MeshingNet3D: Efficient Generation of Adapted Tetrahedral Meshes for Computational Mechanics

Zheyan Zhang, Peter K. Jimack, He Wang
School of Computing, University of Leeds, UK

Abstract

We describe a new algorithm for the generation of high quality tetrahedral meshes using artificial neural networks. The goal is to generate close-to-optimal meshes in the sense that the error in the computed finite element (FE) solution (for a target system of partial differential equations (PDEs)) is as small as it could be for a prescribed number of nodes or elements in the mesh. In this paper we illustrate and investigate our proposed approach by considering the equations of linear elasticity, solved on a variety of three-dimensional geometries. This class of PDE is selected due to its equivalence to an energy minimization problem, which therefore allows a quantitative measure of the relative accuracy of different meshes (by comparing the energy associated with the respective FE solutions on these meshes). Once the algorithm has been introduced it is evaluated on a variety of test problems, each with its own distinctive features and geometric constraints, in order to demonstrate its effectiveness and computational efficiency.

Keywords: Optimal mesh generation, Finite element methods, Machine learning, Artificial neural networks

1. Introduction

The finite element method (FEM) is one of the most widely used approaches for solving systems of partial differential equations (PDEs), which arise across multiple applications in computational mechanics [1, 2]. The key feature in determining the efficiency of the FEM on any given problem is the quality of the mesh: in general terms, the finer the mesh the better the solution but the greater the computational cost of obtaining it. This trade-off has led to a vast body of research into the generation of high-quality FE meshes over decades. Typically, the objective is either to generate a mesh for which the corresponding FE solution has a prescribed accuracy using a minimal number of degrees of freedom (e.g. [3]), or to generate the best possible mesh for a predetermined number of degrees of freedom (e.g. [4]). In this paper we focus primarily on the latter, however the two approaches are very closely related.

Interest in the use of data driven methods to obtain solutions of PDEs has grown significantly in recent years, largely due to the increase in computing power that supports the application of deep neural networks (NNs) [5, 6]. In this work however, we do *not* aim to apply NNs to estimate PDE solutions directly: instead we consider their use to estimate optimal meshes on which to compute traditional FE approximations. The rationale for this is our hypothesis that, for a given approximation error, a larger representation error can be tolerated in a NN to estimate the FE meshes than for a NN to estimate a family of PDE solutions directly. We present a universal deep-learning-based mesh generation system, *MeshingNet3D*, that extends our initial 2D ideas, [7], by building upon classical *a posteriori* error estimation techniques and adopting a new local coordinate system. Consequently, *MeshingNet3D* is able to guide non-uniform mesh generation for a wide range of PDE systems with rich variations of geometries, boundary conditions and PDE parameters.

In the remainder of this section we provide brief overviews of classical non-uniform mesh generation methods, artificial neural networks and mean value coordinates (which are core to the generality of our algorithm). Key areas of related research are also highlighted. Section 2 then describes our methodology in full, whilst Section 3 provides detailed validation and testing. The paper concludes with a discussion of our findings and of the outlook for further developments.

1.1. Non-uniform mesh generation

When applying the FEM to approximate the solution of a computational mechanics problem, it is necessary to define both the type of elements and the computational mesh upon which the approximation is sought. The simplest elements are piecewise linear functions on simplexes (triangles in 2D and tetrahedra in 3D) however other choices are widely used. In 3D, these include higher order Lagrange elements (also defined on tetrahedra), tri-linear and triquadratic elements (defined on octahedra) and more general elements associated with discontinuous Galerkin methods, which may be applied on hybrid meshes [8]. In this paper we restrict our consideration to unstructured tetrahedral meshes [9, 10]. Structured meshes of octahedra do have some advantages, such as requiring less memory, however they are less flexible when considering complex geometries or when targeting highly non-uniform meshes, with optimal approximation properties, which is the goal of this work.

When the volumes of the elements in a given mesh are approximately equal, and the aspect ratio of each tetrahedron is bounded by a small constant, we refer to the mesh as being *uniform*. Theoretical results about the asymptotic convergence of the FEM typically hold for sequences of finer and finer uniform meshes [11]. For many problems such meshes are not the best choice however: since the error in the corresponding FE solution may be much greater on some elements than on others. In such cases it would usually be far better to have more elements in the "high error" regions and fewer elements in the "low error" regions. The resulting mesh may have the same number of elements in total (with a non-uniform size distribution) but permit a much more accurate FE representation of the true solution. Ideally, we would like to identify an element size distribution to ensure that a prescribed global error tolerance can be obtained with the fewest possible number of elements [12]. In practice this is attempted through the use of prior knowledge to control the mesh size distribution (e.g. geometrical information or *a priori* analysis [13]), or through an iterative process based around *a posteriori* error estimates for intermediate solutions [3].

This iterative approach to mesh generation consists of three steps: (i) compute an FE solution on a coarse mesh; (ii) estimate the error locally throughout this solution; (iii) adapt the mesh based upon this estimate. At the next iteration these steps are repeated, beginning with the mesh produced in (iii).

There is a large body of work on the development of cheap and reliable *a posteriori* error estimators. Popular approaches include those which involve solving a set of local problems on each element, or on small patches of elements, to directly estimate the error function [14, 15], and those based upon the recovery of derivatives of the solution field by sampling at particular points and then interpolating with a higher degree polynomial [16, 17]. For example, in the context of linear elasticity problems, the elasticity energy density of a computed solution is evaluated at each element and the recovered energy density value at each vertex is defined to be the average of its adjacent elements. The local stress error is then proportional to the difference between the recovered piece-wise linear energy density and the original piece-wise constant values, [16]. Considering its wide application in engineering practice, this "ZZ error estimate" has been used as the baseline in our work, to generate comparative data (and meshes) from which *MeshingNet3D* will be trained, and against which it will be evaluated.

The third step in the iterative approach to mesh generation is to adapt the existing coarse mesh based upon the estimated local error distribution. This may be achieved through the creation of an entirely new mesh, with target element sizes guided by the local error estimate [9], or via local adaptivity. In the latter case the mesh can be moved locally (r-refinement) [4] and/or locally refined/coarsened (h-refinement) [18]. No matter the type of refinement, the iterative process will generally require multiple passes to obtain a high quality final mesh. This therefore becomes a time-consuming pre-processing step – which we seek to avoid in this work.

1.2. Deep neural networks

Artificial neural networks (ANNs) are used to approximate mappings between specified inputs and outputs. They achieve this through a composition that is loosely based upon the neurons in a biological brain: there are a number of layers of nodes which are connected in a predetermined manner, and each node combines inputs received from the previous layer to generate an output that is passed to the next layer (with the first layer representing the input vector and the last layer the output vector). A number of free parameters are associated with each node, defining the action of that node, and these are prescribed based upon the minimization of a chosen training loss function. This learning problem is therefore equivalent to a nonlinear, multivariate optimization. Furthermore, since the loss function is designed to be differentiable with respect to the network parameters, an ANN can be trained using gradient decent methods such as stochastic gradient descent [19].

In recent years, with the developments in parallel hardware, so-called deep neural networks (DNNs), with many layers and very large numbers of parameters, have been proven to be remarkably effective at high-level tasks such as object recognition [20]. Within computational science, DNNs have also been explored to solve ordinary differential equations (ODEs) and PDEs in both supervised [21, 22] and unsupervised [23] settings. In the latter cases the network parameters are evaluated based upon a residual minimization, rather than using a labelled training data set, as in the supervised case. In each approach however, whilst the results are very promising, it is difficult to obtain high accuracy in the solutions and it is currently not possible to provide any guarantees on the accuracy. Consequently, rather than solving PDEs directly, our focus in what follows is to use DNNs to provide an estimate of the optimal finite element mesh, with the goal of obtaining the most efficient possible finite element solution.

1.3. Mean value coordinates

An important feature of our algorithm is the use of mean value coordinates (MVCs). These are a generalization of the barycentric coordinate system for simplexes [24, 25], to polygons in 2D and polyhedra with triangular faces in 3D [26], whereby the coordinates of any point within the polygon/polyhedron may be expressed as a convex combination of the positions of the boundary vertices. Consequently, all interior points in the neighbourhood of an arbitrary boundary vertex have a high value of the corresponding MVC component. MVCs also have a number of properties that make them attractive choices as input parameters to a DNN, for example their local smoothness with respect to spatial variations, as well as being both scale and rotationally invariant.

