Machine Learning for Computational Science and Engineering (2025) 1:12
https://doi.org/10.1007/s44379-025-00013-3

RESEARCH

Non-iterative generation of optimized meshes for finite element simulations with deep learning

Martin Legeland[1] · **Kevin Linka**[1] · **Roland C. Aydin**[1,2] · **Christian J. Cyron**[1,2]

Received: 6 December 2024 / Accepted: 23 January 2025

Abstract
The finite element method is one of the most widely used computational methods in engineering and science. It provides approximate solutions to boundary value problems. The quality of these solutions critically depends on the underlying discretization, the so-called mesh. To optimize the mesh, adaptive refinement methods have been proposed over the last years that can improve mesh quality over a series of iteration steps. Herein, we propose a novel deep learning architecture that can cut short the process of mesh optimization. This architecture exploits fundamental invariance and equivariance properties to keep the amount of training data modest. It can generate high-quality meshes for a given boundary value problem and a desired target approximation error in a direct, non-iterative way. We demonstrate the performance of our method by the application to standard two-dimensional linear-elastic elasticity problems. There, our method generates meshes that reduce the solution error by 22.6% (median) compared to uniform meshes with the same computational demand.

Keywords Mesh generation · Machine learning · Finite element · Adaptive mesh refinement

1 Introduction

In modern computational engineering and science, it is essential to ensure high accuracy of finite element (FE) approximations, while keeping the numerical cost manageable. One common way of approaching this is to use meshes with heterogeneous element size distributions across the domain of interest. The element size distribution is often chosen in an attempt to ensure that the approximation error meets a certain criterion, for example, a uniform distribution over the domain. However, it is typically *a priori* unknown what element size distribution is required to meet the desired error criterion. To solve this problem, current methods of mesh optimization typically rely on iterative adaptive mesh refinement (AMR). Thereby, starting from some initial mesh, in each iteration step the best currently available FE solution is used to estimate approximation errors and infer where regional refinements of the mesh are expected to improve the accuracy of the FE solution [21, 37]. This process is repeated until a stopping criterion is met, usually a uniform distribution of the error estimate or a maximal number of elements (or degrees of freedom). Typically, such approaches are computationally expensive, as they require the repeated solution of a given boundary value problem (BVP) over many iteration steps (Fig. 1a).

We aim to develop a more computationally efficient method that directly produces an optimal mesh for a given BVP, eliminating the need for iterative refinement. machine learning (ML) offers a promising framework for such a method. Currently proposed methods for mesh generation using ML can be grouped into two categories, namely augmenting methods and direct methods:

The group of augmenting methods enhance the AMR process. For example, Manevitz, Bitar, and Givoli [23] used a supervised feed-forward multilayer perceptron (MLP) architecture to predict the gradients of the solution to improve error indicators. Chen and Fidkowski [7] instead applied a convolutional neural network (CNN) to predict an error indicator from simulation results at each mesh iteration, while other methods circumvent the need for an error indicator entirely

✉ Martin Legeland
martin.legeland@tuhh.de

[1] Institute for Continuum and Material Mechanics, Hamburg University of Technology, Eißendorfer Straße 42, 21073 Hamburg, Germany

[2] Institute of Material Systems Modeling, Helmholtz-Zentrum Hereon, Max-Planck-Straße 1, 21502 Geesthacht, Germany

Published online: 05 February 2025

by using a supervised artificial neural network (ANN) [25] or reinforcement learning (RL) Yang et al. [33] to determine mesh refinements directly. Both Foucart, Charous, and Lermusiaux [11] as well as Lorsung and Farimani [22] employed RL in time-dependent problems to adjust the mesh density during time steps. However, these methods are bound to the costly iterative AMR procedure, requiring a solution at each step. Direct methods instead aim to replace the iterative AMR process altogether and predict an optimized mesh directly.

Early efforts were made using an ANN to predict the element size [10] or an element density [6] for two-dimensional problems in electromagnetism. Notably, some thought was given in these works to generalization to more complex geometries by choosing a certain representation of the input data, thus respecting rotational invariance in the predicting network. However, these works seem to have remained without consequence until recently. Zhang et al. [35] proposed an ANN to predict error values at each point in the domain, converted heuristically into an upper bound on the element area. The architecture applies only to the very narrow class of seven-sided polygons, meaning that virtually every new geometry would require training of a new network. The same limitation applies to the follow-up work [34] treating problems in three dimensions.

Lock et al. [20] parametrized the element size density prediction in terms of source points in the problem domain, each associated with an element size and an influence radius. The positions along with their prescribed element size and radius were predicted by a supervised ANN. Invariances of the output, for example permutation invariance of the predicted points, is not discussed and given that the ANN output is fixed to certain locations in the domain, its capabilities to generalize is limited.

All of the above direct methods use simple supervised feed-forward MLPs and suffer from the inherent restriction to a fixed number of inputs and outputs, limiting their scalability to larger problems.

This has been remedied with some success by transforming the problem into the image domain and building on architectures proven in image processing: Chan, Scholz, and Takacs [16] and Huang et al. [5] utilized modified versions of the same CNNs [27] to predict a pixel-by-pixel element size in the problem domain for one specific plane computational fluid dynamics (CFD) problem [16] and a limited class of plane stress problems in linear elasticity [5] including reentrant corners (stress singularities are not treated). Nevertheless, Chan, Scholz, and Takacs [5] achieved predicted meshes that fall between a uniform mesh and an AMR baseline in terms of quality. CNNs allow the generalization to unseen geometries but limits the resolution of the prediction to a certain number of pixels and rotational invariance must be weakly enforced by using data augmentation, increasing the computational cost of training. Furthermore, CNNs struggle with the extension to three-dimensional problems [12].

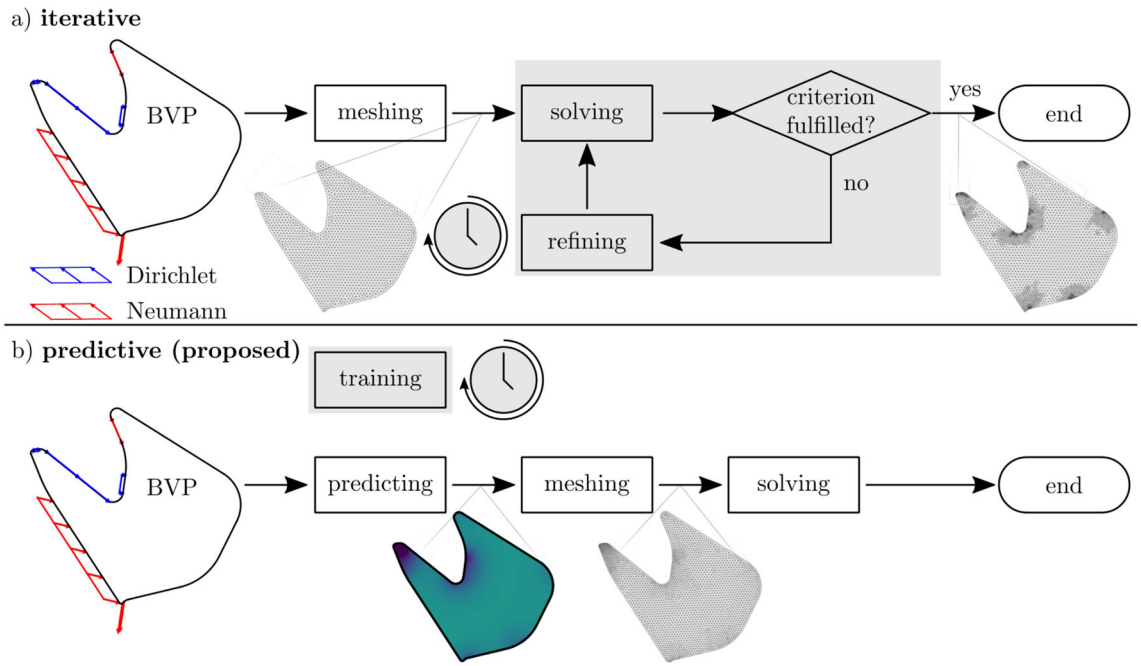


Fig. 1 Comparison between conventional adaptive mesh refinement and the proposed method. a) The currently used iterative method to obtain heterogeneous meshes is computationally expensive. b) The proposed method requires training but produces heterogeneous meshes without iterations at inference

To overcome theses issues, Freymuth et al. [12] proposed a graph neural network (GNN) that operated directly on an initial BVP discretization, predicting refinements for individual elements. This process is repeated several times until the final mesh is obtained, eliminating the need for a solving the BVP at each iteration, thus qualifying as a direct method. Pfaff et al. [26] predicted an adaptive mesh as support for their simulation surrogate using a GNN, where the local representation at each vertex was essential for successful generalization to larger models.

Despite these advances, direct methods for mesh generation still face challenges in generalizing to a wide range of problems, yet effective generalization is essential for realizing an efficient direct mesh generation method. The computationally expensive training would ideally occur only once, allowing the computational savings from subsequent predictions to outweigh the initial cost.

In this proof-of-concept study, we propose a deep learning method to generate high-quality h-adaptive meshes in a direct, non-iterative manner (Fig. 1b). An ANN is initially trained to predict the local relation between element size and approximation error for a certain class of BVPs. This relation enables the inference of a maximum element size at each point in the domain for a desired error threshold. After training, the method predicts optimized meshes for new BVPs without requiring iterative error estimates. As an important improvement compared to Zhang, Jimack, and Wang [35] and Zhang et al. [34], our architecture processes general polygons with an arbitrary number of sides. By leveraging invariance to cyclic permutations and rigid-body motions, it generalizes beyond the polygon types in the training data, eliminating the need for retraining for new polygon classes. This ability to generalize to new classes of polygons without requiring retraining is a key advantage of the proposed method.

While the initial training of the artificial neural network is associated with a significant computational cost, this is a one-time investment. Once trained, the model efficiently generates optimized meshes for a wide range of geometries, offering dramatic long-term savings compared to conventional iterative adaptive mesh refinement (AMR) techniques. This ability to generalize, coupled with the non-iterative nature of the mesh generation process, shows that the proposed method bears the potential of a computationally superior alternative for repeated applications.

In Section 2, we introduce our novel deep learning architecture and how it incorporates knowledge about BVPs. In Section 3, we describe the BVP data and parameters used for training. Section 4 reports the results and performance of our method in specific numerical examples. Section 5 summarizes our main findings and results.

