



Deposited via The University of Leeds.

White Rose Research Online URL for this paper:

<https://eprints.whiterose.ac.uk/id/eprint/173988/>

Version: Accepted Version

Article:

Zhang, Z, Jimack, PK and Wang, H (2021) MeshingNet3D: Efficient generation of adapted tetrahedral meshes for computational mechanics. *Advances in Engineering Software*, 157-158. 103021. ISSN: 0965-9978

<https://doi.org/10.1016/j.advengsoft.2021.103021>

© 2021 Elsevier Ltd. This manuscript version is made available under the CC-BY-NC-ND 4.0 license <http://creativecommons.org/licenses/by-nc-nd/4.0/>.

Reuse

This article is distributed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivs (CC BY-NC-ND) licence. This licence only allows you to download this work and share it with others as long as you credit the authors, but you can't change the article in any way or use it commercially. More information and the full terms of the licence here: <https://creativecommons.org/licenses/>

Takedown

If you consider content in White Rose Research Online to be in breach of UK law, please notify us by emailing eprints@whiterose.ac.uk including the URL of the record and the reason for the withdrawal request.

MeshingNet3D: Efficient Generation of Adapted Tetrahedral Meshes for Computational Mechanics

Zheyang 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

2 The finite element method (FEM) is one of the most widely used ap-
3 proaches for solving systems of partial differential equations (PDEs), which
4 arise across multiple applications in computational mechanics [1, 2]. The
5 key feature in determining the efficiency of the FEM on any given problem is
6 the quality of the mesh: in general terms, the finer the mesh the better the
7 solution but the greater the computational cost of obtaining it. This trade-
8 off has led to a vast body of research into the generation of high-quality FE

9 meshes over decades. Typically, the objective is either to generate a mesh
10 for which the corresponding FE solution has a prescribed accuracy using a
11 minimal number of degrees of freedom (e.g. [3]), or to generate the best
12 possible mesh for a predetermined number of degrees of freedom (e.g. [4]).
13 In this paper we focus primarily on the latter, however the two approaches
14 are very closely related.

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

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

36 1.1. *Non-uniform mesh generation*

37 When applying the FEM to approximate the solution of a computational
38 mechanics problem, it is necessary to define both the type of elements and
39 the computational mesh upon which the approximation is sought. The sim-
40 plest elements are piecewise linear functions on simplexes (triangles in 2D
41 and tetrahedra in 3D) however other choices are widely used. In 3D, these
42 include higher order Lagrange elements (also defined on tetrahedra), tri-
43 linear and triquadratic elements (defined on octahedra) and more general
44 elements associated with discontinuous Galerkin methods, which may be ap-
45 plied on hybrid meshes [8]. In this paper we restrict our consideration to

46 unstructured tetrahedral meshes [9, 10]. Structured meshes of octahedra do
47 have some advantages, such as requiring less memory, however they are less
48 flexible when considering complex geometries or when targeting highly non-
49 uniform meshes, with optimal approximation properties, which is the goal of
50 this work.

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

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

73 There is a large body of work on the development of cheap and reliable *a*
74 *posteriori* error estimators. Popular approaches include those which involve
75 solving a set of local problems on each element, or on small patches of ele-
76 ments, to directly estimate the error function [14, 15], and those based upon
77 the recovery of derivatives of the solution field by sampling at particular
78 points and then interpolating with a higher degree polynomial [16, 17]. For
79 example, in the context of linear elasticity problems, the elasticity energy
80 density of a computed solution is evaluated at each element and the recov-
81 ered energy density value at each vertex is defined to be the average of its
82 adjacent elements. The local stress error is then proportional to the differ-
83 ence between the recovered piece-wise linear energy density and the original

84 piece-wise constant values, [16]. Considering its wide application in engineer-
85 ing practice, this “ZZ error estimate” has been used as the baseline in our
86 work, to generate comparative data (and meshes) from which *MeshingNet3D*
87 will be trained, and against which it will be evaluated.

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

97 1.2. Deep neural networks

98 Artificial neural networks (ANNs) are used to approximate mappings be-
99 tween specified inputs and outputs. They achieve this through a composition
100 that is loosely based upon the neurons in a biological brain: there are a num-
101 ber of layers of nodes which are connected in a predetermined manner, and
102 each node combines inputs received from the previous layer to generate an
103 output that is passed to the next layer (with the first layer representing the
104 input vector and the last layer the output vector). A number of free pa-
105 rameters are associated with each node, defining the action of that node,
106 and these are prescribed based upon the minimization of a chosen training
107 loss function. This learning problem is therefore equivalent to a nonlinear,
108 multivariate optimization. Furthermore, since the loss function is designed
109 to be differentiable with respect to the network parameters, an ANN can be
110 trained using gradient decent methods such as stochastic gradient descent
111 [19].

112 In recent years, with the developments in parallel hardware, so-called
113 deep neural networks (DNNs), with many layers and very large numbers of
114 parameters, have been proven to be remarkably effective at high-level tasks
115 such as object recognition [20]. Within computational science, DNNs have
116 also been explored to solve ordinary differential equations (ODEs) and PDEs
117 in both supervised [21, 22] and unsupervised [23] settings. In the latter cases
118 the network parameters are evaluated based upon a residual minimization,
119 rather than using a labelled training data set, as in the supervised case. In
120 each approach however, whilst the results are very promising, it is difficult

121 to obtain high accuracy in the solutions and it is currently not possible to
122 provide any guarantees on the accuracy. Consequently, rather than solving
123 PDEs directly, our focus in what follows is to use DNNs to provide an esti-
124 mate of the optimal finite element mesh, with the goal of obtaining the most
125 efficient possible finite element solution.

126 1.3. Mean value coordinates

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

138 1.4. Related work

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

155 Some previous authors have also considered the application of ML to
156 mesh-related problems, sharing our aim of enhancing traditional FE solvers

157 rather than replacing them completely. Examples include mesh quality as-
158 sessment [31, 32] and mesh partitioning algorithms, such as [33], to comple-
159 ment parallel distributed solvers. Research into mesh generation using ML
160 has also been undertaken, both for pure shape representation [34, 35], and
161 as the basis for an efficient finite element solver [36]. This latter approach
162 is based upon a self-organizing network but is restricted to fitting (and op-
163 timizing) a mesh of a fixed topology to a prescribed geometry. In [37] a
164 recurrent network is used to enhance the traditional iterative approach to
165 mesh generation through the use of ML to control the mesh adaptivity step.
166 They are able to show results that match the quality of iteratively refined
167 meshes using conventional error estimates and refinement strategies. Whilst
168 these works each replace some aspects of the conventional mesh generation
169 step with a NN, none of them address the specific problem tackled in this
170 paper, where we seek to generate a single, non-uniform, tetrahedral mesh
171 that provides a pseudo-optimal finite element representation of the solution
172 of an unseen problem.

