Received: Added at production | Revised: Added at production | Accepted: Added at production

DOI: xxx/xxxx

RESEARCH ARTICLE

DeepPhysics: a physics aware deep learning framework for real-time simulation

Alban Odot | Ryadh Haferssas | Stephane Cotin

[1]MIMESIS team, Inria, 1 Place de l'Hopital, 67000 Strasbourg, France

Correspondence
Stephane Cotin
Email: stephane.cotin@inria.fr

Abstract

Real-time simulation of elastic structures is essential in many applications, from computer-guided surgical interventions to interactive design in mechanical engineering. The Finite Element Method is often used as the numerical method of reference for solving the partial differential equations associated with these problems. Yet, deep learning methods have recently shown that they could represent an alternative strategy to solve physics-based problems [1,2,3]. In this paper, we propose a solution to simulate hyper-elastic materials using a data-driven approach, where a neural network is trained to learn the non-linear relationship between boundary conditions and the resulting displacement field. We also introduce a method to guarantee the validity of the solution. In total, we present three contributions: an optimized data set generation algorithm based on modal analysis, a physics-informed loss function, and a Hybrid Newton-Raphson algorithm. The method is applied to two benchmarks: a cantilever beam and a propeller. The results show that our network architecture trained with a limited amount of data can predict the displacement field in less than a millisecond. The predictions on various geometries, topologies, mesh resolutions, and boundary conditions are accurate to a few micrometers for non-linear deformations of several centimeters of amplitude.

KEYWORDS:
Neural Network, Deep Learning, Finite Element Method, Newton-Raphson, Real-Time, Physics Informed Neural Network

1 | INTRODUCTION

Recently, with the increase in GPU computational power, machine learning started to revolutionise several fields, in particular computer vision, language processing, and image processing. Deep-learning is an area of machine learning that has demonstrated a strong ability at extracting high-level representations of the relation between a given input and its corresponding output as opposed to task specific algorithms. In other words, given enough inputs, such techniques can approximate the relation to the corresponding outputs without any prior knowledge.

Numerical methods such as the Finite Element Method [4] (FEM) are vastly used in science to solve partial differential equations (PDE) on complex domains, for which analytical solutions are not possible. Computing the non-linear deformation of mechanical structures is one of the fields which deals with such equations and uses FEM to approximate the solution. The main benefits of the FEM are its accuracy and well-grounded mathematical foundations. However, when the problem complexity increases

and accurate solutions are sought, the combination of high resolution meshes and non-linear constitutive laws usually leads to computation times that are too high for certain applications.

Our main objective here is to leverage the advantages of deep learning methods (in particular the ability to learn complex relations between inputs and outputs of a model) and the solid scientific foundations of the FEM to obtain very fast, yet accurate solutions of nonlinear elasticity problems. Since multiple strategies have been proposed over the last decades to reduce the computation time of FEM, the following sections describes some of these techniques and discusses about their limits compared to our expectations.

Domain decomposition

As the name suggests, Domain Decomposition [5] relies on a smart decomposition of the initial domain into multiple sub-domains. The associated (simpler) sub-problems are coupled via disjoint or overlapping boundaries. Different algorithms have been proposed to solve such problems. Sub-structuring algorithms such as BNN [6] or FETI [7] can only handle non-overlapping domain decomposition while others like Schwarz [8] or Lions [9] methods can work on both overlapping and non-overlapping sub-domains. Combined with parallel processing, it is possible to achieve significant speedups while maintaining good convergence properties [10]. However, the optimal gain for Domain Decomposition methods is observed when dealing with very large problems (i.e. with millions of degrees of freedom) [10]. On smaller resolution meshes, or when real-time computation is needed, the speedup obtained by this approach is more limited as the computation of boundary interactions becomes predominant [11]. Since we want to favour fast or real-time computation of nonlinear elastic structures, domain decomposition techniques are not well suited.

Proper Orthogonal Decomposition

Another approach to improve computation times is Model Order Reduction. Among this class of methods, Proper Orthogonal Decomposition (POD) [12] has proved to perform well on a wide variety of problems ranging from fluid dynamics [13 14], soft robotics [15], aeronautics [16], optimal design [17] and many others. This approach aims at reducing the number of Degrees of Freedom (DOFs) by analysing deformation modes and discarding the least significant ones. These modes are obtained via a "data set" of P samples that are put together as a matrix $\mathbf{Q}$ of size $N \times P$ where N is the number of DOFs. The eigenvalues and their associated eigenvectors are then computed from the self-adjoint matrix $\mathbf{QQ}^T$. When the displacement field is well behaved, the magnitude of the eigenvalues decreases quickly, showing that the object is mostly characterised by the first few modes of deformation (i.e. the eigenvalues with the highest norm). Follows a step where the lowest eigenvalues (usually anything 10^{-8} times smaller than the magnitude of the first one) are removed. The simulation is computed in the reduced space thus speeding up the solving of the system. This method relies on a trade-off between accuracy and speed. There are multiple factors impacting the quality of the simulation and the gain such as how magnitudes of eigenvalues decrease and the magnitude of the cutoff value. Although it introduce errors in the simulation, the computed reduced space can be used to solve "similar" problems with slight changes in the boundary conditions or model parameters [18 19]. This allows to reuse the reduced model for a variety of scenarios, therefore reducing the overall computational cost of the method. Artificial neural networks have already proven to be resilient to geometry variations, Pfeiffer *et al.* [20] trained a neural network on randomly generated meshes to predict displacement fields. We will demonstrate in this article that the presented approach is at least as fast as the POD while providing more flexibility to model parameters variations.

GPU-acceleration

Many publications have addressed the problem of computational performance for FE simulations through GPU-accelerated approaches. While the use of a parallel implementation can speedup the computation of the local, element-wise, stiffness matrices, these methods essentially aim at solving the global linear system associated with the linearization of the problem.

In the case of an iterative method, such as the conjugate gradient, most of the gain can be achieved by improving the sparse matrix-vector operations. Multiple methods have been explored to implement efficiently these operations on the GPU [21]. Mueller *et al.* [22] or Martínez-Frutos *et al.* [23] use the fact that CG iterations can be performed without explicitly assembling the whole matrix. This has the advantage of reducing the memory bandwidth while being fast and stable. As an example, Allard *et al.* [24] proposed a cache optimisation process for a co-rotated formulation of a linear elastic model. With this method, a mesh composed of 20k tetrahedra can be simulated in about 2 ms [24]. This approach is however too specific to be applied to other materials such as hyperelastic ones.

In the case of direct solvers, the assembly of the global matrix is required to compute the decomposition or factorisation of the system. Dziekonski *et al.* [25] and Mueller *et al.* [26] proposed to assemble the matrix directly on the GPU thus reducing memory transfer and speeding up the assembly. In this case also, the method requires a model specific algorithm and cannot provide

a good combination of heterogeneous CPU/GPU simulations. One could also optimise the construction of the matrices, using very efficient implementation such as the one provided by the Eigen[27] library.

Overall, despite some limitations, GPU-based FEM algorithms provide fast and accurate results. Yet, as for the POD, GPU-based FEM falls under the category of task specific algorithms. It de facto encounters the same limitations when using different hyper-elastic laws to represent complex objects.

Physics Informed Neural Network

Recently, Raissi et al.[1] proposed a novel method for the approximation of the solution of PDEs using Artificial Neural Networks. Named Physics Informed Neural Networks (PINN), they leverage the recent developments in automatic differentiation[28] to differentiate neural networks with respect to their input coordinates and model parameters using an unsupervised learning process. This method relies on a novel formulation of the objective/loss function. In physics, problems are usually composed of three part: initial conditions, boundary conditions, and a governing equation. In order to satisfy the problem, a solution has to hold true the conditions while solving the governing equation. Raissi *et al.* use this pattern to formulate the loss function where each part of the problem will be evaluated with the output of the network. The Mean Squared Error (MSE) computed for each part from the output is then summed up and, via an optimisation algorithm, the weights and biases of the network are updated. This approach has proved successful in several fields and application cases such as cardiac activation mapping[29], modelling of fractures[30] or fluid dynamics[31]. PINN is a fast, efficient and elegant way of training a neural network to solve ordinary and partial differential equations. However, the approach constrains the inputs to be the differentiation variables. Furthermore it does not support variations of the boundary conditions since their formulation appears in the loss function. As a result, each time the problem is changed (i.e. different Neumann boundary conditions, different model parameters, etc.) the network has to be trained again. This limits significantly the benefit of the approach when compared, for instance, to GPU-based methods or POD techniques. Our method allows for the computation of various deformations from a single training process. The method uses external forces applied on the object as inputs of the network which are not the differentiation variables of the constitutive laws, thus, making the Raissi *et al.*'s hypothesis not applicable in this case.