2 Computational framework

2.1 Boundary value problem parametrization

For this proof-of-concept study, we limit our scope to plane-stress elasticity problems, assuming a homogeneous linear-elastic material. The problem domain Ω is assumed to be a polygon with piecewise constant Dirichlet or Neumann boundary conditions on its edges Γ and a constant body force. The constitutive equation of the linear elastic material is expressed in terms of the two Lamé parameters λ and μ. By normalizing the body force and Neumann boundary conditions by the second Lamé parameter μ, one express the Cauchy stress $\boldsymbol{\sigma}$ in terms of a normalized stress

$$\tilde{\boldsymbol{\sigma}} = \frac{\boldsymbol{\sigma}}{\mu} = \beta \,\mathrm{tr}\,(\boldsymbol{\epsilon})\, \boldsymbol{I} + 2\boldsymbol{\epsilon} \quad \text{with} \quad \beta = \frac{\lambda}{\mu}. \tag{1}$$

Using these normalized quantities, the BVP can be formulated without loss of generality in terms of only a single material parameter, β, which reduces the dimensionality of the data space defining our BVPs of interest. $\boldsymbol{I}$ denotes the identity tensor and $\boldsymbol{\epsilon}$ the linear strain tensor. We denote a parametrization of such a BVP by $\boldsymbol{B}$. Its domain boundaries are described in terms of the polygon's n corners, indexed in counter-clockwise direction by $i = 1, 2, \ldots, n$. Each of the polygon sides has a number of associated features. These features are the two position coordinates of the polygon side's first corner $\boldsymbol{p}_i$, a Boolean flag $\delta_i \in \{0, 1\}$ to indicate whether the side is subject to a Dirichlet condition, and two components defining either a Dirichlet boundary condition $\boldsymbol{u}_i$ or a Neumann boundary condition $\boldsymbol{t}_i$. Each boundary segment is either subject to a Dirichlet or a Neumann condition. A Dirichlet condition is encoded by $\delta_i = 1$. Free boundaries are described imposing a zero value Neumann condition. Since we have either Dirichlet or Neumann boundary conditions on a polygon segment, it is sufficient to describe the values of the boundary conditions using the vector

$$\boldsymbol{v}_i = \begin{cases} \boldsymbol{u}_i & \text{if} \quad \delta_i = 1, \\ \boldsymbol{t}_i & \text{if} \quad \delta_i = 0. \end{cases} \tag{2}$$

These features are collected in the two-dimensional array $\boldsymbol{S} \in \mathbb{R}^{n\times 5}$, whose n rows are formed by the row vectors $\left(\boldsymbol{p}_i^T, \delta_i, \boldsymbol{v}_i^T\right)$ with i = 1, …, n. Furthermore, there are three global features, namely the two components of the body force $\boldsymbol{f}$ and the natural logarithm of the scalar material parameter β, collected in $\boldsymbol{G} = (\ln(\beta), \boldsymbol{f}^T) \in \mathbb{R}^3$. Hence, the type of boundary value problem (BVP) we study herein is fully

described by the parameter set $\boldsymbol{B} = \{\boldsymbol{S}, \boldsymbol{G}\}$. Figure 2a) illustrates this type of BVP.

2.2 Pointwise convergence regression model

The core idea in our approach is to predict the convergence of the error of the FE approximation. In the literature, the convergence behavior of a particular error measure is usually expressed in terms of some norm and the element size h of homogeneous meshes, see for example [4, 15, 36]. Here, we consider a pointwise convergence of a measure E at each point $\boldsymbol{x}$. We heuristically assume that it can be expressed as

$$E(h, \boldsymbol{x}, \boldsymbol{B}) = C(\boldsymbol{x}, \boldsymbol{B}) h^{M(\boldsymbol{x}, \boldsymbol{B})} \tag{3}$$

with the two parametrized scalar-valued functions C and M. The error is thus defined as a function over the entire BVP domain with the parameters collected in $\boldsymbol{B}$. It can be evaluated at any position $\boldsymbol{x}$, yielding an explicit relationship between the error and the largest side h of the element containing $\boldsymbol{x}$, as indicated in Fig. 2b).

Modeling the convergence behavior by such a gray box model, rather than by a black box model where the error is computed altogether as the output of a monolithic end-to-end artificial neural network (ANN), allows us to invert the model such that the element size can be computed as

$$h^*(\boldsymbol{x}, E^*, \boldsymbol{B}) = \left[\frac{E^*}{C(\boldsymbol{x}, \boldsymbol{B})} \right]^{\frac{1}{M(\boldsymbol{x}, \boldsymbol{B})}} . \tag{4}$$

This equation can be exploited as follows. First, the convergence behavior for a large number of BVPs is determined by computing the finite element (FE) solutions associated with various discretizations of these BVPs. From these data, the functions C and M in Eq. 3 can be learned by ANNs. Subsequently, the trained ANNs can be used in Eq. 4 to predict for new, unseen BVPs the (in general) non-uniform element size field h^* that is required to ensure a given target error E^* at each point. Finally, this element size field can be used to generate a proper mesh with the desired element size distribution using some existing general purpose mesh generation algorithm. This procedure is illustrated in Fig. 3.

2.2.1 Assumptions and requirements

For our machine learning model we define several assumptions and requirements. These can be understood as a form of *a priori* information grounded on general mathematical insights and principles. Endowing our machine learning architecture with this *a priori* information substantially reduces the amount of training data required and helps the trained network to generalize beyond the domain of training data.

Positivity From Eq. 4 it is clear that we must ensure $C > 0$. Our experiments have shown that instead of learning C directly, it is beneficial to learn c such that

$$C = \exp(c), \tag{5}$$

which automatically ensures $C > 0$. Furthermore, we assume a positive pointwise rate of convergence $M > 0$

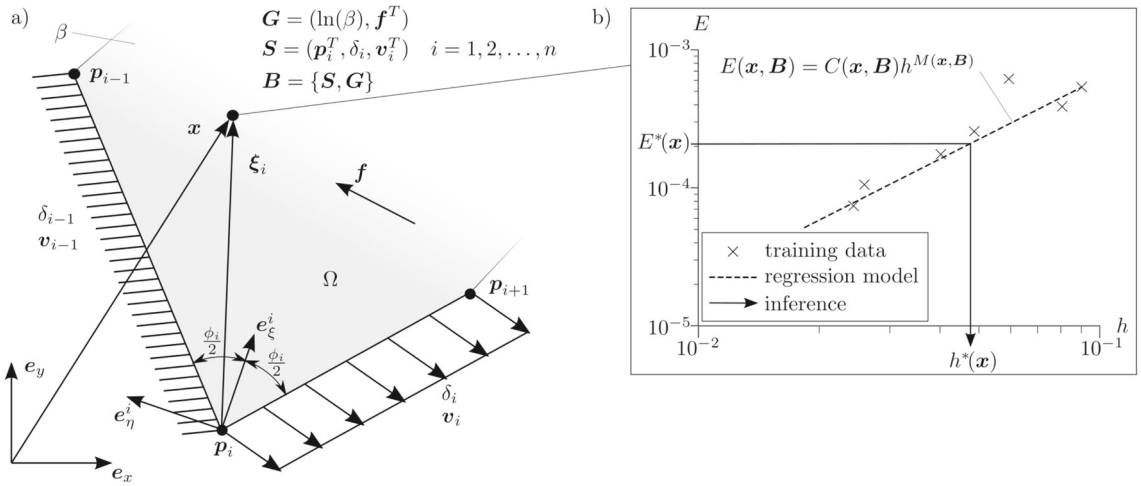


Fig. 2 Pointwise regression model in the neighborhood of a corner in the BVP: a) A part of an example BVP domain Ω (gray) is shown with boundary conditions $\{\boldsymbol{v}_{i-1}, \delta_{i-1}\}$ and $\{\boldsymbol{v}_i, \delta_i\}$. The vectors $\boldsymbol{e}_x$ and $\boldsymbol{e}_y$ form the basis of the global coordinate system, ϕ_i is the internal angle of the i-th corner. b) From training data, we learn how at each point $\boldsymbol{x}$ discretization length h and error E are related depending on the parameters $\boldsymbol{B}$ of the boundary value problem. Once we have learned this, we can predict the discretization length h^* that can be expected to ensure a certain desired error E^* at each position $\boldsymbol{x}$

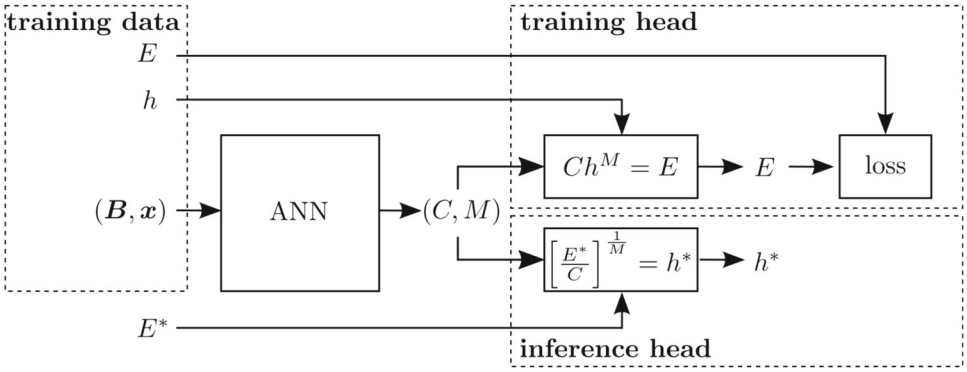


Fig. 3 Convergence model. During training, ANNs learn to provide at each point $\boldsymbol{x}$, for a given boundary value problem $\boldsymbol{B}$, an estimate of the functions C and M, governing (local) convergence. Once training has been accomplished, the trained networks can be exploited to provide pointwise estimates of C and M for new, unseen BVPs, that can be used to infer the local element size h^* to achieve a given desired error E^*

everywhere. To this end, we found that it is beneficial to not directly learn M with our ANN but rather m with

$$M = \text{softplus}(m) = \log(1 + e^m), \tag{6}$$

as suggested in [9].

Cyclic invariance We require that the model be independent of the corner point numbering, i.e., it must be invariant with respect to simultaneous cyclic permutations of the boundary specification (geometry and boundary conditions). This requirement ensures that all equivalent representations of a BVP result in the same prediction. To achieve cyclic invariance of the model, three components are employed:

First, the topology of the polygon must be represented correctly. The first row in $\boldsymbol{S}$ represents a side that neighbors the side represented by the last row.