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

187 2. Methodology

188 The goal of this research is to develop a robust, and widely applicable,
189 mesh generation procedure for the efficient FE solution of systems of elliptic
190 PDEs. Our particular emphasis here is on the equations of linear elasticity,
191 however the approach described in this section may equally be applied to
192 any family of problems for which a reliable *a posteriori* local error estimator

193 is available to support the training phase for our neural network. In the first
194 subsection we provide an overview of our methodology, with further details
195 on the software design, training data generation and the training of the deep
196 network given in the following subsections.

197 2.1. Theory

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

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

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

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

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

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

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

221 *2.2. Software and evaluation*

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

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

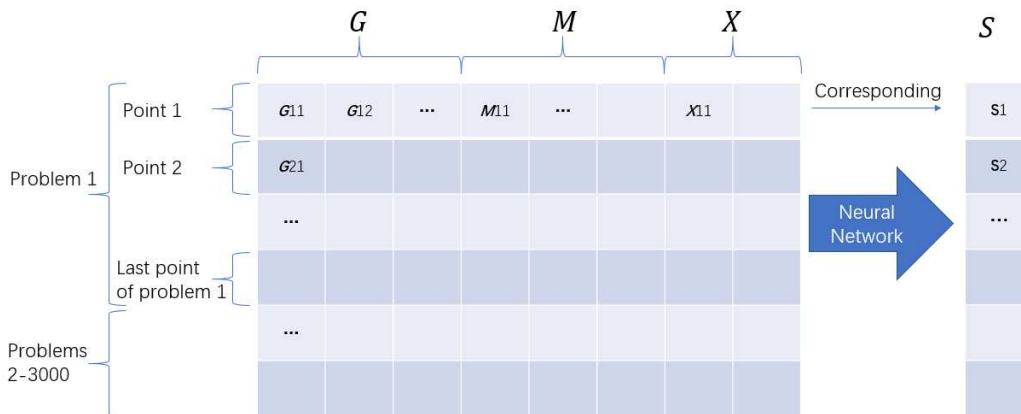


Figure 1: Illustration of the training data for *MeshingNet3D*: each individual problem is defined by the geometry G , the PDE parameters M and the boundary condition parameters B (not shown here). However for each such problem there are multiple sample points, X , in the domain, with the corresponding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

256 2.3. Data generation

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

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

276 a target element size using an inverse relationship such as

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

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

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

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

300 2.4. Training and using the neural network

301 The deep learning platform that we use in this work is *Keras* [43] based
302 on *Tensorflow* [44]. Our networks are fully connected, typically with six
303 hidden layers, though we find that our results are not especially sensitive to
304 the number of layers or the precise number of neurons per layer. We do ob-
305 serve however that it is advantageous to first increase and then decrease the
306 number of neurons per layer as we pass forward through the network. The
307 activation functions selected in this model are rectified linear functions [45]
308 for the hidden units, with linear activation in the output layer. Before train-
309 ing, the input data is linearly normalised and 10% is selected for validation

310 (monitoring the validation loss during training can help to identify and pre-
311 vent over-fitting). The training itself uses mean square error loss and the
312 stochastic gradient descent optimiser, *Adam*[46], with batch sizes of 128: for
313 each of the examples considered in this paper this takes no longer than 3
314 hours on a Nvidia RTX 2070 graphics card.

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

328 3. Computational Experiments

329 We present four computational tests which allow us to analyse the perfor-
330 mance of *MeshingNet3D* across a range of different problems, geometries,
331 boundary conditions and PDE parameters. The first and the third case in-
332 volve prismatic geometries, which permit the description of spatial locations
333 based upon “2.5D MVCs”. These are composed of regular 2D MVCs in the
334 x-y planes plus an additional z-coordinate. The second case uses general
335 3D Cartesian coordinates, whilst the final example uses general polyhedral
336 geometries and fully 3D MVCs.

337 For each of the examples we provide a brief description of the problem,
338 followed by a discussion of the network topology used (including the specific
339 input vector) and the training undertaken. We then present results based
340 upon 500 unseen test problems. These results compare the FE solutions
341 computed on the NN-guided mesh with those computed on a “ground truth”
342 mesh of similar size, generated using the same *ZZ a posteriori* error estimator
343 that was applied to train the network. We also compare against the FE
344 solution computed on a uniform mesh with a similar number of elements. To
345 facilitate these comparisons, for each of the 500 test problems, we compute

346 the difference between the total energy of the FE solution on the NN-guided
347 mesh with that of the FE solution on the comparison mesh. We then provide
348 a histogram to illustrate the proportion of the test cases in different binned
349 error ranges. A negative value of the difference indicates that the solution
350 on the NN-based mesh has a lower energy and is therefore superior.

351 3.1. Clamped beam

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

357 3.1.1. Problem specification

358 The beam is a right prism with a convex quadrilateral cross section as
359 illustrated in Figure 2. This cross section has vertices at $(x_0, y_0) = (0, 0)$ and
360 $(x_1, y_1) = (0, 2)$, and also at (x_2, y_2) and (x_3, y_3) which are randomly sampled
361 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
362 each problem. The length of the beam is fixed ($0 \leq z \leq 6$) and a boundary
363 shear, with components $(f_x, f_y, 0)$, is applied at the face $z = 6$. The face $z = 0$
364 is clamped and the bottom face is clamped between $z = 0$ and $z = \zeta$, where
365 $2 < \zeta < 4$ (randomly sampled for each problem). All other boundaries are
366 free, subject to zero normal stress. Hence the input vector for this problem
367 requires values for $x_2, y_2, x_3, y_3, \zeta, f_x$ and f_y , along with the MVCs of
368 the point at which the mesh spacing is required. In these examples, the
369 parameters f_x and f_y are constrained to lie in the range $(-10^6, 10^6)$.

