Research Repository UCD

Title	Finite volume-based supervised machine learning models for linear elastostatics
Authors(s)	Tandis, Emad, Cardiff, Philip
Publication date	2023-02
Publication information	Tandis, Emad, and Philip Cardiff. “Finite Volume-Based Supervised Machine Learning Models for Linear Elastostatics.” Elsevier, February 2023. https://doi.org/10.1016/j.advengsoft.2022.103390.
Publisher	Elsevier
Item record/more information	http://hdl.handle.net/10197/26172
Publisher's statement	This is the author’s version of a work that was accepted for publication in Advances in Engineering Software. Changes resulting from the publishing process, such as peer review, editing, corrections, structural formatting, and other quality control mechanisms may not be reflected in this document. Changes may have been made to this work since it was submitted for publication. A definitive version was subsequently published in Advances in Engineering Software (VOL 176, (February 2023)) DOI: https://doi.org/10.1016/j.advengsoft.2022.103390
Publisher's version (DOI)	10.1016/j.advengsoft.2022.103390

Downloaded 2026-05-01 23:49:53

The UCD community has made this article openly available. Please share how this access benefits you. Your story matters! (@ucd_oa)

© Some rights reserved. For more information

Finite Volume-Based Supervised Machine Learning Models for Linear Elastostatics

Emad Tandis and Philip Cardiff

School of Mechanical and Materials Engineering, University College Dublin, Ireland

Corresponding author: Philip Cardiff (philip.cardiff@ucd.ie)

Abstract

This article proposes two approaches for combining finite volume and machine learning techniques to solve linear elastostatic problems. The first approach adopts a classical supervised machine learning model and generates the training dataset by classical finite volume-based solvers. The second approach applies a physics-informed model to enforce the governing equations without requiring a priori ground-truth data; as a result, all training cases are solved within the training process. Although the methods presented apply to a wide range of computational problems, this study is limited to linear elastostatics to demonstrate the concept. To develop a physics-informed approach consistent with a finite volume discretisation, we create symbolic Gauss-based gradient and divergence operators as a function of the displacement field. This allows for a finite volume-based residual of the momentum equation to be used as the loss of the network within the training process. For both approaches, the trained models can be used as surrogates or initialisers for classical solvers. The results for three problems are presented: a plate with a hole, a curved plate, and a cantilever beam. It is demonstrated that both approaches can be used as a surrogate or initialiser with an acceptable level of accuracy; however, the classical supervised approach requires much less computational effort than the physics-informed approach. In particular, employing the classical supervised model as an initialiser for the solution of 500 configurations from the cantilever beam case can reduce the overall computational time by up to 461%.

Keywords: Machine learning, Finite volume method, Linear elastostatics, Physics-informed neural network, Solution acceleration, Code emulators.

1- Introduction

Finite volume/element (FV/FE) methods are numerical techniques commonly used to solve boundary value problems approximately. Despite their tremendous success, these solution procedures can be computationally expensive for many problems. In recent years, hybrid FE/FV approaches have been proposed exploiting machine learning methods. These hybrid approaches have several potential advantages in terms of accuracy and solution time. However, they also suffer disadvantages such as sacrificing accuracy for computational speed or vice versa. This paper proposes two novel approaches for combining finite volume methods with machine learning techniques to accelerate the solution of boundary value problems.

The issue of computational expense for FV/FE methods is apparent when the solution process needs to be repeated many times, e.g., optimisation problems, uncertainty quantification, and parametric studies. On the other hand, machine learning/deep learning (ML/DL) approaches have impressive capabilities for learning complex underlying rules in diverse areas, including computer vision, speech recognition, and natural language processing. These approaches offer an opportunity to enhance the solution procedure with techniques orders of magnitude faster in computational problems.

Reviewing the literature reveals two approaches to combining machine learning methods and classical simulations in computational mechanics: the ML-enhanced approach, where machine learning is used to boost some functionality of the solution approach, and the standalone approach, where machine learning is employed to infer the solution. The ML-enhanced method replaces a part of the classical solution procedure with a pre-trained neural network to improve the solver's capabilities or mitigate specific limitations. As a subset of this approach, neural networks have been employed as a surrogate to efficiently handle computationally expensive tasks. Among these approaches are the studies by Yang et al. [1] and Tompson et al. [2], where a neural network was used to solve the pressure equation in a fluid dynamics algorithm. In both cases, the machine learning-enhanced approaches offered order-of-magnitude speedups while maintaining similar accuracy to approaches commonly used for real-time simulation. In addition to reducing computational expense, improving the accuracy of classical solution methods via machine learning techniques has received attention from researchers in computational mechanics. For example, in Oishi and Yagawa [3], machine learning methods extract optimised quadrature rules in the finite element method to improve accuracy without adding computational time.

The standalone ML approach can be further divided into general and problem-specific architectures. In the former, the neural network has an architecture similar to those used in other applications, e.g., image processing. For the latter, an architecture is developed based on specific application area knowledge. From the category of standalone ML approach with general architecture, Guo and Lorio [4] used a supervised machine learning model with a general convolutional architecture and trained it over ground-truth data generated by classical computational solvers. They then employed this trained model as a surrogate for predicting the velocity field in 2D and 3D non-uniform steady laminar flows. Thuerey et al. [5] adopted the same approach to predict velocity and pressure in turbulent flow around an airfoil as a function of free-stream velocity and airfoil shape. In addition to being used as a surrogate, such machine-learning models have been used to generate initial solution fields to accelerate iterative solvers. For instance, in CFDNet [6], a predictive machine learning model, trained over various geometries of a specific CFD problem, is employed to initialise the pressure and velocity fields in unseen problems and, consequently, reduce the required iterations for the solver convergence. As a drawback, training the models in this approach necessitates ground-truth data consisting of the corresponding solution fields for each sample in the training dataset. This can require significant computational efforts; however, as shown later in this article, the overhead for generating such ground-truth data can be acceptable in some cases when compared to the benefits.

Among standalone ML approaches with problem-specific architecture, PDE-Net [7] and FEA-Net [8] are physics-driven machine-learning approaches aimed at discovering governing PDEs directly from ground-truth data. The two approaches are based on an analogy between convolutional kernels (filters) and discrete mathematical operators, e.g. gradient, over a Cartesian mesh domain. This analogy has led to a purposeful design of the network architecture. In this context, purposeful means that each unknown weight in the kernel can be interpreted as physical parameters, e.g. material constant or geometrical parameter. Therefore, training these networks leads to reconstructing the underlying PDEs and, consequently, the physical phenomena behind the data. These types of networks are not only able to predict the result for unseen cases but are also capable of revealing associated physical phenomena hidden in the data. However, these approaches adopt a supervised learning approach and hence must access ground-truth data gathered from experimental measurements or numerical solvers.

Another recently proposed standalone machine learning approach is the Physics-Informed Neural Network (PINN) [9-13], whose versatility has been demonstrated for various applications [14-18]. PINNs learn the solution of a specific case via training over several spatio-temporal points randomly generated within the computational domain. This way, the training procedure is an alternative to the conventional numerical solution. As a result, the final trained model is a function of space and time. The innovation in PINN defines the training loss as residual of the PDE, where automatic differentiation [19] is combined with a symbolic

relationship between the network inputs and outputs for evaluating derivatives in the equations. A strong point of such an approach is that it avoids discretisation errors within the solution process. Another benefit is that the final function is a symbolic solution to the problem, which is fully differentiable with respect to all input coordinates. However, as mentioned by Raissi et al. [13], the standard version of this approach is not comparable to the conventional solvers in terms of solution speed and robustness.

Several variants have been proposed to enhance the capabilities and mitigate the disadvantages of the standard PINN. Among them are cPINN [20], XPINN [21], and bc-PINN [22]. cPINN and XPINN apply a separate PINN for each sub-domains of the entire computational domain to enhance capacities such as parallelisation, representation, and trainability. cPINN ensures conservative fluxes across the sub-domains interfaces. XPINN applies C^0 continuity at the sub-domains interfaces and aims to enhance the representation and parallelisation capacity of cPINN by allowing multiple neural networks in the smaller sub-domains. In another study, Mattey and Ghosh [22] demonstrated that the standard PINN suffers from accuracy issues for solving PDEs with strong nonlinearity or higher-order derivative operators.

Despite tremendous success in various applications, building a neural network to recognise the underlying relationships between a boundary value problem definition and the solution field is a challenging task. Typically, one has to compromise between training computational cost and accuracy of predictions, i.e. if an accurate prediction is given priority, it requires a complex network and large training dataset, which imposes additional training and data generation time. One reason is rooted in the fact that accurately identifying a nonlinear map from a few and potentially high-dimensional input and output data pairs introduces a significant challenge in training a machine learning model. While the PINN approach alleviates this issue, it is only applicable to a single physical problem with a specific configuration and is not as efficient as classical solvers. On the other hand, for the supervised models to satisfy the desired accuracy, the ground-truth data require many prior solutions, which might impose prohibitive expenses in some cases.

Additionally, working with predictive deep learning models in practice often comes with the question of how reliable the predictions of that model are, i.e. there is still no guarantee that predictions for unseen samples meet the required accuracy. In the context of elastostatics, slight deviations from actual values for the deformation field can lead to significant errors for dependent quantities such as stress and strain. One option to overcome this issue is to employ the predictive model not as a surrogate but as an initialiser for classical iterative solvers. Starting with an accurate initial solution, the classical iterative solvers are expected to converge rapidly to the final solution with the required accuracy. These arguments have motivated the current study to investigate two approaches for combining computational mechanics techniques with machine learning models to provide faster solutions for a desired level of accuracy.

This article proposes two related approaches for combining iterative finite volume solvers with supervised machine learning techniques to accelerate solving problems in linear elastostatics. The first approach uses a supervised machine learning model trained on solution results from a finite volume solver. The second approach employs a physics-informed neural network, where the loss function is defined in terms of finite volume-based discrete governing equations instead of prior solutions. Depending on the required accuracy, both approaches can be used as either initialisers for iterative solvers or code surrogates (emulators). Compared to the PINN approach, the second approach applies a similar method for defining the loss function but differs in the evaluation of derivative operators and its application. It uses prior knowledge of the finite volume method to reconstruct gradient and divergence operators as a symbolic function of the predictions. In terms of application, the distinguishing feature of the second proposed approach is its capability to simultaneously solve and learn enforcement of the PDEs for multiple configurations associated with the same physics; this is also different from the PINN approach, which is typically applied for solving a single configuration, where the trained model is not applicable for the solution of a new configuration. The initial focus is on linear elastostatics, but the approaches are expected to generalise to other systems governed by partial differential equations.

The remainder of this article is structured as follows: Section 2 introduces the problem of linear elastostatics; Section 3 details the proposed supervised and physics-informed machine learning models; Section 4 describes the symbolic definition of the loss within the physics-informed approach; Section 5 details the implementation of the algorithms; Section 6 presents the performance of both approaches on three test cases; finally, Section 7 ends the article with concluding remarks.

2- Problem definition

For demonstration purposes, this study is limited to the analysis of 2D solid bodies (Figure 1) undergoing steady-state small linear elastic deformations, where the boundary, $\boldsymbol{\Gamma} = \boldsymbol{\Gamma_U} \cup \boldsymbol{\Gamma_T}$, of the body $\boldsymbol{\Omega}$ is composed of two regions: $\boldsymbol{\Gamma_U}$ where displacement is prescribed, and $\boldsymbol{\Gamma_T}$ where traction is prescribed.