Data-driven learning process

Data-driven methods are a class of machine learning techniques that can deal with a wide variety of physics-based problems ranging from radio-frequency or microwave modeling[32] to fluid mechanics[33] and solid mechanics[34]. Data are generated from real world acquisitions[35] or can be computer-generated[36]. Once the data are processed or generated, the inputs are fed to the network, and the outputs are compared to the ground truth via a loss function (usually the MSE). The error gradient is then computed and, finally, the weights and biases of the artificial neural network are updated using an optimization algorithm. Such an approach is helpful to approximate problems that do not have a mathematical model, such as image animation[37] or problems that do not fit into the PINN framework.

Prior work has been done on the topic of simulating hyper-elastic materials using data-driven approaches. Cloth deformation is an important subject in the field of physics-based animation, and Wang *et al.*[38] used measured data to build a piece-wise bending and stretching model to compute the non-linear dynamic of cloth material. Xu *et al.*[39] following the idea of Kim *et al.*[40] proposed a technique of mix and match to generate meshes that fit the desired pose. More recently, Santestaben *et al.*[41] used a neural network to generate non-linear garment wrinkles. For applications involving the deformation of elastic solids, Brunet *et al.*[42] proposed a method to compute hyper-elastic volume deformation of a liver in real-time from a set of Dirichlet boundary conditions. Finally, Meister *et al.*[43] and Mendizabal *et al.*[34] proposed neural networks able to predict the deformation of an elastic structure given variable external forces. The later approximate static deformations using a Convolutional Neural Network (CNN). This method requires to immerse the object within a topological grid in order to perform convolutions. Neumann boundary conditions are transferred to the grid through a mapping. The deformation is then computed on the grid by the network and applied on the object using the inverse mapping. These methods provide a fast estimations of the displacement field. Their principal drawbacks are related to the data generation process, and the limited accuracy of the solution. The amount of simulations required to train the network (several hundred simulations per boundary node) becomes time consuming when the grid resolution is increased, and so does the training time. The predictions accuracy is in the range $[10^{-4}; 10^{-3}]$ of the object scale, which is acceptable for some applications but not sufficient for others.

In this article we present an improvement of this latter class of methods. We first briefly describe the network architecture. In a second time we propose a novel technique to generate meaningful deformations based on modal analysis. To follow this, we present a physics-aware loss function that significantly improves the training and prediction accuracy. As our last contribution

we propose a variation of the Newton-Raphson algorithm that uses the prediction of the ANN. Finally, we analyse and discuss the results of our method.

2 | METHOD

The deep learning aided simulation can be split into two main techniques: the PINNS introduced by Raissi et al.[1] in 2019 and the simulation-based approach. In this paper we focus on the latter, with the long-term objective to learn also from real data and to obtain as-generic-as-possible neural networks, able to predict elastic deformations for a wide range of cases.

2.1 | Neural network architecture

In recent years, Deep Neural Networks (DNN) have led to many successful applications such as image recognition and natural language processing thanks to their exceptional expressibility. Starting from a simple feed forward fully-connected neural network, scientists have proposed more complex architectures such as Convolutionnal Neural Network (CNN), and combined multiple architectures to suit their problems and constraints. CNNs have shown some incredible results in their generalisation capabilities and high-level features extraction.

In a context similar to this paper, Mendizabal et al.[34] used a specific CNN architecture named Unet. This network takes as input a $3 \times n_x \times n_y \times n_z$ force tensor. It corresponds to the force interpolated on each node of the axis aligned bounding grid. It outputs a similarly sized tensor corresponding to the grid displacement at each nodes.

In this section we will demonstrate that even though CNN are a good and viable option in this context, simpler is better. With a simple Fully Connected Neural Network (FCNN) composed of two hidden layers, we manage to obtain an excellent prediction accuracy with a mean squared error between 10^{-7} and 10^{-10}.

2.1.1 | Proposed Architecture

A feed forward FCNN can be assumed to be the stack of the input layer, multiple hidden layers and the output layer. The connection between two adjacent layers can be expressed in the form of a tensor, as follows

$$\mathbf{z}_i = \alpha(\mathbf{W}_i \mathbf{z}_{i-1} + \mathbf{b}_i), \text{ for } 1 \leq i \leq n+1 \qquad (1)$$

where n is the total number of layers, $\alpha(.)$ denotes the activation function acting element wise, $\mathbf{z}_0$ and $\mathbf{z}_{n+1}$ denotes the input and output tensors respectively, $\mathbf{W}_i$ and $\mathbf{b}_i$ are the trainable weight matrices and biases in the i^{th} layer. Activation functions play an important role in the learning process of the neural networks. Their role is to apply a (nonlinear) function to the output of the previous layer and pass it to the next one.

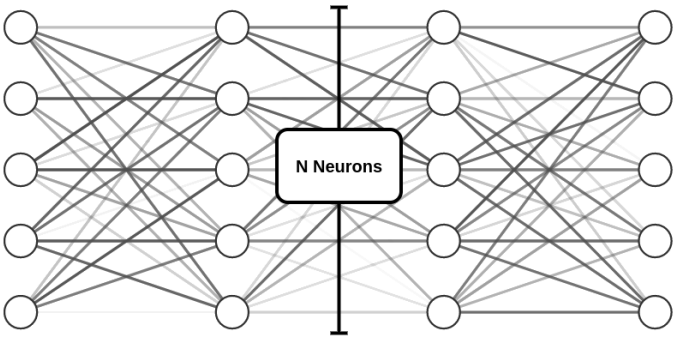


FIGURE 1 The proposed architecture is composed of 4 fully connected layers of size the number of dofs with a PReLU activation function.

Only non-linear differential functions are considered since they allow for a useful information to be computed during the back propagation. Here the activation function $\sigma(.)$ is a PReLU[44], which stands for Parametric Rectified Linear Unit. This function has multiple advantages such as: valuation and gradient computation speed, and neuron-wise adaptive activation.

$$\text{PReLU}(x) = \begin{cases} x & x \in \mathbb{R}^+ \\ ax & x \in \mathbb{R}^- \ \ a \in \mathbb{R}^+ \end{cases}$$

The negative part has a learnable parameter a, allowing us to adaptively consider both positive and negative inputs.

Such architecture is easy to implement and fast, but has two drawbacks. The first is the number of parameters that need to be learnt by the network. This number is given by the polynomial $4N^2 + 7N$, where N is the number of degrees of freedom. The

quadratic growth is a limitation for both GPU memory and gradient computation. The second drawback of such an architecture is that FCNN are prone to over-fitting even though this phenomenon has not been observed in any of our scenarios.

2.2 | Mode-based data set generation

In simulation-based approaches, the generation of a synthetic data set often represents an important part of the whole training process. The naive technique, which consists in applying random external forces at each node of the FE domain) provides minimal control over the representativeness of the data set. Defining an optimal combination of external forces is difficult and depends on the model and has to be done mesh wise in order to obtain specific and interesting deformations.

Our method is based on modal analysis of the global stiffness matrix $\mathbb{K}'$ arising from a finite element method. Thus, it can be applied with any constitutive model, mesh resolution or finite element type. While a modal analysis is used to produce the external forces, no reduction is performed as shown in equations 10 and 11 thus preserving the physical properties of the model. It allows the computation of external forces that are responsible for the most significant deformations of a given mesh.

In order to explain both the method and the notations we start this section by a quick overview of the theory behind continuum mechanics. More detailed knowledge can be found, for instance, in[45]. Finally, we demonstrate how to compute the external forces from the deformation modes.

2.2.1 | Numerical simulation of hyper-elasticity problems

For the analysis of nonlinear initial boundary value problem in continuum mechanics, a coupled system of partial differential equations has to be solved which consist of kinematics relations, local balance of momentum and the constitutive equations. We first present the hyper-elastic strong form and its alternative weak formulation Finally we proceed to present one of the approaches to solve it.

The boundary-value problem reads as follows : find $\mathbf{d} : \bar{\Omega} \longrightarrow \mathbb{R}^d$

$$\begin{aligned} -\nabla \cdot \underline{\mathbf{P}}(\mathbf{u}) &= \mathbf{f} && \text{in } \Omega \\ \mathbf{u} &= \mathbf{g} && \text{on } \Gamma_D \subset \partial\Omega \\ \underline{\mathbf{P}}(\mathbf{u})\mathbf{n} &= \mathbf{h} && \text{on } \Gamma_N \subset \partial\Omega \\ \underline{\mathbf{P}}(\mathbf{u})\mathbf{n} + \alpha\mathbf{u} &= \mathbf{0} && \text{on } \Gamma_R \subset \partial\Omega \end{aligned} \tag{2}$$

Where $\Omega \subset \mathbb{R}^d (d = 2, 3)$, $\partial\Omega$ the boundary of Ω, $\underline{\mathbf{P}}$ is the first Piola-Kirchhoff, $\mathbf{f}$ is the vector of the external forces, $\mathbf{u}$ is the displacement vector, $\mathbf{g}$ a suitable Dirichlet's boundary conditions on Γ_D, $\mathbf{h}$ a Neumann's boundary conditions on Γ_N, $\mathbf{n}$ the normals on $\partial\Omega$, α the elastic coefficient associated with the Robin-Type boundary condition on Γ_R.