370 3.1.2. Network information

371 In this example our fully-connected network has six hidden layers with
372 32, 64, 128, 64, 32 and 8 neurons respectively. Training data is generated
373 based upon solving 3000 individual problems, each of which is obtained us-
374 ing a random choice for each input parameter (selected uniformly from its
375 range), leading to 10,740,746 individual input-output pairs. Of these, 10%
376 are selected for validation and the remainder are used for training using a
377 batch size of 128. The training takes 10 epochs, meaning that each item of
378 data has been used an average of 10 times. Figure 13 shows the rates of
379 convergence for the training, along with the corresponding validation curve.

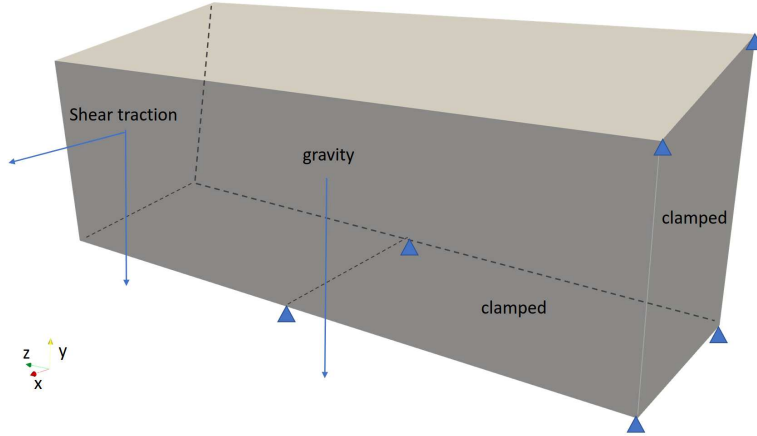


Figure 2: The geometry and boundary conditions for the *Clamped beam*, with constant cross section along the z -axis. The gravity is uniformly distributed over the volume. The surfaces bounded by four vertices with blue triangles are clamped.

380 3.1.3. Results

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

389 3.2. Laminar material

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

393 3.2.1. Problem specification

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

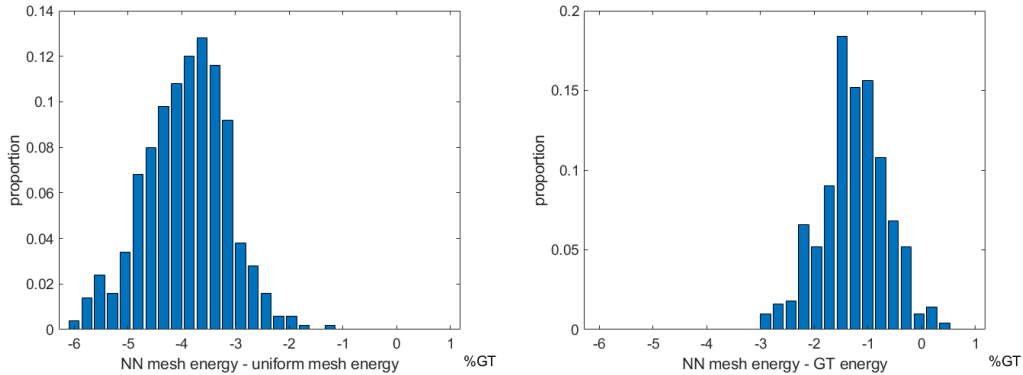


Figure 3: For the *Clamped beam*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x -axis (as a percentage of the ground truth energy).

399 ($y = 0$, $0 < z < 2.5$) is clamped, as is the surface $z = 0$. On the surface
 400 $z = 5$ a traction of amplitude 10000 is applied in the x direction, with all
 401 other boundaries free to displace under zero normal-stress conditions. Hence
 402 the input vector for this problem requires values for E_{top} , E_{bot} , ν_{top} , ν_{bot}
 403 and h , along with the coordinates of the point at which the mesh spacing
 404 is required. We actually use $\log_{10}(E_{\text{top}})$ and $\log_{10}(E_{\text{bot}})$ as the first two
 405 input parameters.

406 3.2.2. Network information

407 In this example our fully-connected network has five hidden layers with 32,
 408 64, 32, 16 and 8 neurons respectively. Training data is generated based upon
 409 solving 3000 individual problems, each of which is obtained using a random
 410 choice for each input parameter (selected uniformly from its range), leading
 411 to 19,719,750 individual input-output pairs. Of these, 10% are selected for
 412 validation and the remainder are used for training using a batch size of 128.
 413 The training takes 15 epochs, and Figure 13 shows the rates of convergence
 414 for this training, along with the corresponding validation curve.

415 3.2.3. Results

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

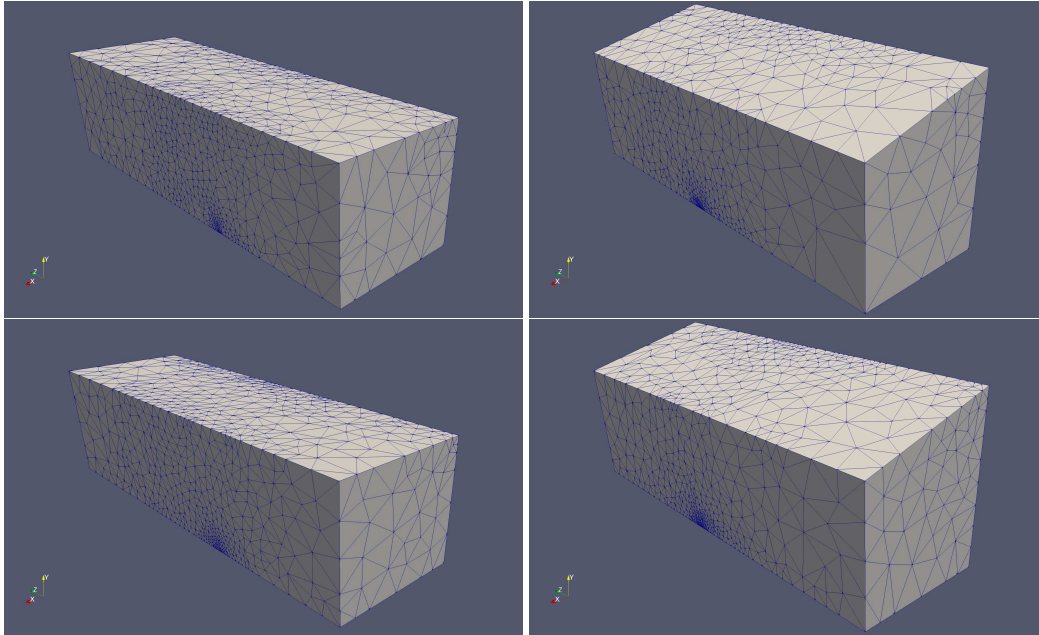


Figure 4: For the *Clamped beam*, ground truth meshes (top) and NN-guided meshes (bottom) for two test cases.