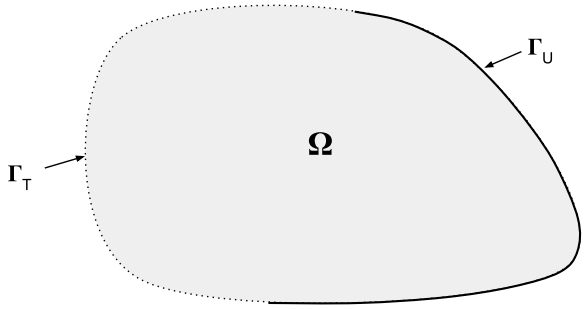

Figure 1- Schematic of a 2D elastostatic problem with domain $\boldsymbol{\Omega}$ and boundary $\boldsymbol{\Gamma} = \boldsymbol{\Gamma_U} \cup \boldsymbol{\Gamma_T}$

Ignoring body forces, the displacement, $\boldsymbol{U}$, of such bodies is governed by the conservation of linear momentum and related boundary conditions:

$$\boldsymbol{\nabla} \cdot \boldsymbol{\sigma} = 0 \qquad \forall (x, y) \in \boldsymbol{\Omega} \tag{1a}$$

$$\boldsymbol{U} = \boldsymbol{U_\Gamma}(x, y) \qquad \forall (x, y) \in \boldsymbol{\Gamma_U} \tag{1b}$$

$$\boldsymbol{n} \cdot \boldsymbol{\sigma} = \boldsymbol{T_\Gamma}(x, y) \qquad \forall (x, y) \in \boldsymbol{\Gamma_T} \tag{1c}$$

Where $\boldsymbol{U_\Gamma}$ and $\boldsymbol{T_\Gamma}$ are prescribed displacement and traction vectors at $\boldsymbol{\Gamma_U}$ and $\boldsymbol{\Gamma_T}$ boundaries, respectively. Assuming Hooke's law, the stress tensor, $\boldsymbol{\sigma}$, is given in terms of the deformation gradient, $\boldsymbol{\nabla U}$, by

$$\boldsymbol{\sigma} = \mu \left(\boldsymbol{\nabla U} + \boldsymbol{\nabla U^T} \right) + \lambda \, \mathrm{tr}(\boldsymbol{\nabla U}) \mathbf{I} \tag{2}$$

where μ and λ are the Lamé parameters. Eq. (1a) represents the momentum equations in differential form; its integral form is expressed as $\oint_S \boldsymbol{n} \cdot \boldsymbol{\sigma} \, dS = 0$ or $\int_V \boldsymbol{\nabla} \cdot \boldsymbol{\sigma} \, dV = 0$, where S and V are the surrounding surface and volume of any arbitrary space within the body $\boldsymbol{\Omega}$ and $\boldsymbol{n}$ is the normal outward vector on the surface S. The conventional FV/FE approach to the solution of this problem is to discretise the domain $\boldsymbol{\Omega}$ into smaller volumes, called cells or elements, and then apply the strong or weak integral forms of the governing equations for individual cells or collections of cells/elements.

3- Machine learning methods

The two machine learning approaches proposed in this study each consist of two parts (Figure 2): a neural network for predicting the displacement field, and a loss evaluator, where the physics-informed and the classical supervised approaches differ.

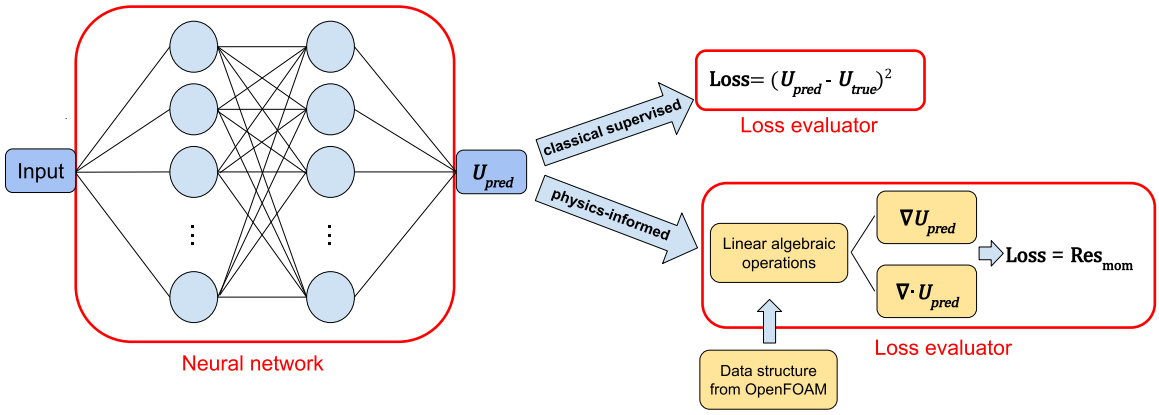

Figure 2- Neural network and loss evaluator for classical supervised and physics-informed machine learning models. The orange/yellow colour boxes indicate the role of the finite volume method in the physics-informed model.

Both machine learning models use a fully connected network for the model architecture. For evaluating the loss, the classical supervised model applies the mean square difference between the predictions and true displacement field, whereas the physics-informed model defines the loss function as the squared residual of the discrete momentum equation. The architecture of the network and loss evaluator for these two approaches are discussed in the subsequent sections.

The network input can be any parameter that uniquely describes each configuration from a base physical problem. In this study, the focus is to learn the effect of geometry on the solution field. So, a variable corresponding to the geometry is defined as the input. The network input is discussed in Section 4-1.

3-1- Neural network architecture

A variety of architectures for the neural network can be used for the classical supervised and physic-informed approaches. However, due to some potential benefits and for the sake of comparison, this study applies the same fully connected architecture for both approaches. The main advantage of fully connected networks is that they are structure agnostic, i.e. no particular assumptions need to be made about the input, and hence they are more general. As a result, this architecture gives flexibility in defining the network input [23]. Additionally, the suitability of such an architecture in recognising the underlying relations between inputs and outputs within computational mechanics problems has been demonstrated in the literature [1, 3, 24]. After the input, a batch normalisation layer is employed to reduce the number of training iterations by standardising the input for each mini-batch [25]. Moreover, this layer eliminates the requirement for prior normalisations of the input for the prediction task, which facilitates data organisation, implementation, and subsequently employing the predictive models in a computational mechanics solver.

3-2- Loss function

Training a neural network is an optimisation problem that aims to reduce the samples' loss by modifying the network weights. Therefore, the loss function is an essential component of neural networks, affecting their learning dynamics. Several studies addressed the effects of various loss functions on the training of neural networks [26, 27]. The current study uses different definitions of loss for the classical supervised and physics-informed approaches. For the classical supervised approach, the loss is defined for each sample as the squared difference between the network output, $\boldsymbol{U}_{pred}$, and the true solution field, $\boldsymbol{U}_{true}$.

$$\textbf{Loss} = \left\| \boldsymbol{U}_{pred} - \boldsymbol{U}_{true} \right\|^2 \tag{3}$$

The network output and the loss are defined here as two-dimensional tensors, i.e. tensors with the shape (n, m), where n refers to the number of cells and m represents the number of spatial components. Here, m = 2 as the current study is limited to 2D problems. As Eq. (3) suggests, the loss for the classical supervised model requires the true displacement field. This study employs iterative finite volume solvers within the solids4Foam [28, 29] toolbox for OpenFOAM [28, 29] to generate the true displacements.

The definition of the loss in the proposed physics-informed approach employs the discrete momentum equation and eliminates the need for the true displacement field. As the reconstruction of the momentum equation requires spatial derivatives, this study defines finite volume-based gradient and divergence operations as symbolic functions of the displacement field. Consequently, a finite volume-based residual of the momentum equation to be used as a symbolic function within the network output. Furthermore, a symbolic definition of the loss function conveniently allows the use of automatic differentiation during the back-propagation step, which is a core component of state-of-the-art optimisations within the training procedures. Figure 2 illustrates the role of the finite volume method in the loss of the physics-informed model. Having the displacement fields predicted by the first part of the physics-informed model, the second part evaluates the residual of the discrete momentum equation as the training loss. Section 4 discussed the reconstruction of the finite-volume-based discrete momentum equation in more detail.

It should be noted that in machine learning terminology and also in this paper, the cost function indicates the prediction error for the entire training dataset. The cost function is a scalar defined as the average loss for all cells and samples within the training dataset. In either approach, the training process seeks the weights in the neural network to minimise the cost function.

3-3- Neural network training

The adaptive learning rate optimisation algorithm (Adam) [30] is employed to train both models. Adaptive learning rate means that it computes individual learning rates for each network weight w. Like gradient descent algorithms [31], Adam uses the gradient of the cost with respect to the network weights to modify the weights within each training iteration (epoch). However, since the gradient of the cost function is usually a stochastic/random variable due to the potentially small batch sizes, Adam uses the estimates of the first and second moments of the gradient, m_t and v_t, respectively, computed as [32]

$$\begin{aligned} m_t &= \beta_1 m_{t-1} + (1-\beta_1) g_t \\ v_t &= \beta_2 v_{t-1} + (1-\beta_2) {g_t}^2 \end{aligned} \tag{4}$$

Here, g_t is the cost gradient with respect to the network weights at epoch t; β_1 and β_2 are hyper-parameters controlling the contribution of the gradient at previous epochs. Prior to modification of the network weights, Adam uses the following relations to compute the biased corrected first and second moments, $\hat{m}_t$ and $\hat{v}_t$:

$$\hat{m}_t = \frac{m_t}{1-{\beta_1}^t}$$
$$\hat{v}_t = \frac{v_t}{1-{\beta_2}^t} \quad (5)$$

The weight modification at epoch t is calculated as

$$\mathrm{w}_t = \mathrm{w}_{t-1} - \eta \frac{\hat{m}_t}{\sqrt{{\hat{v}_t}^t} + \varepsilon} \quad (6)$$

where η is a hyper-parameter denoting the step size of the training, and ε is a small value.

Increasing β_1 and β_2 causes more contribution of the cost gradient at the previous epochs to that at the current epoch and is expected to make the training process smoother but slower. In this study, the effects of these hyper-parameters on the convergence of the cost are investigated.

3-4- Assessment of proposed models

This section presents assessment strategies to evaluate the performance of the classical supervised and physics-informed models in terms of training time, prediction accuracy, and effectiveness as initialisers for iterative solvers. The procedure for each case is to train the machine learning models over randomly generated configurations within a particular parameter space and test them on unseen samples in the same parameter space.

3-4-1- Accuracy as an emulator

In order to quantify the difference between the displacement field predicted by the machine learning models, $\boldsymbol{U}_{Pred}$, and that given by a classical solver, $\boldsymbol{U}_{ref}$, the relative error for each sample is defined as:

$$Err = \frac{max(|\boldsymbol{U}_{pred} - \boldsymbol{U}_{ref}|)}{max(|\boldsymbol{U}_{ref}|)} \times 100\% \quad (7)$$

In this study, average and maximum *Err* quantities are reported to assess the accuracy. Additionally, the computational time for creating an emulator as well as its accuracy using the proposed models are discussed in Section 6-1.

3-4-2- Effectiveness as an initialiser

The effectiveness of the trained model as an initialiser (ML-initialiser approach) is assessed by embedding the predictive model into an iterative solver to generate an initial solution field; then, the resultant acceleration for unseen configurations is measured.

Two benefits of the ML-initialiser approach are studied. The first one, "speedup", compares the number of solution iterations required within a finite volume iterative solver *with* and *without* the ML-initialiser approach. The second aspect accounts for data generation time and training time; as a result, the overall benefit of the ML-initialiser, in terms of reduced computational time, can be assessed. Two effectiveness parameters are defined; the first, Eff_1, is defined as

$$Eff_1 = \frac{t_M}{t_N + t_T + t_{Mml}} \times 100\% \quad (8)$$

where the numerator (t_M) represents the time required to solve the unseen cases *without* the ML-initialiser. The denominator represents the total time required to solve the same unseen cases *with* the ML-initialiser, which includes the time taken to solve the training cases (t_N), the time taken to train the machine learning model (t_T), and the time required to solve the unseen samples *with* the ML-initialiser (t_{Mml}). From Eq. (8), increasing the size of the training dataset (N) leads to a larger value for t_N, which may reduce the effectiveness Eff_1; however, an increased training dataset size may lead to a more accurate machine learning model, which potentially increases Eff_1 by reducing the time to solve unseen cases with the ML-initialiser (t_{Mml}).

