rightarrow 16) \ \& \ (0 \rightarrow -16)],$ $[(0 \rightarrow -16) \ \& \ (0 \rightarrow 16)], [(0 \rightarrow -16) \ \& \ (0 \rightarrow -16)]$
		$[s_y \ \& \ \theta_y]$	$[(0 \rightarrow 16) \ \& \ (0 \rightarrow 5.0)], [(0 \rightarrow 16) \ \& \ (0 \rightarrow -5.0)],$ $[(0 \rightarrow -16) \ \& \ (0 \rightarrow 5.0)], [(0 \rightarrow -16) \ \& \ (0 \rightarrow -5.0)]$
		$[s_z \ \& \ \theta_x]$	$[(0 \rightarrow 16) \ \& \ (0 \rightarrow 1.5)], [(0 \rightarrow 16) \ \& \ (0 \rightarrow -1.5)],$ $[(0 \rightarrow -16) \ \& \ (0 \rightarrow 1.5)], [(0 \rightarrow -16) \ \& \ (0 \rightarrow -1.5)]$
		$[s_z \ \& \ \theta_y]$	$[(0 \rightarrow 15) \ \& \ (0 \rightarrow 5.0)], [(0 \rightarrow 16) \ \& \ (0 \rightarrow -5.0)],$ $[(0 \rightarrow -16) \ \& \ (0 \rightarrow 5.0)], [(0 \rightarrow -16) \ \& \ (0 \rightarrow -5.0)]$
		$[\theta_x \ \& \ \theta_y]$	$[(0 \rightarrow 1.5) \ \& \ (0 \rightarrow 6.5)], [(0 \rightarrow 1.5) \ \& \ (0 \rightarrow -6.5)],$ $[(0 \rightarrow -1.5) \ \& \ (0 \rightarrow 6.5)], [(0 \rightarrow -1.5) \ \& \ (0 \rightarrow -6.5)]$

Acknowledgments. The authors gratefully acknowledge the support provided by CNRS, EPF, ALTAIR, and STELLANTIS for this research.

References

- [1] A. Reille, V. Champaney, F. Daim, Y. Tourbier, N. Hascoet, D. Gonzalez, E. Cueto, J.L. Duval, F. Chinesta, Learning data-driven reduced elastic and inelastic models of spot-welded patches, *Mech. Ind.* **22**, 32 (2021)
- [2] P. Salvini, F. Vivio, V. Vullo, A spot weld finite element for structural modelling, *Int. J. Fatigue* **22**, 645–656 (2000)
- [3] A. Rupp, K. Storzel, V. Grubisic, Computer aided dimensioning of spot-welded automotive structures, *SAE Tech. Pap.* 950711 (1995)
- [4] R. Courant, K. Friedrichs, H. Lewy, On the partial difference equations of mathematical physics, *IBM J. Res. Dev.* **11**, 215–234 (1967)
- [5] J. Mandel, Balancing domain decomposition, *Int. J. Numer. Methods Biomed. Eng.* **9**, 233–241 (1993)
- [6] C. Farhat, F.-X. Roux, A method of finite element tearing and interconnecting and its parallel solution algorithm, *Int. J. Numer. Meth. Eng.* **32** 1205–1227 (1991)
- [7] P. Ladevèze, O. Loiseau, D. Dureisseix, A micro–macro and parallel computational strategy for highly heterogeneous structures, *Int. J. Numer. Meth. Eng.* **52**, 121–138 (2001)
- [8] H. Ben Dhia, G. Rateau, The Arlequin method as a flexible engineering design tool, *Int. J. Numer. Methods Eng.* **62**, 1442–1462 (2005)
- [9] L. Gendre, O. Allix, P. Gosselet, Non-intrusive and exact global/local techniques for structural problems with local plasticity, *Comput Mech.* **44**, 233–245 (2009)
- [10] L. Gendre, O. Allix, P. Gosselet, A two-scale approximation of the Schur complement and its use for non-intrusive coupling, *Int. J. Numer. Meth. Eng.* **87**, 889–905 (2011)
- [11] J.C. Passieux, J. Réthoré, A. Gravouil, M.C. Baietto, Local/global non-intrusive crack propagation simulation using a multigrid X-FEM solver, *Comput Mech.* **56**, 1381–1393 (2013)
- [12] M. Duval, J.C. Passieux, M. Salaün, et al., Non-intrusive coupling: recent advances and scalable nonlinear domain decomposition, *Arch. Computat. Methods Eng.* **23**, 17–38 (2016)
- [13] G. Guguin, O. Allix, P. Gosselet, On the computation of plate assemblies using realistic 3D joint model: a non-intrusive approach, *Adv. Model. Simul. Eng. Sci.* **3** (2016)
- [14] T. Chantrait, J. Rannou, A. Gravouil, Low intrusive coupling of implicit and explicit time integration schemes for structural dynamics: application to low energy impacts on composite structures, *Finite Elem. Anal. Des.* **86**, 23–33 (2014)