1.4. Related work

As noted above, recent developments in DNNs have led to renewed interest in the application of machine learning (ML) to the direct solution of PDEs and PDE systems [27]. The majority of this research is based upon supervised learning strategies, such as [28], which requires the use of a conventional solver to generate training data. Once trained, the NN is able to solve problems of the same type much more quickly than the original solver. Recently there has also been a growth in interest in the development and application of unsupervised learning methods, which act as independent PDE solvers without the need to refer to external supervisory information. These have been investigated particularly in the context of physics-informed algorithms [29, 30] or those targeting high-dimensional PDEs, [5, 6]. However, the issue still remains that, whether solving computational mechanics problems directly via supervised or unsupervised learning, current capabilities do not provide any *a priori* guarantees of accuracy. Indeed, even when a such solution has been produced, it is not generally possible to estimate how accurate it is.

Some previous authors have also considered the application of ML to mesh-related problems, sharing our aim of enhancing traditional FE solvers rather than replacing them completely. Examples include mesh quality assessment [31, 32] and mesh partitioning algorithms, such as [33], to complement parallel distributed solvers. Research into mesh generation using ML has also been undertaken, both for pure shape representation [34, 35], and as the basis for an efficient finite element solver [36]. This latter approach is based upon a self-organizing network but is restricted to fitting (and optimizing) a mesh of a fixed topology to a prescribed geometry. In [37] a recurrent network is used to enhance the traditional iterative approach to mesh generation through the use of ML to control the mesh adaptivity step. They are able to show results that match the quality of iteratively refined meshes using conventional error estimates and refinement strategies. Whilst these works each replace some aspects of the conventional mesh generation step with a NN, none of them address the specific problem tackled in this paper, where we seek to generate a single, non-uniform, tetrahedral mesh that provides a pseudo-optimal finite element representation of the solution of an unseen problem.

In this context of using ML to guide high-quality non-uniform mesh generation there is relatively little prior research. In [38], for example, early knowledge-based approaches were considered, though with limited success. Time-dependent remeshing is studied by [39], where an NN is used to undertake time-series predictions that identify areas of greater (and less) refinement at different times, though on a domain with a simple geometry. In [40, 41] NNs were applied successfully to generate high quality finite element meshes for elliptic PDEs, however the input vectors are highly problem-dependent: requiring specific *a priori* knowledge of the geometries being considered. The challenge of using DNNs to generate psuedo-optimal FE meshes on quite general geometries was first considered in [7] for selected two-dimensional PDEs. This paper extends these ideas to problems in three dimensions, to consider PDE systems with rich variation in geometry, boundary conditions and material properties.

2. Methodology

The goal of this research is to develop a robust, and widely applicable, mesh generation procedure for the efficient FE solution of systems of elliptic PDEs. Our particular emphasis here is on the equations of linear elasticity, however the approach described in this section may equally be applied to any family of problems for which a reliable *a posteriori* local error estimator is available to support the training phase for our neural network. In the first subsection we provide an overview of our methodology, with further details on the software design, training data generation and the training of the deep network given in the following subsections.

2.1. Theory

We seek to automatically generate high quality FE meshes for arbitrary instances within a given family of PDE problems, where each instance is defined by the domain geometry G (from a predefined family of possible geometries), the PDE parameters M, and the applied boundary conditions B. For any given mesh, the corresponding FE solution is assumed to be a unique solution, for which we have available a means of determining the local error. This computed *a posteriori* error is also assumed to be unique, and provides a mechanism for determining a desired FE element size for each location within the domain. Consequently, in order to generate a pseudo-optimal FE mesh we seek to estimate a mapping F that represents an ideal spatial distribution of the FE element sizes:

$$F : X \to S \tag{1}$$

Here, X is the specified location in the domain and S is the target element size (for example average edge length) at X. Noting that we define each instance by its specific geometry, parameter values and boundary conditions, we may express this mapping more precisely as:

$$F : X \to S(G, B, M; X) \tag{2}$$

Our goal is to make use of offline training to create a neural network that is able to learn the mapping

$$F : G, B, M, X \to S \tag{3}$$

After training, the NN is able to predict a pseudo-optimal mesh-size distribution for unseen problems. Specifically, given G, B, M for any problem, and an arbitrary sample point X, the NN outputs a target element size at that sample point. This is precisely the information required by a 3D mesh generator in order to generate a non-uniform, unstructured finite element mesh.

2.2. Software and evaluation

In this paper we use *Tetgen* for tetrahedral mesh generation, [9], and *FreeFem++* to assemble and solve the corresponding global FE systems [42]. The input to *Tetgen* includes a *.poly* file containing vertices and edges of polygons that define the boundary of the computational domain. From this file *Tetgen* is able to generate a uniform mesh based upon a single parameter, indicating a constant target element size. To generate a non-uniform mesh, *Tetgen* reads a background mesh from *.b.ele*, *.b.node* and *.b.face* files, and an element size list file *.b.mtr*, that defines target element sizes corresponding to the vertices of the background mesh. Having defined a valid mesh, *FreeFem++* is able to solve variational problems that are user defined. To do this it executes a *.edp* script file containing information such as: how to import the mesh; what type of finite element to use, what the specific variational form is; and which solver to apply. In this paper all examples are based upon the use linear tetrahedral elements (as gerenated by *Tetgen*) and the Lamé solver (for the equations of linear elasticity).

To mass produce training problems a simple script has been produced that allows an appropriate *.poly* file to be generated for a given geometry G. Then, for each geometry this calls *FreeFem++* to obtain linear elasticity solutions for a range of material parameters (M) and boundary conditions (B). Note that *FreeFem++* not only solves the elasticity equations but also computes the total stored energy, which may be used to evaluate the quality of a given FE solution. This is because the underlying PDE system corresponds to an energy minimization problem, so the analytic solution minimizes the energy functional over all functions from the appropriate Sobolov space ($(H_E^1)^3$ in this case). On the other hand, the FE solution minimizes the energy over all functions in the space of piecewise linear functions on the given tetrahedral mesh. Since this is a subspace of $(H_E^1)^3$, the energy corresponding to the FE solution is always greater than the energy of the analytic solution. Therefore, the lower the energy associated with the computed FE solution the better the solution. Consequently, the quality of any given set of tetrahedral meshes, for a particular problem, may be ranked based upon the computed energy corresponding to the finite element solution on each mesh: the lower the energy the better the mesh. We will use this observation as part of the evaluation of our approach.

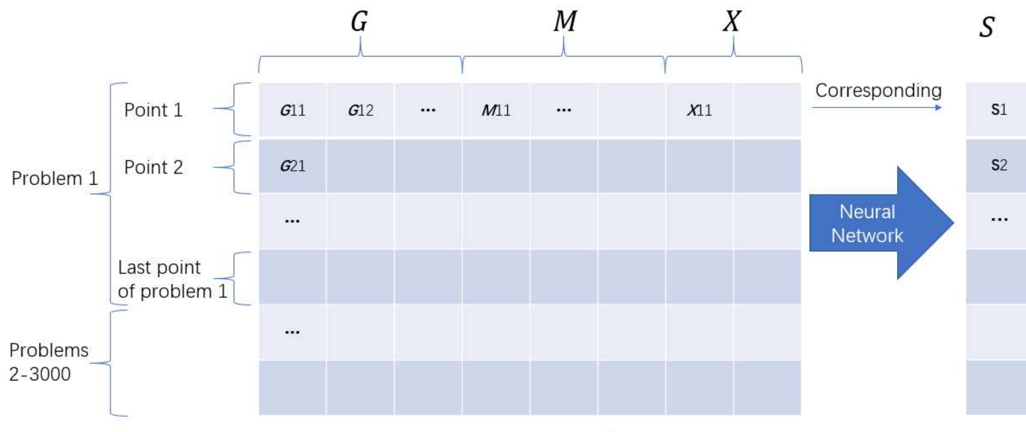

Figure 1: Illustration of the training data for *MeshingNet3D*: each individual problem is defined by the geometry G, the PDE parameters M and the boundary condition parameters B (not shown here). However for each such problem there are multiple sample points, X, in the domain, with the corresponding local mesh size S specified.