This first effectiveness parameter Eff_1 measures the benefit of the machine learning model when the solution of the training cases is not of interest, i.e. if the goal is to solve M unseen cases and additional N cases are required by training a machine learning model, then Eff_1 can reliably measure the benefit of using an ML-initialiser. However, if the goal is to solve K unseen cases, where a portion of these cases (N) are used to train the machine learning model (Figure 3), then it is appropriate to define a second effectiveness parameter, Eff_2, as

$$Eff_2 = \frac{t_M + t_N}{t_N + t_T + t_{Mml}} \times 100\% \quad (9)$$

where, in this case, the numerator ($t_M + t_N$) represents the time required to solve the $K = M + N$ total cases *without* the ML-initialiser. The denominator is the same as in Eff_1; however, notice that in this case, the N training cases represent a portion of the total cases to be solved. As a concrete example, consider 500 total cases that are to be solved; the machine learning approach must decide how many of these cases (N) it should use to train the machine learning model such that it can accelerate the remaining M cases. If Eff_2 is greater than 100% then it demonstrates solving 500 cases is faster using a machine-learning-enhanced approach than not, even taking into account the data generation and training time. Section 6-2 examines the effect of this N/M split on the overall benefit of the ML-initialiser approach.

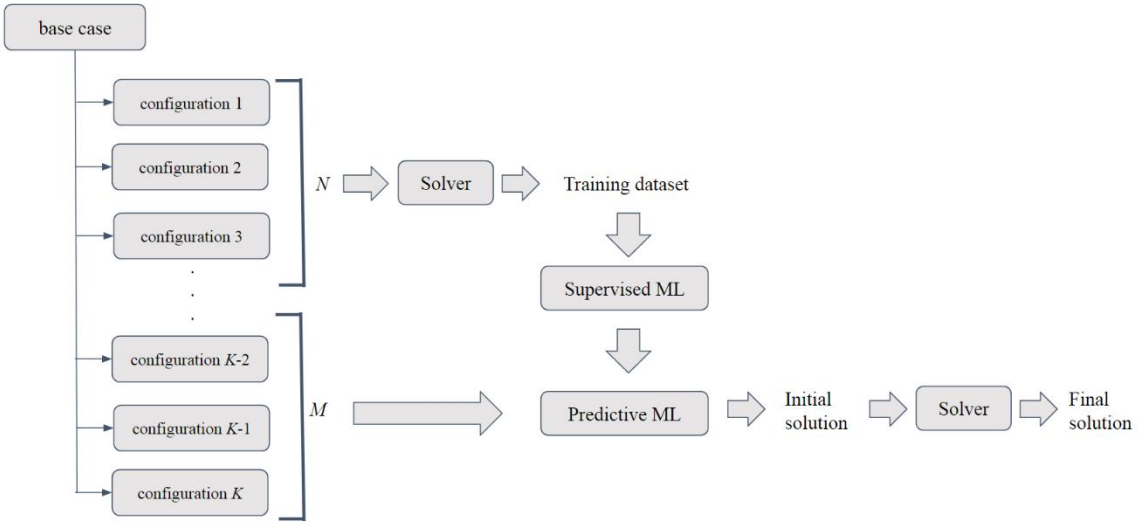

Figure 3- The procedure of the ML-initialiser approach, which includes data generation (N samples), training the machine learning model, and applying the predictive model as an initialiser for the unseen dataset (M samples)

4- Symbolic momentum residual

For the physics-informed model, the prediction of the displacement field is followed by computing the residual of the discrete momentum equation. Calculating this residual requires evaluating gradient and divergence terms in the momentum equation for the predicted displacement fields. A symbolic relationship between the loss and predictions conveniently allows for automatic differentiation [33], a core component of optimisation within the training process. This section establishes this symbolic relation between the displacement field and derivative operators, i.e. gradient and divergence, over a discrete domain. This task splits the role of the displacement field and mesh structure on these operators. This role separation, together with introducing mesh structure information as additional inputs, allows the training loss to be symbolically defined as a function of the displacement field. The following section discusses the theoretical basis of this relation between momentum residual, derivative operators, displacement field, and mesh data in the context of the cell-centred finite volume method.

4-1- Symbolic momentum residual in finite volume method

In the finite volume method, the domain is discretised into a finite number of cells (control volumes). Figure 4 shows a schematic of a computational cell P, its neighbours N, and the cell-face parameters. The integral form of the momentum equation is discretised over individual cells, leading to a system of algebraic equations. For the sake of simplicity and to prove the concept, this study applies conventional discretisation schemes, but the approach is expected to work with more sophisticated and accurate schemes.

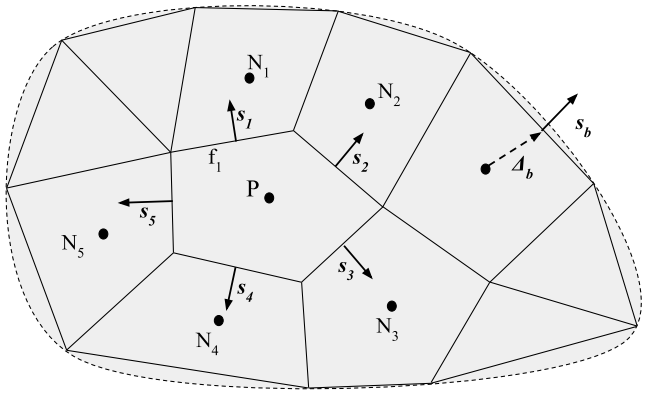

Figure 4- Discretisation of computational domain: P is the centre/centroid of the computational cell P; N_i is centre of *i*th neighbour cell; f_i are faces surrounding the cell P and $\boldsymbol{s_i}$ are their corresponding outward surface vectors; Δ_b is a vector connecting the centre of boundary cells to the adjacent boundary face.

To reconstruct residuals of the momentum equation as the training loss for the physics-informed model, computation of the gradient and divergence operators is required, as Eq. (2) suggests. The volume-averaged residual of the momentum equation over cell P, surrounded by a closed surface *S*, is defined as

$$\boldsymbol{R_P} = \frac{1}{V_P}\oint_S \boldsymbol{n}\cdot\boldsymbol{\sigma}\, dS = \frac{1}{V_P}\sum_f \int_f \boldsymbol{n}\cdot\boldsymbol{\sigma}\, ds \tag{10}$$

where V_P indicates the volume of cell P; f represents individual faces around cell P, and s denotes their corresponding surface area. Following the conventional approach in the finite volume method for assuming a uniform distribution of the stress at face f, the residual at each cell P can be calculated in terms of the face-averaged stress, $\boldsymbol{\sigma_f}$, as

$$\boldsymbol{R_P} \approx \frac{1}{V_P}\sum_f \boldsymbol{s_f}\cdot\boldsymbol{\sigma_f} \tag{11}$$

Defining $\overline{\Phi_P}$ as the volume-averaged quantity Φ within cell P, the residual in cell P can be interpreted as the volume-averaged divergence of the stress within that cell:

$$\boldsymbol{R_P} = \overline{(\boldsymbol{\nabla}\cdot\boldsymbol{\sigma})_P} \tag{12}$$

To evaluate the stress, the displacement gradient is required, and is computed here within cell P using the Green-Gauss theorem [34]:

$$\overline{(\boldsymbol{\nabla U})_P} = \frac{1}{V_P}\sum_f \boldsymbol{U_f}\otimes\boldsymbol{s_f} \tag{13}$$

Two types of cells exist in the computational domain: internal cells whose gradient is a function of the mesh data and displacement in neighbour cells; and boundary cells whose gradient also depends on the adjacent boundary conditions. At first, we explain the evaluation of these operators at internal cells, and then address the boundaries.

Various interpolation schemes are available in the literature to evaluate quantities at the internal face. In this study, the linear interpolation scheme is used. According to this scheme, the stress tensor at each internal face, $\boldsymbol{\sigma_f}$, is computed by

$$\boldsymbol{\sigma_f} = w_P\ \boldsymbol{\sigma_P} + w_N\ \boldsymbol{\sigma_N} \tag{14}$$

Subscript P and N indicate two cells sharing the face f (Figure 4); w_P and w_N are scalars proportional to the inverse distance between face f and the corresponding cell-centres. Combining Eqs. (12) and (14) gives the volume-averaged divergence of stress within cell P as

$$\overline{(\boldsymbol{\nabla}\cdot\boldsymbol{\sigma})_P} = \boldsymbol{\sigma_P}\cdot\boldsymbol{W_P} + \sum_{\mathrm{N}=1}^{Q}\boldsymbol{\sigma_N}\cdot\boldsymbol{W_N} \tag{15}$$

Here, $\boldsymbol{W_P}$ and $\boldsymbol{W_N}$ are vectors coefficients that are functions of w_P, w_N, and cell geometry; Q is the number of neighbours around cell P, e.g. Q = 5 in Figure 4. In addition, the stress is a function of the displacement gradient according to Eq. (2). By applying the linear interpolation for displacement at the internal faces, Eq. (13) can be formulated in terms of cell-centre displacements as

$$\overline{(\boldsymbol{\nabla U})_P} = \boldsymbol{U_P}\otimes\boldsymbol{W_P} + \sum_{\mathrm{N}=1}^{Q}\boldsymbol{U_N}\otimes\boldsymbol{W_N} \tag{16}$$

Note that the $\boldsymbol{W_P}$ and $\boldsymbol{W_N}$ in Eqs. (15) and (16) are the same, referred to here as *divergence coefficients*. Given the displacement field and divergence coefficients at each cell, the momentum residual can be evaluated via Eqs. (2), (15), and (16).

Considering w_P and w_N in Eq. (14) as geometric-based values, one can realise that $\boldsymbol{W_P}$ and $\boldsymbol{W_N}$ are only functions of the mesh parameters. So, Eqs. (15) and (16) indicate that the gradient and divergence operators are functions of the displacement field and parameters associated with the mesh. Additionally, for problems where samples differ only in the geometry, these coefficients can potentially be used as the network input as is proposed here, i.e. an input tensor $\boldsymbol{W}$ for each cell consisting of the divergence coefficients $\boldsymbol{W_P}$ and $\boldsymbol{W_N}$. The input tensor for each cell, $\boldsymbol{W}$, is a tensor with d × (k + 1) components where d indicates dimensions (here we have used 3-D) and k indicates the maximum number of cell neighbours in the mesh. So, for a structured quadrilateral mesh (each cell has a maximum of 4 neighbours) with 100 cells, the input for each sample is a rank 2 tensor with the shape (100, 15).
It should be noted that the input can be defined by any other quantities that can describe the variation of the geometry across the samples, e.g. coordinates of the cells; however, a comparison of the results for a few other inputs suggested that the divergence coefficient $\boldsymbol{W}$ can produce a faster training process.

4-2- Boundary conditions

For the boundary cells, each boundary type is considered separately. Here, we focus on displacement and traction as two common types of boundary conditions for solid deformation problems. For cells adjacent to boundaries, Eq. (16) is modified as

$$\overline{(\boldsymbol{\nabla U})_P} = \boldsymbol{U_P} \otimes \boldsymbol{W_P} + \sum_{\mathrm{N}=1}^{Q} \boldsymbol{U_N} \otimes \boldsymbol{W_N} + \underbrace{\sum_{\mathrm{b}=1}^{N_b} \boldsymbol{U_b} \otimes \boldsymbol{W_b}}_{\substack{\text{contribution of} \\ \text{boundary faces}}} \tag{17}$$

where

$$\boldsymbol{W_B} = \frac{\boldsymbol{S_b}}{V_P}$$

and N_b refers to the number of boundary faces within cell P. $\boldsymbol{U_b}$ represents the displacement value at each boundary face *b*. The third term on the right-hand side represents the contribution of the boundary faces to the gradient tensor at the adjacent cell P.

For displacement boundaries $\boldsymbol{\Gamma}_{\mathrm{U}}$, the displacement value is known on the boundary faces. So, the effect of these boundary faces on the gradient at adjacent cells is directly applied by substituting $\boldsymbol{U}_{\mathbf{b}}$ with $\boldsymbol{U}_{\boldsymbol{\Gamma}}$ in Eq. (17).