Let's introduce two functional spaces:

$$\begin{aligned} V &= \{\mathbf{w} \in \left[H^1(\Omega)\right]^d \ : \ \mathbf{w}|_{\Gamma_D} = \mathbf{g}\} \\ V_D &= \{\mathbf{w} \in \left[H^1(\Omega)\right]^d \ : \ \mathbf{w}|_{\Gamma_D} = \mathbf{0}\} \end{aligned} \tag{3}$$

The weak formulation of the boundary value problem 2 reads:
Find $\mathbf{u} \in V_D$ such that

$$\int_\Omega \underline{\mathbf{P}}(\mathbf{u}) \ : \ \nabla\mathbf{w} \, d\Omega + \int_{\Gamma_R} \alpha\mathbf{u} \cdot \mathbf{w} \, d\Gamma = \int_\Omega \mathbf{f} \cdot \mathbf{w} \, d\Omega + \int_{\Gamma_N} \mathbf{h} \cdot \mathbf{w} \, d\Gamma \quad \forall \mathbf{w} \in V \tag{4}$$

Let's introduce a Finite Element partition Ω_h of the domain Ω from which we construct a conforming FE space

$$X_h = \left\langle x_1, \dots, x_{N_h} \right\rangle \subset \left[H^1(\Omega)\right]^d \tag{5}$$

We also define $V_h = X_h \cap V$ and $V_{Dh} = X_h \cap V_D$. The finite element approximation of equation 4 can be written as :
Find $\mathbf{u}_h \in V_{Dh}$ such that:

$$\int_\Omega \underline{\mathbf{P}}(\mathbf{u}_h) \ : \ \nabla\mathbf{w} d\Omega + \int_{\Gamma_R} \alpha\mathbf{u}_h \cdot \mathbf{w} d\Gamma = \int_\Omega \mathbf{f} \cdot \mathbf{w} d\Omega + \int_{\Gamma_N} \mathbf{h} \cdot \mathbf{w} d\Gamma \quad \forall \mathbf{w} \in V_h \tag{6}$$

$$\mathbf{f}^i(\mathbf{u}_h) = \mathbf{f}^e \tag{7}$$

$$\mathbf{R}(\mathbf{u}_h) = \mathbf{f}^i(\mathbf{u}_h) - \mathbf{f}^e = \mathbf{0} \tag{8}$$

Where $\mathbf{f}^i(\mathbf{u}_h)$ are the internal forces and $\mathbf{f}^e$ the external ones. $\mathbf{R}(\mathbf{u}_h)$ is called the residual vector.

From now on, we will only work with equation 6 and in order to do not overload the notations we will denote $\mathbf{u} \in \mathbb{R}^{N_h}$ the vector of coefficients in the expansion of $\mathbf{u}_h$ with respect to the FE basis function φ_i.

Finding the roots of a function has been a hot topic for multiples centuries now. From these researches emerges the well studied Newton-Raphson process that has proven to be very effective at finding roots of non-linear equations. As of today, this algorithm and its variations are commonly used to solve FE problems and is the one we picked to solved such problems. In the next subsection we will see how we use the tensor $\mathbb{K}(u)$ in order to obtain a given displacement, and its corresponding forces.

2.2.2 | Modal forces

External force distribution on the object boundary (i.e. the choice of Neumann boundary conditions) has an direct impact on the resulting deformation, yet it is not always intuitive to know how to create a given displacement from a set of external forces. The opposite is also true, where two different distributions of forces can actually give a similar deformation. This is why it is difficult to generate an optimal training data set composed of a large variety of (non-redundant) deformations. For example, Mendizabal et al.[34] use a heuristics to generate the training data set, which requires to be tuned for each application in order to be computationally stable.

One way to maximise the information that goes through the network would be to construct a deformation basis. This basis can then be interpolated within this space to obtain an approximation of the constitutive law, as shown in Figure 2. This might

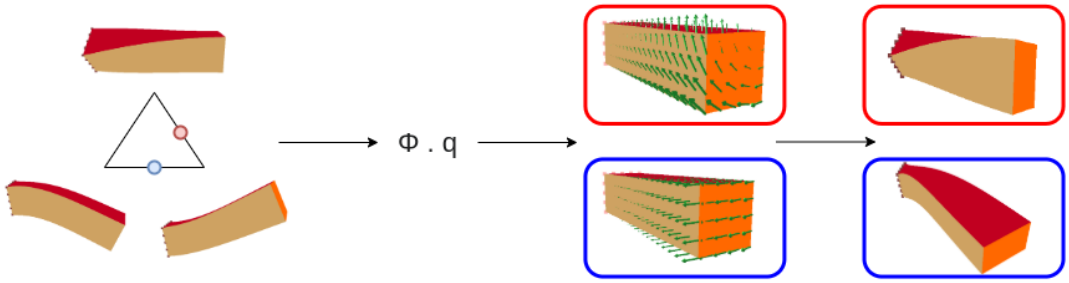


FIGURE 2 Example of a deformation basis and its resulting forces / linear deformations. Φ and $\mathbf{q}$ are the matrix and vector described in equation 10. The volume gain is a known artefact of linear elasticity laws when rotations are involved.

be sufficient for linear elasticity problems, but the hyper-elastic materials have a more complex relation between forces and displacements. For a non-linear elastic model, the stiffness matrix is itself a function of the DOF values (or their derivatives). We need an iterative approach, such as the Newton-Raphson method, to solve the nonlinear equations and compute the correct deformation caused by the external forces. This is described in figure 3.

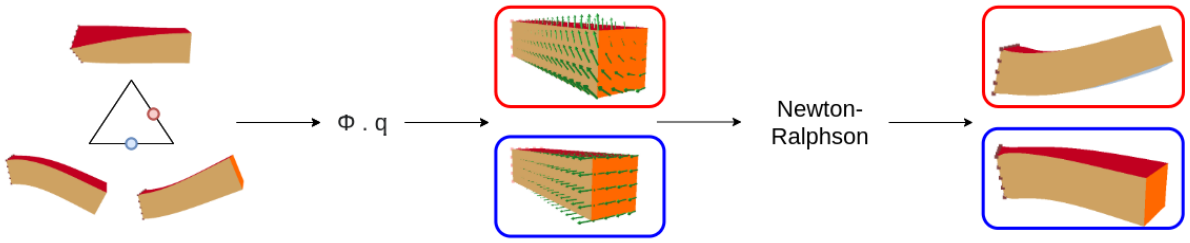


FIGURE 3 Example of a deformation basis and its resulting forces / non-linear deformations. Φ and $\mathbf{q}$ are the matrix and vector described in equation 10.

In this subsection we explain how we generate this deformation basis and compute external forces from modes. Modal analysis relies on the eigenvectors of a given matrix, in this case we consider the tangent stiffness matrix $\mathbb{K}(\mathbf{u})$ from equation 19.

Each eigenvector corresponds to a deformation mode. As a result, any deformation of an elastic object can be represented as a combination of these vectors. Hyper-elasticity problems cannot be represented by a single matrix hence the system of equations is solved using a sequence of linearizations. Since this process is time consuming, we choose here to apply the modal analysis only the tangent stiffness matrix associated with the rest configuration of the elastic body.

The linearisation process of the Newton-Raphson (equation 16 and 19) leads to the equation

$$\mathbb{K}(\mathbf{u}^k) \cdot \boldsymbol{\delta}\mathbf{u}^k = \mathbf{R}(\mathbf{u}^k) \tag{9}$$

For simplicity we only write $\mathbb{K}$ instead of $\mathbb{K}(\mathbf{u} = \mathbf{u}^k)$. From its eigen decomposition $\mathbb{K} = \Phi\Lambda\Phi^T$ we can write

$$\begin{aligned} \mathbb{K} \cdot \boldsymbol{\delta}\mathbf{u}^0 = \mathbf{R}(\mathbf{u}^0) \iff \\ \Phi\Lambda\Phi^T \cdot \boldsymbol{\delta}\mathbf{u}^0 = \mathbf{f}^i(\mathbf{u}^0) - \mathbf{f}^e \quad \text{with} \quad \mathbf{R}(\mathbf{u}^k) = \mathbf{f}^i(\mathbf{u}^k) - \mathbf{f}^e \iff \\ \Phi(\Lambda\Phi^T \cdot \boldsymbol{\delta}\mathbf{u}^0) = \mathbf{f}^i(\mathbf{u}^0) - \mathbf{f}^e \iff \\ \Phi \cdot \mathbf{q} = -\mathbf{f}^e \quad \text{with} \quad \mathbf{f}^i(\mathbf{u}^0) = \mathbf{0} \end{aligned} \tag{10}$$

where Φ is the matrix composed of the eigenvectors of $\mathbb{K}(\mathbf{u}^0)$, Λ is the diagonal matrix composed of the corresponding eigenvalues of $\mathbb{K}$, and finally $\mathbf{q} = \Lambda\Phi^T\boldsymbol{\delta}\mathbf{u}^0$. Each coefficient of $\mathbf{q}$ reflect how much its associated deformation mode is present in the applied external force.