419 in the case (a) and (c)

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

420 and for (b) and (d)

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

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

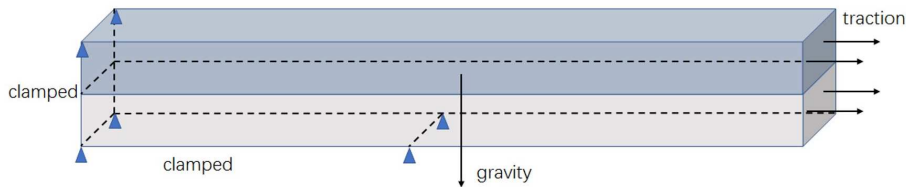


Figure 5: The boundary conditions and loads of the *laminar material* where the height of the interface is random

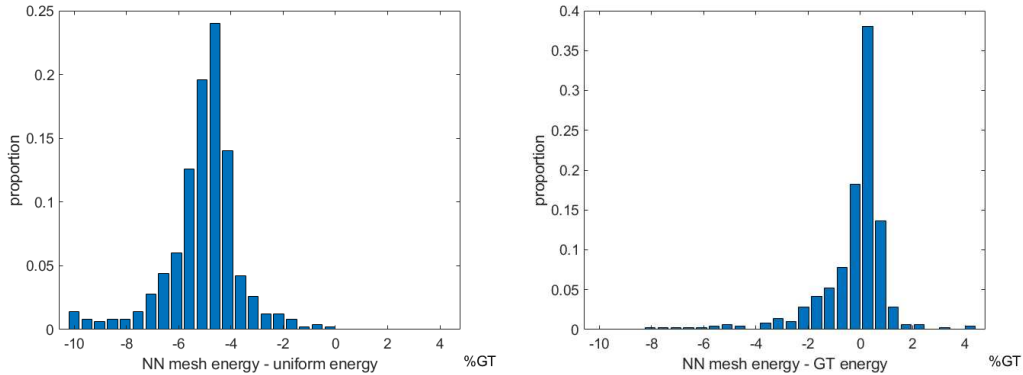


Figure 6: For the *Laminar material*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x -axis (as a percentage of the ground truth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

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

427 3.3. *hex-bolt with a hole*

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

432 3.3.1. *Problem specification*

433 A regular hexagonal prism has an octagonal prism hole inside it where
 434 the height of the prism is $h = 4$ (Figure 8 left). On the cross section, the
 435 edge length of the regular hexagon is 4 and the octagon is coaxial with the
 436 hexagon. The eight vertices of the octagon lie on the same circle, whose
 437 radius varies $r \in (0.2, 1.0)$. The arc angles between vertices are random.
 438 Linear distributed pressures are applied to create a torque on the top ($p =$
 439 $-10000x + 10000$) and bottom ($p = -10000x - 10000$) surfaces. The eight
 440 surfaces of the hole are clamped. The input vectors for this problem include
 441 the position of the octagon's eight vertices and the MVCs of the target point

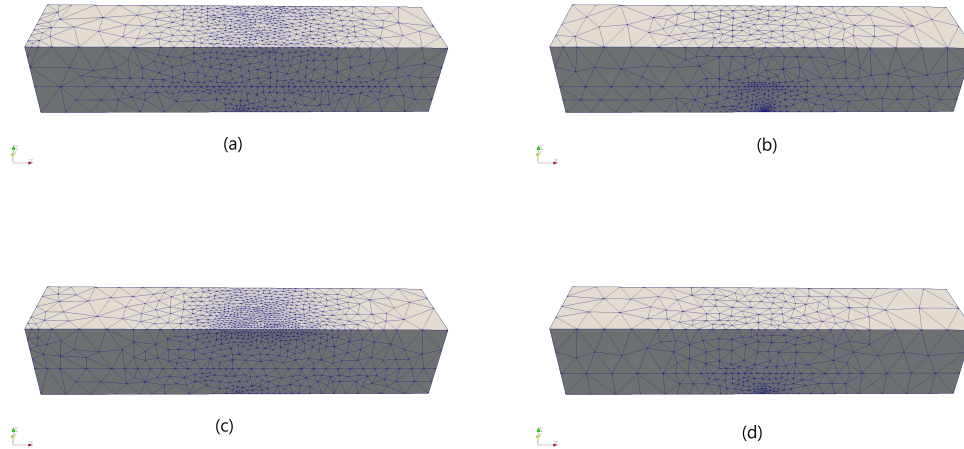


Figure 7: (a)(c) and (b)(d) are two problems in the *laminar material* experiments. (a) and (b) are ground truth meshes and (c) and (d) are non-uniform meshes guided by the neural network

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

444 3.3.2. Network information

445 In this example our fully-connected network has four hidden layers with
 446 32, 64, 16 and 8 neurons respectively. Training data is generated based upon
 447 solving 3000 individual problems, each of which is obtained using a random
 448 choice for each input parameter (selected uniformly from its range), leading
 449 to 10,748,618 individual input-output pairs. Of these, 10% are selected for
 450 validation and the remainder are used for training using a batch size of 128.
 451 The training takes 10 epochs and Figure 13 shows the convergence for this
 452 training, along with the corresponding validation curve.

453 3.3.3. Results

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

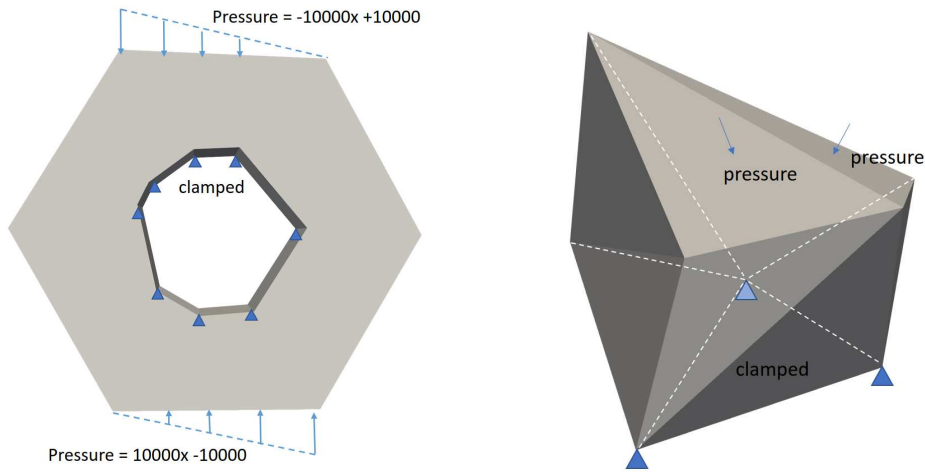


Figure 8: The boundary conditions and loads of the *hex – bolt* (left) and *irregular polyhedron* (right). On *hex – bolt*, eight surfaces of the hole are clamped, linear distributed pressure is applied on top and bottom surfaces.