2.3. Data generation

Training data is required in order to sample the mapping of equation (2). Each training problem is defined by parameters that uniquely define the geometry (G), PDE parameters (M) and boundary conditions (B). For each such problem, multiple training data are generated by specifying numerous points, X, at which the target mesh size is given. This is illustrated in Figure 1: for which there are 3000 test problems, each of which generates multiple inputs, corresponding to different points X in the domain. The precise number of points X is problem-dependent which should be sufficient to represent the spatial mesh size variation throughout the domain (too many points will not decrease the training performance but will slow down the data generation). For each input the generated output, used to train the NN, is the target mesh size, S, for that point and that problem.

The value of S is computed using a variation of the iterative approach to mesh generation, based upon *a posteriori* error estimation, described in Subsection 1.1. For each problem we generate a relatively coarse uniform mesh and compute the corresponding FE solution and error estimate. In this work we use the "ZZ" energy estimate of [16]. However, for different problems or different quantities of interest, other choices are possible. For each sample point, X, the estimated local error, $E(X)$, can be converted to a target element size using an inverse relationship such as

$$S(X) = \frac{K}{E(X)} , \tag{4}$$

for some scaling coefficient K. This is the value of $S(X)$ used to define (2), as illustrated in Figure 1. The effect of the scaling coefficient is to control the total number of elements in the non-uniform mesh that is generated based upon the target local size distribution $S(X)$. Hence, for each test problem, K may be adjusted iteratively in order to obtain a target number of elements in the non-uniform mesh (or a target total error in the FE solution).

Note that the precise definition of the input vector in Figure 1 has to be problem dependent: a parameterization of the family of domains is required to define G; the number of free parameters in the PDE systems has to be predetermined; and the possible boundary conditions must also be parameterized. For each example shown in the Experiments section of this paper a different input vector has therefore been prescribed. Nevertheless, the methodology described here is shown to work robustly on all settings.

The final component required for the data generation is the algorithm to select the sample points X for each of the training problems. This is achieved via two steps: first an initial non-uniform mesh with predefined target element number is generated (e.g. by *Tetgen*) based upon the *a posteriori* error computed on the coarse uniform mesh; then we sample a fixed percentage of the elements of this mesh (we find that 10% is adequate), choosing each X to be the MVCs of the centroids of the sampled elements. Note that the advantage of sampling from the non-uniform, rather than the uniform, mesh is that the training data is weighted based upon the error distribution: our experiments show this to be advantageous.

2.4. Training and using the neural network

The deep learning platform that we use in this work is *Keras* [43] based on *Tensorflow* [44]. Our networks are fully connected, typically with six hidden layers, though we find that our results are not especially sensitive to the number of layers or the precise number of neurons per layer. We do observe however that it is advantageous to first increase and then decrease the number of neurons per layer as we pass forward through the network. The activation functions selected in this model are rectified linear functions [45] for the hidden units, with linear activation in the output layer. Before training, the input data is linearly normalised and 10% is selected for validation (monitoring the validation loss during training can help to identify and prevent over-fitting). The training itself uses mean square error loss and the stochastic gradient descent optimiser, *Adam*[46], with batch sizes of 128: for each of the examples considered in this paper this takes no longer than 3 hours on a Nvidia RTX 2070 graphics card.

Once trained, the NN can be used to guide mesh generation for unseen problems in real time. Given a new problem, defined by G, M and B, a uniform background mesh is generated based upon G alone. For each element in this background mesh we compute the MVC of its centre and concatenate this with the problem parameters to form an input vector for the NN. The corresponding output is the target element size at the centre of that element. The background mesh, with its associated target element size distribution, is then used to allow *TetGen* to generate the desired non-uniform mesh. If the total number of elements in this mesh is outside of the required range then each $S(X)$ may be scaled linearly before generating an updated non-uniform mesh. In this way, an adapted tetrahedral mesh of a specified size is generated directly, without the need to compute a sequence of FE solutions and *a posteriori* error estimates, as would otherwise be the case.

3. Computational Experiments

We present four computational tests which allow us to analyse the performance of *MeshingNet3D* across a range of different problems, geometries, boundary conditions and PDE parameters. The first and the third case involve prismatic geometries, which permit the description of spatial locations based upon "2.5D MVCs". These are composed of regular 2D MVCs in the x-y planes plus an additional z-coordinate. The second case uses general 3D Cartesian coordinates, whilst the final example uses general polyhedral geometries and fully 3D MVCs.

For each of the examples we provide a brief description of the problem, followed by a discussion of the network topology used (including the specific input vector) and the training undertaken. We then present results based upon 500 unseen test problems. These results compare the FE solutions computed on the NN-guided mesh with those computed on a "ground truth" mesh of similar size, generated using the same ZZ *a posteriori* error estimator that was applied to train the network. We also compare against the FE solution computed on a uniform mesh with a similar number of elements. To facilitate these comparisons, for each of the 500 test problems, we compute the difference between the total energy of the FE solution on the NN-guided mesh with that of the FE solution on the comparison mesh. We then provide a histogram to illustrate the proportion of the test cases in different binned error ranges. A negative value of the difference indicates that the solution on the NN-based mesh has a lower energy and is therefore superior.

3.1. Clamped beam

We consider the problem of an over-hanging beam (under gravity), with different cross sections (G) and variable boundary conditions (B). In this case the material parameters (M) are not varied (the specific inputs to the Lamé solver in *FreeFEM++* being: density = 8000, Young's modulus = 210×10^9 and Poisson's ratio = 0.27).

3.1.1. Problem specification

The beam is a right prism with a convex quadrilateral cross section as illustrated in Figure 2. This cross section has vertices at $(x_0, y_0) = (0, 0)$ and $(x_1, y_1) = (0, 2)$, and also at (x_2, y_2) and (x_3, y_3) which are randomly sampled within $x_2 \in (1.5, 2.5)$, $y_2 \in (1.5, 2.5)$, $x_3 \in (-0.5, 0.5)$ and $y_3 \in (1.5, 2.5)$ for each problem. The length of the beam is fixed $(0 \leq z \leq 6)$ and a boundary shear, with components $(f_x, f_y, 0)$, is applied at the face $z = 6$. The face $z = 0$ is clamped and the bottom face is clamped between $z = 0$ and $z = \zeta$, where $2 < \zeta < 4$ (randomly sampled for each problem). All other boundaries are free, subject to zero normal stress. Hence the input vector for this problem requires values for x_2, y_2, x_3, y_3, ζ, f_x and f_y, along with the MVCs of the point at which the mesh spacing is required. In these examples, the parameters f_x and f_y are constrained to lie in the range $(-10^6, 10^6)$.

3.1.2. Network information

In this example our fully-connected network has six hidden layers with 32, 64, 128, 64, 32 and 8 neurons respectively. Training data is generated based upon solving 3000 individual problems, each of which is obtained using a random choice for each input parameter (selected uniformly from its range), leading to $10,740,746$ individual input-output pairs. Of these, 10% are selected for validation and the remainder are used for training using a batch size of 128. The training takes 10 epochs, meaning that each item of data has been used an average of 10 times. Figure 13 shows the rates of convergence for the training, along with the corresponding validation curve.

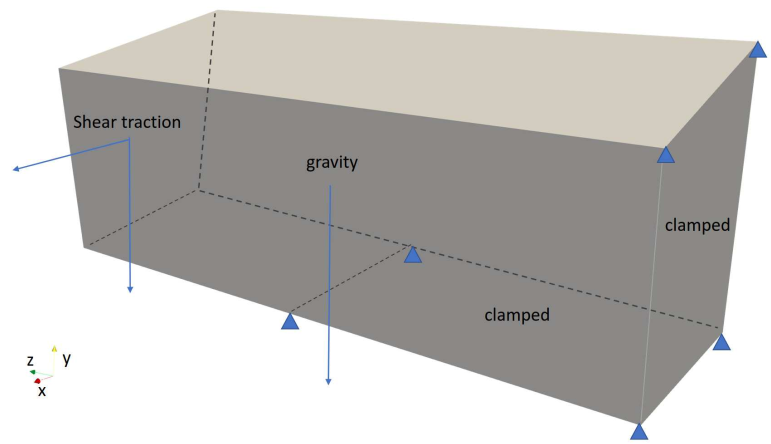

Figure 2: The geometry and boundary conditions for the *Clamped beam*, with constant cross section along the z-axis. The gravity is uniformly distributed over the volume. The surfaces bounded by four vertices with blue triangles are clamped.