Second, all polygon sides (rows in $\boldsymbol{S}$) must be treated equally. Hence, we cannot simply input $\boldsymbol{S}$ altogether into the same ANN because the ordering of the input features would influence the output. Instead, the subsets of features associated with the respective boundary segments (rows in $\boldsymbol{S}$) are processed separately, yielding an output for each boundary segment as a contribution to the total output. The processing of the boundary segments' features is performed by identically designed networks with shared weights to assign equal importance to all segments' contributions.

Third, the contribution of a boundary segment to the total output must depend on the features of the boundary segment and, crucially, its neighbors' features. If the contribution did not depend on a boundary segment's neighbors, the model would become invariant also with respect to non-cyclic permutations, which is undesired. In our implementation, we use the features associated with a polygon's side and its two immediate neighbors. Thus, only such neighborhood information is used independently of the global context.

As discussed below, we use 2D convolutional layers with cyclic padding of the input in our implementation, because they respect these three requirements naturally.

Objectivity Furthermore, the model must be objective, i.e., unaffected by superposed rigid-body motions. The requirement of objectivity is realized by transforming the BVP parametrization into local ξ_i-η_i-coordinate systems (Fig. 2), attached to each to each i-th corner in the polygon with $i = 1, ..., n$. In doing so, the BVP features are represented independently of the choice of any global coordinate system. Two parts of a polygon (or different polygons), that are similar in the geometric sense (i.e., one being a rotated copy of the other) may exhibit different components of associated vectorial characteristics (like Neumann boundary conditions) in a fixed global coordinate system, respectively, but they are ensured to exhibit the same components in their respective local coordinate systems. Since we assume that the error is determined locally, we require that similar geometries are associated with locally similar errors. Moreover, the local representation of the features is beneficial in the ML task at hand, because objectivity is enforced by design and does not need to be learned by other means such as data augmentation.

Parametrized contributions Finally, we regard the convergence behavior of the error E as a global function over the BVP domain, depending only on the BVP features. This is achieved by modeling c and m as parametrized functions. In our approach, we choose to represent c and m in terms of Gaussian functions whose parameters depend only on the BVP features. The position $\boldsymbol{x}$ is only used to evaluate the Gaussian functions for a particular position. We can therefore be sure that our model does not learn spurious correlations with the representation of $\boldsymbol{x}$.

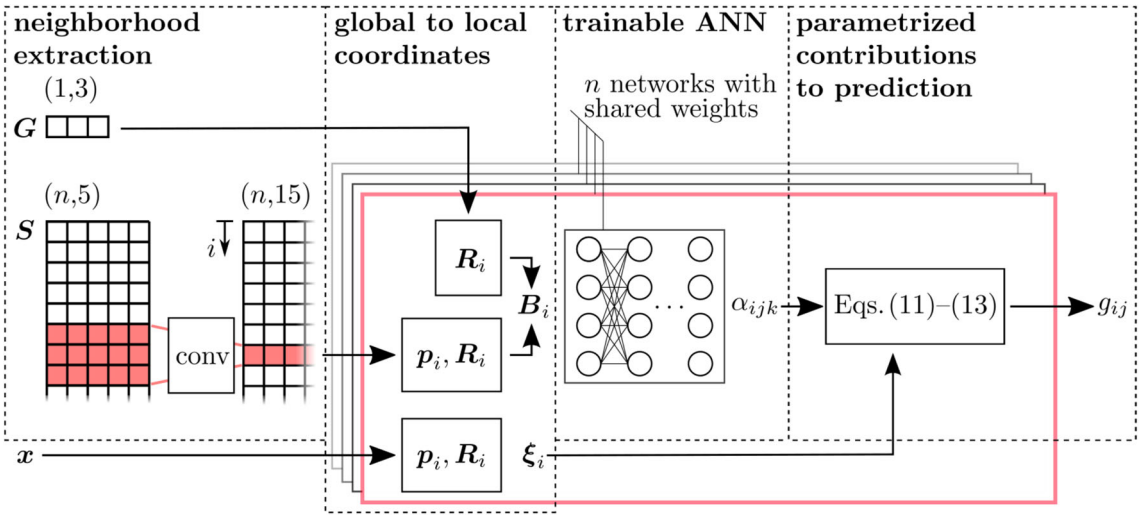


Fig. 4 Artificial neural network architecture. For each of the n corners of the polygon, first the relevant neighborhood information is extracted from the global feature matrix $\boldsymbol{S}$ by an appropriate convolutional layer with fixed weights and concatenated into a 1×15 row vector (illustrated in red on the left). Subsequently, to ensure objectivity, global coordinates are transformed to (unique) local coordinates using the procedure described in Section 2.2.2, yielding thereby a selection of the locally relevant features in local coordinates that is referred herein as $\boldsymbol{B}_i$. The stacked frames represent the separate processing of each corner's features. The red frame processes the i-th corner's features. $\boldsymbol{B}_i$ is processed by a trainable ANN consisting of fully connected layers, whose weights are shared across all n corners. Its outputs are the parameters α_{ijk}, which yield, via Eqs. 11 – 13, the Gaussian functions g_{ij} that contribute to C and M (Fig. 5)

In the following three subsections, we explain the model's architecture in detail. Figures 4 and 5 provides a visual representation of its composition. Its elements are explained in detail in the following subsection.

2.2.2 Feature representation

The input to the model consists of the BVP parametrization $\boldsymbol{B} = \{\boldsymbol{S}, \boldsymbol{G}\}$ and the position vector $\boldsymbol{x}$. The first layer in our network architecture is an untrainable convolutional layer

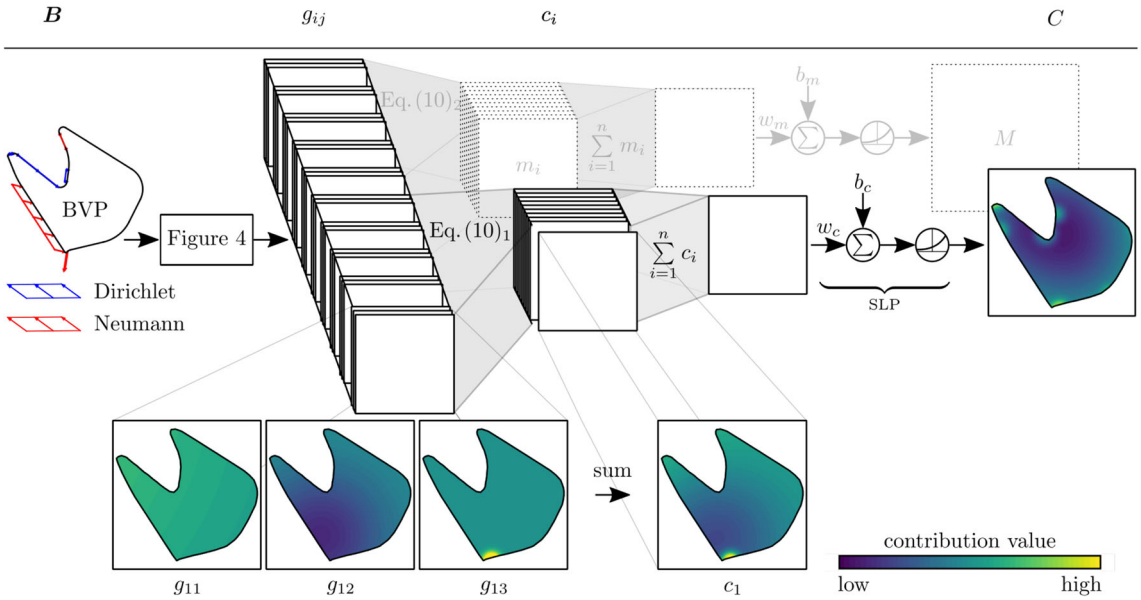


Fig. 5 Contributions from Gaussian functions to predicted C. For a given BVP, the contributions of the Gaussian functions g_{ij} to C are shown. The computation of M is completely analogous and indicated by dashed outlines in the background. The BVP's corners and Gaussian functions are indexed by i and j, respectively. The three individual Gaussians for corner $i = 1$ are shown (bottom left) as well as their sum (bottom right). A single layer perceptron (SLP) transforms the summed contributions from all n corners to obtain C

with a kernel size of $(1, 2W + 1)$, whose kernel weights are initialized with values of either 0 or 1, such that the features from adjacent sides are simply concatenated. That is, with our choice $W = 1$, the three 1×5 row vectors with the features of a specific side of the polygon and the two sides directly adjacent to it are concatenated into a 1×15 row vector with the same content (illustrated in red on the left side of Fig. 4). We perform this operation for each of the n sides of our polygon. The purpose of this layer is merely the selection of the neighborhood features around a polygon corner from the features assembled in the global matrix $\boldsymbol{S}$. We apply cyclic padding of the input features in this layer, as suggested by Schubert et al. [28], in order to respect the cyclic structure of the polygon.

For a software implementation it is usually helpful to define geometric features like corners or boundary conditions initially in terms of some uniform global coordinate system. However, to achieve objectivity (that is, independence of the choice of this global coordinate system) we transform the coordinates of geometric features first into a unique local Cartesian coordinate system before processing them in our machine learning architecture. For machine learning steps related to the i-th corner, this local coordinate system is assumed to have the origin $\boldsymbol{p}_i$ and the unit basis vectors $\boldsymbol{e}_\xi^i$ and $\boldsymbol{e}_\eta^i$. The former basis vector is defined pointing inwards parallel to the angle bisector at the i-th corner. The latter is constructed orthogonal to $\boldsymbol{e}_\xi^i$ such that a right-handed coordinate system is obtained.

To transform position vectors into the local coordinate system, one first applies a translation by subtracting the position of the i-th corner, that is, one defines relative positions

$$\boldsymbol{\rho}_i^- = \boldsymbol{p}_{i-1} - \boldsymbol{p}_i, \quad \boldsymbol{\rho}_i^+ = \boldsymbol{p}_{i+1} - \boldsymbol{p}_i \quad \text{and} \quad \boldsymbol{\xi}_i = \boldsymbol{x} - \boldsymbol{p}_i, \tag{7}$$

where $\boldsymbol{x}$ denotes some point in space. As a next step, both position vectors and other vectors (e.g., those defining loads and displacements) are written as column vectors and multiplied with a rotation matrix $\boldsymbol{R}_i \in SO(2)$ whose columns are the unit vectors $\boldsymbol{e}_x$ and $\boldsymbol{e}_y$ of the global system, represented in the i-th local coordinate system. This rotational transformation provides the coordinates of all vectors in terms of the local $\boldsymbol{e}_\xi^i$-$\boldsymbol{e}_\eta^i$-basis, which is required to ensure objectivity of the data processing of our machine learning architecture.