- [15] O. Bettinotti, O. Allix, U. Perego, V. Oancea, B. Malherbe, Simulation of delamination under impact using a global–local method in explicit dynamics, *Finite Elem. Anal. Des.* **125**, 1–13 (2017)
- [16] F. Chinesta, A. Huerta, G. Rozza, K. Willcox, Model order reduction, in: E. Stein, R. de Borst, T. Hughes (Eds.), *The Encyclopedia of Computational Mechanics*, 2nd ed., John Wiley & Sons Ltd., 2015
- [17] G. Rozza, D.B.P. Huynh, A.T. Patera, Reduced basis approximation and a posteriori error estimation for affinely parametrized elliptic coercive partial differential equations, *Arch. Computat. Methods Eng.* **15**, 229 (2008)
- [18] S.K. Kauwe, J. Graser, A. Vazquez, T.D. Sparks, Machine learning prediction of heat capacity for solid inorganics, *Integr. Mater. Manuf. Innov.* **7**, 43–51 (2018)
- [19] P. Baldi, K. Bauer, C. Eng, P. Sadowski, D. Whiteson, Jet substructure classification in high-energy physics with deep neural networks, *Phys. Rev. D* **93**, 9 (2016)
- [20] C. Tesche, C.N. De Cecco, S. Baumann, et al., Coronary CT angiography–derived fractional flow reserve: machine learning algorithm versus computational fluid dynamics modeling. *Radiology* **288**, 64–72 (2018)
- [21] B. Alipanahi, A. Delong, M. TWeirauch, B.J. Frey, Predicting the sequence specificities of DNA-and RNA-binding proteins by deep learning, *Nat. Biotechnol.* **33**, 831–838 (2015)
- [22] J. Willard, X. Jia, S. Xu, et al., Integrating scientific knowledge with machine learning for engineering and environmental systems, *ACM Comput. Surv.* **55**, 1–37 (2022)
- [23] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Computat. Phys.* **378**, 686–707 (2019)
- [24] F. As’ad, P. Avery, C. Farhat, A mechanics-informed artificial neural network approach in data-driven constitutive modelling, AIAA 2022-0100. AIAA SCITECH 2022 Forum, 2022
- [25] J.S. Read, X. Jia, J. Willard, A.P. Appling, et al., Process-guided deep learning predictions of lake water temperature, *Water Resour. Res.* **55**, 9173–9190 (2019)
- [26] P. Sturmfels, S. Rutherford, M. Angstadt, et al., A domain guided CNN architecture for predicting age from structural brain images, arXiv:1808.04362 (2018)
- [27] A. Daw, R.Q. Thomas, C.C. Carey, J.S. Read, A.P. Appling, A. Karpatne, Physics-guided architecture (PGA) of neural networks for quantifying uncertainty in lake temperature modelling, arXiv:1911.02682 (2019)
- [28] F. Hamilton, A.L. Lloyd, K.B. Flores, Hybrid modeling and prediction of dynamical systems, *PLoS Computat. Biol.* **13**, pp. e1005655 (2017)
- [29] T. Belytschko, W.K. Liu, B. Moran, *Nonlinear Finite Elements for Continua and Structures*, John Wiley & Sons, Ltd, 2000
- [30] O. Bettinotti, O. Allix, U. Perego, V. Oancea, B. Malherbe, A fast weakly intrusive multiscale method in explicit dynamics, *Int. J. Numer. Methods Eng.* **100**, 577–595 (2014)
- [31] M. Mozaffar, R. Bostanabad, W. Chen, K. Ehmann, J. Cao, M. Bessa, Deep learning predicts path-dependent plasticity, *Proc. Natl. Acad. Sci. U.S.A.* **116**, 26414–26420 (2019)
- [32] M.B. Gorji, M. Mozaffar, J.N. Heidenreich, J. Cao, D. Mohr, On the potential of recurrent neural networks for modeling path dependent plasticity, *J. Mech. Phys. Solids* **143**, 103972 (2020)
- [33] F. Masi, I. Stefanou, P. Vannucci, V. Maffi-Berthier, Thermodynamics-based artificial neural networks for constitutive modelling, *J. Mech. Phys. Solids* **147**, 1–28 (2021)
- [34] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, Z.B. Wojna, Rethinking the inception architecture for computer vision, *CoRR*, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 2016, pp. 2818–2826 2016.

Cite this article as: A. Pulikkathodi, E. Lacazedieu, L. Chamoin, J. P. Berro Ramirez, L. Rota, and M. Zarroug, A Neural Network-Based Data-Driven local modeling of spotwelded plates under impact, *Mechanics & Industry* **24**, 34 (2023)