3.1.3. Results

Figure 3 demonstrates that the NN-guided meshes generally perform at least as well as the ground truth meshes (generated from explicitly-computed *a posteriori* error estimates) and, as expected, much better than uniform meshes. Two typical examples are shown in Figure 4, which compares NN-guided meshes (bottom) with their ground-truth counterparts (top). In each case the high mesh density near $y = 0$ and $z = \zeta$ is easily captured. More significantly however, high and low mesh density regions are captured well throughout the domain, with a smooth variation between these regions.

3.2. Laminar material

In this example we consider a variation of the previous problem for which the material parameters (M) are now permitted to vary but the geometry (G) and the boundary conditions (B) are kept fixed.

3.2.1. Problem specification

A beam of dimensions $1 \times 1 \times 5$ is composed of two horizontal layers, as illustrated in Figure 5. Each layer has a Young's modulus (E_{top} and E_{bot}) between 10^9 and 10^{11}, and a Poisson's ratio (ν_{top} and ν_{bot}) between 0.05 and 0.45. The densities of the two materials are both 8000 and the interface between the layers is at a height $y = h \in (0.2, 0.8)$. Half of the bottom surface

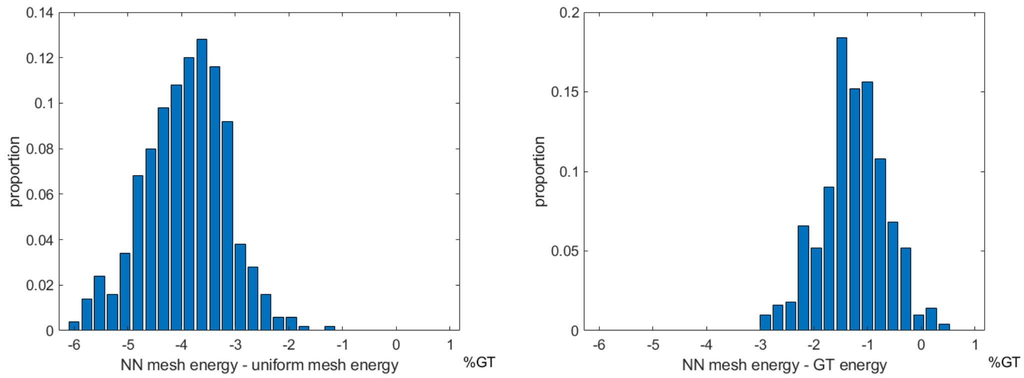

Figure 3: For the *Clamped beam*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy).

$(y = 0,\ 0 < z < 2.5)$ is clamped, as is the surface $z = 0$. On the surface $z = 5$ a traction of amplitude 10000 is applied in the x direction, with all other boundaries free to displace under zero normal-stress conditions. Hence the input vector for this problem requires values for E_{top}, E_{bot}, ν_{top}, ν_{bot} and h, along with the coordinates of the point at which the mesh spacing is required. We actually use $\log_{10}(E_{\text{top}})$ and $\log_{10}(E_{\text{bot}})$ as the first two input parameters.

3.2.2. Network information

In this example our fully-connected network has five hidden layers with 32, 64, 32, 16 and 8 neurons respectively. Training data is generated based upon solving 3000 individual problems, each of which is obtained using a random choice for each input parameter (selected uniformly from its range), leading to $19,719,750$ individual input-output pairs. Of these, 10% are selected for validation and the remainder are used for training using a batch size of 128. The training takes 15 epochs, and Figure 13 shows the rates of convergence for this training, along with the corresponding validation curve.

3.2.3. Results

Figure 6 demonstrates that, as in the previous example, the NN-guided meshes typically perform on a par with the ground truth meshes, and much better than uniform meshes. Two typical examples are shown in Figure 7:

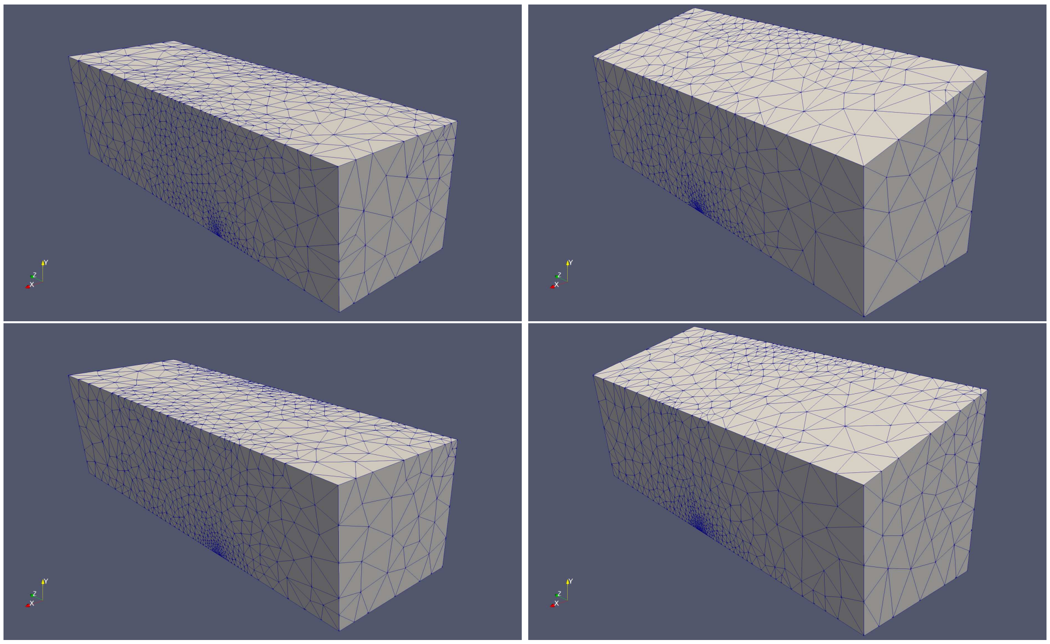

Figure 4: For the *Clamped beam*, ground truth meshes (top) and NN-guided meshes (bottom) for two test cases.

in the case (a) and (c)

$$(\log_{10}(E_{\text{top}}), \log_{10}(E_{\text{bot}}), \nu_{\text{top}}, \nu_{\text{bot}}, h) = (10.82, 9.17, 0.34, 0.20, 0.34)\ ,$$

and for (b) and (d)

$$(\log_{10}(E_{\text{top}}), \log_{10}(E_{\text{bot}}), \nu_{\text{top}}, \nu_{\text{bot}}, h) = (9.17, 10.33, 0.44, 0.21, 0.41)\ .$$

In the first example the top layer has the higher Young's modulus, which leads to a higher mesh density in this layer (for both the NN-guided and

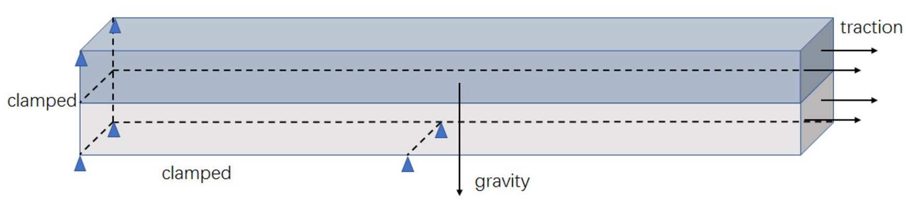

Figure 5: The boundary conditions and loads of the *laminar material* where the height of the interface is random

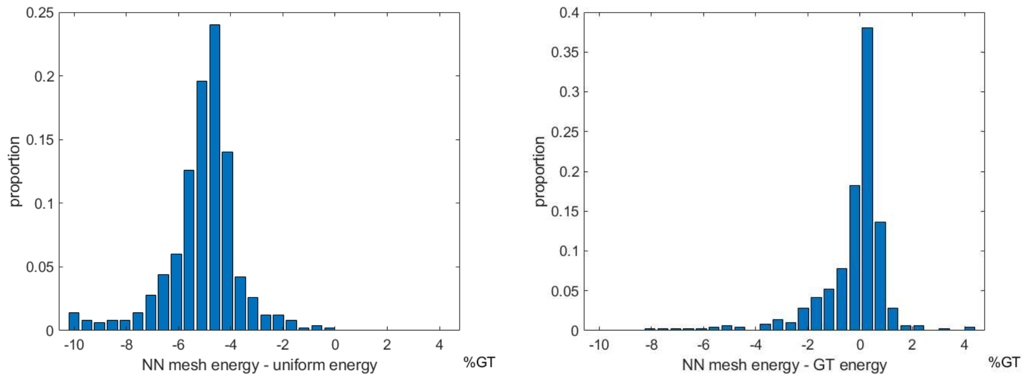

Figure 6: For the *Laminar material*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy).

the ground-truth meshes). Conversely, in the second example the bottom material is stiffer than the top and we see a very different distribution of the element size. In each case there is a strong correlation between the NN-guided mesh and the ground-truth case.