Figure 2 exemplarily shows the description of an arbitrary position $\boldsymbol{x}$ by $\boldsymbol{\xi}_i$. Note that in the local representation, the coordinates of the corner point $\boldsymbol{p}_i$ are always zero, because it is the origin of the local coordinate system. Hence, it can be omitted in the input to the trainable layers. Furthermore, from the features associated with side $i + 1$, only the corner point $\boldsymbol{p}_{i+1}$ is used because it constitutes the end point of the i-th side. Hence, only 10 of the 15 features extracted from $\boldsymbol{S}$ are needed to determine the neighboring sides in both forward and backward direction from the i-th corner, namely $\left\{\boldsymbol{\rho}_i^-, \delta_{i-1}, \boldsymbol{v}_{i-1}, \delta_i, \boldsymbol{v}_i, \boldsymbol{\rho}_i^+\right\}$. These 10 selected features are concatenated together with the global features $\ln(\beta)$ and $\boldsymbol{f}$ into the 1×13 array $\boldsymbol{B}_i$, resulting in the representation of the i-th corner's neighborhood in its own coordinate system. These localized features constitute the input to the trainable layers.

2.2.3 Parametrized contributions

The mesh size required to ensure a certain error at a certain point $\boldsymbol{x}$ depends on its position relative to each boundary segment as well as on the boundary conditions imposed on the respective boundary segments because this information (together for all points in the domain) defines the BVP as a whole. Without loss of generality, one can thus express the functions c and m as

$$c(\boldsymbol{x}, \boldsymbol{B}) = w_c \sum_{i=1}^{n} c_i(\boldsymbol{x}, \boldsymbol{B}) + b_c \quad \text{and} \quad m(\boldsymbol{x}, \boldsymbol{B}) = w_m \sum_{i=1}^{n} m_i(\boldsymbol{x}, \boldsymbol{B}) + b_m \tag{8}$$

with contributions c_i and m_i from all n sides of the BVP geometry, where w_c, b_c, w_m and b_m are scalars. Note that the c_i and m_i in Eq. 8 in general depend on all parameters in $\boldsymbol{B}$ so that also complex (e.g., long-range or nonlinear) interactions between the effect of different sides of the polygon can in principle be represented by suitable functions c_i and m_i so that Eq. 8 does not imply any loss of generality.

However, to reduce the complexity of our problem for practical purposes, we herein make the additional heuristic assumption that the contributions of the i-th boundary segment c_i and m_i can be computed by information about this boundary segment itself as well as its two adjacent boundary segments, which corresponds to the relevant neighborhood information $\boldsymbol{B}_i$ illustrated in red in the section *neighborhood extraction* of Fig. 4. That is, we assume

$$c(\boldsymbol{x}, \boldsymbol{B}) = w_c \sum_{i=1}^{n} c_i(\boldsymbol{x}, \boldsymbol{B}_i) + b_c \quad \text{and} \quad m(\boldsymbol{x}, \boldsymbol{B}) = w_m \sum_{i=1}^{n} m_i(\boldsymbol{x}, \boldsymbol{B}_i) + b_m. \tag{9}$$

We underline that Eq. 9 neglects compared to Eq. 8 certain nonlinear interactions between boundary segments. However, the success of this assumption in the examples below demonstrates that it is sufficient as a heuristic basis for the time being.

We allow any boundary segment to influence the desired mesh size at any point $\boldsymbol{x}$ in the domain, not only in the neighborhood of the segment. However, physically it is reasonable to assume that the influence can generally be described by some sort of function decaying with spatial distance. Herein,

we hence assume that the influence of the i-th boundary segment on the convergence at point $\boldsymbol{x}$ can be modeled through a series of Gaussian functions. That is, we assume

$$c_i(\boldsymbol{x}, \boldsymbol{B}_i) = \sum_{j=1}^{K_c} g_{ij}(\boldsymbol{\xi}_i, \boldsymbol{B}_i) \quad \text{and} \quad m_i(\boldsymbol{x}, \boldsymbol{B}_i) = \sum_{j=K_c+1}^{K_c+K_m} g_{ij}(\boldsymbol{\xi}_i, \boldsymbol{B}_i) \tag{10}$$

with Gaussian functions $g_{ij}(\boldsymbol{\xi}_i, \boldsymbol{B}_i)$ defined as

$$g_{ij}(\boldsymbol{\xi}_i, \boldsymbol{B}_i) = A_{ij} \exp\left[-\frac{1}{2}(\boldsymbol{\xi}_i - \boldsymbol{\mu}_{ij})^\top \boldsymbol{\Sigma}_{ij}^{-1}(\boldsymbol{\xi}_i - \boldsymbol{\mu}_{ij})\right] \tag{11}$$

with

$$A_{ij} = \alpha_{ij1}, \quad \boldsymbol{\mu}_{ij} = \begin{pmatrix}\alpha_{ij2}\\ \alpha_{ij3}\end{pmatrix}, \quad \boldsymbol{\Sigma}_{ij} = \boldsymbol{Q}_{ij}\boldsymbol{\Lambda}\boldsymbol{Q}_{ij}^\top, \tag{12}$$

where

$$\boldsymbol{\Lambda} = \begin{bmatrix}\alpha_{ij4}^2 + 10^{-7} & 0\\ 0 & \alpha_{ij5}^2 + 10^{-7}\end{bmatrix}, \quad \boldsymbol{Q}_{ij} = \begin{bmatrix}\cos(\alpha_{ij6}) & -\sin(\alpha_{ij6})\\ \sin(\alpha_{ij6}) & \cos(\alpha_{ij6})\end{bmatrix}. \tag{13}$$

Note that $\boldsymbol{\Sigma}_{ij}$ is positive definite by construction. The summands 10^{-7} in $\boldsymbol{\Lambda}$ are added for numerical reasons to ensure that the variance of the Gaussian functions is always sufficiently different from zero to avoid numerical problems. For ease of notation we dropped in Eqs. 11, 12 and 13 the dependence of A_{ij}, $\boldsymbol{\mu}_{ij}$, $\boldsymbol{\Sigma}_{ij}$, $\boldsymbol{Q}_{ij}$ and the α_{ijk} on the characteristic properties of the BVP captured by the $\boldsymbol{B}_i$. We note that the parameters A_{ij}, $\boldsymbol{\mu}_{ij}$, $\boldsymbol{\Sigma}_{ij}$, and $\boldsymbol{Q}_{ij}$ of the Gaussian functions g_{ij} are directly governed by the altogether six scalar parameters $\alpha_{ij1}, \ldots, \alpha_{ij6}$. These are determined by the $\boldsymbol{B}_i$ by some in general non-trivial relation. This relation can, for example, be learned by a dedicated ANN.

The number of Gaussian functions is controlled by the hyperparameters K_c and K_m. These hyperparameters are comparable to the model degree in polynomial regression in the sense that a larger number of functions results in a greater expressivity of the contributions but bears the risk of overfitting. Hyperparameter tuning revealed that $K_c = K_m = 3$ provides enough expressivity to model the i-th side's contributions to the convergence behavior and more Gaussian functions do not result in a better performance. Hence, in total, there are $K_c + K_m = 6$ Gaussians associated with every corner. The first $K_c = 3$ Gaussian functions contribute to c_i, whereas the following $K_m = 3$ functions contribute to m_i. Given that each Gaussian function has 6 unknown parameters $\alpha_{ij1}, \ldots, \alpha_{ij6}$, this yields altogether 36 unknown parameters.

The Eqs. 5, 6, 9 - 13 enable the computation of functions $C(\boldsymbol{x}, \boldsymbol{B})$ and $M(\boldsymbol{x}, \boldsymbol{B})$, as visualized in Fig. 5. That is, they allow modeling the convergence behavior in a way that depends on altogether $n \times (K_c + K_m) \times 6$ scalar parameters $\alpha_{ijk}(\boldsymbol{B}_i)$. How to choose these parameters depending on the characteristics $\boldsymbol{B}_i$ of the BVP is beyond what can be grasped analytically. Therefore, machine learning is used to determine the relation between the $\boldsymbol{B}_i$ (input) and the corresponding α_{ijk} (output). Details are discussed in the following subsection.

2.2.4 Trainable artificial neural network

ANNs are used to map the localized features to the values of the functions c_i and m_i. More precisely, the output of the ANN are the parameters α_{ijk} that are used to determine the parametrized functions c_i and m_i over Ω. We have

$$\alpha_{ijk} = f_i(\boldsymbol{B}_i, \boldsymbol{\theta}_i), \tag{14}$$

where the f_i are the ANNs and $\boldsymbol{\theta}_i$ their trainable parameters. As described in Section 2.2.1, to achieve cyclic invariance of the prediction, all corners must be processed separately and equally. That is, we must use $f_i = f$ to map the neighborhood features of each corner to its corresponding parameters. Equivalently, to process a polygon with n corners, we require n identical ANNs and hence $\boldsymbol{\theta}_i = \boldsymbol{\theta}$.

From an implementation point of view, it is convenient to use a CNN [13, 18]. Typically, such networks are used to process images and have the following characteristics: The input is a multi-dimensional array, typically an image with one or more color channels. The input is scanned by a filter that "slides" over the elements of the input in two dimensions and maps the features in further dimensions (e.g., color values) within a window-view of the input to an output. However, if the window size is 1-by-1, only the features of a single input element are used. In this case, the neighborhood relation between adjacent elements in the input is discarded, but still all pixels are processed using the same filter. Effectively, such a CNN consists of a fully connected network that is applied to the input pixel-by-pixel, where the input consists of the input pixel's channels [19].

In our implementation, we utilize this CNN architecture to process all corners' features simultaneously. The representations $\boldsymbol{B}_i$ are concatenated and the resulting array of shape $(n, 13)$ is passed through several convolutional layers, all using 1-by-1 kernels. In analogy to the image domain, the input corresponds to an image of height 1 and width n with 13 channels. Owing to the use of filters in CNNs, only the last dimension of the input is fixed, meaning that the number of corners n that can be processed using this architecture is variable, allowing us to process polygons with an arbitrary number of corners.