The computation of the eigenvectors can be time consuming on very fine meshes, but is reasonable in the scenarios we are considering (around 4 minutes and 20 seconds for 15,000 DOFs). Yet, as it occurs only once in the whole training phase hence, this computation time remains negligible when compared to the whole training process.

The selection of the first few modes (setting the first values of $\mathbf{q}$ to a non zero value) generates force vectors which correspond to the principal deformations of the object. We only consider a few modes to generate the external forces. The left-hand side of the equation is unchanged hence, the physical properties of the systems remain the sames. The equation simply becomes:

$$\mathbb{K} \cdot \boldsymbol{\delta}\mathbf{u} = \mathbf{f}^i(\mathbf{u}) + \Phi \cdot \mathbf{q} \tag{11}$$

The size of the training data set is then controlled by the sampling of those modes. From tests performed on various meshes we noticed that with only the first three to five modes and an overall sampling of 1,000 to 2,000 simulations we can create a learnable and generalisable data set leading to accurate predictions. To further improve the generalisation ability of the network, we apply the previously computed forces on random areas of random size over the mesh boundary. This gives us the ability to better represent local deformations. With this approach the parameter tuning only consists on choosing the number of modes and the sampling of each mode. In the following sections we will consider that we have no prior knowledge on the use case scenario hence, we will use a uniform sampling of each mode. However, application knowledge can be used to select particular modes and specific sampling in order to have a more representative training data set.

Depending on the material characteristics, one might need to apply very large or very small forces, to obtain the expected amplitude of deformation. Making a neural network robust to such extreme inputs and outputs is proven difficult, and we use a batch normalisation process on the input forces and output displacements to prevent this issue.

2.3 | Loss function

Deep learning uses the gradient of the error to learn from its previous mistakes. The error is computed via a function called loss function. There are various factors involved in choosing a specific loss function such as: ease of calculating the derivatives, the percentage of outliers in the data set, the type of machine learning algorithm, etc. One of the most known and used function is the mean squared error (MSE).

$$\text{MSE}(\mathbf{u}) := \frac{\sum_{i=0}^{N-1}(\mathbf{u}_i - \mathbf{v}_i)^2}{N} \tag{12}$$

Where $\mathbf{u}$ is the value predicted by the ANN and $\mathbf{v}$ the ground truth. It has nice mathematical and computational properties and values at 0 only when the output / prediction of the network exactly matches the expected value. $\mathbf{v}$ is the ground truth divided by the length of the longest axis of the object-oriented bounding box in order to become scale independent.

However, the MSE metric being based solely on geometry, it does not enforce physically-correct configurations, as illustrated in figure 4. This can introduce important errors in the prediction when complex deformations take place. Since the MSE alone is not a good enough discriminant for such problems, another metric needs to be proposed.

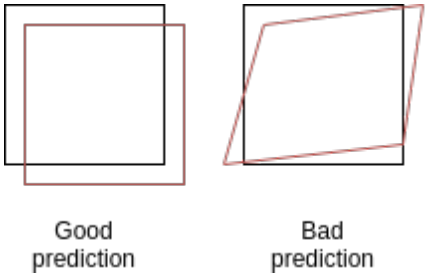


FIGURE 4 Example of two predictions (red) that have similar errors compared with the ground truth (black). The right-hand side prediction is 22% more accurate than the left one according to the MSE, yet we would prefer the left one that better preserve some physical properties.

To improve the discriminative aspect of the loss function, and enforce its physics-based nature, we propose to rely on the internal forces of the elastic model. We know that at equilibrium the total amount of force in a system sums up to zero. Using equation 9 we can then compute the *residual forces* represented by the vector $\mathbf{R}(\mathbf{u})$ and we call residual its euclidean norm. This criterion and its relative counterpart describe with a good accuracy the state of the system.
While we want to keep the geometric convergence from the MSE we also want to enhance it with some physical knowledge. From these statements two basic approaches emerge.

$$L_r^+(\mathbf{u}) = \text{MSE}(\mathbf{u}) + \frac{\|\mathbf{R}(\mathbf{u})\|}{\|\Phi \mathbf{q}_{max}\|} \tag{13}$$

$$L_r^*(\mathbf{u}) = \text{MSE}(\mathbf{u}) \times \frac{\|\mathbf{R}(\mathbf{u})\|}{\|\Phi \mathbf{q}_{max}\|} \tag{14}$$

Where $\mathbf{q}_{max}$ is given by $\max_i \|\mathbf{q}_i\|$. We recall that Φ is the matrix composed of the eigenvectors of $\mathbb{K}$. Although numerically correct, the formulation of the L_r^+ loss poses some issues when it comes to its implementation. In our approach, and others dealing with similar problems, the simulations are computed using an external software library. In order to evaluate the residual forces, outputs of the ANN have to be cast in the corresponding framework type. During that process, we loose the derivation graph previously computed using autograd[46]. Due to this lack of derivation graph, during the back propagation, the gradient of the normalised residual cannot be computed.

The formulation of the L_r^* loss fixes previously mentioned problems. During the computation of the back propagation, and using the linearity of the gradient, the normalised residual will be multiplied to the gradient of the MSE, thus, its value will propagate through the network. The normalised residual acts as a meaningful non-constant multiplicative coefficient which positively affect the learning process. Furthermore, the normalised residual is robust to scale-independent optimisation algorithms such as ADAM[47] since its value differs at each batch of data.

In general, one can enhance the MSE loss function by any combination of the loss and the residual as long as the dimension analysis allows it. We want to bring the attention of the reader on the fact that the behaviour of the differential is also important and has to be taken into account in the conception process. In our case, and for the remainder of this method and associated results, we will use the L_r^* loss function.

2.4 | Hybrid Newton-Raphson algorithm

Since we are interested in the simulation of hyper elastic materials, we need a method that can handle nonlinear systems of equations. The Newton-Raphson algorithm (NRA) is a method often used technique for iteratively solving non-linear problems. One can present the process as follow:
We define the displacement $\mathbf{u}$ as the limit of a sequence $(\mathbf{u}^k)_{k\in\mathbb{N}}$ such that:

$$\mathcal{F}(\mathbf{u}) = 0 \quad \text{with} \quad \mathbf{u}^0 = 0 \tag{15}$$

Then the Newton-Raphson process is given by

$$(\mathbf{u}^k - \mathbf{u}^{k+1}) = \mathcal{J}^{-1}_{\mathcal{F}(\mathbf{u}^k)} \mathcal{F}(\mathbf{u}^k) \tag{16}$$

which is the same as

$$\mathbf{u}^{k+1} = \mathbf{u}^k - \mathcal{J}^{-1}_{\mathcal{F}(\mathbf{u}^k)} \mathcal{F}(\mathbf{u}^k) = \mathbf{u}^k + \boldsymbol{\delta}\mathbf{u}^k \tag{17}$$

Using equation 8 we have :

$$\mathcal{F}(u^k) = \mathbf{R}(\mathbf{u}^k) \tag{18}$$

$$\mathcal{J}_{\mathcal{F}(\mathbf{u}^k)} = \frac{\partial \mathbf{f}^i(\mathbf{u}^k)}{\partial \mathbf{u}} = \mathbb{K}(u^k) \tag{19}$$

In the case of non-linear elasticity problems the jacobian of such function $\mathcal{F}$ is called the tangent stiffness matrix noted, $\mathbb{K}$. Under the right conditions, the method converges and its convergence rate is quadratic. An example of good condition could be that the initial guess is close to the actual solution and there is no stationary point within the $(\mathbf{u}^k)_{k\in\mathbb{N}}$ sequence. In order to set optimal convergence conditions for the Newton-Raphson algorithm, we use predictions from our neural network. For now, let us consider the classical method. It can be rewritten as: $\mathcal{J}_{\mathcal{F}(\mathbf{u}^k)} \cdot (\mathbf{u}^{k+1} - \mathbf{u}^k) = -\mathcal{F}(\mathbf{u}^k)$. Therefore, the Newton-Raphson algorithm reads:

Algorithm 1: Newton-Raphson algorithm

Data: $k = 0$, $\mathbf{u}^0 = 0$, $\boldsymbol{\delta u} = 0$
1 **repeat**
2 Compute $\mathbb{K}(\mathbf{u}^k)$
3 Compute $\mathbf{R}(\mathbf{u}^k)$
4 Solve $\mathbb{K}(\mathbf{u}^k) \cdot \boldsymbol{\delta u} = \mathbf{R}(\mathbf{u}^k)$ for $\boldsymbol{\delta u}$
5 $\mathbf{u}^{k+1} = \mathbf{u}^k + \boldsymbol{\delta u}$
6 $k = k + 1$
7 **until** $\|\mathbf{R}^k\| < \epsilon$ *or* $\|\boldsymbol{\delta u}\| < \eta$

It exists multiple options in order to speedup the Newton-Raphson algorithm. One can choose to improve the computation speed of *line* 3 using parallel computation as an example. Usually the bottle neck of the algorithm is at *line* 4 where most of the time is spent inverting the matrix $\mathbb{K}(\mathbf{u}^k)$ which is composed of the number of dofs squared coefficients. Multiple approaches have been proposed to improve such computation. The Quasi-Newton-Method[48] create a matrix $\mathcal{B}$ which is an approximation of $\mathcal{J}^{-1}_{\mathbf{F}(\mathbf{u}^k)}$, recently, Duff *et al.*[49] proposed a new formulation of the LDL^T solver using an *a posteriori* threshold pivoting.