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

463 3.4. Irregular polyhedron

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

469 3.4.1. Problem specification

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

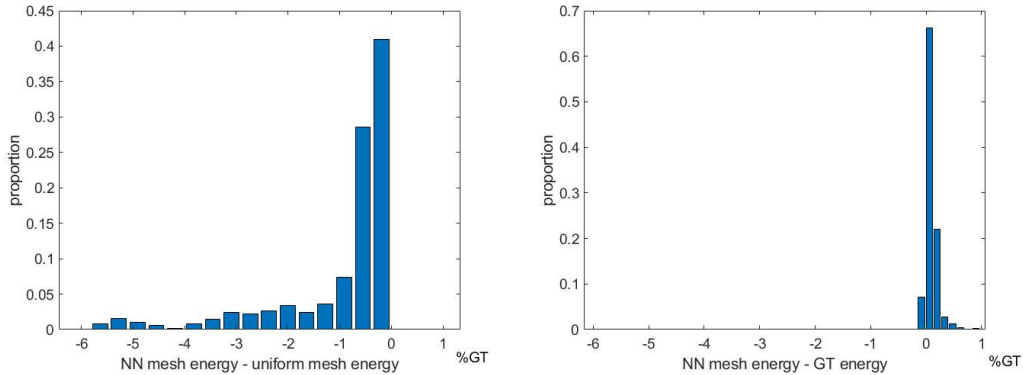


Figure 9: For *hex-bolt with a hole*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x -axis (as a percentage of the ground truth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

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

481 3.4.2. Network information

482 In this example our fully-connected network has four hidden layers with
 483 32, 64, 32, 16, and 8 neurons respectively. Training data is generated based
 484 upon solving 3000 individual problems, each of which is obtained using a ran-
 485 dom choice for each input parameter, leading to 7,383,999 individual input-
 486 output pairs. Of these, 10% are selected for validation and the remainder are
 487 used for training using a batch size of 128. We use the network after training
 488 10 epochs and Figure 13 shows the convergence for this training, along with
 489 the corresponding validation curve.

490 3.4.3. results

491 From Figure 11 it is clear that the *MeshingNet3D* meshes are signifi-
 492 cantly better than uniform meshes and that the solution energies are rela-
 493 tively close to those of the ground truth: though in some cases the ground
 494 truth mesh is slightly superior. One such example is shown in Figure 12
 495 (three views of the same problem), where we see that the NN mesh appears
 496 to be more conservative in some aspects of its local refinement. Nevertheless,

497 even in this worst-case scenario, the *MeshingNet3D* mesh generally has the
498 same regions of refinement as the ground truth mesh.

499 3.5. Discussion

500 Across the four experiments described in this section we have shown re-
501 sults over a range of geometries, boundary conditions and material paramete-
502 rs. For each problem the input layer of the NN is necessarily of a different
503 dimension, which is dependent on the problem specification (along with the
504 MVCs of the target point), whereas the output is always a single value rep-
505 resenting the predicted mesh spacing at the target point. The number and
506 size of the hidden layers is not a critical choice, but does naturally have some
507 impact on the performance of the network.

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

524 Note that our NNs are always “spindly”, with the greatest number of neu-
525 rons in the inner layers. We find from experiment that this kind of network
526 appears to have the best performance for the set of tasks considered in this
527 work. Given that our problems have a relatively small number of inputs and
528 a very small number of outputs (typically one) this is perhaps not surpris-
529 ing: to capture the highly nonlinear relationships between the inputs and the
530 mesh spacing across the domain, significant complexity must be introduced
531 into the network between the input and output layers.

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

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

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

545 4. Conclusions

546 We have proposed a new framework for the generation of non-uniform
547 three-dimensional finite element meshes. This is designed to produce meshes
548 of the same quality as those obtained using traditional approaches, based
549 upon *a posteriori* error estimates and local mesh refinement, but at a sub-
550 stantially reduced computational cost. This has been implemented as *Mesh-*
551 *ingNet3D*, building upon the 3D mesh generator *Tetgen* and the finite ele-
552 ment package *FreeFem++*. By selecting the linear elasticity solver within
553 *FreeFem++* we have been able to undertake quantitative comparisons of
554 different meshes based upon the energy minimization property of the elasto-
555 static equations. Specifically, we can compare any two meshes by solving the
556 finite element system on each mesh and then computing the stored energy of
557 the solutions: the lower one being superior.

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

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

584 **References**

- 585 [1] P. M. Gresho, R. L. Sani, Incompressible flow and the finite element
586 method. volume 1: Advection-diffusion and isothermal laminar flow
587 (1998).
- 588 [2] O. C. Zienkiewicz, R. L. Taylor, The finite element method for solid and
589 structural mechanics, Elsevier, 2005.
- 590 [3] R. Stevenson, Optimality of a standard adaptive finite element method,
591 Foundations of Computational Mathematics 7 (2007) 245–269.