According to the results of hyperparameter tuning, we use eight ELU-activated layers [8] with 182, 91, 45, 45, 45, 45, 91 and 182 filters respectively, with an additional bias input in each. These layers are followed by a single unactivated layer with 36 filters. The number of neurons in the final layer is determined by our particular choice for the corners' contributions on the prediction, as described in Section 2.2.3. We use dropout [29] with a retaining probability of $p = 0.7$ (dropout rate of 0.3) between all nine hidden layers of the network. As suggested by Srivastava et al. [29], we impose a max-norm constraint on the weights of the layers with a value of 4. In total, the layers have 57,069 trainable parameters. From the trainable layers, we obtain an array of parameters of shape $(n, 36)$, which comprises the outputs from all sides in the BVP geometry. The parameters are rearranged into an array with elements α_{ijk}, with $i = 1, 2, \ldots, n$; $j = 1, 2, \ldots, (K_c + K_m)$; $k = 1, 2, \ldots, 6$.

Note that we can capture Eqs. 5, 6 and 9 by two SLPs with a single neuron in each and correspondingly chosen activation functions, thereby adding 4 additional trainable parameters, raising the total to 57,073 trainable parameters.

3 Numerical examples

3.1 Definition of BVPs

To generate rich training data, we produced a wide variety of random BVPs. We use the 2-opt algorithm implementation of Auer and Held [2] as a random polygon generator to create polygonal BVP domains with n corners within the unit circle. This algorithm is not limited to a specific class of polygons (e.g., convex or star-shaped), but produces polygons that are general in nature [2].

Once a polygonal domain has been created, boundary conditions are randomly assigned to each side with probabilities of 40% for a free boundary (Neumann boundary with zero load), 30% for a Neumann boundary with non-zero load, and 30% for a Dirichlet boundary. The values of Dirichlet conditions are determined by a random angle, drawn uniformly from $[0, 2\pi)$, and a magnitude drawn uniformly from the interval $[0, 10^{-5})$ in units of displacement. Similarly, Neumann conditions are determined by uniformly drawing an angle of attack from $[0, 2\pi)$ and a magnitude from $[0, 10^{-4})$ in units of force per length. Free boundaries are equivalent to Neumann conditions with all zero components. The magnitude of the body force is drawn uniformly from the interval $[0, 10^{-3})$ in units of force per area and its angle is drawn from the interval $[0, 2\pi)$. The unitless material parameter β is drawn from a reciprocal distribution, such that $\ln(\beta)$ is uniformly distributed in the interval $\left[\ln(10^{-4}), \ln(10^{4})\right)$. To obtain valid BVPs, any problem without a Dirichlet condition is discarded.

3.2 Singularities

The generated geometries for the BVPs are in general not convex. In the neighborhood of reentrant corners, the stresses may approach infinity [24, 32] and the approximations fail to converge. Notably, such singularities may "pollute" the solution quality also in a different region of the domain [3]. To prevent stress singularities at reentrant corners and the associated problems, the randomly generated BVPs are modified. All corners are rounded, as illustrated in Fig. 6. The radius R_i of the fillet at the i-th corner is determined on the basis of the shorter of the two sides adjacent to it. To this end, we use a third of the length $\ell_{i,\min} = \min\left\{\|\boldsymbol{p}_{i+1} - \boldsymbol{p}_i\|, \|\boldsymbol{p}_i - \boldsymbol{p}_{i-1}\|\right\}$ to define

$$R_i = \frac{\ell_{i,\min}}{3} \tan\left(\frac{2\pi - \phi_i}{2}\right). \tag{15}$$

It ensures a smooth transition of the fillet into both adjacent sides at a distance of $\ell_{i,\min}/3$ from the corner. The fillet geometry is fully determined by the BVP parametrization encoded in $\boldsymbol{B}$. Hence, adding fillets does not require any amendments to $\boldsymbol{B}$ to hand over a complete description of the respective BVP to a machine learning algorithm. We impose zero value Neumann boundary conditions (free boundaries) in all fillet regions.

Herein we use linear finite elements (with straight sides) only. Thus, the piecewise linear approximation of the geometry converges to the exact round geometry as the mesh size h approaches zero.

3.3 Mesh specifications

In this proof-of-concept study, we only consider meshes of linear triangular elements. All meshes were generated with Gmsh [14].

For each BVP, a reference solution $\boldsymbol{u}$ is computed with an adaptive refinement procedure. To this end, an initial homogeneous mesh with the element size $h = 0.005$ is gradually refined using the so-called ZZ indicator [38], until the relative percentage error of $\bar{\eta} = 0.1$ is reached in each element. The solution computed on the basis of the resulting mesh is considered a (quasi-exact) reference solution. It is used in lieu of the exact (in general unknown) solution to determine the approximation quality. In the following, we understand the solution $\boldsymbol{u}$, strains $\boldsymbol{\epsilon}$ and stresses $\boldsymbol{\sigma}$ to be quasi-exact.

To generate data revealing the relation between mesh and convergence, seven additional approximations were produced for every BVP, each using a homogeneous mesh with a different element size, spaced logarithmically on the interval $[0.02, 0.1]$ (Fig. 7) such that the quasi-exact approximation has smaller elements everywhere. Quantities derived from computations with these meshes are denoted by a subscript

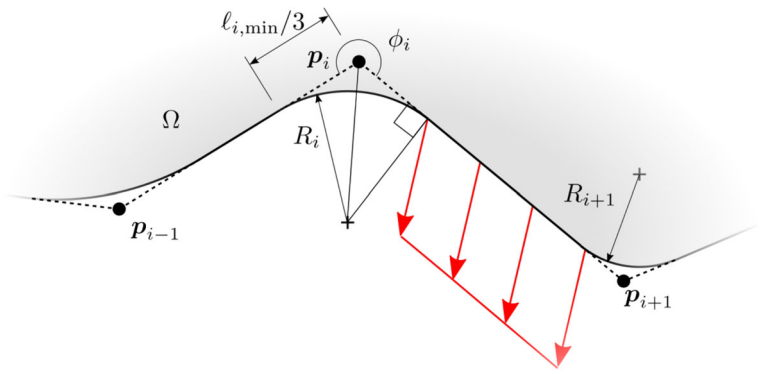


Fig. 6 Rounded corners. All corners of the polygons are rounded to avoid singularities

h. All approximate solutions to the BVPs were computed using the finite element solver FEniCS [1].

To recover the gradient of the displacement at the mesh nodes, superconvergent patch recovery (SPR) [39] was applied. Using this gradient, the stress and strain tensors could be interpolated over the domain using the element shape functions, allowing the computation of a pointwise error measure $E = \sqrt{(\boldsymbol{\sigma} - \boldsymbol{\sigma}_h) : (\boldsymbol{\epsilon} - \boldsymbol{\epsilon}_h)}$ in terms of the quasi-exact stress and strain tensors $\boldsymbol{\sigma}$ and $\boldsymbol{\epsilon}$ and their approximate counterparts and $\boldsymbol{\sigma}_h$ and $\boldsymbol{\epsilon}_h$. The associated total energy error norm is

$$\|\boldsymbol{u} - \boldsymbol{u}_h\|_E = \left(\int_\Omega E^2 \, \mathrm{d}\boldsymbol{x} \right)^{\frac{1}{2}}. \tag{16}$$

3.4 Model training

The model was trained exclusively on data from BVPs with $n = 10$ corners. To generate training data, the pointwise error was computed for each of the seven approximating meshes of every BVP. Then, 200 points were sampled uniformly in the intersection of the seven approximating mesh interiors. Note that by approximating the geometry using linear elements, the boundaries of the seven meshes are not identical and small regions exist near the boundary that are not part of the interior of all eight meshes. By requiring the samples to lie in the intersection of the interiors, we ensure that the samples lie in the interior of all meshes.

At every sample point, the pointwise error measure E was evaluated for each of the seven approximations. Together with the largest side length h of the element containing

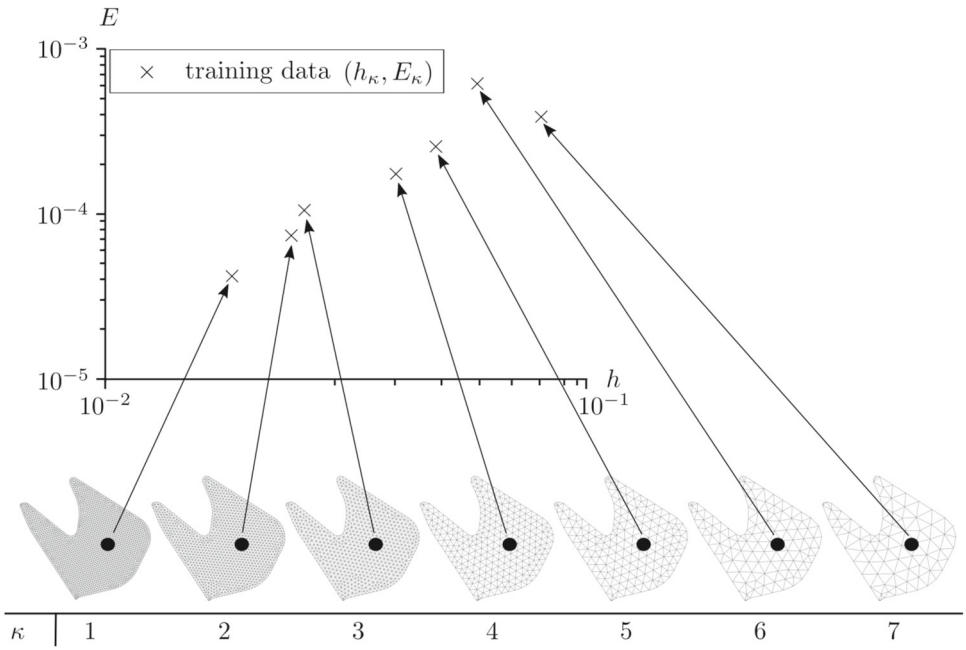


Fig. 7 Training data. The training data at a single sample point consists of the seven pairs (h_κ, E_κ), $\kappa = 1, 2, \ldots, 7$

the sample point in each of the eight meshes, the local convergence behavior is characterized (Fig. 7). A training sample consists of $\boldsymbol{x}$, $\boldsymbol{B}$ and the seven pairs (h_κ, E_κ) with $\kappa = 1, \ldots, 7$. In Appendix A, we illustrate the convergence behavior that an ideal model would extract from the training data.