Finally one can also reduce the number of iterations needed to satisfy the condition on *line* 2 using a good initial guess $\mathbf{u}^0 = \mathbf{u_p}$. In this work we choose this approach, and use the prediction of the network to set an optimal starting point. This reduces the number of iterations of the algorithm most of the time, while also guaranteeing that we obtain a correct solution to our problem even if the prediction is not accurate or even incorrect.

The Hybrid Newton-Raphson-Algorithm reads as follows:

Algorithm 2: Hybrid Newton-Raphson algorithm

Data: $k = 0$, $\mathbf{u}^0 = 0$, $\boldsymbol{\delta u} = 0$
1 Compute $\mathbf{R}(\mathbf{u^0})$
2 $\mathbf{u}_p = Prediction(\, R(\mathbf{u}^0)\,)$
3 Compute $\mathbf{R}(\mathbf{u_p})$
4 **if** $\|\mathbf{R}(\mathbf{u_p})\| < \epsilon$ **then**
5 exit
6 **end**
7 **repeat**
8 Compute $\mathbb{K}(\mathbf{u}^k)$
9 Compute $\mathbf{R}(\mathbf{u}^k)$
10 Solve $\mathbb{K}(\mathbf{u}^k) \cdot \boldsymbol{\delta u} = \mathbf{R}(\mathbf{u}^{k)}$ for $\boldsymbol{\delta u}$
11 $\mathbf{u}^{k+1} = \mathbf{u}^k + \boldsymbol{\delta u}$
12 **if** $k = 0$ *and* $\|\mathbf{R}(\mathbf{u}^1)\| > \|\mathbf{R}(\mathbf{u}_p)\|$ **then**
13 $\mathbf{u}^1 = \mathbf{u_p}$
14 **end**
15 $k = k + 1$
16 **until** $\|\mathbf{R}(\mathbf{u}^k)\| < \epsilon$ *or* $\|\boldsymbol{\delta u}\| < \eta$

The logic of the algorithm remains unchanged. There are multiple scenarios to consider, first the trivial one where $\mathbf{u_p} = \mathbf{u}^0$, algorithm 2 introduces a slight computational overhead at *line* 3 but produces the same answer. The second scenario is the one where $\|\mathbf{R}(\mathbf{u}^1)\| <= \|\mathbf{R}(\mathbf{u}_p)\|$ at *line* 12. In this case the prediction does not provide any gain to the simulation. This could be due to $\mathbf{R}(\mathbf{u^0})$ being too different from the training data. The artificial neural network cannot generalise enough to produce a good answer. This can also happen if the force is too small and produces a displacement field in the order of magnitude of the noise generated by the network. The third and last scenario is when the condition at either *line* 4 or *line* 12 is satisfied. In the best case when it stops at *line* 4 we can compute the displacement field in a couple of milliseconds. In the other case, the prediction usually reduces the number of iterations needed to satisfy the condition on *line* 7, thus speeding up the algorithm when compared to its classical version.

In the next section we will discuss the results of the method and more specifically the Hybrid Newton-Raphson algorithm in subsection 3.2.4.

3 | RESULTS

In this section, we start by presenting the chosen metrics to study our results. In order to assess the different assumptions made in this article, the method will be applied over four different experiments. The first experimentation aims at showing the variety of use cases in which our method works. To do so, we discuss the prediction errors of two use cases that differ by the meshes, the

hyper-elasticity law, the young modulus and the scale. The second experimentation deals with the impact of the residual in the loss. Both ANN will have the same starting set of weights, data set, and training process. The only difference will appear in the formulation of the loss function where one uses a classic MSE while the other uses L_r^*. The third experimentation explores the generalisation abilities of the ANN. We consider a similar scenario as previously shown, but here, simulations and predictions are computed with random Young's modulus. The fourth experimentation investigates the weaknesses and strengths of the Hybrid Newton-Raphson algorithm presented in the previous section. We will consider extreme cases with important and dim forces and also the average use cases.

3.1 | Metrics

We use three different metrics to assert performances of the artificial neural networks. Those values are computed from 100 independent simulations.

3.1.1 | Maximum relative L2

The first metric used is the maximum relative L2 error noted e_{max}. It gives the distance of the 3D node that is the furthest away from the ground truth over the 100 simulations.

$$e_{max} = \max_{s\in[1,S]} \left(\max_{i\in[1,M]} \|\mathbf{u}_i^s - \mathbf{v}_i^s\| \right) \tag{20}$$

Where M is the number of 3D nodes, S the number of independent simulations, $\mathbf{u}$ the output of the network and finally $\mathbf{v}$ the solution computed by the solver. While this metric is a good indicator of how wrong the worst part of the object is deformed, it does not take into account more global phenomenons. Thus, it cannot be the sole metric in our analysis.

3.1.2 | Mean relative L2

The second metric is the relative mean L2 error noted e_{mean}. It gives us the average per dofs relative error.

$$e_{mean} = \frac{1}{N \times S} \sum_{s=0}^{S-1} \|\mathbf{u}^s - \mathbf{v}^s\| \tag{21}$$

Where N is the number of dofs, S the number of independent simulations, $\mathbf{u}$ the output of the network and finally $\mathbf{v}$ the solution computed by the solver. This metric is used in order to compute global information about the deformation error. This, combined with the previously detailed loss allows for a better understanding of the type of inaccuracy that the neural network provides. Yet, for inaccurately predicted small deformations and an accurately predicted big deformation, the previously mentioned error metrics will have low values. In order to properly interpret these low values and their relevance, a last metric is needed.

3.1.3 | Signal-to-noise ratio

The third and last metric used is the signal-to-noise ratio noted SNR. It quantifies the power of noise/inaccuracy in a given signal.

$$\mathrm{SNR}_{min}^{dB} = \min_{s\in[1,S]} 10 \times \log_{10} \left(\frac{\|\mathbf{u}_s\|}{\|\mathbf{u}_s - \mathbf{v}_s\|} \right) \tag{22}$$

Where S is the number of independent simulations, $\mathbf{u}$ the output of the network and finally $\mathbf{v}$ the solution computed by the solver. This metric will allow us to differentiate between the two previously mentioned problematic cases. Given a certain accuracy of the prediction, small displacements will provide small SNR while bigger displacements will provide bigger SNR.

3.2 | Experiments

The following experiments will be realised over 100 randomly generated force samples. Given a mesh and a (hyperelastic) material law, we start a training using the previously defined process. Once the network is trained, each prediction is then compared to its corresponding FEM simulation, which serves as a ground truth. In order to work with comparable values, all the predictions $\mathbf{u}$ and ground-truth $\mathbf{v}$ are scaled using the length of the longest side of the object-aligned bounding box noted L.