- 592 [4] R. Mahmood, P. K. Jimack, Locally optimal unstructured finite element
593 meshes in 3 dimensions, *Computers & structures* 82 (2004) 2105–2116.
- 594 [5] E. Weinan, B. Yu, The deep ritz method: a deep learning-based nu-
595 merical algorithm for solving variational problems, *Communications in*
596 *Mathematics and Statistics* 6 (2018) 1–12.
- 597 [6] J. Sirignano, K. Spiliopoulos, Dgm: A deep learning algorithm for solv-
598 ing partial differential equations, *Journal of computational physics* 375
599 (2018) 1339–1364.
- 600 [7] Z. Zhang, Y. Wang, P. K. Jimack, H. Wang, Meshingnet: A new
601 mesh generation method based on deep learning, *arXiv preprint*
602 *arXiv:2004.07016* (2020).
- 603 [8] J. Chan, Z. Wang, A. Modave, J.-F. Remacle, T. Warburton, Gpu-
604 accelerated discontinuous galerkin methods on hybrid meshes, *Journal*
605 *of Computational Physics* 318 (2016) 142–168.
- 606 [9] H. Si, Tetgen, a delaunay-based quality tetrahedral mesh generator,
607 *ACM Transactions on Mathematical Software (TOMS)* 41 (2015) 11.
- 608 [10] C. Geuzaine, J.-F. Remacle, Gmsh: A 3-d finite element mesh generator
609 with built-in pre-and post-processing facilities, *International journal for*
610 *numerical methods in engineering* 79 (2009) 1309–1331.
- 611 [11] G. Strang, G. J. Fix, *An analysis of the finite element method* (1973).
- 612 [12] W. Dörfler, A convergent adaptive algorithm for poisson’s equation,
613 *SIAM Journal on Numerical Analysis* 33 (1996) 1106–1124.
- 614 [13] T. Apel, O. Benedix, D. Sirch, B. Vexler, A priori mesh grading for an
615 elliptic problem with dirac right-hand side, *SIAM journal on numerical*
616 *analysis* 49 (2011) 992–1005.
- 617 [14] M. Ainsworth, J. T. Oden, *A posteriori error estimation in finite element*
618 *analysis*, volume 37, John Wiley & Sons, 2011.
- 619 [15] R. E. Bank, A. Weiser, Some a posteriori error estimators for elliptic
620 partial differential equations, *Mathematics of computation* 44 (1985)
621 283–301.

- 622 [16] O. C. Zienkiewicz, J. Z. Zhu, A simple error estimator and adaptive
623 procedure for practical engineering analysis, *International journal for*
624 *numerical methods in engineering* 24 (1987) 337–357.
- 625 [17] O. C. Zienkiewicz, J. Z. Zhu, The superconvergent patch recovery and a
626 posteriori error estimates. part 1: The recovery technique, *International*
627 *Journal for Numerical Methods in Engineering* 33 (1992) 1331–1364.
- 628 [18] W. Speares, M. Berzins, A 3d unstructured mesh adaptation algorithm
629 for time-dependent shock-dominated problems, *International Journal*
630 *for Numerical Methods in Fluids* 25 (1997) 81–104.
- 631 [19] L. Bottou, Large-scale machine learning with stochastic gradient de-
632 scent, in: *Proceedings of COMPSTAT’2010*, Springer, 2010, pp. 177–
633 186.
- 634 [20] A. Krizhevsky, I. Sutskever, G. E. Hinton, Imagenet classification with
635 deep convolutional neural networks, in: *Advances in neural information*
636 *processing systems*, pp. 1097–1105.
- 637 [21] T. Q. Chen, Y. Rubanova, J. Bettencourt, D. K. Duvenaud, Neural
638 ordinary differential equations, in: *Advances in neural information pro-*
639 *cessing systems*, pp. 6571–6583.
- 640 [22] Z. Long, Y. Lu, X. Ma, B. Dong, Pde-net: Learning pdes from data,
641 *arXiv preprint arXiv:1710.09668* (2017).
- 642 [23] J. Han, A. Jentzen, E. Weinan, Solving high-dimensional partial dif-
643 ferential equations using deep learning, *Proceedings of the National*
644 *Academy of Sciences* 115 (2018) 8505–8510.
- 645 [24] K. Hormann, M. S. Floater, Mean value coordinates for arbitrary planar
646 polygons, *ACM Transactions on Graphics (TOG)* 25 (2006) 1424–1441.
- 647 [25] M. S. Floater, Mean value coordinates, *Computer aided geometric*
648 *design* 20 (2003) 19–27.
- 649 [26] M. S. Floater, G. Kós, M. Reimers, Mean value coordinates in 3d,
650 *Computer Aided Geometric Design* 22 (2005) 623–631.
- 651 [27] S. L. Brunton, B. R. Noack, P. Koumoutsakos, Machine learning for
652 fluid mechanics, *Annual Review of Fluid Mechanics* 52 (2020) 477–508.

- 653 [28] W. Tang, T. Shan, X. Dang, M. Li, F. Yang, S. Xu, J. Wu, Study on
654 a poisson's equation solver based on deep learning technique, in: 2017
655 IEEE Electrical Design of Advanced Packaging and Systems Symposium
656 (EDAPS), IEEE, pp. 1–3.
- 657 [29] M. Raissi, P. Perdikaris, G. E. Karniadakis, Physics-informed neural
658 networks: A deep learning framework for solving forward and inverse
659 problems involving nonlinear partial differential equations, *Journal of
660 Computational Physics* 378 (2019) 686–707.
- 661 [30] L. Sun, H. Gao, S. Pan, J.-X. Wang, Surrogate modeling for fluid flows
662 based on physics-constrained deep learning without simulation data,
663 *Computer Methods in Applied Mechanics and Engineering* 361 (2020)
664 112732.
- 665 [31] S. Iqbal, G. F. Carey, Neural nets for mesh assessment, Technical Re-
666 port, TEXAS UNIV AT AUSTIN, 2005.
- 667 [32] X. Chen, J. Liu, Y. Pang, J. Chen, L. Chi, C. Gong, Developing a new
668 mesh quality evaluation method based on convolutional neural network,
669 *Engineering Applications of Computational Fluid Mechanics* 14 (2020)
670 391–400.
- 671 [33] A. Bahreininejad, B. Topping, A. Khan, Finite element mesh partition-
672 ing using neural networks, *Advances in Engineering Software* 27 (1996)
673 103–115.
- 674 [34] Y. Feng, Y. Feng, H. You, X. Zhao, Y. Gao, Meshnet: Mesh neural
675 network for 3d shape representation, in: *Proceedings of the AAAI Con-
676 ference on Artificial Intelligence*, volume 33, pp. 8279–8286.
- 677 [35] W. Yifan, N. Aigerman, V. G. Kim, S. Chaudhuri, O. Sorkine-Hornung,
678 Neural cages for detail-preserving 3d deformations, in: *Proceedings of
679 the IEEE/CVF Conference on Computer Vision and Pattern Recogni-
680 tion*, pp. 75–83.
- 681 [36] L. Manevitz, M. Yousef, D. Givoli, Finite-element mesh generation
682 using self-organizing neural networks, *Computer-Aided Civil and In-
683 frastructure Engineering* 12 (1997) 233–250.