For traction boundaries, Eqs. (1c), (2), and (3) are combined and then rearranged in terms of the normal displacement gradient, leading to [35]

$$(\boldsymbol{n}.\boldsymbol{\nabla U})_{\boldsymbol{b}} = \frac{\boldsymbol{T_\Gamma} - \boldsymbol{n_b} \cdot [\mu \boldsymbol{\nabla U_b} - (\lambda + \mu)\boldsymbol{\nabla U_b}] - \boldsymbol{n_b}\lambda \mathrm{tr}(\boldsymbol{\nabla U})}{\lambda + 2\mu} \tag{18}$$

This equation relates the normal displacement gradient, $(\boldsymbol{n}.\boldsymbol{\nabla U})_{\boldsymbol{b}}$, at each boundary face with the specified traction $\boldsymbol{T_\Gamma}$ at that face. Here, subscript $\boldsymbol{b}$ represents the boundary face, and $\boldsymbol{n_b}$ denotes the outward normal vector at the boundary face. The gradient tensor at a traction boundary is evaluated as

$$\boldsymbol{\nabla U_b} = [(\mathbf{I} - \boldsymbol{n_b} \otimes \boldsymbol{n_b}) \cdot \boldsymbol{\nabla U_P} + \boldsymbol{n_b} \otimes (\boldsymbol{n}.\boldsymbol{\nabla U})_{\boldsymbol{b}}] \tag{19}$$

Moreover, displacement at a traction boundary is extrapolated from the internal domain as

$$\boldsymbol{U_b} = \boldsymbol{U_P} + \boldsymbol{\Delta_b} \cdot \boldsymbol{\nabla U}_b + [(\mathbf{I} - \boldsymbol{n_b} \otimes \boldsymbol{n_b}) \cdot \boldsymbol{\Delta_b}] \cdot \boldsymbol{\nabla U_b} \tag{20}$$

where $\boldsymbol{\Delta_b}$ is a vector connecting the centre of the boundary cell to the adjacent boundary face (Figure 4). Having the displacement $\boldsymbol{U_b}$ computed from Eq. (19), the effect of traction boundaries on the gradient at the adjacent cell can be applied. Eqs. (17) to (20) are coupled together, and conventional finite volume solvers use an iterative procedure to add the effect of this boundary on the gradient in adjacent cells.

As apparent from Eq. (17) to (20), implementing the contribution of the boundaries to the gradient tensor requires 'boundary data' such as $\boldsymbol{S_b}$, $\boldsymbol{V_P}$, $\boldsymbol{U_\Gamma}$, and $\boldsymbol{T_\Gamma}$. These data are given to the loss evaluator as additional inputs.

4-3- Automatic differentiation and symbolic momentum residual

Following the symbolic definition of the gradient and divergence operators and hence residual for all cells in terms of the displacement field, the loss is defined as the normalised momentum residual:

$$\mathbf{Loss} = \frac{\|\mathbf{R}\|^2}{\mathbf{R_0}} \tag{21}$$

Note that the **Loss** is computed for each sample, and the division is component-wise. For a sample with *n* cells in a 2-D elastostatic problem, **R** is a tensor with the shape (*n*, 2) whose components for each cell are defined in Eq. (12) and (15); $\mathbf{R_0}$ is the cell-wise average of the momentum residual for a zero displacement field, and, hence, is a tensor with the shape (1, 2). So, for each sample, the **Loss** is a tensor with the shape (*n*, 2).

The training aims to find the set of network weights that minimise the cost function, a scalar defined as the average loss across all cells and samples. During training, the backpropagation algorithm updates the network weights using the gradient of the cost function, calculated using automatic differentiation. Automatic differentiation works by decomposing the derivatives into a sequence of elementary arithmetic by the chain rule. Thanks to the symbolic definition of the momentum residual in terms of the displacement field, there is a chain of symbolic relations between the network weights and the cost function. Consequently, this allows the chain rule to be applied when computing the gradient of the cost function with respect to the network weights within optimisers.

5- Implementation

All procedures and cases from this article are publicly shared with the community at https://bitbucket.org/philip_cardiff/smartsim. The neural networks for both the classical supervised and physics-informed approaches are implemented in TensorFlow (version 2.8.0) [33], where the Keras library (version 2.4.3) [34, 35] is used to facilitate coding. The relevant piece of code for building and training the machine learning models is explained below. The computational mechanics methods have been embedded in solids4foam [28, 29], a toolbox for OpenFOAM [36]. Maulik et al. [37] described the practical procedure

for embedding a predictive model into the OpenFOAM package, with an alternative approach recently described by [38].

The code in Listing 1 shows the implementation of both approaches, where it is demonstrated how the classical supervised and physics-informed models share the same neural network and differ only in the loss definition. The network in this code uses only one hidden layer with 50 neurons to ease readability. As is clear from the code, the input is flattened at first, then normalised within each batch, and is finally fed to the hidden layer. The function of *tf.reshape* in the output layer ensures that the shape of the network output is consistent with the expected solution field, i.e. the number of cells and 2-D solution field. The variable *pinn* is a bool that directs the flow of the code according to either approach, i.e. between the classical supervised or physics-informed models. As is apparent, the *momRes* is the last layer of the network for the physics-informed model. However, it should be noted that after training this model, the predictions are provided by one layer before this layer, namely *output*.

Listing 1:

```
def fullyConnectNet(input, nNeuron):
    inputLayer = Flatten()(input)
    normalizedInput = BatchNormalization()(inputLayer)
    hiddenLayer = Dense(nNeuron, activation='relu')(normalizedInput)
    output = tf.reshape(Dense(numberOfCells*2)(hiddenLayer), [-1, numberOfCells, 2])
    return output
def BuildModel(neurons):
    inputNet = Input()
    displacement = fullyConnectNet(inputNet, neurons)
    if (Pinn): # use physics-informed approach
        caseInitLossInp = Input()
        fvDataInp = Input()
        momRes = Lambda(momentumResidual)([displacement, fvDataInp, caseInitLossInp])
        model = Model(inputs=[inputNet, fvDataInp, caseInitLossInp], outputs=momRes)
    else: # use classic approach
        model = Model(inputNet, displacement)
mlModel = BuildModel(50)
```

The keywords *Dense*, *Lambda*, *Input*, and *Model* are built-in Keras functions. In particular, *Lambda*, which creates a custom layer, is used to apply the *momentumResidual* function as the final layer on the physics-informed model. *momentumResidual* is a function that receives the predicted displacement from the network, finite volume-based discrete operator data, and initial residual of the momentum; then, as the output, it gives the dimensionless residual of the discrete momentum equation. Section 4 explains the theoretical basis for this function.

The code in Listing 2 demonstrates the application of this library for an iterative elastostatic solver in the soilds4foam toolbox:

Listing 2:

```
mlInitialiser.set
(
     new machineLearningInitialiser
     (
          modelPath,
          inputLayerNames,
          outputLayerNames,
          D(),
        )
 );
….
if (predictor_)
{
    mlInitialiser->predict(D().internalField());
}
```

As is apparent in this code, an object from the class *machineLearningInitialiser* is created by feeding the path of the pre-trained model, names of the input and output layers, and the variable for prediction, i.e. displacement field. When the *predictor_* flag is enabled, the machine learning model updates the displacement field with a prediction. In Section 6-2, this library is employed to assess the effectiveness of the predictive model as an initialiser.

6- Test Cases

This section aims to assess the performance of the proposed machine learning models in terms of training time, prediction accuracy, and effectiveness as initialisers for iterative finite volume solvers. These assessments are carried out by separately training machine learning models over three test cases: a curved plate, a plate with a hole, and a cantilever beam. The architecture of the neural network for each case, including the number of hidden layers and the number of neurons within each hidden layer, is chosen by trial and error to reduce the network size and the training time. Since exhaustive optimisation of the network architecture is not the focus of this work, the trial and error approach is deemed sufficient here to demonstrate the proposed approach. The procedure for each case is to train the machine learning models over randomly generated geometries within a determined parameter space to learn the solution field in terms of varying geometry. The capability of these predictive networks as emulators and initialisers is then evaluated via the parameters introduced in Sections 3-4 (*Err*, speedup, Eff_1, and Eff_2).

In Section 6-1, the training time and prediction accuracy of the proposed approaches are compared, where the reference solution is given by an iterative solver in the solids4foam toolbox for OpenFOAM. Also, the effect of the β optimisation parameters on cost convergence is studied. Subsequently, Section 6-2 examines

the potential computational benefits of applying the proposed machine learning models as initialisers for an iterative solver in solids4Foam; moreover, the effect of mesh size and geometry on the benefits is investigated.

6-1- Machine learning models as an emulator

The curved plate and a plate with a hole cases are used to study the accuracy and training time of the machine learning models. In order to assess the emulation capability of both models, the predictions by the models for unseen geometries are compared with results obtained from the standard numerical solvers.

6-1-1- Curved plate

The geometry for the curved plate case is defined by two arcs, each specified by three points (Figure 5).

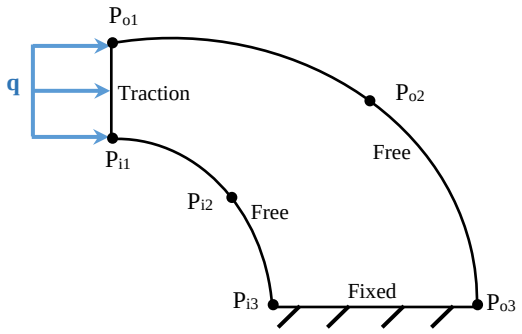

Figure 5- Schematic of the curved plate case: each of the two arcs in the geometry is defined by three random points

A uniform distributed load, q = 1.0 Pa, is applied to the left boundary. The material properties are Young's modulus E = 2.0 Pa and Poisson's ratio v = 0. The training dataset consists of various geometries generated by random positioning of the six points within the following parameter space: $P_{o1,y}$ = 1.0 to 2.0 m, $P_{i1,y}$ = 0.3 to 0.7 × $P_{o1,y}$, $P_{o2,x}$ = (0.8 to 1.0) × $P_{o1,y}$, $P_{i2,x}$ = (0.3 to 0.7) × $P_{o2,x}$, $P_{i3,y}$ = (0.85 to 0.95) × $P_{i1,y}$, $P_{o3,y}$ = (0.85 to 0.95) × $P_{o1,y}$. The mesh for all of the samples consist of 15 × 6 = 90 cells. Figure 6 shows geometry of 4 samples in the dataset.

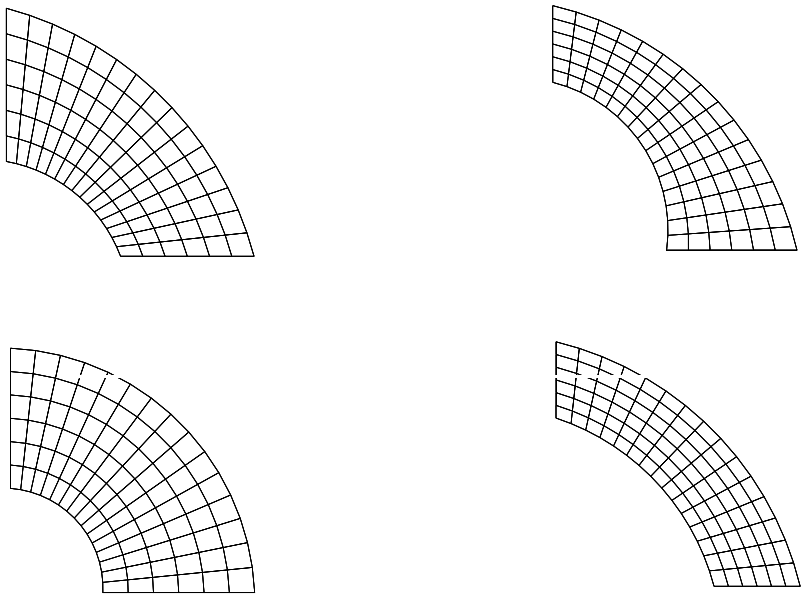

Figure 6- Mesh and geometry of 4 samples from the curved plate dataset

For the physics-informed approach, a fully connected network with 753 k parameters is applied, consisting of 12 hidden layers with 200 neurons in each and the output layer with 180 neurons. The number of samples used in the training dataset is 300, each with 90 cells. As a result, the network input is an array with the shape (300, 90, 15), where 15 refers to the number of divergence coefficients components, i.e. $\boldsymbol{W_P}$ and $\boldsymbol{W_N}$, for each cell. Flattening the input layer delivers (300, 1350) to the normalisation layer, where its output is fed to the first hidden layer. The training process is carried out through the Adam optimiser with different values for the β_1 and β_2.