3.3. hex-bolt with a hole

We consider the problem of a hex-bolt (under torque), with different cross sections (G). In this case the material parameters (M) are not varied (the specific inputs to the Lamé solver in *FreeFEM++* being: density $= 8000$, Young's modulus $= 210 \times 10^9$ and Poisson's ratio $= 0.27$).

3.3.1. Problem specification

A regular hexagonal prism has an octagonal prism hole inside it where the height of the prism is $h = 4$ (Figure 8 left). On the cross section, the edge length of the regular hexagon is 4 and the octagon is coaxial with the hexagon. The eight vertices of the octagon lie on the same circle, whose radius varies $r \in (0.2, 1.0)$. The arc angles between vertices are random. Linear distributed pressures are applied to create a torque on the top ($p = -10000x + 10000$) and bottom ($p = -10000x - 10000$) surfaces. The eight surfaces of the hole are clamped. The input vectors for this problem include the position of the octagon's eight vertices and the MVCs of the target point

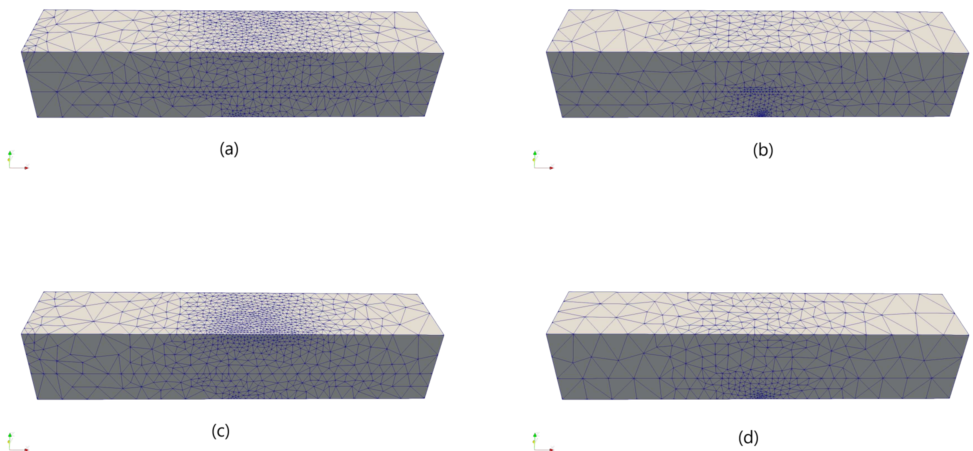

Figure 7: (a)(c) and (b)(d) are two problems in the *laminar material* experiments. (a) and (b) are ground truth meshes and (c) and (d) are non-uniform meshes guided by the neural network

expressed with respect to both the vertices of the outer hexagon and the inner octagon (combined with its z coordinate, $z \in (-1.0, 0.0)$).

3.3.2. Network information

In this example our fully-connected network has four hidden layers with 32, 64, 16 and 8 neurons respectively. Training data is generated based upon solving 3000 individual problems, each of which is obtained using a random choice for each input parameter (selected uniformly from its range), leading to $10,748,618$ individual input-output pairs. Of these, 10% are selected for validation and the remainder are used for training using a batch size of 128. The training takes 10 epochs and Figure 13 shows the convergence for this training, along with the corresponding validation curve.

3.3.3. Results

Figure 9 shows that the *MeshingNet3D* meshes are again better than uniform meshes and that the NN mesh energies are very close to those of the ground truth. As illustrated in Figure 10, the NN can successfully guide non-uniform mesh generation on very different geometries. This example also illustrates the success of the proposed approach on non-simply-connected

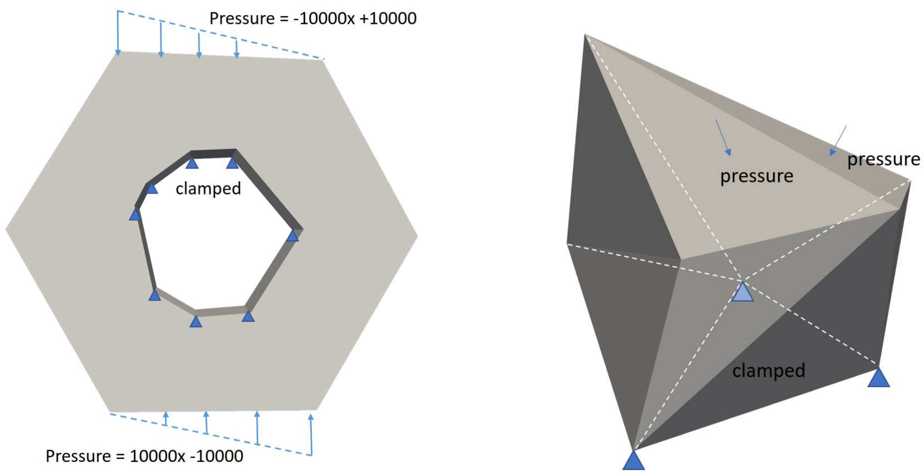

Figure 8: The boundary conditions and loads of the $hex - bolt$ (left) and *irregular polyhedron* (right). On $hex - bolt$, eight surfaces of the hole are clamped, linear distributed pressure is applied on top and bottom surfaces.

domains. Note that the second problem (on the right) in Figure 10 illustrates one of the worst performing cases for the NN mesh relative to the ground truth: here, the NN mesh is more uniform than the ground truth (though still a vast improvement on a standard uniform mesh).

3.4. Irregular polyhedron

We now consider the problem of mesh generation on arbitrary twelve-faced polyhedra, with a range of geometries (G) and variable boundary conditions (B). In this case the material parameters (M) are not varied (the specific inputs to the Lamé solver in *FreeFEM++* being: density = 8000, Young's modulus = 210×10^9 and Poisson's ratio = 0.27).

3.4.1. Problem specification

An irregular polyhedron with twelve triangular faces and eight vertices is illustrated in Fig 8 (right). The four "bottom" vertices are constrained to be co-planar and one of the two bottom triangular surfaces (i.e. the two triangles whose union is bounded by the four co-planar vertices) is clamped. In all training and testing problems the geometries are subject to the restriction that the four bottom vertices always lie in the same plane. A normal pressure of amplitude 10000 is applied on the two "top" surfaces (i.e. the triangular

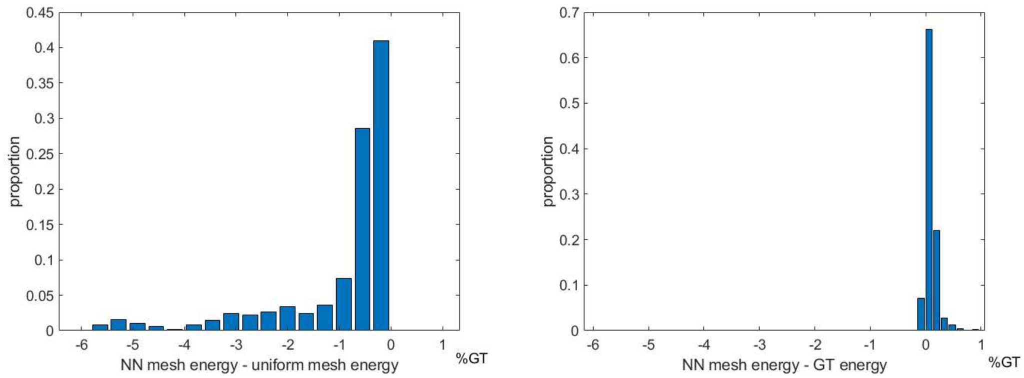

Figure 9: For $hex-bolt\ with\ a\ hole$, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy).

faces whose union is bounded by the other four vertices) and zero normal stress is applied on the other nine triangular faces. The input vectors for this problem define the Cartesian coordinates of the eight vertices and the corresponding MVCs of the point at which the mesh spacing is required.