- 684 [37] J. Bohn, M. Feischl, Recurrent neural networks as optimal mesh refine-
685 ment strategies, arXiv preprint arXiv:1909.04275 (2019).
- 686 [38] B. Dolšak, A. Jezernik, I. Bratko, A knowledge base for finite element
687 mesh design, Artificial intelligence in engineering 9 (1994) 19–27.
- 688 [39] L. Manevitz, A. Bitar, D. Givoli, Neural network time series forecasting
689 of finite-element mesh adaptation, Neurocomputing 63 (2005) 447–463.
- 690 [40] R. Chedid, N. Najjar, Automatic finite-element mesh generation using
691 artificial neural networks-part i: Prediction of mesh density, IEEE
692 Transactions on Magnetics 32 (1996) 5173–5178.
- 693 [41] D. Dyck, D. Lowther, S. McFee, Determining an approximate finite ele-
694 ment mesh density using neural network techniques, IEEE transactions
695 on magnetics 28 (1992) 1767–1770.
- 696 [42] F. Hecht, New development in freefem++, J. Numer. Math. 20 (2012)
697 251–265.
- 698 [43] F. Chollet, et al., Keras, <https://keras.io>, 2015.
- 699 [44] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S.
700 Corrado, A. Davis, J. Dean, M. Devin, et al., Tensorflow: Large-scale
701 machine learning on heterogeneous distributed systems, arXiv preprint
702 arXiv:1603.04467 (2016).
- 703 [45] V. Nair, G. E. Hinton, Rectified linear units improve restricted boltz-
704 mann machines, in: ICML.
- 705 [46] D. P. Kingma, J. Ba, Adam: A method for stochastic optimization,
706 arXiv preprint arXiv:1412.6980 (2014).

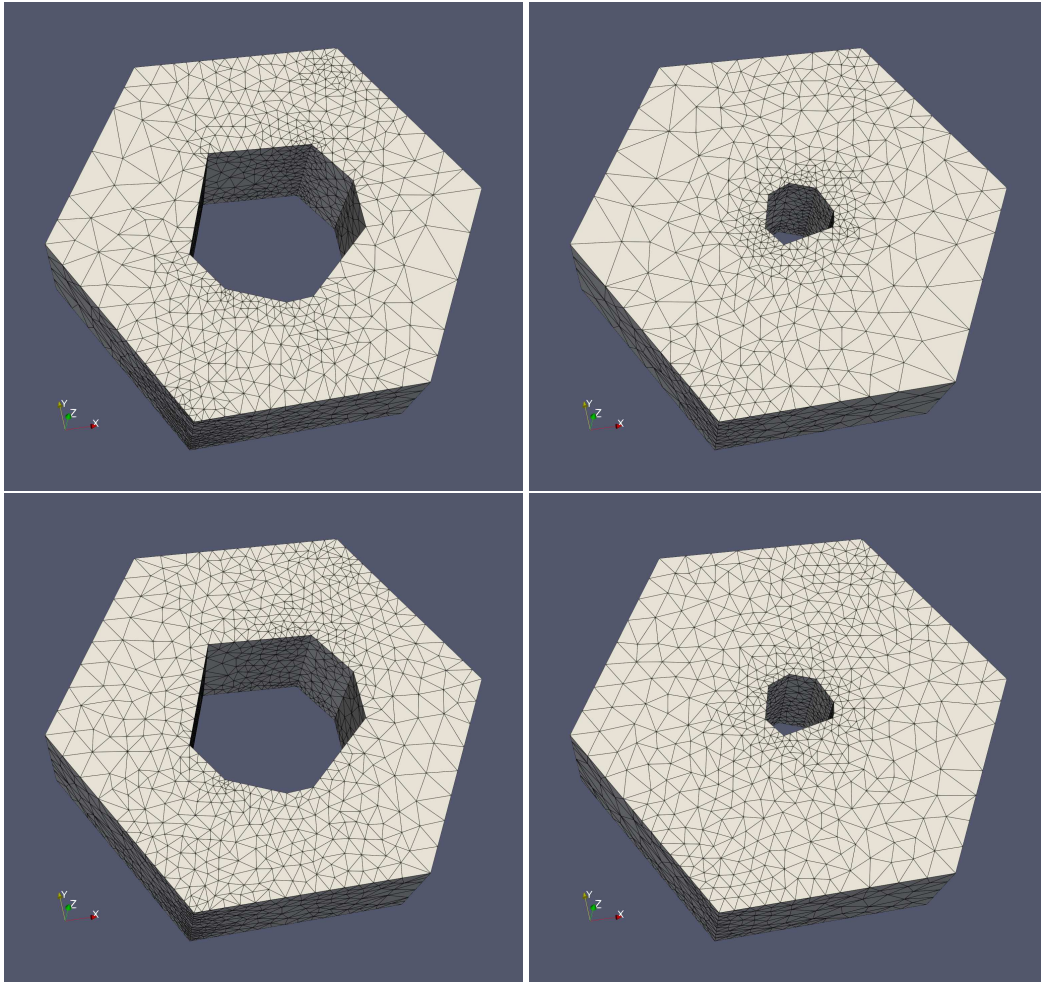


Figure 10: hex-bolt experiment, ground truth meshes (top) and NN meshes (bottom) , the left and right are two problems that only have different geometries

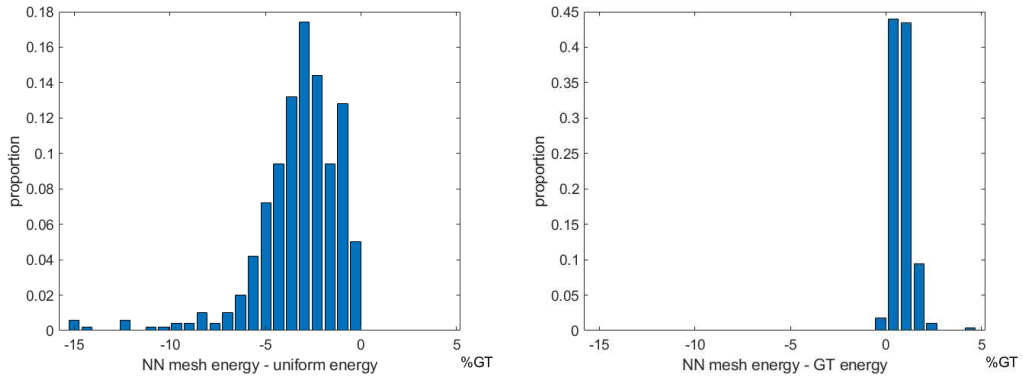


Figure 11: For *irregular polyhedron*, FE energies of neural network (NN) generated meshes versus uniform mesh FE energies and ground truth (GT) energies. The height of each bar represents the proportion of experiment results in the energy range shown on the x -axis (as a percentage of the ground truth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