Figure 7 demonstrates the cost convergence during training for different values of beta parameters. As is evident, β_1 = 0.999 and β_2 = 0.99999 provides the best convergence rate; larger values give a slower convergence, and smaller values result in oscillatory behaviour.

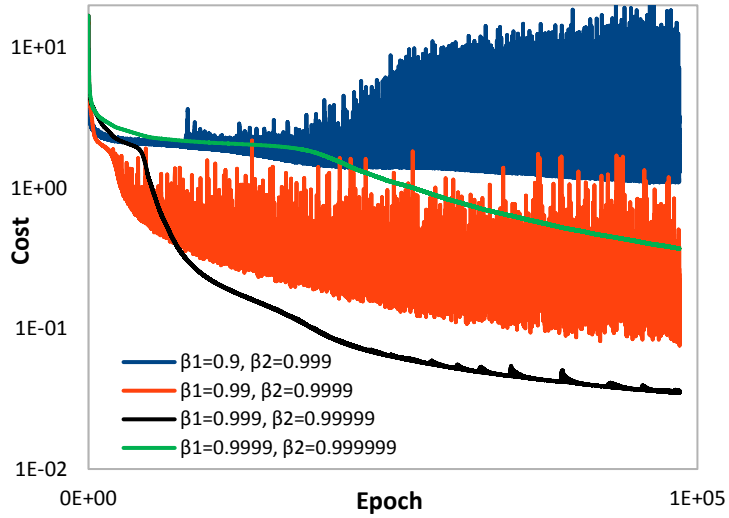

Figure 7- Convergence of the cost within the training process of the physics-informed model used for the curved plate case

For the classical supervised approach, a fully connected network with 43 k parameters consists of 6 hidden layers with 20, 50, 50, 50, 50, 20 neurons and an output layer with 180 neurons. The type of inputs, number of samples, and the normalisation layer for this network are the same as in the physics-informed one. However, for the training process, true displacement fields are required. Here, they are generated by the solids4Foam OpenFOAM solver and then standardised to give a mean of zero and a standard deviation of one. Finally, they are given to the network as the ground-truth data. As before, the training process is carried out through the Adam optimiser with different values for the beta parameters.

As presented in Figure 8, β_1 = 0.99 and β_2 = 0.9999 leads to smoother convergence behaviour. Due to the different definitions of the loss function for the supervised and physics-informed networks, their cost values during the training should not be compared. However, as is evident from the final epoch numbers in Figure 6 and Figure 7, training of the physics-informed network requires five times more epochs for nearly the same accuracy.

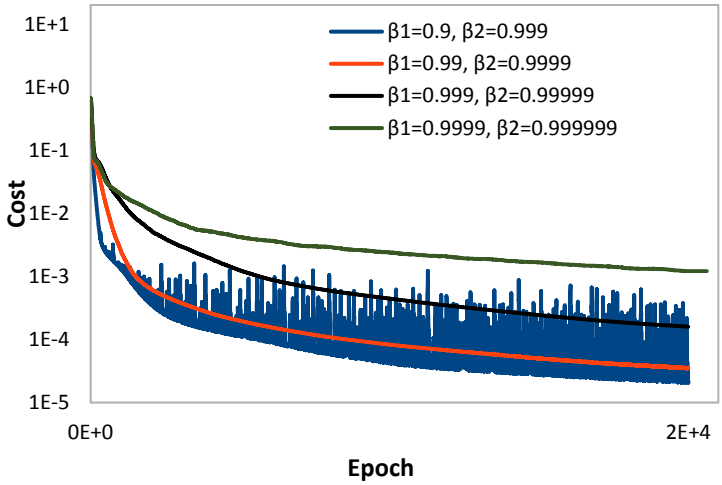

Figure 8- Convergence of the cost within the training process of the classical supervised for the curved plate case

The trained models are then employed as an emulator for the classical numerical solvers to predict solution fields for unseen configurations. Figure 9 compares the displacement fields given by the supervised and physics-informed models for two example unseen configurations with the results obtained by the corresponding solids4Foam OpenFOAM solver.

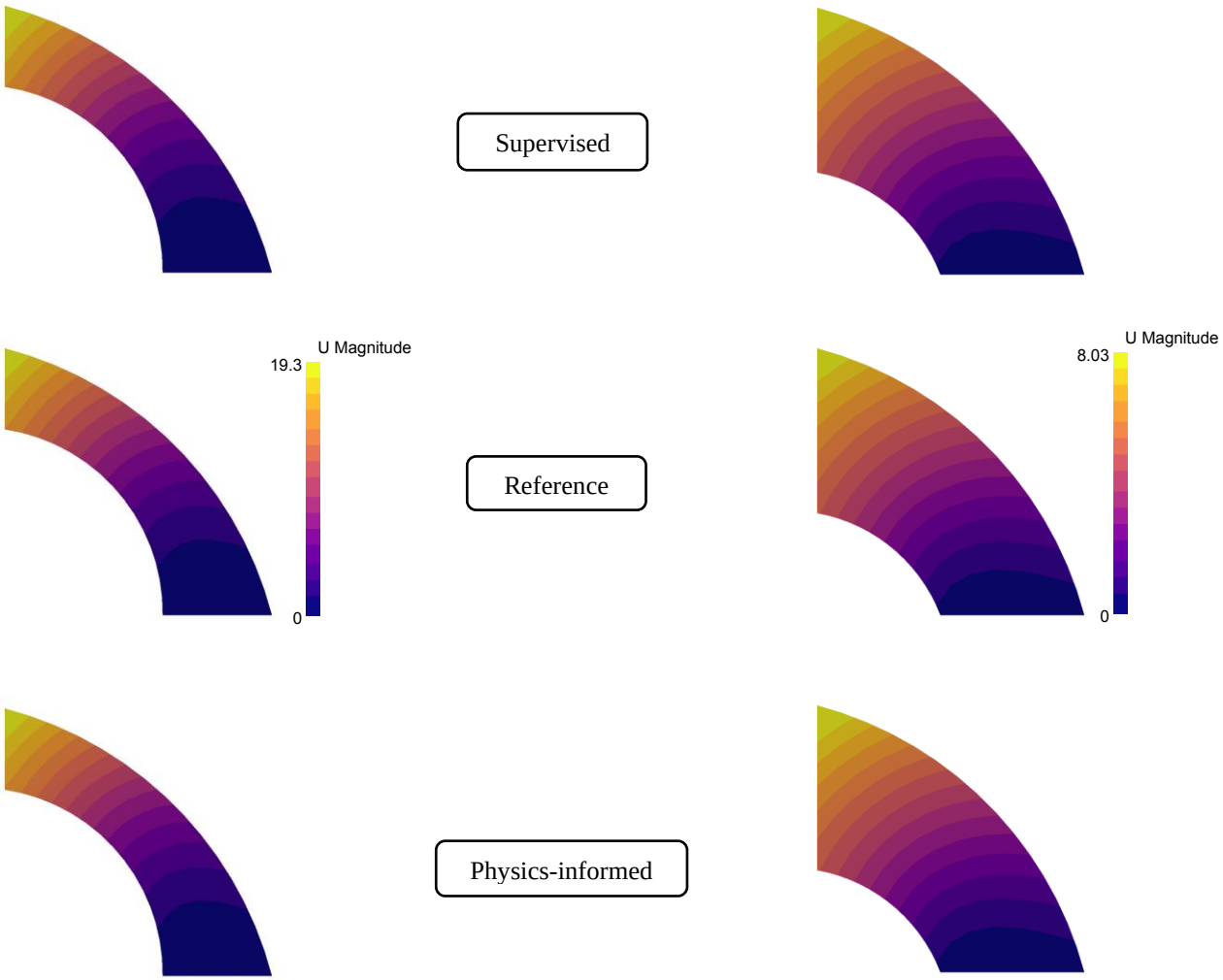

Figure 9- Contour of displacement magnitude for the curved plate case, given by supervised model, corresponding solids4Foam OpenFOAM solver, and physics-informed model at top, middle, and bottom, respectively, for two geometries at left and right

Table 1 presents the training time and accuracy of the classical and physics-informed models for the training and unseen datasets. The classical supervised approach outperforms the physics-informed approach in terms of computational time for the training process at a similar level of accuracy. In this case, it is over 100 times faster. For smaller physics-informed networks, e.g. a network with 43 k parameters, the training process does not converge; additionally, each epoch is 25 times slower than that of the supervised approach with

the same network size. As a result, compared to the training time in the physics-informed model, the time required for generating data in the classical supervised model is negligible for this case.

Table 1-Training results for the curved plate case with 300 training samples and 100 unseen samples. t_N is the solution time to generate the training samples, and t_T is the training time.

Model type	Weights [k]	t_N [min]	t_T [min]	Ave. *Err* [%]		Max. *Err* [%]	
				training	unseen	training	unseen
Classical supervised	95	1.5	3.5	1.01	1.38	5.56	3.93
Physics informed	753	-	373	2.23	2.58	5.49	5.95

6-1-2- A plate with a hole

The geometry of this case (*Figure 10*) is fully described by the length of the plate, L = H, and the radius of the hole, R. The fixed and varying parameters of this case are: the pressure on the top, P_T = 1.0 Pa, the pressure on the right, P_R = 1.0 Pa, the pressure on the hole, P = 1.0 Pa, geometrical parameters, L = H = (1 to 2) m, R = (0.2 to 0.4) L, the Young's modulus, E = 2.0 Pa, and the Poisson's ratio, ν = 0.3. The neural network for this case is trained over randomly generated L and R values within parameter spaces 1.0 to 2.0 m and 0.2 L to 0.4 L, respectively.

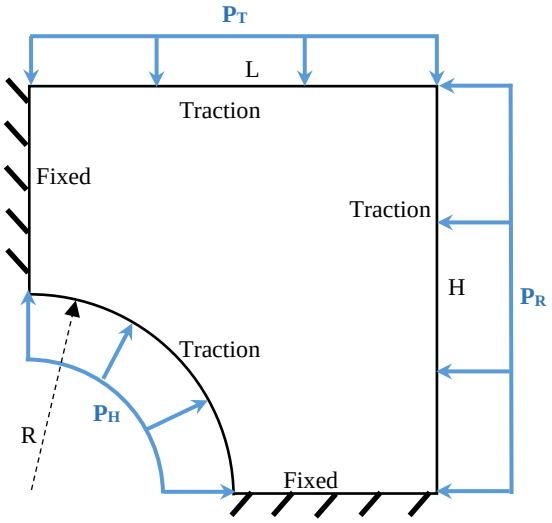

Figure 10- The plate with a hole case: R and L are randomly varied to generate samples in the dataset

The samples for this case are allowed to have different mesh sizes (190 to 250 cells) to preserve the mesh quality for various geometries. Mesh quality is preserved by enforcing a uniform distribution of boundary nodes on each domain edge. Figure 11 shows four samples in the dataset.