3.4.2. Network information

In this example our fully-connected network has four hidden layers with 32, 64, 32, 16, and 8 neurons respectively. Training data is generated based upon solving 3000 individual problems, each of which is obtained using a random choice for each input parameter, leading to $7,383,999$ individual input-output pairs. Of these, 10% are selected for validation and the remainder are used for training using a batch size of 128. We use the network after training 10 epochs and Figure 13 shows the convergence for this training, along with the corresponding validation curve.

3.4.3. results

From Figure 11 it is cleear that the *MeshingNet3D* meshes are significantly better than uniform meshes and that the solution energies are relatively close to those of the ground truth: though in some cases the ground truth mesh is slightly superior. One such example is shown in Figure 12 (three views of the same problem), where we see that the NN mesh appears to be more conservative in some aspects of its local refinement. Nevertheless, even in this worst-case scenario, the *MeshingNet3D* mesh generally has the same regions of refinement as the ground truth mesh.

3.5. Discussion

Across the four experiments described in this section we have shown results over a range of geometries, boundary conditions and material parameters. For each problem the input layer of the NN is necessarily of a different dimension, which is dependent on the problem specification (along with the MVCs of the target point), whereas the output is always a single value representing the predicted mesh spacing at the target point. The number and size of the hidden layers is not a critical choice, but does naturally have some impact on the performance of the network.

As an example, to illustrate this, Table 1 shows the performance of five different networks when applied to the fourth of the test problems above. In each case the networks have been trained on the same data set, with validation losses having converged after 10 epochs. The networks are then used to compute meshes on the same testing set of 500 unseen problems and the finite element solutions computed on all meshes. The energy of each solution is normalised against the energy of the finite element solution computed on the "ground truth" mesh so as to allow a meaningful average to be taken across all 500 cases. This is the value shown in the "normalised average energy" column of Table 1: so, the lower this energy the better the meshes are on average. The results shown in Subsection 3.4 are generated using NN3 from the table but NN2 and NN4 produced meshes of very similar quality. The network denoted by NN1 appears to have too few degrees of freedom to be able to model the non-uniform mesh patterns satisfactorily, whereas the network denoted by NN5 likely has too many degrees of freedom for the size of our training data set.

Note that our NNs are always "spindly", with the greatest number of neurons in the inner layers. We find from experiment that this kind of network appears to have the best performance for the set of tasks considered in this work. Given that our problems have a relatively small number of inputs and a very small number of outputs (typically one) this is perhaps not surprising: to capture the highly nonlinear relationships between the inputs and the mesh spacing across the domain, significant complexity must be introduced into the network between the input and output layers.

Finally, we note that *MeshingNet3D* has the potential to make simulations more efficient for designers who use pre-built 3D models provided

NN	NN structure	training epochs	normalised average energy
NN1	32-16-8	10	$9.0 \times 10^{-}3$
NN2	32-64-16-8	10	$8.1 \times 10^{-}3$
NN3	32-64-32-16-8	10	$7.9 \times 10^{-}3$
NN4	32-64-128-32-16-8	10	$8.1 \times 10^{-}3$
NN5	32-64-128-64-32-16-8	10	$8.6 \times 10^{-}3$

Table 1: Comparison of 5 different fully connected NNs based upon normalised average energies of the finite element solutions. NN3 gives the lowest average energy and therefore provides the best mean performance.

within Computer Aided Design (CAD) software to accelerate design. From screws and bolts, to washers and bearings, CAD can not only define geometries but also materials. Embedding pre-trained *MeshingNet3D* in these CAD libraries could save meshing cost and provide high-quality non-uniform meshes. Similarity, *MeshingNet3D* can help parametric design where the NN is pre-trained for each geometry topology: under the guidance of the NN an appropriate mesh is generated in response to each iteration of the design. To implement this efficiently the challenge will be in defining a suitable family of boundary conditions as NN inputs, where forces due to interacting objects are unknown *a priori*. However, for components in a specific assembly, if contacts are defined, the load may be inferred by data-driven methods.

4. Conclusions

We have proposed a new framework for the generation of non-uniform three-dimensional finite element meshes. This is designed to produce meshes of the same quality as those obtained using traditional approaches, based upon *a posterori* error estimates and local mesh refinement, but at a substantially reduced computational cost. This has been implemented as *MeshingNet3D*, building upon the 3D mesh generator *Tetgen* and the finite element package *FreeFem++*. By selecting the linear elasticity solver within *FreeFem++* we have been able to undertake quantitative comparisons of different meshes based upon the energy minimization property of the elastostatic equations. Specifically, we can compare any two meshes by solving the finite element system on each mesh and then computing the stored energy of the solutions: the lower one being superior.

We have assessed the performance of *MeshingNet3D* on four different problem families for which the optimal finite element mesh is generally highly non-uniform. In all cases we are able to demonstrate the capability to generate meshes which are not only substantially better than uniform meshes for the same geometry, but which are comparable in quality to non-uniform meshes that are generated based upon the traditional (and expensive) approach of undertaking a sequence of local adaptive steps involving finite element solves and *a posteriori* error estimates. Perhaps not surprisingly, the benefits of *MeshingNet3D* are most apparent on those problems for which the optimal finite element mesh is far from uniform.

The main limitation of our approach is associated with the need to define a different set of inputs for each family of problems that is to be considered. Hence, for each new family of problems being considered, it is necessary to define a set of inputs that fully reflects the richness of that family, and then to undertake training for a new network. Furthermore, as with most supervised learning approaches, there is a trade-off to be made between the level of generality of the family of problems that the user of *MeshingNet3D* wishes to consider and the amount of work that must be undertaken in the training phase of the algorithm. Nevertheless, in situations where many solutions are required for large numbers of related problems (such as design and optimization problems for example) this is likely to be a worthwhile expense. Finally, we note that, in cases where engineers may have limited confidence in their ability to define the most appropriate inputs (to define the geometry or boundary conditions for example), data analysis techniques such as principle components analysis may be used to find the most critical parameters.

References

[1] P. M. Gresho, R. L. Sani, Incompressible flow and the finite element method. volume 1: Advection-diffusion and isothermal laminar flow (1998).

[2] O. C. Zienkiewicz, R. L. Taylor, The finite element method for solid and structural mechanics, Elsevier, 2005.

[3] R. Stevenson, Optimality of a standard adaptive finite element method, Foundations of Computational Mathematics 7 (2007) 245–269.

[4] R. Mahmood, P. K. Jimack, Locally optimal unstructured finite element meshes in 3 dimensions, Computers & structures 82 (2004) 2105–2116.

[5] E. Weinan, B. Yu, The deep ritz method: a deep learning-based numerical algorithm for solving variational problems, Communications in Mathematics and Statistics 6 (2018) 1–12.

[6] J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solving partial differential equations, Journal of computational physics 375 (2018) 1339–1364.

[7] Z. Zhang, Y. Wang, P. K. Jimack, H. Wang, Meshingnet: A new mesh generation method based on deep learning, arXiv preprint arXiv:2004.07016 (2020).

[8] J. Chan, Z. Wang, A. Modave, J.-F. Remacle, T. Warburton, Gpu-accelerated discontinuous galerkin methods on hybrid meshes, Journal of Computational Physics 318 (2016) 142–168.

[9] H. Si, Tetgen, a delaunay-based quality tetrahedral mesh generator, ACM Transactions on Mathematical Software (TOMS) 41 (2015) 11.

[10] C. Geuzaine, J.-F. Remacle, Gmsh: A 3-d finite element mesh generator with built-in pre-and post-processing facilities, International journal for numerical methods in engineering 79 (2009) 1309–1331.

[11] G. Strang, G. J. Fix, An analysis of the finite element method (1973).