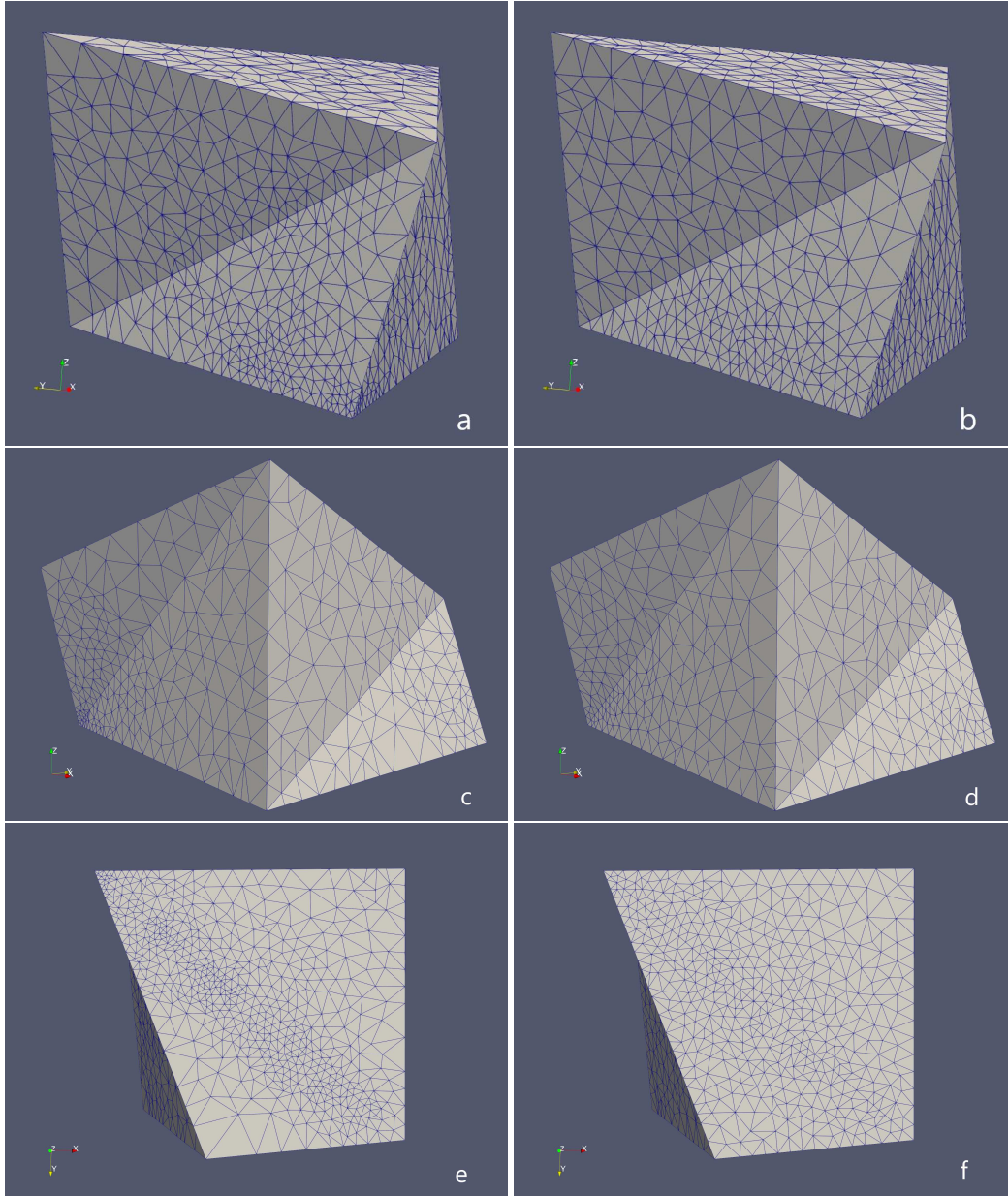


Figure 12: A ground truth mesh (a, c and e) and corresponding NN mesh (b, d and f) selected from 500 testing problems, they are in front (a and b), right (c and d) and bottom (e and f) views.

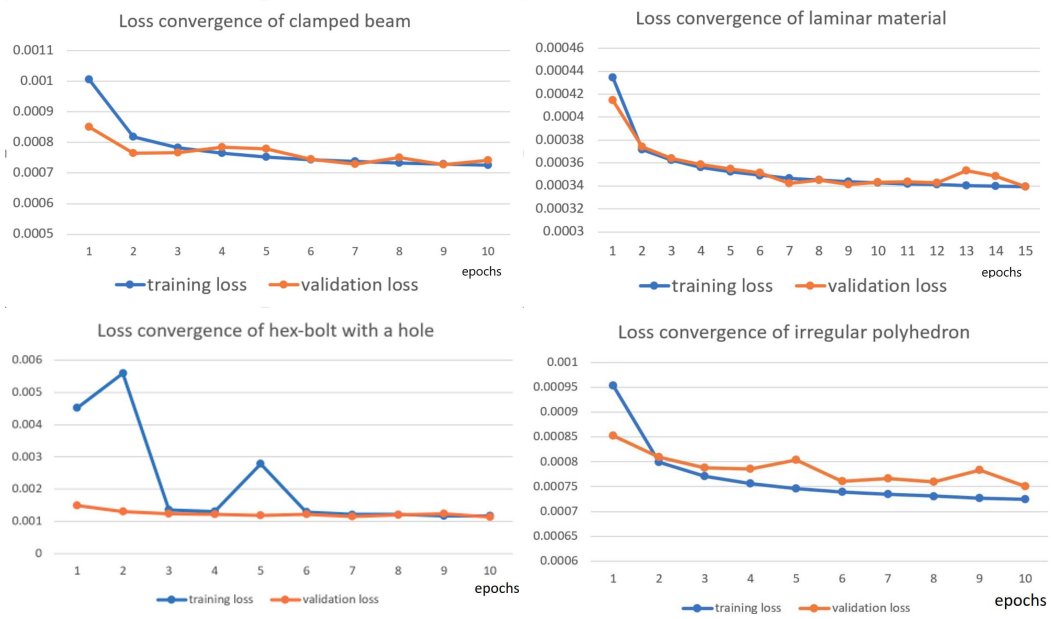


Figure 13: Training and validation loss of the four experiment