The input features $\ln(\beta)$, $\boldsymbol{f}$, $\boldsymbol{u}_i$ and $\boldsymbol{t}_i$ were scaled by the respective maximum magnitude over the entire dataset, such that the value ranges are $[-1, 1]$ for β and $[0, 1]$ for all others.

The training data set consists of sample points from 9,161 BVPs, resulting in 1,832,200 individual samples. The samples were shuffled over all BVPs, such that all mini-batches contain samples from different BVPs in general. The validation set consists of 435,000 sample points from 2,175 BVPs. Samples from any particular BVP were only used for either the training or the validation set. The mean squared error was used to fit the log-transformed output from the regression model Eq. 3 to the log-transformed true error values $\ln(E_\kappa)$ of the samples. Hence, the loss function reads

$$\mathcal{L} = \frac{1}{b}\sum_{t=1}^{b}\frac{1}{7}\sum_{\kappa=1}^{7}\left[\ln\left(E(h_\kappa, \boldsymbol{x}^t, \boldsymbol{B}^t)\right) - \ln\left(E_\kappa^t\right)\right]^2. \tag{17}$$

The samples within each batch are numbered by the superscript t and $\boldsymbol{B}^t$ denotes the associated BVP. For training, we used the Adam optimizer [17] with a learning rate of 10^{-4}, $\beta_1 = 0.95$ and $\beta_2 = 0.999$ and a mini-batch size of $b = 128$. For inference, we used the network state at the epoch with the lowest validation loss, which occurred after 218 epochs.

4 Results and discussion

4.1 Evaluation method

As discussed already in Section 1, our approach aims at generating meshes without any prior information about the solution of a BVP. Without any prior information, one would in general have to rely on a uniform mesh. Therefore, we evaluate the performance of our approach by comparing the quality of meshes it suggests for so far unseen BVPs to the quality of uniform meshes that would require a similar computational effort for the solution. For this kind of evaluation, we use BVPs from a test set that is generated using the same parameters as described in Sections 3.1 and 3.2, but never used for training or validation. For these BVPs we aim to generate meshes fulfilling a constraint regarding the computational budget.

The uniform baseline meshes were generated for the BVPs with a constant element size

$$\bar{h} = \sqrt{\frac{2|\Omega|}{\sqrt{3}(V-1)}}, \tag{18}$$

where V is the desired number of vertices in the mesh, used as a proxy for computational cost. To ensure a fair comparison, we prescribed a budget of $V = 3{,}000$ vertices with a tolerance of roughly $\pm 5\%$ for both predicted and baseline meshes.

Subsequently, we used the trained model to predict values of C and M for each BVP. Given these parameters, we used the inference model Eq. 4 to generate an element size distribution h^*. We observe that h^* indicates refinement largely in the same regions as the quasi-exact meshes obtained with AMR (not shown), suggesting that the model has indeed learned some key aspects regarding the distribution of the error. However, the model has not captured the global convergence behavior of the BVPs perfectly. Some non-local effects are disregarded, which manifests in large gradients of the element size distribution in h^*, resulting in meshes of inferior quality (Fig. 8).

To regularize the prediction, we used the weighted geometric mean

$$h_V(\boldsymbol{x}, E^*) = \exp\left[s\ln\left(\bar{h}\right) + (1-s)\ln\left(h^*(\boldsymbol{x}, E^*)\right)\right] \tag{19}$$

by merging a certain amount of insight from h^* with a uniform distribution $\bar{h}$. This amount is controlled by the parameter $s \in [0, 1]$, where $s = 1$ would result in a uniform mesh. Tuning s on the training data set showed that a value of $s = 0.8$ performs best overall and yielded meshes of superior quality compared to both $\bar{h}$ and h^*.

By tuning E^* in Eq. 19, we can produce meshes with different element sizes, i.e. computational demand. Using a root finding algorithm, we can determine E_V^* such that $h_V(\boldsymbol{x}, E_V^*)$ represents the best element size distribution for the prescribed computational budget V. During the root finding, several evaluations of the number of vertices are performed. To avoid repeated meshing during tuning of E^*, we estimated the number of vertices to be expected in a mesh generated with h_V as described in Appendix B.

The element size field h_V was used to guide Gmsh's Frontal Delaunay algorithm for meshing the BVP. After meshing, we applied ten iterations of GETMe smoothing [30]. Figure 9 shows some examples of predicted meshes for BVPs from the test set.

Finally, the predicted and baseline meshes were compared for each BVP, using two different quality measures. We com-

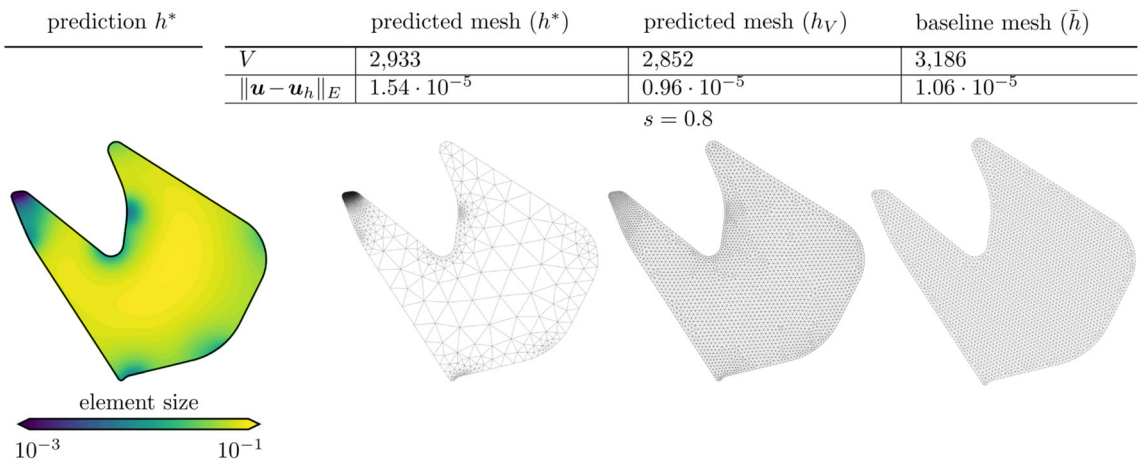


Fig. 8 Predicted mesh. The prediction h^* is shown for a BVP from the training set (left). On the right, we compare the meshes obtained from the prediction h^*, the modified ($s = 0.8$) prediction h_V and the the uniform baseline $\bar{h}$ in terms of the number of vertices V and total energy norm error

pared the meshes with regard to a) total energy norm error of the solution and b) solution time, comprising the meshing (including prediction of h_V for predicted meshes) and solving steps for both predicted and baseline meshes (Fig. 10).

For each BVP and quality measure, we computed a performance value P as the ratio of the measure value for the prediction to the measure value for the baseline. A performance value $P = 1.0$ indicates equal performance in terms of the respective measure, values of $P < 1.0$ indicate superior performance of the proposed method.

4.2 Performance

The proposed method achieves an expected increase in solution quality of 22% (measured in the total energy norm error of the solution) compared to the baseline with a uniform mesh of roughly the same computational demand. The computational cost is measured in terms of vertices in the mesh and the number of vertices in the prediction is within ±10% of the number of vertices in the baseline. The predictions yielded better meshes than the baseline for 94% of BVPs in the test set, making the improvement very reliable (Fig. 10a). Once the model is trained, the proposed method requires roughly

		a)	b)	c)	d)	e)	f)
V	prediction	2,959	2,961	2,974	3,004	2,996	2,918
	baseline	3,220	3,250	3,240	3,208	3,214	3,202
$\|u-u_h\|_E$	prediction	$0.140 \cdot 10^{-5}$	$0.339 \cdot 10^{-5}$	$0.951 \cdot 10^{-5}$	$0.770 \cdot 10^{-5}$	$0.417 \cdot 10^{-5}$	$0.216 \cdot 10^{-5}$
	baseline	$0.230 \cdot 10^{-5}$	$0.411 \cdot 10^{-5}$	$1.01 \cdot 10^{-5}$	$0.996 \cdot 10^{-5}$	$0.529 \cdot 10^{-5}$	$0.233 \cdot 10^{-5}$

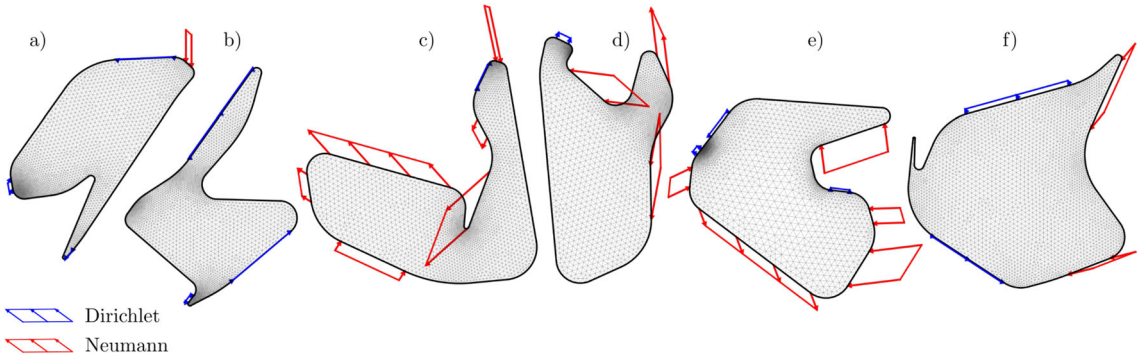
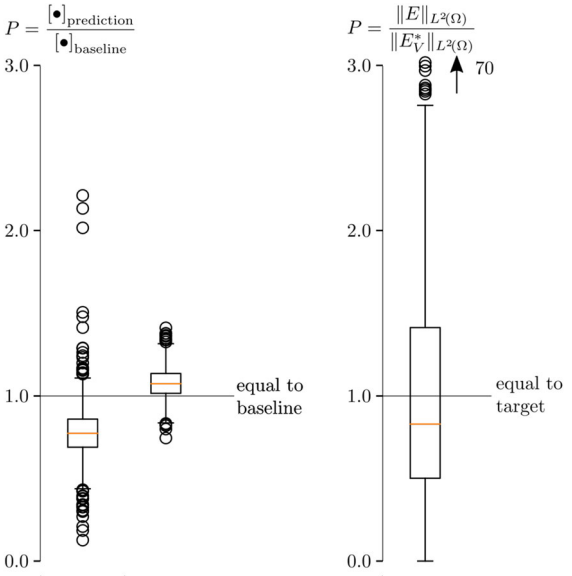