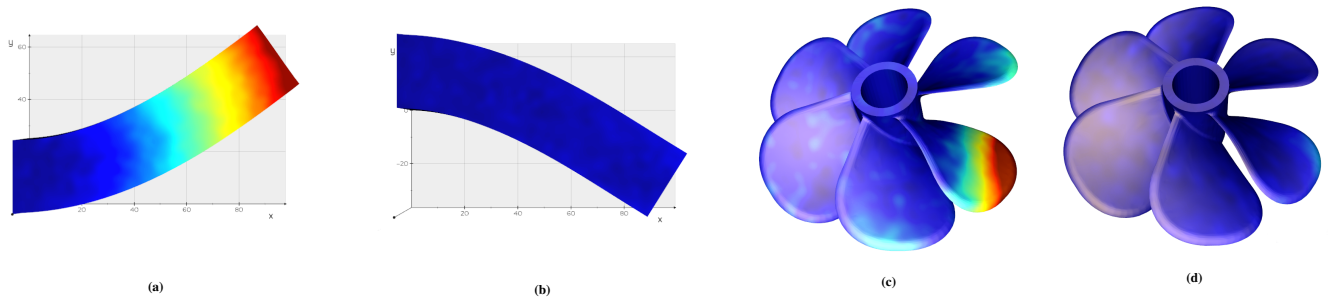
3.2.1 | General results

This section discusses about the ability of the network to approximate deformations on various geometries with different elastic laws. To highlight the ability for our approach to learn on very different shapes and material properties, we selected a propeller mesh with a Neo-Hookean hyper-elasticity law and a cantilever beam with a Saint-Venant-Kirrchhof hyper-elasticity law. Both have about 12,000 degrees of freedom. Their material parameters are, on the other hand, very different with a low stiffness for the beam model, and very high stiffness for the propeller, in order to assess the validity of the training process and the predictions. Dimensions of the two structures were also chosen to be quite different for the same reason. The beam which is extremely soft provides displacements in the order of tens of meters from small external forces. Such extremes scenarios are usually the weak point of artificial neural networks. In addition, given its geometry, it tends to deform globally even from local forces. An accurate prediction requires that the network can transfer the necessary deformation information to all the nodes. On the opposite, for the propeller, high forces are needed to provide relatively small displacements, in the order of the centimetre. Yet, its geometry is such that with local forces it displays blade-wise deformations without any displacement on the neighbouring blades. This is a another important test for the neural network. The different parameters of the 2 test cases are summarised in table 1. Both training lasted about 12 hours (including data set generation) on an NVidia TITAN RTX, with 4095 generated samples each.

Name	#DOFs	Hyper-elastic law	L [m]	E [Pa]	Time [ms]	e_{mean}	e_{max}	SNR_{min}^{dB}
Beam	12,000	Saint-Venant-Kirchhoff	100	4.5×10^{3}	0.4	8.0×10^{-6}	0.03	8.4
Propeller	12,075	Neo-Hookean	1.0	2.03×10^{11}	0.4	2.7×10^{-6}	0.01	18.9

TABLE 1 Results of a comparison between FEM simulations and ANN predictions over 100 randomly distributed forces with random amplitudes.

Both models having complete different sets of simulation parameters provide similar mean and max error over 100 simulations. This, although being demonstrated only on two models in this article, can argue the point that the network and its framework provide accurate global and local deformations of a mesh while handling a wide range of simulation parameters. The displacement field is computed in a steady $0.4ms$ which is up to three orders of magnitude faster than the reference FEM simulation. To compare, the simulation of a deformation of the propeller takes around $500ms$ to compute. One hypothesis to explain the big difference in SNR relies on the amount of near null deformations seen by both models. The beam has few samples where the displacement of the whole body is almost null hence is less trained at generating value close to 0. Where with the propeller most of the samples require a null or almost null displacement field for the vast majority of the points. A proper in-depth analysis on multiple models is required in order to conclude on this hypothesis. Works on this perspective are currently being held.



(a) (b) (c) (d)

FIGURE 5 Examples of large nonlinear elastic deformations predicted by our neural network. The colours represent the node-wise euclidean distance to the solution of the Newton-Raphson algorithm. For both beams the colour gradient goes from 3×10^{-4}m (blue) to 3×10^{-2}m (red), for both propellers the colour gradient goes from 3×10^{-5}m (blue) to 2×10^{-3}m (red).

Name	e_{mean}	e_{max}	SNR^{dB}
Beam (a)	5.4×10^{-5}	0.037	22.2
Beam (b)	6.6×10^{-6}	0.0050	40.2
Propeller (c)	2.82×10^{-6}	0.0036	33.9
Propeller (d)	8.2×10^{-7}	0.0008	42.3

TABLE 2 Error values and SNR of the deformations shown at figure 5. The displayed deformations are highly non-linear, yet the error values and SNR remain in the range of values displayed at table 1.

Beams (a) and (b) in figure 5 undergo roughly the same amount of deformation (with a deflection at the tip of $\approx$ 40m) yet lead to very different error values and patterns. Beam (a) has an maximum error of 37mm near the free end of the beam reducing gradually to 54 µm near the fixed end. Beam (b) on the other hand provides an homogeneous error with a maximum error of 5.0 mm and an average error of 6.6 µm. The same behaviour can be observed with the propeller model. Propeller (c) while having a similar deformation than (d), has and average prediction error of 2.82 µm with a maximum value at 3.6 mm, about four time as much as in the second scenario.

3.2.2 | Physics aware loss function

Loss functions play an important role in the learning process of an artificial neural network. It exists multiple formulations for such functions and one can choose the one that suits better his data set and problem. When dealing with physics, loss functions that only rely on geometric difference might not be sufficient to properly train an artificial neural network. The formulation of the L_r^* at equation 14 provides physical knowledge about the validity of the deformation by introducing the norm of the residual forces generated by the ANN prediction. The experiment will compare the accuracy of two ANN that only differ by the loss function used during the training (same initial weight, data set, normalisation process, etc...). The ANN is trained on a $100 \times 25 \times 25$ meters beam composed of 12,000 DOFs with a Neo-Hookean hyper-elasticity law and a Young's modulus of 4,500 Pa. Both training took about 12 hours of computation (including data set generation)on a NVidia TITAN RTX, with 4095 generated samples each.

Loss function	e_{mean}	e_{max}	SNR_{min}^{dB}
L_r^*	7.7×10^{-6}	0.03	2.7
MSE	5.2×10^{-5}	0.05	-5.1

TABLE 3 Results of a comparison between FEM simulations and the ANN predictions over 100 randomly distributed forces with random amplitudes. This table describes the accuracy of two ANN that only differ by the loss function used during the training.

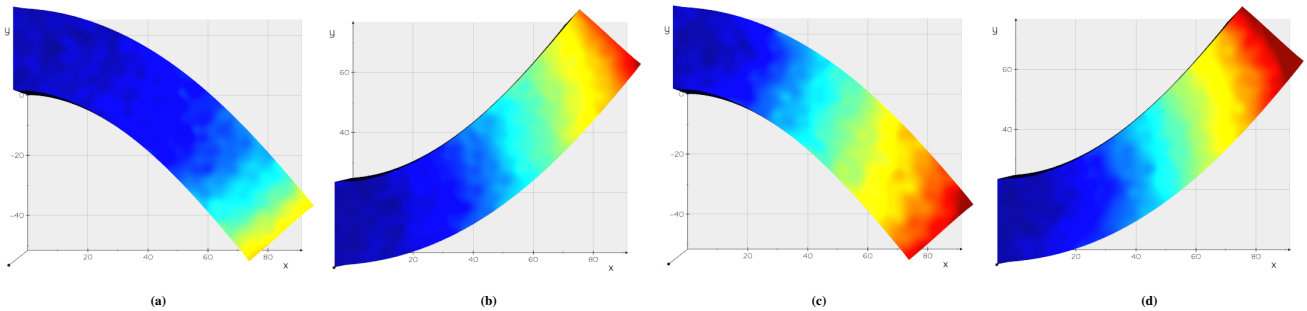


FIGURE 6 Nonlinear elastic deformations of a beam model predicted by our artificial neural network. The colours represent the node-wise euclidean distance to the solution of the Newton-Raphson algorithm. For all four configurations, the colour scale goes from 3×10^{-4}m (blue) to 3×10^{-2}m (red).

While the prediction of the neural network is very good in all cases, we can notice a difference due to the choice of loss function. Both networks have been trained from the same set of starting weights, data set, shuffle seed, normalisation process,

etc., the only difference between the two training procedures being the loss function. The ANN trained using the L_r^* loss function provides results similar to the ones obtained in section 3.2.1 which display a certain stability in the training capacities of the method. The overall performances of the network trained using the proposed method are up by five to ten folds. The average error is the one that provides the best gain with a factor of 10, going down from 5.2×10^{-5} to 7.7×10^{-6}. The maximum error sees an improvement by a factor of five while the noise is reduced by 6. The change in loss function has a strong impact over the learning of an ANN improving both its accuracy and generalisation abilities.

Name	e_{mean}	e_{max}	SNR^{dB}
L_r^* (a)	6.1×10^{-6}	0.027	38.21
L_r^* (b)	1.5×10^{-5}	0.034	34.7
MSE (c)	5.3×10^{-5}	0.034	25.42
MSE (d)	5.5×10^{-5}	0.037	24.9

TABLE 4 Error values and SNR of the deformations shown in figure 6. The displayed deformations are highly non-linear, yet the error values and SNR remain in the range of values displayed at table 3.

The deformations displayed at figure 6, the Beams (a) and (b) are both predicted using the neural network trained with the L_r^* loss function while Beams (c) and (d) are predicted using the network trained with the MSE loss function. All predictions display similar pattern where the error increases toward the tip of the beam. The variations appear in the error metric where predictions (a) and (b) perform better overall than predictions (c) and (d). Forces applied on the beams are very important compared to what has been seen during the training process and one can notice that the prediction (c) and (d) remain around the mean values displayed in the table 3 while predictions (a) and (b) display errors up to two times bigger than average.