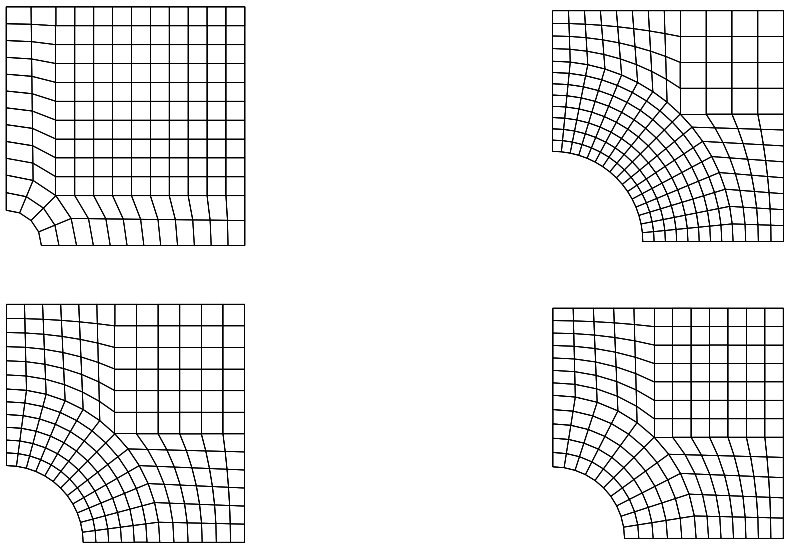

Figure 11- Mesh and geometry of 4 samples in the plate with a hole dataset

The network for both supervised and physics-informed models has 115 k parameters, consisting of 7 hidden layers with 30 neutrons in each and the output layer with 250 neurons. The number of samples used in the training dataset is 300. As a result, the network input is an array with the shape (300, 250, 15). Since the samples are free to have various numbers of cells, the mesh with the maximum cell number is chosen to determine the shape of the input. For the samples with smaller cells, the loss is enforced to be zero for non-existent cells.

For both supervised and physics-informed models, the Adam optimiser with various values for β_1 and β_2 is applied to train the networks. As demonstrated in Figure 12, β_1 = 0.9 and β_2 = 0.999 for the supervised model, and β_1 = 0.99 and β_2 = 0.9999 for the physics-informed model, give the best convergence rate.

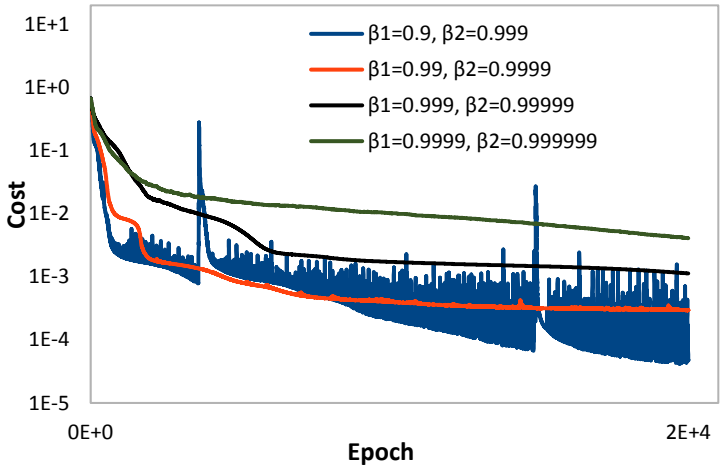

Figure 12- Convergence history of the cost for the supervised model trained on the plate with a hole case

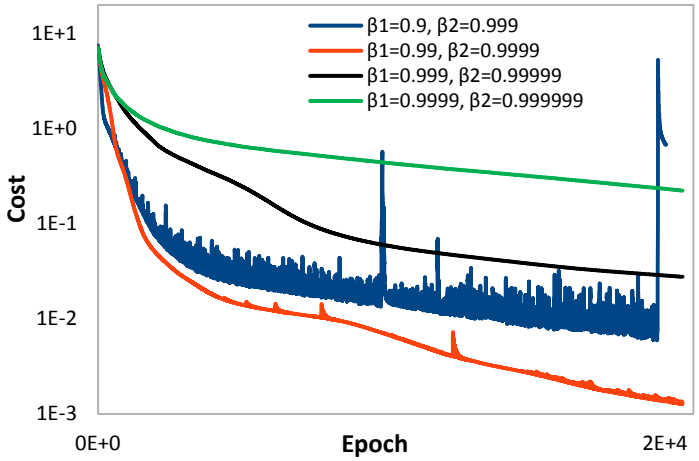

Figure 13- Convergence history of the cost for the physics-informed model trained on the plate with a hole case

Following the training, the solution fields generated by the trained models and the corresponding solids4Foam solver are compared for two cases (Figure 14).

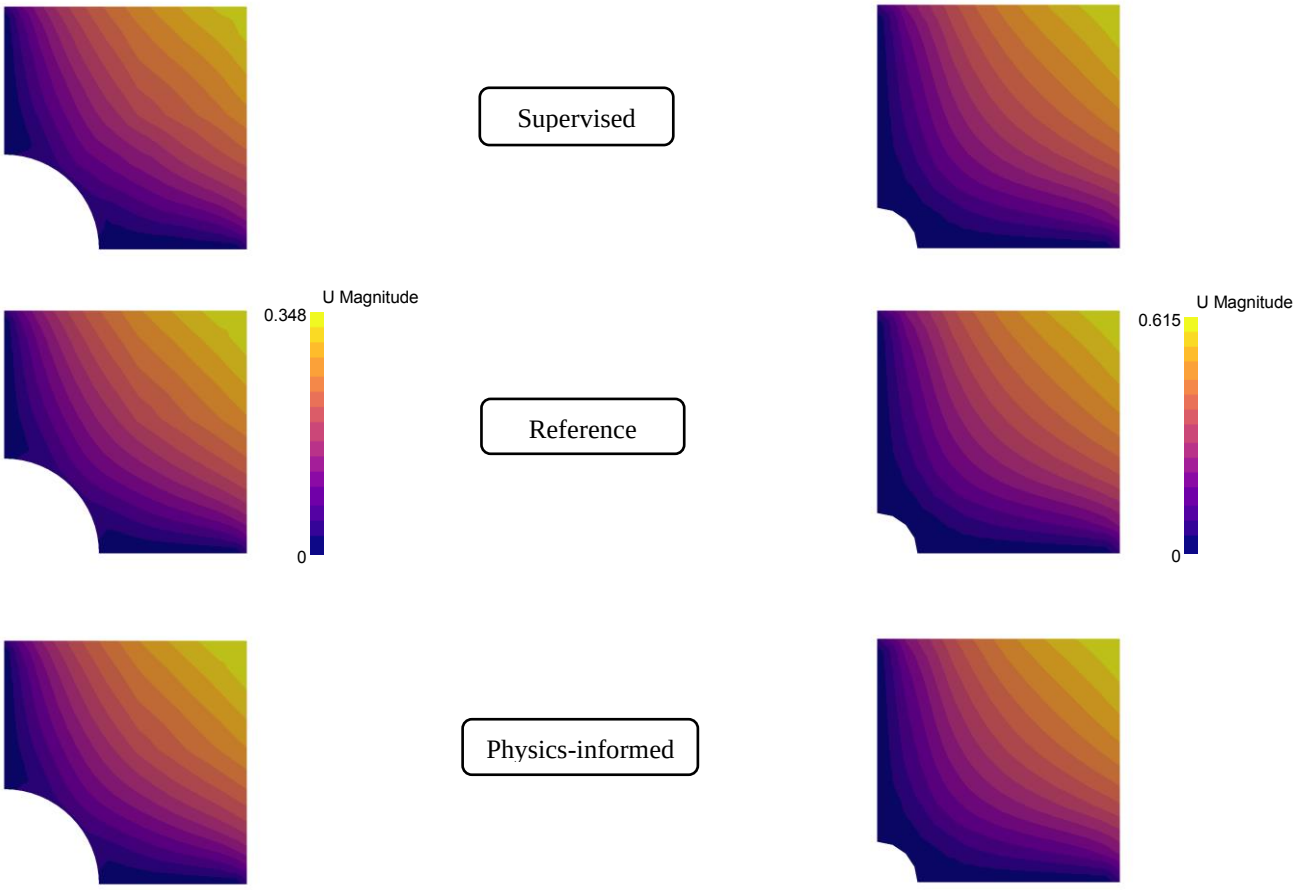

Figure 14- Contour of displacement magnitude for the plate with a hole case, given by supervised model, corresponding solids4Foam OpenFOAM solver, and physics-informed at the top, middle, and bottom, respectively, for two geometries at left and right

Table 2 quantitatively compares the predictions of both machine learning models with the solids4Foam solver for training and unseen datasets.

Table 2- Training results for the plate with a hole case with 300 training samples and 100 unseen samples. t_N is the solution time to generate the training samples, and t_T is the training time.

Model type	Weights [k]	t_N [min]	t_T [min]	Ave. *Err* [%]		Max. *Err* [%]	
				training	unseen	training	unseen
Classical supervised	118	1.2	3.5	0.35	0.82	0.86	1.77
Physics informed	118	-	267	2.62	1.72	3.53	2.67

Once again, the time for building a predictive model for the classical supervised approach, which consists of data generation, t_N, and training process, t_T, is two orders of magnitude less than that for the proposed physics-informed model. In this case, the classical supervised approach also achieved lower average and maximum errors on both training and unseen datasets.

The previous two cases show that the reason for the superiority of the classical supervised model over the physics-informed one is mainly due to the large number of algebraic computations needed for evaluating the loss within the physics-informed model. These computations cause each epoch in the training process of the physics-informed model to be nearly 25 times slower than in the classical supervised approach. Another reason originates from the fact that the classical supervised model defines the loss as the mean square error which is known for its favourable characteristics, such as convexity, which benefits the training process.

6-2- Machine learning models as initialisers

The procedure for using a machine learning approach as an initialiser consists of data generation, training the machine learning model, initialising the solution field for unseen problems, and finally solving the problems using a classical iterative solver. Various hyper-parameters associated with the machine learning model play a role in the solution acceleration and the overall benefits. Among them are the training dataset size, the network size, and the number of epochs for the training process. One challenge is finding the optimal values for these parameters to increase the overall computational efficiency. Here, we aim to provide insight into the role of these parameters on the overall computational time. An unseen dataset with 500 samples is used to decrease the effect of randomness in the data generation process on the average speedup and effectiveness; moreover, the entire procedure for each case is repeated three times, and the reported results are the average of these three runs.

6-2-1- Physics-informed model as an initialiser

To demonstrate the capability of the physics-informed model as an initialiser, the predictive model trained over 300 samples within Sections 6-1 is embedded in an iterative solver in solids4Foam. Figure 15 presents the acceleration obtained for unseen configurations from the curved plate and a plate with a hole cases. As is clear from the convergence histories, initialisation with the machine learning model can accelerate the solver up to 20 times. Additionally, the training results from several models show that the speedup depends on the accuracy of the machine learning predictions. This is influenced by the number of samples within

the training dataset and the magnitude of the loss at the final epoch of the training. Despite the speedup given by the physics-informed model, the computational efficiency of this overall approach was not found to be beneficial compared to the non-ML-enhanced approach due to the excessive training time.