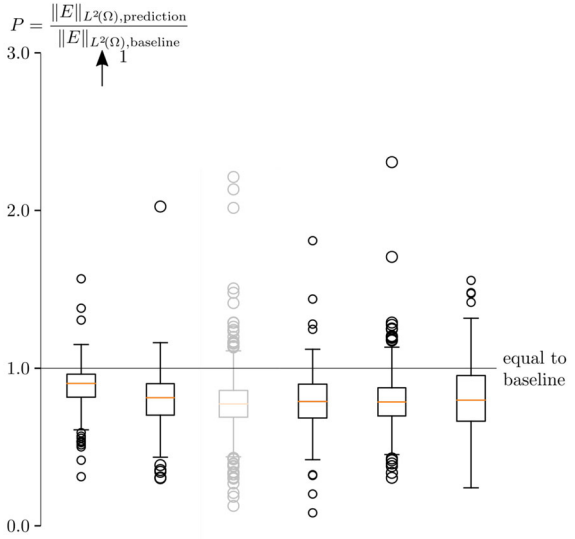
Fig. 9 Prediction examples. Predicted meshes are shown for some randomly chosen BVPs with $n = 10$ from the test set. The number of vertices V and total energy norm error are compared against the corresponding uniform baseline meshes (not shown)



	a)	b)
N	936	936
mean	0.774	1.078
median	0.774	1.078
max	2.212	1.410
$p(P < 1)$	0.944	0.200

	c)
N	936
mean	1.715
median	0.830
max	184.9
$p(P < 1)$	0.589

Fig. 10 Performance of the proposed method. The performance is quantified in terms of the two measures a) total energy norm error and b) solution time. Values of $P < 1.0$ signify an improvement over the baseline. Furthermore, the target accuracy of our method is assessed in c) by the ratio of the total energy norm error to the integrated pointwise target E_V^*, i.e., the expected total energy norm error. N is the number of BVPs evaluated for the respective metric



n	4	6	10	14	16	20
N	276	253	936	155	244	96
mean	0.892	0.798	0.774	0.793	0.800	0.810
median	0.904	0.814	0.774	0.789	0.786	0.798
max	3.017	2.025	2.212	1.808	2.307	1.556
$p(P < 1)$	0.864	0.955	0.944	0.910	0.898	0.896

Fig. 11 Generalization. The performance of the proposed method trained on BVPs with $n = 10$ corners when applied to domains with a different number of corners, measured in terms of total energy norm error is shown. We evaluated BVPs with $n = 4, 6, 14, 16, 20$ corners. For better comparison, the performance on the test set with $n = 10$ is shown in gray. Values $P < 1.0$ signify an improvement over the baseline. N is the number of BVPs evaluated for the respective class of BVP. Apparently, the improvements in the error norm that our architecture can achieve for polygons with 10 corners properly generalize also to polygons with different numbers of corners although no such polygons were present in the training data

8% more processing time in the mean to generate a mesh and solution than the uniform baseline (Fig. 10b). The predicted and baseline meshes are roughly equal in size (i.e., number of vertices), therefore the additional time taken is attributed entirely to the additional step of producing the predicted element size distribution h_V.

We measured the accuracy of the learned convergence behavior by comparing the total energy norm error to the expected energy norm error given the pointwise error target E^*. Most predictions undercut the expected error, although the mean is above parity, due to the many outliers (Fig. 10c).

4.3 Generalization

We evaluated the performance for groups of BVPs with 4, 6, 14, 16 and 20 corners in terms of the total energy norm error (Fig. 11). It can be seen that our model generalizes very well to such problems not present in the training data. This is attributed to the structure of our architecture, which decomposes the geometry into local neighborhoods, whose contributions are combined to make a global prediction. The performance of the proposed method is consistent across different classes of BVPs with $n \neq 10$, showing a reliable improvement over uniform meshes also for BVPs outside the training data domain.

4.4 Limitations and further work

Target accuracy Using the predicted h^* directly for meshing without a subsequent smoothing step by Eq. 19 leads to meshes of insufficient quality. This is attributed to the assumptions described in Section 2.2 that the convergence behavior can be modeled by a simple power law in a pointwise manner and only depends on the local neighborhood at each point, which is just a coarse approximation of the actual situation. We have shown that this problem can be addressed in a heuristic manner by interpolating between the predicted h^* and a uniform element size distribution

through Eq. 19, see also Fig. 8. However, the considerable number of examples where our procedure surpasses the error target E^* (Fig. 10c) suggests that developing a more sophisticated strategy could be beneficial. A possible reason for the shortcomings of the current approach is that the trained model neglects non-local interactions between BVP features, causing a larger than expected error at certain points in the domain. One approach to circumvent such a behavior could be to provide more global information about the BVP to the machine learning architecture in future work.

Objective function Herein, we have focused on the prediction of optimal element size distributions with respect to one particular error measure, namely the energy norm. Practical applications may require a different error measure, such as the error in the displacements or a consideration of several error measures. Replacing the current error norm by some alternative error norm (or a combination of several) in our framework would require, however, only a very minor modification.

Material Herein we focus on linear-elastic isotropic materials. A generalization to non-linear or anisotropic materials would require additional input parameters and likely a significantly increased amount of training data but is definitely a promising avenue of future research.

Geometry So far our implementation includes simple plane polygons without holes and with rounded reentrant corners, as described in Section 3.2. The extension to more complex geometries (e.g., domains including holes) is not straightforward and will require the development of meaningful BVP descriptors. The features of a two-dimensional boundary value problem over a simply connected domain can be enumerated in a meaningful way using lists, e.g. in counterclockwise order. However, with three-dimensional geometries, the connectivity of the vertices prohibits such a simple representation. Hence, an extension to three-dimensional problems will require a different representation, for example using graphs with features on vertices or edges, necessitating substantial changes to the proposed method.

Differential operator Herein we focus on BVPs governed by the differential operator of linear elasticity. However, the framework proposed could in fact be applied without substantial changes to any time-independent two-dimensional problem, i.e. also to BVPs like diffusion-reaction problems. In fact, by adding information about the differential operator to the input of the machine learning architecture, even a form of transfer learning appears possible where a neural network is trained for BVPs governed by differential operators from a specific training set and can then make predictions even for BVPs with an unknown (but somehow similar) differential operator.

5 Conclusion

In our paper, we introduced a new non-iterative mesh generation framework based on machine learning. After training the framework with a large dataset of various BVPs, it can predict non-uniform element size distributions for unseen BVPs in a single step. Our proof-of-concept study focused on isotropic two-dimensional linear elasticity and considered simply polygonal domains without holes (topological genus of zero). Our new method can generate optimized meshes to comply with certain accuracy requirements. Through numerous numerical examples, we demonstrated that for our problem class, the proposed trained ML model could produce meshes that provide 22% more accurate solutions than uniform meshes with a comparable computational demand.

A key strength of our framework lies in its ability to generalize beyond the training data. Remarkably, We found that our framework could be trained on polygonal domains with a specific number of corners (in this case $n = 10$) and still generalize well to different polygonal domains without requiring retraining. This generalization capability underscores the potential for long-term computational efficiency, as the initial training effort becomes negligible with repeated applications.

While these results are promising, it is important to note that this proof-of-concept study focuses on the introduction of a novel machine learning framework and the discussion of its foundations, not primarily on its practical application. In fact, as discussed in more detail in Section 4.4, the method we propose is still subject to a number of limitations. Addressing these limitations will be crucial to further develop our framework for broad-scale application in practice.

Appendix A Training data visualization

We perform a pointwise least squares fit of the convergence model Eq. 3 sampled at 5000 positions of one BVP and thereby determine the optimal parameters $C_{LS}(\boldsymbol{x}, \boldsymbol{B})$ and $M_{LS}(\boldsymbol{x}, \boldsymbol{B})$ for this BVPs in a point-by-point manner. On the right of Fig. 12, we have interpolated linearly between the sampled points to obtain a better visualization. The resulting plots represent what an ideal model would predict for the respective BVP and serve only illustrative purposes. Note that in contrast to this, the ANN described in Section 2.2.4

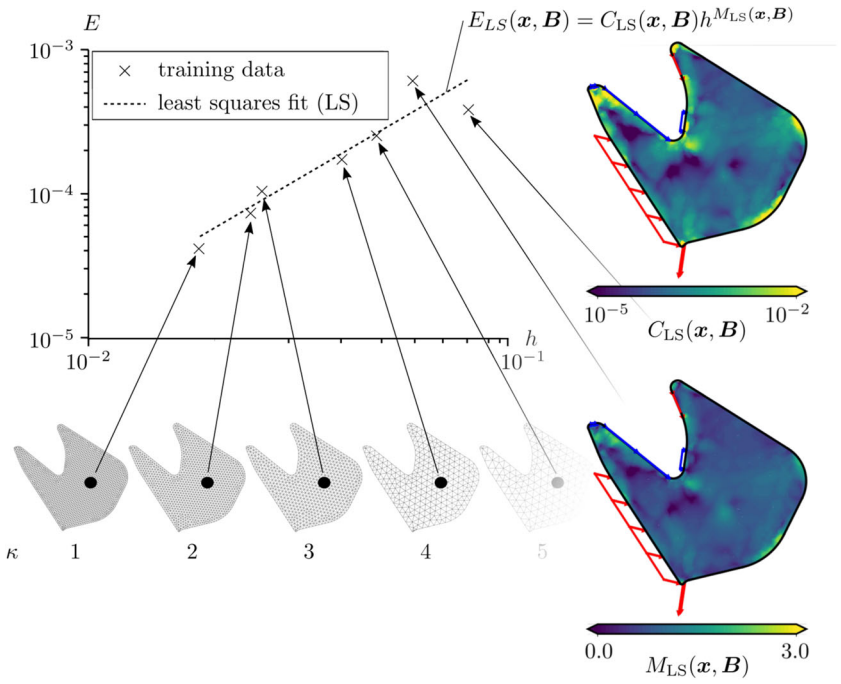


Fig. 12 Training data with pointwise least squares fit

instead fits a parametrized function to all samples and BVPs simultaneously.

Appendix B Root finding

Given the prediction h_V that is dependent on the parameter E^*, we employ a root finding algorithm to find E^*_V such that the resulting mesh has V vertices, within a certain tolerance.