[12] W. Dörfler, A convergent adaptive algorithm for poisson's equation, SIAM Journal on Numerical Analysis 33 (1996) 1106–1124.

[13] T. Apel, O. Benedix, D. Sirch, B. Vexler, A priori mesh grading for an elliptic problem with dirac right-hand side, SIAM journal on numerical analysis 49 (2011) 992–1005.

[14] M. Ainsworth, J. T. Oden, A posteriori error estimation in finite element analysis, volume 37, John Wiley & Sons, 2011.

[15] R. E. Bank, A. Weiser, Some a posteriori error estimators for elliptic partial differential equations, Mathematics of computation 44 (1985) 283–301.

[16] O. C. Zienkiewicz, J. Z. Zhu, A simple error estimator and adaptive procedure for practical engineerng analysis, International journal for numerical methods in engineering 24 (1987) 337–357.

[17] O. C. Zienkiewicz, J. Z. Zhu, The superconvergent patch recovery and a posteriori error estimates. part 1: The recovery technique, International Journal for Numerical Methods in Engineering 33 (1992) 1331–1364.

[18] W. Speares, M. Berzins, A 3d unstructured mesh adaptation algorithm for time-dependent shock-dominated problems, International Journal for Numerical Methods in Fluids 25 (1997) 81–104.

[19] L. Bottou, Large-scale machine learning with stochastic gradient descent, in: Proceedings of COMPSTAT'2010, Springer, 2010, pp. 177–186.

[20] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in neural information processing systems, pp. 1097–1105.

[21] T. Q. Chen, Y. Rubanova, J. Bettencourt, D. K. Duvenaud, Neural ordinary differential equations, in: Advances in neural information processing systems, pp. 6571–6583.

[22] Z. Long, Y. Lu, X. Ma, B. Dong, Pde-net: Learning pdes from data, arXiv preprint arXiv:1710.09668 (2017).

[23] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial differential equations using deep learning, Proceedings of the National Academy of Sciences 115 (2018) 8505–8510.

[24] K. Hormann, M. S. Floater, Mean value coordinates for arbitrary planar polygons, ACM Transactions on Graphics (TOG) 25 (2006) 1424–1441.

[25] M. S. Floater, Mean value coordinates, Computer aided geometric design 20 (2003) 19–27.

[26] M. S. Floater, G. Kós, M. Reimers, Mean value coordinates in 3d, Computer Aided Geometric Design 22 (2005) 623–631.

[27] S. L. Brunton, B. R. Noack, P. Koumoutsakos, Machine learning for fluid mechanics, Annual Review of Fluid Mechanics 52 (2020) 477–508.

[28] W. Tang, T. Shan, X. Dang, M. Li, F. Yang, S. Xu, J. Wu, Study on a poisson's equation solver based on deep learning technique, in: 2017 IEEE Electrical Design of Advanced Packaging and Systems Symposium (EDAPS), IEEE, pp. 1–3.

[29] 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.

[30] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data, Computer Methods in Applied Mechanics and Engineering 361 (2020) 112732.

[31] S. Iqbal, G. F. Carey, Neural nets for mesh assessment, Technical Report, TEXAS UNIV AT AUSTIN, 2005.

[32] X. Chen, J. Liu, Y. Pang, J. Chen, L. Chi, C. Gong, Developing a new mesh quality evaluation method based on convolutional neural network, Engineering Applications of Computational Fluid Mechanics 14 (2020) 391–400.

[33] A. Bahreininejad, B. Topping, A. Khan, Finite element mesh partitioning using neural networks, Advances in Engineering Software 27 (1996) 103–115.

[34] Y. Feng, Y. Feng, H. You, X. Zhao, Y. Gao, Meshnet: Mesh neural network for 3d shape representation, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pp. 8279–8286.

[35] W. Yifan, N. Aigerman, V. G. Kim, S. Chaudhuri, O. Sorkine-Hornung, Neural cages for detail-preserving 3d deformations, in: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp. 75–83.

[36] L. Manevitz, M. Yousef, D. Givoli, Finite–element mesh generation using self–organizing neural networks, Computer-Aided Civil and Infrastructure Engineering 12 (1997) 233–250.

[37] J. Bohn, M. Feischl, Recurrent neural networks as optimal mesh refinement strategies, arXiv preprint arXiv:1909.04275 (2019).

[38] B. Dolšak, A. Jezernik, I. Bratko, A knowledge base for finite element mesh design, Artificial intelligence in engineering 9 (1994) 19–27.

[39] L. Manevitz, A. Bitar, D. Givoli, Neural network time series forecasting of finite-element mesh adaptation, Neurocomputing 63 (2005) 447–463.

[40] R. Chedid, N. Najjar, Automatic finite-element mesh generation using artificial neural networks-part i: Prediction of mesh density, IEEE Transactions on Magnetics 32 (1996) 5173–5178.

[41] D. Dyck, D. Lowther, S. McFee, Determining an approximate finite element mesh density using neural network techniques, IEEE transactions on magnetics 28 (1992) 1767–1770.

[42] F. Hecht, New development in freefem++, J. Numer. Math. 20 (2012) 251–265.

[43] F. Chollet, et al., Keras, https://keras.io, 2015.

[44] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: Large-scale machine learning on heterogeneous distributed systems, arXiv preprint arXiv:1603.04467 (2016).

[45] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltzmann machines, in: ICML.

[46] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization, arXiv preprint arXiv:1412.6980 (2014).

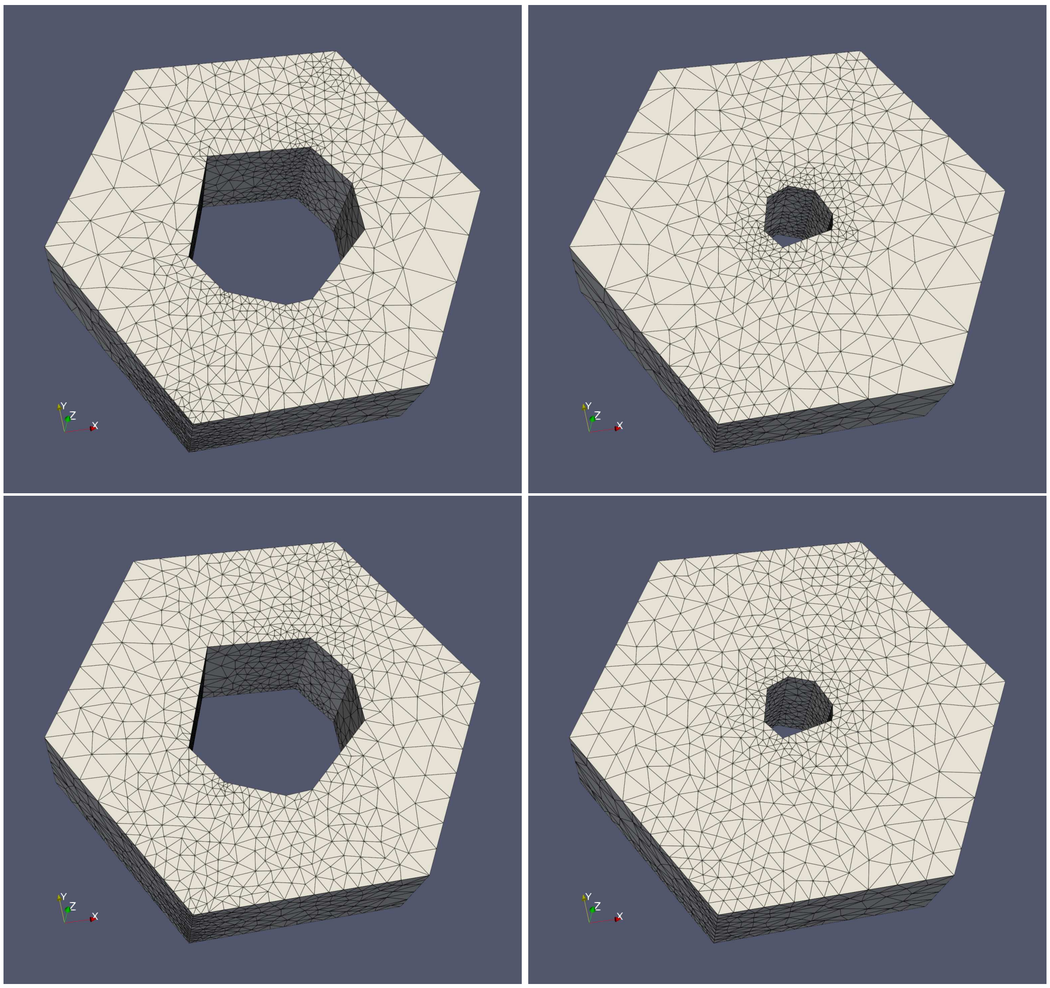

Figure 10: hex-bolt experiment, ground truth meshes (top) and NN meshes (bottom) , the left and right are two problems that only have different geometries