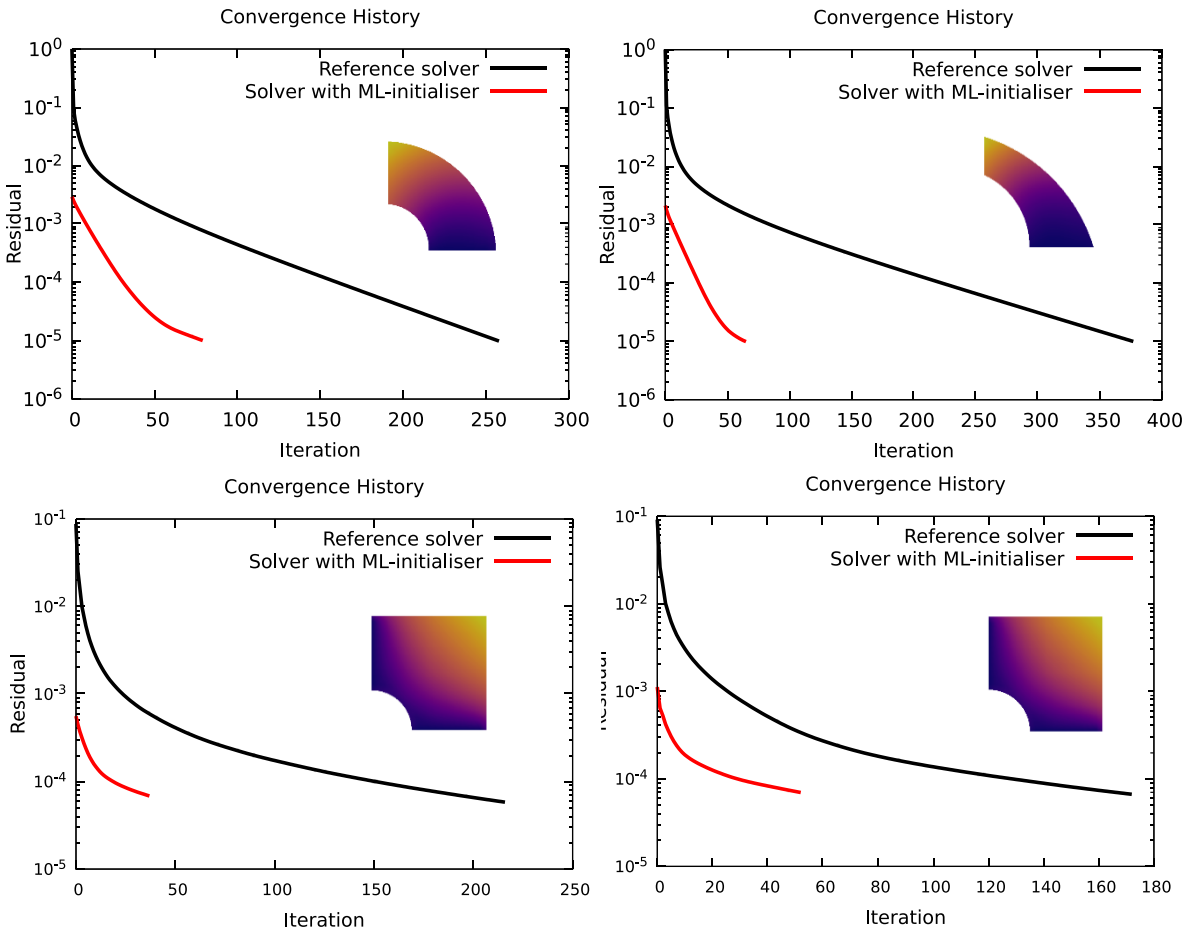

Figure 15- The iterative solver convergence with and without the physics-informed ML-initialiser

6-2-2- Classical supervised model as an initialiser

Here, two cases are studied: the curved plate and the cantilever beam. The reason for choosing the latter case is that it poses computational challenges for segregated classical iterative finite volume solvers due to the strong inter-displacement-component coupling [36].

The classical supervised neural network and training parameters described in Section 6-1-1 are used to create a predictive machine learning model for the curved plate case. Table 3 presents the results for this case. As the minimum speedup of 1.8 suggests, the ML-initialiser accelerates the convergence rate for all

500 unseen samples. Despite the average speedup for all 500 unseen samples being 2.43, a comparison between the total time and the reference time (solution without ML-initialiser) reveals that incorporating an ML-initialiser slowed down the overall solutions and is not beneficial for this case. The main reason for this is the required time for data generation and training.

Table 3- Results of the ML-initialiser for 500 unseen samples from the curved plate case using a network with 43 k parameters trained by 10,000 epochs and the average *Err* of 4.38% for unseen samples. t_N is the solution time to generate the training samples, t_M is the solution time for unseen samples without ML-initialiser, t_T is the training time; t_{Mml} is the solution time for unseen samples using ML-initialiser.

N	Speedup			**Time with ML-initialiser** [sec]				**Time without ML-initialiser**
	Ave.	Min.	Max.	t_N	t_T	t_{Mml}	Total	t_M [sec]
300	2.43	1.8	18.2	145.8	115	213	474	243

For the cantilever beam case (Figure 16), the beam is subjected to a traction T on the right side while the left side is fixed. The fixed parameters for this case are: T = (0 -1.0) Pa, E = 2.0 Pa, ν = 0.3 and the varying parameters for the training and unseen datasets are the length, L = (0.5 to 1.0) m, and height, H = (0.05 to 0.1) m. The mesh size for all cases is 50 × 8 = 400 cells unless stated otherwise.

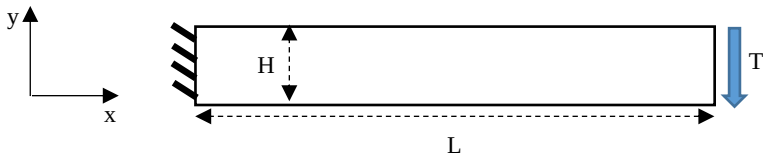

Figure 16- Schematic of cantilever beam

The neural network for this case contains 365 k weights constructed by 7 hidden layers with 20, 50, 50, 50, 50, 20 neurons, and the output layer with n neurons (n denotes the number of cells multiplied by the number of components of the displacement vector, i.e. 2). The number of training epochs applied for these cases is 10,000. Table 4 presents the results for various numbers of the training dataset samples, *N*.

Table 4- Effect of training dataset size on ML-initialiser results for 500 unseen samples from the cantilever beam case. The network has 365 k parameters and is trained by 10,000 epochs. *N* is the number of samples in the training dataset; t_N and t_M are solution time for training and unseen samples, respectively; t_T is the training time; t_{Mml} is solution time for unseen samples using ML-initialiser.

N	Ave. *Err*	Speedup			**Time with ML-initialiser** [sec]				**Reference time** [sec]	Eff_1
	[%]	Ave.	Min.	Max.	t_N	t_T	t_{Mml}	Total	t_M	[%]
10	36.5	1.8	1.01	188.7	81	102	2221	2239	4053	181
20	35.81	2.24	1.02	347.5	165	103	1801	2068	4307	208.3
40	21.2	3.22	1.14	295.6	337	110.7	1265.3	1714	4222	246.3
80	9.86	4.58	1.3	387.7	674	121.7	932.7	1728	4210	243.6
150	8.2	6.2	1.36	363.7	1235	156	716.7	2108	4117	195.3
300	2.52	11.8	1.86	454.1	2443	202.7	461	3096	4055	131
600	1.5	16.3	2.14	300.2	4958	271.7	387	5615	4132	73.6

As is apparent in *Table 4*, the ML-initialiser provides a speedup up to 454 for some samples in the dataset. Also, the minimum speedup of >1 suggests that it always produces a solution speedup for individual cases. The last column in Table 4 shows the effectiveness of the ML-initialiser when accounting for the training dataset time, i.e. Eff_1, up to 246.3%. Given the definition of Eff_1 as the ratio of the solution time without the ML-initialiser to the time with the ML-initialiser plus training and data generation time, the ML-initialiser approach is seen to reduce the overall solution time, even when accounting for the additional ML-related computations. Moreover, comparing the average error (Ave. *Err*) with the speedup implies that the models with higher accuracy result in more speedup, as expected. However, a comparison of these two columns (Ave. *Err* and speedup) with the results of the last column (Eff_1) reveals that the increase in accuracy of the predictive model and related speedup does not necessarily lead to higher effectiveness. In fact, in addition to the accuracy and speedup, the computational time for creating the associated predictive model is another parameter that should be considered for studying the effectiveness. Thus, hyper-parameters other than the size of the training dataset (e.g. training epochs or network architecture) can influence the effectiveness of the ML-initialiser by altering the training time and accuracy of the predictive model. Figure 17, obtained from Table 4, gives more insights into the Eff_1. In this case, the maximum Eff_1 occurs at 40 training samples. For training datasets with a size greater than 40, the additional computational time imposed by data generation and training outweighs the benefits from the accuracy of the predictive model and the resulting speedup for the unseen dataset. Hence, this reduces the overall computational benefits of the ML-initialiser.

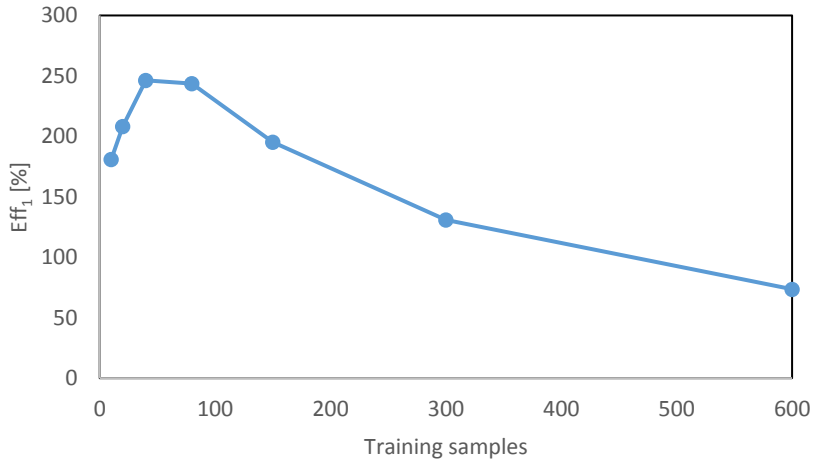

Figure 17- Effectiveness of ML-initialiser (Eff_1) for the case of cantilever beam in terms of training sample size (N)

Figure 18 presents Eff_2 for the results in Table 4 in two states. For the first state, the total number of samples (K) is assumed to be fixed at 500 samples. As the number of training samples N increases, the number of unseen samples M decreases based on $K = N + M$. For instance, for $N = 40$ within Table 4, Eff_2 = 4222 / [337 + 110 + 1265 × (460/500)] × 100% = 262%. Figure 18 shows the maximum of the Eff_2 for the first state is 266% which occurs at $N = 80$. In the second state, the number of unseen samples is kept fixed ($M = 500$) and, as a result, increasing the number of training samples N leads to an increase in the total samples. According to the results in Table 4 for this state, the maximum of the Eff_2 is 282% at $N = 80$. For the same number of training samples N, the total number of samples K in state 2 is always larger

than that in state 1 ($K_2 = K_1 + N_2$, where subscripts 1 and 2 refer to states 1 and 2, respectively). Therefore, we can conclude that Eff_2 increases as the number of total samples to be solved K increases, or to put it another way, the benefit of using the ML-initialiser approach increases as the number of cases to be solved increases.

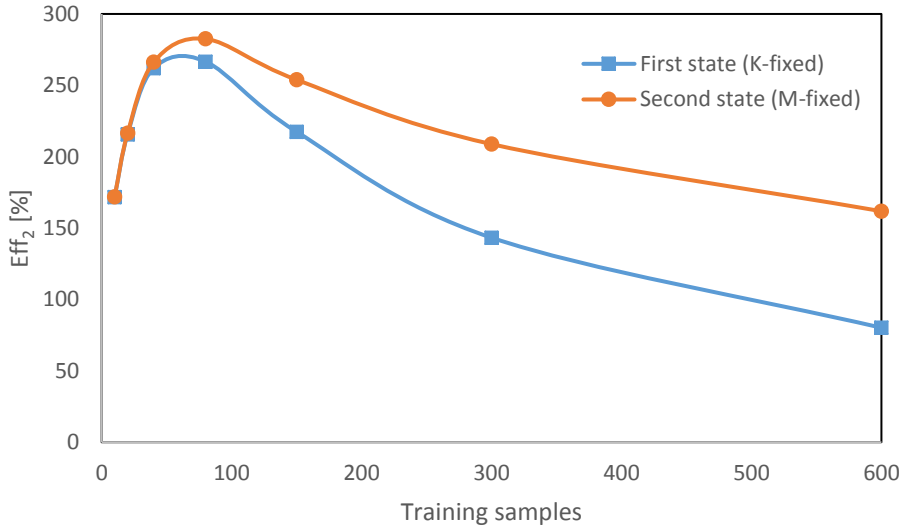

Figure 18- Effectiveness of the ML-initialiser (Eff_2) for the cantilever beam case as a function of training dataset size

All results confirm that the ML-initialiser approach makes the convergence faster by starting from initial solutions closer to the final solutions. However, a comparison of the results for the cantilever beam and curved plate cases suggests that the additional computational cost imposed by the ML-initialiser approach is only justifiable for problems in which the iterative solvers require a considerable number of iterations for convergence, and the number of cases to solve is sufficiently large.