3.2.3 | Young's modulus generalisation

The simulation has multiple parameters that can greatly influence the physics of an object. One of which is the Young's modulus. This positive coefficient represents the stiffness of the objects with values such as $1.96 \times 10^{11}\, Pa$ (iron) or $6.0 \times 10^{6}\, Pa$ (collagen). Given a force and a model, an increase in the Young's modulus produces smaller displacements and vice versa. Methods such as Model Order Reduction can solve problems with "slights" variations in simulation parameters [18] [19]. The goal of this section is to show that the network can work with a wide variety of Young's modulus thus bypassing MOR limitations regarding this parameter. The ANN is trained on a $1 \times 0.25 \times 0.25$ meters beam composed of 12000 dofs with a Neo-Hookean hyper-elasticity law. The training lasted for 47h (including data set generation) on a NVidia TITAN RTX, with 10650 generated samples each. During the training, each computed sample is generated with a random Young's modulus between 1.0×10^9 and 2.03×10^{11}.

Young's Modulus	e_{mean}	e_{max}	SNR_{min}^{dB}
$[1.0 \times 10^9, 2.03 \times 10^{11}]$	4.6×10^{-5}	0.09	-2.6

TABLE 5 Results of a comparison between FEM simulations and ANN predictions over 100 randomly distributed forces with random amplitudes and random Young's modulus within the given range.

The input tensor is then modified by adding a coefficient corresponding to the normalised value of the Young's modulus. The Young's modulus generalisation displays some losses compared to the presented results in section 3.2.1 but remain comparable to the MSE trained network presented in section 3.2.2. With an average error of 4.6×10^{-5} it performs on average better than the MSE, but displays bigger max error with a 0.09 against 0.05 for the later.

When the range of Young's modulus becomes too important, the network cannot learn to generate deformations and tend to only create noise. With a range of $[1.0 \times 10^6, 2.03 \times 10^{11}]$ the network barely reaches the 1.0×10^{-4} at the end of the training thus showing that it requires much more samples to converge to more accurate predictions.

Beam (b) and Beam (c) are respectively near the lower and upper bound of the Young's modulus range and provide similar errors, both being comparable to the other two beams in term of errors too.

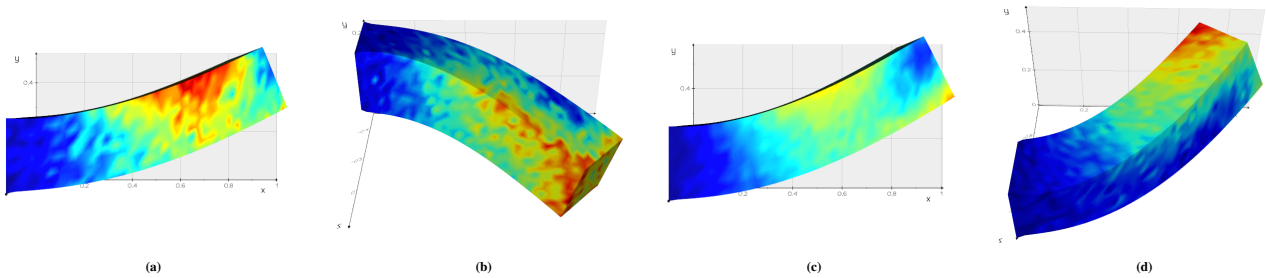


FIGURE 7 Nonlinear elastic deformations of a beam model predicted by our artificial neural network. The colours represent the node-wise euclidean distance to the solution of the Newton-Raphson algorithm. For all four configurations, the colour scale goes from 3×10^{-4}m (blue) to 3×10^{-2}m (red).

Name	Young's modulus	e_{mean}	e_{max}	SNR^{dB}
Beam (a)	9.6×10^{10}	3.9×10^{-5}	0.028	25.66
Beam (b)	7.6×10^{9}	4.8×10^{-5}	0.038	25.42
Beam (c)	1.0×10^{11}	4.1×10^{-5}	0.034	24.25
Beam (d)	6.4×10^{10}	6.8×10^{-5}	0.060	23.73

TABLE 6 Error values and SNR of the deformations shown at figure 7. The displayed deformations are highly non-linear, yet the error values and SNR remain in the range of values displayed at table 5.

The error doesn't seem to have any specific pattern over few samples. In the long run, the pattern seen in the previous sections appears. All beams have similar errors in every metric with an average error around 5×10^{-5} with a SNR around of 24.5. The only discrepancy appears with Beam (d) where the maximum error reaches 0.060 and doubles the other values. The chosen Young's modulus doesn't seem to affect the accuracy of the network.

3.2.4 | ANN accelerated Newton-Raphson

Artificial neural network has proven to be precise up to a couple of micro meters on average. Where one could be satisfied with the precision, another may want to ensure the respect of some properties such as incompressibility or fixed points on the boundaries. As shown in section 2, the Hybrid Newton-Raphson algorithm proposes to use the prediction of the network in order to speed-up the algorithm. The experiment will compare the speed and solution of the ANN, the classic Newton-Raphson and its hybrid version. The network trained for the beam at section 3.2.1 is used to obtain the following values.

Solver type	Converged simulations	Network prediction picked	Average iterations count
Classic Newton-Raphson	46%	-	6.1
Hybrid Newton-Raphson	71%	66%	5.0

TABLE 7 Results of a comparison between the classic Newton-Raphson algorithm and the presented Hybrid Newton-Raphson algorithm over 100 randomly distributed forces with random amplitudes.

These results are computed from a data set of 100 random external forces. Among them, 50 are within the amplitude range of the training data set and the 50 other have an amplitude up to two times bigger. Although 50 of them have an amplitude that is within the training bounds, they do not share orientation nor location with any of the training data set.

The classic Newton-Raphson manages to converge 46% of the time. On average, when it converges, it does so in 6.1 iterations.

The Hybrid Newton-Raphson converges 71% of the time and 100% of the time when the classic version did too. Our algorithm leads to a gain of 54% in term of convergence. In 66% of the cases, the solution of the neural network is preferred to the first Newton-Raphson iteration. Over the 100 test samples, the Hybrid Newton-Raphson picked two out of three times the prediction of the network as a better starting point than the result of the first iteration of the Newton-Raphson algorithm. From this point, on average, the algorithm converges in 5.0 iterations. This shows that from the prediction the algorithm converges on average in

4.0 iterations adding up to 5.0 to account for the first one that is discarded when the prediction is picked. Overall the proposed algorithm converges more often and faster than the classic method while keeping the ordinary convergence properties of the Newton-Raphson algorithm.

4 | DISCUSSION AND CONCLUSION

We presented a novel machine learning framework that can learn and compute very quickly and accurately complex nonlinear static deformations of elastic structures. This framework is based on modal analysis, a physics aware loss function and an optional Hybrid Newton-Raphson algorithm. Our results show that we can train a neural network with a small data set to achieve micrometer-scale errors in less than a millisecond, thus easily reaching real-time requirements. The method is also robust to Young's modulus generalisation, but requires more samples to achieve the same precision due to the increase in problem complexity. The Hybrid Newton-Raphson algorithm displays an important gain with a 54% increase in convergence rate and 20% gain in average iterations count.

The method shows promising results, but can be further improved. Firstly, some tests have shown that external forces applied very locally are not very well handled by the network and would need a more important representation in the data set. Secondly, the network has no knowledge of the topology, the geometry nor Dirichlet boundary conditions. Thus, any changes in these characteristics will not be taken into account in the predictions. This limits the generalisation abilities of the network, although it remains similar to Model Order Reduction methods. Finally, for small displacements, the noise introduced by the network can be of the same amplitude as the range of motion. However, for small linear deformations, there is no need for any of the methods described in this paper, as more efficient and simpler solutions exist.

In our approach, we rely on a fully connected network for its simplicity and efficiency. However, a change in the architecture of the artificial neural network could help with the generalisation and also reduce the number of parameters of the network. Architectures such as convolutional neural networks are known to generalise well from sparse data sets and we will consider them as a way to further develop our approach. In addition, recent developments in graph network can also alleviate the constraints on the topology typically imposed by CNNs, allowing to use any type of mesh rather than a regular grid structure.

5 | ACKNOWLEDGEMENTS

This work was supported by the DRIVEN project grant agreement No 811099.

References

1. Raissi M, Perdikaris P, Karniadakis GE. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics* 2019; 378: 686–707.

2. Sanchez-Gonzalez A, Godwin J, Pfaff T, Ying R, Leskovec J, Battaglia P. Learning to simulate complex physics with graph networks. In: PMLR. ; 2020: 8459–8468.

3. Kim B, C. Azevedo V, Thuerey N, Kim T, Gross M, Solenthaler B. Deep Fluids: A Generative Network for Parameterized Fluid Simulations. *Computer Graphics Forum (Proc. Eurographics)* 2019; 38(2).