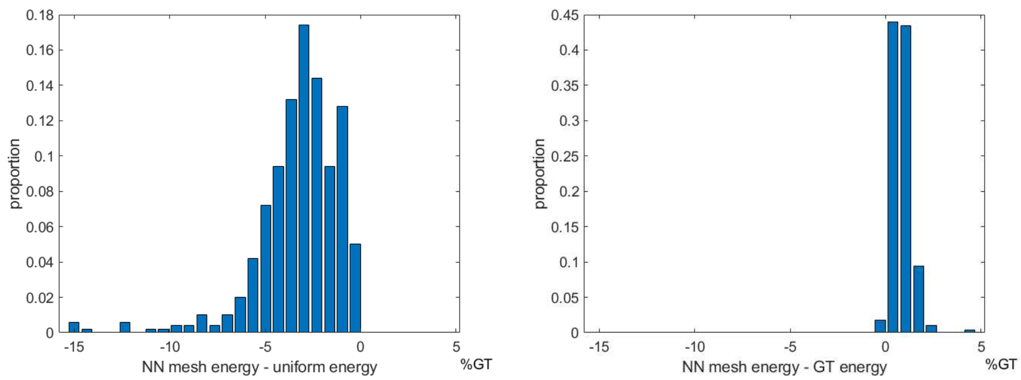

Figure 11: For *irregular polyhedron*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x-axis (as a percentage of the ground truth energy).

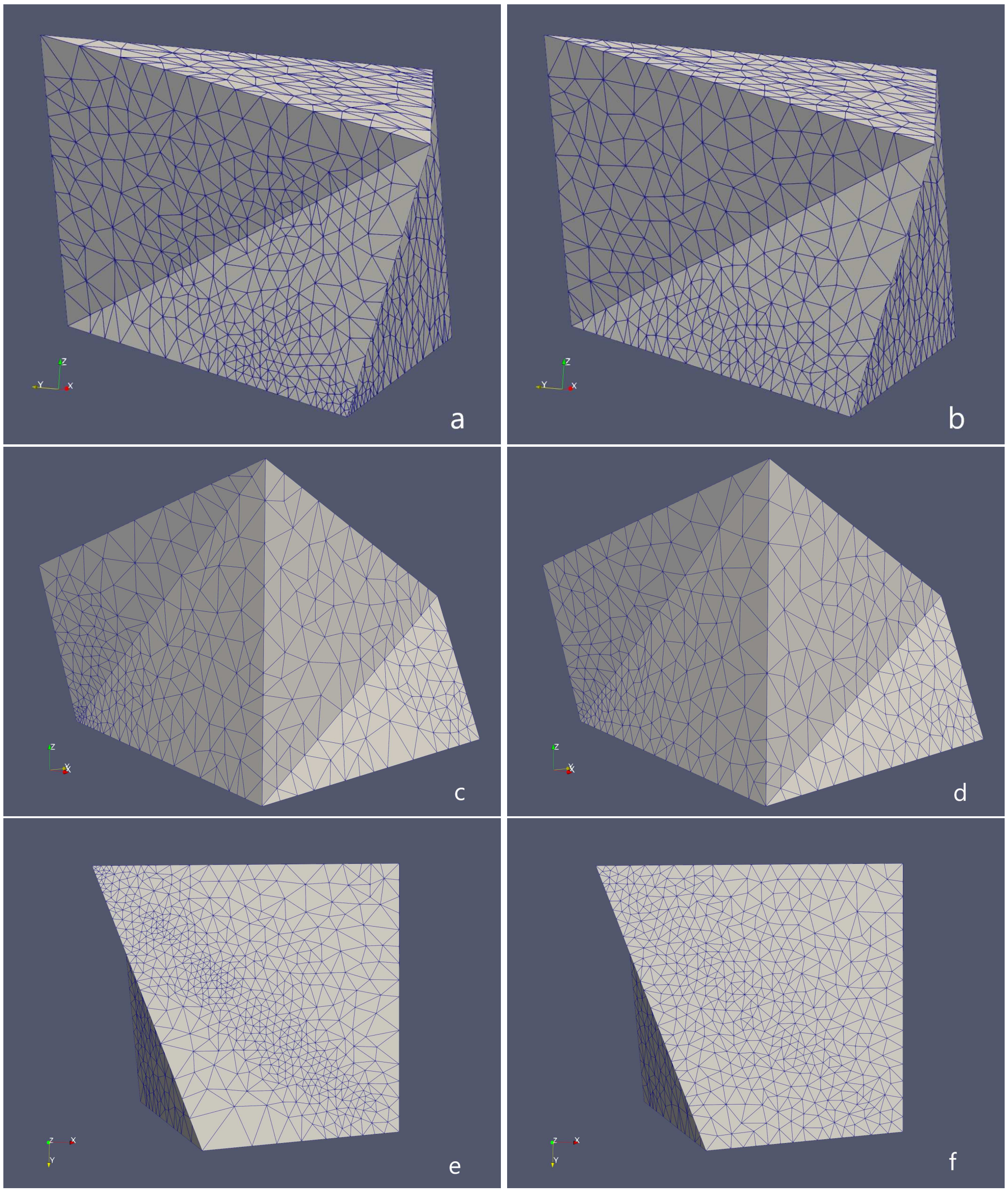

Figure 12: A ground truth mesh (a, c and e) and corresponding NN mesh (b, d and f) selected from 500 testing problems, they are in front (a and b), right (c and d) and bottom (e and f) views.

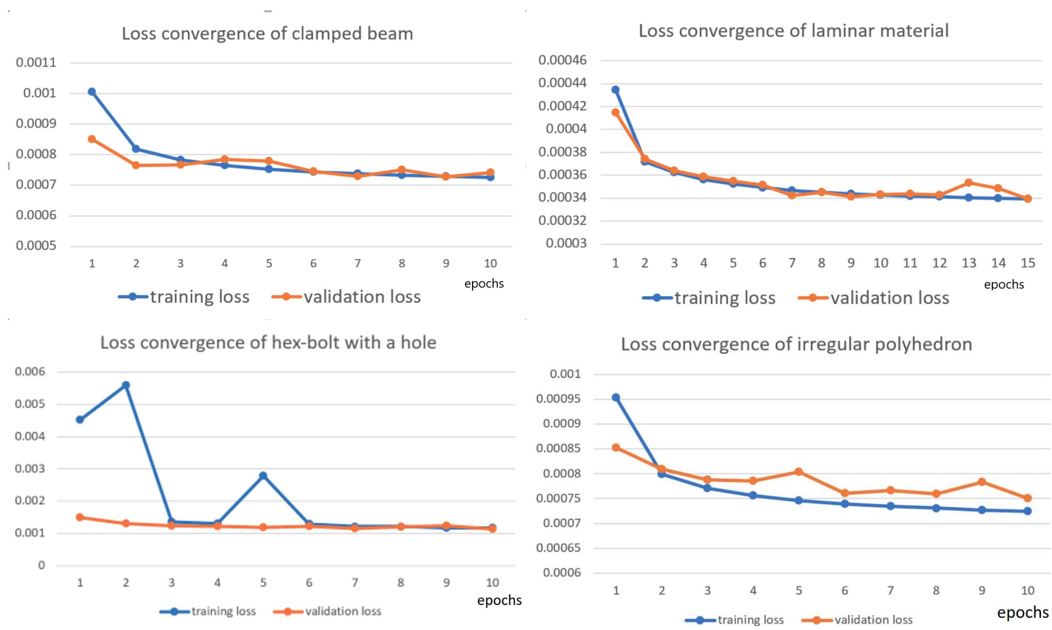

Figure 13: Training and validation loss of the four experiment