To delve deeper, the results for three length-to-height cantilever beam ratios are compared. For higher length-to-height ratios, the inter-displacement-component coupling becomes stronger, and the performance of the segregated iterative solvers degrades. Table 5 presents the results for the cantilever beam with three datasets that apply different length variations while other parameters are fixed. As is clear from this table, speedup and effectiveness (Eff_1) increase as the geometry of the cantilever beam becomes slimmer (increasing aspect ratio).

Table 5- Results of the ML-initialiser for 500 unseen samples from the cantilever beam case in terms of beam length (aspect ratio) variation. The network has 365 k parameters and is trained by 10,000 epochs over 80 training samples. L is the beam length where the thickness is kept constant; t_N and t_M are solution time for training and unseen samples, respectively; t_T is the training time; t_{Mml} is solution time for unseen samples using ML-initialiser.

L	Ave. *Err* [%]	Speedup			**Time with ML-initialiser** [sec]				**Reference time** [sec]	*Eff*$_1$ [%]
		Ave.	Min.	Max.	t_N	t_T	t_{Mml}	Total	t_M	
0.5-1.0	9.86	4.58	1.3	387.7	674	121.7	932.7	1728	4210	250.7
1.0-2.0	19.21	10.2	1.23	1230	1862	122	1200	3185	11643	365.5
2.0-4.0	7.00	19.3	1.93	1147	4309	113.7	1406.7	5835	26930	461.5

In order to study the effect of mesh size on the performance of the ML-initialiser approach, the proposed acceleration procedure is applied for three levels of mesh: 50 × 8 = 400, 100 × 16 = 1600, 150 × 24 = 3600. As seen in the second column of Table 6, the network size changes with the mesh size. The reason is that the number of neurons in the input and output layers are functions of cell numbers in the mesh. The results in Table 6 suggest that although a straightforward trend cannot be found for the effectiveness, the ML-initialiser approach is always beneficial for all three mesh sizes. Additionally, the contribution of the training process, t_T, to the total time decreases as the mesh size increases.

Table 6- Results of the ML-initialiser for 500 unseen samples from cantilever beam case in terms of the mesh size. The networks are trained by 10,000 epochs over 80 training samples. t_N and t_M are solution time for training and unseen samples, respectively; t_T is training time; t_{Mml} is solution time for unseen samples using ML-initialiser.

Mesh [cells]	Weights [k]	Ave. *Err* [%]	Speedup			**Time with ML-initialiser** [min]				**Reference time** [min]	Eff_1 [%]
			Ave.	Min.	Max.	t_N	t_T	t_{Mml}	Total	t_M	
400	365	9.86	4.58	1.3	387.7	11.1	2.02	15.53	28.65	70.2	250.7
1600	605	27.7	4.16	1.07	381.2	49.9	2.8	61.4	113.9	315.4	276.9
3600	1348	18.3	3.68	1.19	388.6	143	4.7	191.4	339.4	895.1	263.7

7- Conclusions

This paper proposes two new approaches for combining machine learning and cell-centred finite volume methods for solving elastostatic problems. The capability of these approaches as emulators and initialisers for three test cases was studied. The conclusions are as follows:

- Both approaches can emulate the classical solvers with an average error of less than 2%. The training process for the physics-informed approach does not require true displacement fields and prior solutions. However, due to a large number of algebraic computations for evaluating the finite volume-based loss, each epoch for training is 25 times slower than that for the supervised approach. Physics-informed networks also require more epochs to achieve convergence for the same level of accuracy. In other words, even when accounting for data generation time, the proposed classical supervised approach outperforms the physics-informed model in terms of overall computational time and trainability.
- The speedup given by the ML-initialiser approach depends on the case and the machine learning prediction accuracy. Generally, if the iterative solvers require a significant number of iterations for the problem, incorporating the ML-initialiser becomes more beneficial in terms of solution time. In particular, the results for the cantilever beam case demonstrate that the supervised model can accelerate the average and maximum solution process of 500 unseen cases up to 19 and 1147 times, respectively. The speedup increases as the beam aspect ratio increases.
- For problems requiring a considerable number of solutions to the same physical problem, e.g. optimisation problems, the effectiveness (overall benefit) of the ML-initialiser approach increases. The more configurations to solve, the more effective the ML-initialiser is. For such problems, the ML-initialiser approach can learn from the solution of part of the configurations and accelerate the rest so that the acquired benefits outweigh the additional machine learning costs. In particular, for the solution of 500 random configurations from the cantilever beam case, the ML-initialiser can accelerate the cases up to 461%.

- The proposed physics-informed model solves and learns elastostatic problems simultaneously and, therefore, leverages machine learning capabilities to solve these problems. From the authors' point of view, this approach can be developed for other physical problems. The downside is that the computational time of training such a model is currently not comparable to the classical numerical methods, as has been the main disadvantage of the PINN approach [13]. The superiority of the classical numerical methods is mainly due to the well-developed linear system solvers, which outperform the optimisation techniques for computational problems. However, the authors believe that with future improvements to the proposed physics-informed model, e.g. conditioning or normalising/standardising the loss, they may provide a useful alternative solution strategy.

8- Acknowledgements

Financial support is gratefully acknowledged from the Irish Research Council (IRC) through the Laureate programme, grant number IRCLA/2017/45. Additionally, the authors want to acknowledge project affialiates, Bekaert, through the Bekaert University Technology Centre (UTC) at UCD (www.ucd.ie/bekaert), and I-Form, funded by Science Foundation Ireland (SFI) Grant Number 16/RC/3872, co-funded under European Regional Development Fund and by I-Form industry partners. The authors wish to acknowledge the DJEI/DES/SFI/HEA Irish Centre for High-End Computing (ICHEC) for the provision of computational facilities and support (www.ichec.ie), and part of this work has been carried out using the UCD ResearchIT Sonic cluster which was funded by UCD IT Services and the UCD Research Office.

References

[1] C. Yang, X. Yang, X. Xiao, Data-driven projection method in fluid simulation, Computer Animation and Virtual Worlds, 27 (2016) 415-424.
[2] J. Tompson, K. Schlachter, P. Sprechmann, K. Perlin, Accelerating eulerian fluid simulation with convolutional networks, in: International Conference on Machine Learning, PMLR, 2017, pp. 3424-3433.
[3] A. Oishi, G. Yagawa, Computational mechanics enhanced by deep learning, Computer Methods in Applied Mechanics and Engineering, 327 (2017) 327-351.
[4] X. Guo, W. Li, F. Iorio, Convolutional neural networks for steady flow approximation, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 481-490.
[5] N. Thuerey, K. Weißenow, L. Prantl, X. Hu, Deep learning methods for Reynolds-averaged Navier–Stokes simulations of airfoil flows, AIAA Journal, 58 (2020) 25-36.
[6] O. Obiols-Sales, A. Vishnu, N. Malaya, A. Chandramowliswharan, CFDNet: A deep learning-based accelerator for fluid simulations, in: Proceedings of the 34th ACM International Conference on Supercomputing, 2020, pp. 1-12.
[7] Z. Long, Y. Lu, B. Dong, PDE-Net 2.0: Learning PDEs from data with a numeric-symbolic hybrid deep network, Journal of Computational Physics, 399 (2019) 108925.
[8] H. Yao, Y. Gao, Y. Liu, FEA-Net: A physics-guided data-driven model for efficient mechanical response prediction, Computer Methods in Applied Mechanics and Engineering, 363 (2020) 112892.

[9] 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, Journal of Computational Physics, 378 (2019) 686-707.
[10] Y. Shin, J. Darbon, G.E. Karniadakis, On the convergence and generalization of physics informed neural networks, arXiv e-prints, (2020) arXiv: 2004.01806.
[11] X. Meng, Z. Li, D. Zhang, G.E. Karniadakis, PPINN: Parareal physics-informed neural network for time-dependent PDEs, Computer Methods in Applied Mechanics and Engineering, 370 (2020) 113250.
[12] H. Wessels, C. Weißenfels, P. Wriggers, The neural particle method–an updated Lagrangian physics informed neural network for computational fluid dynamics, Computer Methods in Applied Mechanics and Engineering, 368 (2020) 113127.
[13] M. Raissi, P. Perdikaris, G.E. Karniadakis, Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, arXiv preprint arXiv:1711.10561, (2017).
[14] M. Liu, L. Liang, W. Sun, A generic physics-informed neural network-based constitutive model for soft biological tissues, Computer methods in applied mechanics and engineering, 372 (2020) 113402.
[15] Z. Mao, A.D. Jagtap, G.E. Karniadakis, Physics-informed neural networks for high-speed flows, Computer Methods in Applied Mechanics and Engineering, 360 (2020) 112789.
[16] G. Kissas, Y. Yang, E. Hwuang, W.R. Witschey, J.A. Detre, P. Perdikaris, Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks, Computer Methods in Applied Mechanics and Engineering, 358 (2020) 112623.
[17] F. Sahli Costabal, Y. Yang, P. Perdikaris, D.E. Hurtado, E. Kuhl, Physics-informed neural networks for cardiac activation mapping, Frontiers in Physics, 8 (2020) 42.
[18] E. Haghighat, M. Raissi, A. Moure, H. Gomez, R. Juanes, A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics, Computer Methods in Applied Mechanics and Engineering, 379 (2021) 113741.
[19] A.G. Baydin, B.A. Pearlmutter, A.A. Radul, J.M. Siskind, Automatic differentiation in machine learning: a survey, Journal of machine learning research, 18 (2018).
[20] A.D. Jagtap, E. Kharazmi, G.E. Karniadakis, Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems, Computer Methods in Applied Mechanics and Engineering, 365 (2020) 113028.
[21] A.D. Jagtap, G.E. Karniadakis, Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations, Communications in Computational Physics, 28 (2020) 2002-2041.
[22] R. Mattey, S. Ghosh, A novel sequential method to train physics informed neural networks for Allen Cahn and Cahn Hilliard equations, Computer Methods in Applied Mechanics and Engineering, 390 (2022) 114474.
[23] R.Z. Bosagh, B. Ramsundar, TensorFlow for Deep Learning, O'Reilly Media, Incorporated, 2018.
[24] B. Wang, Y. Deng, P. Kry, U. Ascher, H. Huang, B. Chen, Learning Elastic Constitutive Material and Damping Models, in: Computer Graphics Forum, Wiley Online Library, 2020, pp. 81-91.
[25] J. Brownlee, A Gentle Introduction to Batch Normalization for Deep Neural Networks, Machine Learning Mastery [online], 16 (2019).
[26] A. Choromanska, M. Henaff, M. Mathieu, G.B. Arous, Y. LeCun, The loss surfaces of multilayer networks, in: Artificial intelligence and statistics, PMLR, 2015, pp. 192-204.
[27] K. Janocha, W.M. Czarnecki, On loss functions for deep neural networks in classification, arXiv preprint arXiv:1702.05659, (2017).
[28] P. Cardiff, Solid mechanics and fluid-solid interaction using the solids4foam toolbox.

[29] P. Cardiff, A. Karač, P. De Jaeger, H. Jasak, J. Nagy, A. Ivanković, Ž. Tuković, An open-source finite volume toolbox for solid mechanics and fluid-solid interaction simulations, arXiv preprint arXiv:1808.10736, (2018).
[30] D.P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980, (2014).
[31] N. Ketkar, Stochastic gradient descent, in: Deep learning with Python, Springer, 2017, pp. 113-132.
[32] V. Bushaev, Adam—latest trends in deep learning optimization, Towards Data Science, Listopad, (2018).
[33] A. Griewank, On automatic differentiation, Mathematical Programming: recent developments and applications, 6 (1989) 83-107.
[34] F. Moukalled, L. Mangani, M. Darwish, The finite volume method in computational fluid dynamics, Springer, 2016.
[35] H. Jasak, H. Weller, Application of the finite volume method and unstructured meshes to linear elasticity, International journal for numerical methods in engineering, 48 (2000) 267-287.
[36] P. Cardiff, Z. Tukovic, H. Jasak, A. Ivankovic, A block-coupled finite volume methodology for linear elasticity, in: 9th OpenFOAM Workshop, 2014.