4. Zienkiewicz OC, Taylor RL, Nithiarasu P, Zhu J. *The finite element method*. 3. McGraw-hill London . 1977.

5. Brenner S, Scott R. *The mathematical theory of finite element methods*. 15. Springer Science & Business Media . 2007.

6. Mandel J. Balancing domain decomposition. *Communications in numerical methods in engineering* 1993; 9(3): 233–241.

7. Farhat C, Lesoinne M, LeTallec P, Pierson K, Rixen D. FETI-DP: a dual–primal unified FETI method—part I: A faster alternative to the two-level FETI method. *International journal for numerical methods in engineering* 2001; 50(7): 1523–1544.

8. Schwarz HA. *Au-dessus d'une frontière "u transition par procédure alternée*. Z "u rcher et Furrer . 1870.

9. Lions PL. On the Schwarz alternating method. III: a variant for nonoverlapping subdomains. In: . 6. SIAM Philadelphia, PA. ; 1990: 202–223.

10. Haferssas R, Tournier PH, Nataf F, Cotin S. Simulation of soft tissue deformation in real-time using domain decomposition. working paper or preprint; 2019.

11. Haferssas R, Tournier PH, Nataf F, Cotin S. Simulation of soft tissue deformation in real-time using domain decomposition. In: INRIA. ; 2019.

12. Chatterjee A. An introduction to the proper orthogonal decomposition. *Current science* 2000: 808–817.

13. Berkooz G, Holmes P, Lumley JL. The proper orthogonal decomposition in the analysis of turbulent flows. *Annual review of fluid mechanics* 1993; 25(1): 539–575.

14. Mendez M, Balabane M, Buchlin JM. Multi-scale proper orthogonal decomposition of complex fluid flows. *Journal of Fluid Mechanics* 2019; 870: 988–1036.

15. Goury O, Carrez B, Duriez C. Real-Time Simulation For Control Of Soft Robots With Self-Collisions Using Model Order Reduction For Contact Forces. *IEEE Robotics and Automation Letters* 2021; 6(2): 3752–3759.

16. MacManus DG, Chiereghin N, Prieto DG, Zachos P. Complex aeroengine intake ducts and dynamic distortion. *AIAA Journal* 2017; 55(7): 2395–2409.

17. Oliveira I, Patera A. Reduced-basis techniques for rapid reliable optimization of systems described by affinely parametrized coercive elliptic partial differential equations. *Optimization and Engineering* 2007; 8(1): 43–65.

18. Park H, Cho D. The use of the Karhunen-Loeve decomposition for the modeling of distributed parameter systems. *Chemical Engineering Science* 1996; 51(1): 81–98.

19. Maday Y, Ronquist EM. The reduced basis element method: application to a thermal fin problem. *SIAM Journal on Scientific Computing* 2004; 26(1): 240–258.

20. Pfeiffer M, Riediger C, Weitz J, Speidel S. Learning soft tissue behavior of organs for surgical navigation with convolutional neural networks. *International journal of computer assisted radiology and surgery* 2019; 14(7): 1147–1155.

21. Bell N, Garland M. Implementing Sparse Matrix-Vector Multiplication on Throughput-Oriented Processors. In: SC '09. NVidia. Association for Computing Machinery; 2009; New York, NY, USA

22. Mueller E, Guo X, Scheichl R, Shi S. Matrix-free GPU implementation of a preconditioned conjugate gradient solver for anisotropic elliptic PDEs. https://arxiv.org/pdf/1302.7193.pdf; 2013.

23. Martínez-Frutos J, Martínez-Castejón PJ, Herrero-Pérez D. Fine-grained GPU implementation of assembly-free iterative solver for finite element problems. *Computers & Structures* 2015; 157: 9-18. doi: https://doi.org/10.1016/j.compstruc.2015.05.010

24. Allard J, Courtecuisse H, Faure F. Implicit FEM Solver on GPU for Interactive Deformation Simulation. In: W. Hwu mW. , ed. *GPU Computing Gems Jade Edition*Applications of GPU Computing Series. Elsevier. 2011 (pp. 281-294)

25. Dziekonski A, Sypek P, Lamecki A, Mrozowski M. Finite Element Matrix Generation on a Gpu. *Progress In Electromagnetics Research* 2012; 128: 249-265. doi: 10.2528/PIER12040301

26. Mueller-Roemer JS, Stork A. GPU-based polynomial finite element matrix assembly for simplex meshes. In: . 37. Wiley Online Library. ; 2018: 443–454.

27. Guennebaud G, Jacob B, others . Eigen v3. http://eigen.tuxfamily.org; 2010.

28. Baydin AG, Pearlmutter BA, Radul AA, Siskind JM. Automatic differentiation in machine learning: a survey. *Journal of machine learning research* 2018; 18.

29. Sahli Costabal F, Yang Y, Perdikaris P, Hurtado DE, Kuhl E. Physics-Informed Neural Networks for Cardiac Activation Mapping. *Frontiers in Physics* 2020; 8: 42. doi: 10.3389/fphy.2020.00042

30. Goswami S, Anitescu C, Chakraborty S, Rabczuk T. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics* 2020; 106: 102447.

31. Cheng C, Zhang GT. Deep Learning Method Based on Physics Informed Neural Network with Resnet Block for Solving Fluid Flow Problems. *Water* 2021; 13(4). doi: 10.3390/w13040423

32. Zhang QJ, Gupta KC, Devabhaktuni VK. Artificial neural networks for RF and microwave design-from theory to practice. *IEEE transactions on microwave theory and techniques* 2003; 51(4): 1339–1350.

33. Kochkov D, Smith JA, Alieva A, Wang Q, Brenner MP, Hoyer S. Machine learning accelerated computational fluid dynamics. *arXiv preprint arXiv:2102.01010* 2021.

34. Mendizabal A, Márquez-Neila P, Cotin S. Simulation of hyperelastic materials in real-time using deep learning. *Medical image analysis* 2020; 59: 101569.

35. Romaguera LV, Plantefève R, Romero FP, Hébert F, Carrier JF, Kadoury S. Prediction of in-plane organ deformation during free-breathing radiotherapy via discriminative spatial transformer networks. *Medical image analysis* 2020; 64: 101754.

36. Chentanez N, Macklin M, Müller M, Jeschke S, Kim TY. Cloth and skin deformation with a triangle mesh based convolutional neural network. *Computer Graphics Forum* 2020; 39(8): 123–134.

37. Holynski A, Curless BL, Seitz SM, Szeliski R. Animating Pictures With Eulerian Motion Fields. In: University of Washington. ; 2021: 5810-5819.

38. Wang H, O'Brien JF, Ramamoorthi R. Data-driven elastic models for cloth: modeling and measurement. *ACM transactions on graphics (TOG)* 2011; 30(4): 1–12.

39. Xu W, Umetani N, Chao Q, Mao J, Jin X, Tong X. Sensitivity-optimized rigging for example-based real-time clothing synthesis.. *ACM Trans. Graph.* 2014; 33(4): 107–1.

40. Kim D, Koh W, Narain R, Fatahalian K, Treuille A, O'Brien JF. Near-exhaustive precomputation of secondary cloth effects. *ACM Transactions on Graphics (TOG)* 2013; 32(4): 1–8.

41. Santesteban I, Otaduy MA, Casas D. Learning-based animation of clothing for virtual try-on. In: . 38. Wiley Online Library. ; 2019: 355–366.

42. Brunet JN, Mendizabal A, Petit A, Golse N, Vibert E, Cotin S. Physics-based deep neural network for augmented reality during liver surgery. In: Springer. ; 2019: 137–145.

43. Meister F, Passerini T, Mihalef V, Tuysuzoglu A, Maier A, Mansi T. Towards fast biomechanical modeling of soft tissue using neural networks. *arXiv preprint arXiv:1812.06186* 2018.

44. He K, Zhang X, Ren S, Sun J. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In: Microsoft Research. ; 2015: 1026–1034.

45. Wriggers P. *Nonlinear finite element methods*. Springer Science & Business Media . 2008.

46. Maclaurin D, Duvenaud D, Johnson M, Adams RP. Autograd: Reverse-mode differentiation of native Python. http://github.com/HIPS/autograd; .

47. Kingma DP, Ba J. Adam: A Method for Stochastic Optimization. http://arxiv.org/abs/1412.6980; 2014. cite arxiv:1412.6980Comment: Published as a conference paper at the 3rd International Conference for Learning Representations, San Diego, 2015.

48. Broyden CG. Quasi-Newton methods and their application to function minimisation. *Mathematics of Computation* 1967; 21(99): 368–381.

49. Duff I, Hogg J, Lopez F. A New Sparse LDLˆT Solver Using A Posteriori Threshold Pivoting. *SIAM Journal on Scientific Computing* 2020; 42(2): C23–C42.