To avoid computationally costly repeated meshing during root finding, we use a background mesh Ω_h with constant element size of 0.01 and compute

$$\tilde{V}(E^*) = \gamma \left(1 + \frac{2}{\sqrt{3}} \sum_{\mathcal{T}_p \in \Omega_h} \frac{|\mathcal{T}_p|}{h_V^2(\boldsymbol{x}_p, E^*)} + \frac{1}{2} \sum_{\mathcal{E}_q \in \Gamma_h} \frac{|\mathcal{E}_q|}{h_V(\boldsymbol{x}_q, E^*)}\right) \tag{20}$$

as an estimate of the number of vertices, where we sum over the interior elements $\mathcal{T}_p$ and boundary edges $\mathcal{E}_q$ and evaluate h_V at their centroid $\boldsymbol{x}_p$ and midpoint $\boldsymbol{x}_q$, respectively. $\gamma = 1.07$ was empirically found to improve the accuracy of the estimate.

Given the vertex estimate $\tilde{V}$, we define

$$\bar{\varphi}(E^*) = \tilde{V}(E^*) - V \quad \text{and seek} \quad E^*_V \quad \text{s.t.} \quad \bar{\varphi}(E^*_V) = 0. \tag{21}$$

In principle, this problem can be solving using Brent's method as implemented in [31], provided an initial bracket. However, we found that the root finding in our case is much more robust if instead we introduce $\eta = \ln(E^*)$, define

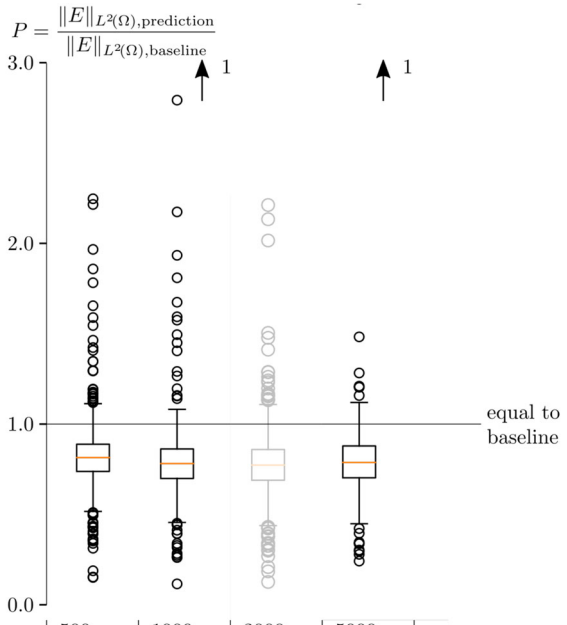
$$\varphi(\eta) = \ln\left(\tilde{V}(\exp(\eta))\right) - \ln(V) \quad \text{and seek} \quad \eta_V \quad \text{s.t.} \quad \varphi(\eta_V) = 0. \tag{22}$$

Note that the roots of Eqs. 21 and 22 imply the same E^*_V. We choose the initial bracket $\eta_{\text{init}} = \{-30, 0\}$.

Appendix C Additional performance analysis

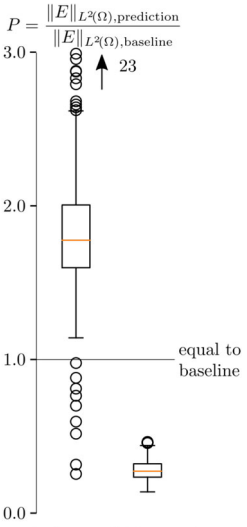
We evaluated the performance in terms of the total energy norm error for different vertex budgets (Fig. 13) compared to uniform meshes of roughly the same respective computational demand. The solution quality is consistent across all the different vertex budgets. This suggests that the proposed method is suitable for a wide range of settings.

Furthermore, we have compared the performance of the proposed method with respect to a second type of baseline mesh, obtained using an AMR method (Fig. 14). To generate the adaptive baseline meshes, we use the procedure for obtaining the quasi-exact solutions (Section 3.3), differing in the number of elements refined in each iteration. The $V_{\text{refine}} = \frac{V - V_{\mathcal{T}}}{3}$ elements showing the largest value of the ZZ refinement indicator are marked for refinement. Here, $V_{\mathcal{T}}$ is the number of vertices in the current mesh. The refinement procedure is halted once $V_{\mathcal{T}}$ is within $\pm 5\%$ of V. We use a vertex budget of $V = 3{,}000$.



V	500	1000	3000	5000
N	642	598	936	613
mean	0.821	0.795	0.774	0.790
median	0.814	0.782	0.774	0.788
max	2.247	3.237	2.212	3.351
$p(P < 1)$	0.924	0.936	0.944	0.931

Fig. 13 Performance with various vertex budgets. Comparison of the total energy norm error in the solution for predictions with various vertex budgets against uniform baseline meshes of roughly the same respective computational demand



	a)	b)
N	433	433
mean	1.917	0.280
median	1.776	0.273
max	7.886	0.463
$p(P < 1)$	0.021	1.000

Fig. 14 Performance against adaptive baseline. Comparison of a) total energy norm error of the solution and b) solution time with respect to an adaptive mesh

Compared to the adaptive baseline meshes, the proposed method produces meshes that are of inferior quality by 36%, in the mean, measured in terms of the total energy norm error. However, the time to obtain a solution amounts to only 31% of the time taken by the adaptive procedure. This speedup is very reliable, given that in the worst case, our method takes 46% of the time taken by the corresponding adaptive method. We believe that we can build on this result and improve the performance of the proposed methods in future work.

Acknowledgements This work was partly funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - 533187597.

Author Contributions M.L.: conceptualisation, methodology, writing - original draft, writing - review/editing, software, formal analysis, data curation, visualisation. K.L.: conceptualisation, methodology, writing - review/editing, formal analysis. R.C.A.: methodology, writing - review/editing, formal analysis. C.J.C.: funding acquisition, conceptualisation, writing - original draft, writing - review/editing, formal analysis.

Funding Open Access funding enabled and organized by Projekt DEAL.

Data Availability No datasets were generated or analysed during the current study.

References

1. Alnæs MS, Blechta J, Hake J, Johansson A, Kehlet B, Logg A, Wells GN. The FEniCS Project Version 1.5. Arch Numeric Softw. 2015;3(100):9–23. https://doi.org/10.11588/ans.2015.100.20553.
2. Auer T, Held M. Heuristics for the generation of random polygons. In: Proc. 8th canadian conference on computational geometry (CCCG'96). 1996. p. 38–43.
3. Babuska I, et al. Pollution Error in the h-Version of the Finite Element Method and the Local Quality of A-Posteriori Error Estimates. Tech. rep. Maryland: Institute for Physical Science and Technology; 1994.
4. Babuska I, Szabo B. On the rates of convergence of the finite element method. Int J Numer Meth Eng. 1982;18(3):323–41. https://doi.org/10.1002/nme.1620180302.
5. Chan CL, Scholz F, Takacs T. Locally refined quad meshing for linear elasticity problems based on convolutional neural networks. Eng Comput. 2022;38(5):4631–52. https://doi.org/10.1007/s00366-022-01677-8 . arXiv:2203.07843

6. Chedid R, Najjar N. Automatic finite-element mesh generation using artificial neural networks-part I: prediction of mesh density. IEEE Trans Magn. 1996;32(5):5173–8.
7. Chen G, Fidkowski K, Output-based error estimation and mesh adaptation using convolutional neural networks: application to a scalar advection-diffusion problem. AIAA Scitech,. Forum (Vol. 1 PartF). Reston, Virginia: American Institute of Aeronautics and Astronautics. 2020. p. 1–23. https://doi.org/10.2514/6.2020-1143.
8. Clevert D-A, Unterthiner T, Hochreiter S. Fast and accurate deep network learning by Exponential Linear Units (ELUs). 4th international conference on learning representations, 2016. 2015. p. 1–14. arXiv:1511.07289
9. Dugas C, et al. Incorporating second-order functional knowledge for better option pricing. In: Advances in neural information processing systems (vol. 13). MIT Press. 2000.
10. Dyck DN, Lowther DA, McFee S. Determining an approximate finite element mesh density using neural network techniques. IEEE Trans Magn. 1992;28(2):1767–70. https://doi.org/10.1109/20.124047.
11. Foucart C, Charous A, Lermusiaux PFJ. Deep reinforcement learning for adaptive mesh refinement. Proc Mach Learn Res. 2022;206:5997–6014. arXiv:2209.12351
12. Freymuth N, et al. Iterative Sizing Field Prediction for Adaptive Mesh Generation From Expert Demonstrations. 2024. arXiv:2406.14161
13. Fukushima K. Neocognitron: a self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. Biol Cybern. 1980;36(4):193–202. https://doi.org/10.1007/BF00344251.
14. Geuzaine C, Remacle J-F. Gmsh: a 3-D finite element mesh generator with built-in pre- and post-processing facilities. Int J Numer Meth Eng. 2009;79(11):1309–31. https://doi.org/10.1002/nme.2579.
15. Grätsch T, Bathe K-J. A posteriori error estimation techniques in practical finite element analysis. Comput Struct. 2005;83(4–5):235–65. https://doi.org/10.1016/j.compstruc.2004.08.011.
16. Huang K, et al. Machine learning-based optimal mesh generation in computational fluid dynamics. 2021. arXiv:2102.12923
17. Kingma DP, Ba J. Adam: a method for stochastic optimization. 3rd International Conference on Learning Representations, ICLR 2015 - Conference track proceedings. 2014. p. 1–15. arXiv:1412.6980
18. Lecun Y, et al. Gradient-based learning applied to document recognition. Proc IEEE. 1998;86(11):2278–324. https://doi.org/10.1109/5.726791.
19. Lin, M., Chen, Q., Yan, S. Network in network. 2nd International Conference on Learning Representations, ICLR 2014 - conference track proceedings. 2013. p. 1–10. arXiv:1312.4400
20. Lock C, et al. Meshing using neural networks for improving the efficiency of computer modelling. Eng Comput. 2023;39(6):3791–820. https://doi.org/10.1007/s00366-023-01812-z.
21. Logg A, Mardal K-A, Wells G (Eds). Automated solution of differential equations by the Finite element method (vol. 84). Berlin, Heidelberg: Springer Berlin Heidelberg; 2012.
22. Lorsung, C., & Farimani, A.B. (2022). MeshDQN: a deep reinforcement learning framework for improving meshes in computational fluid dynamics. arXiv:2212.01428
23. Manevitz L, Bitar A, Givoli D. Neural network time series forecasting of finite-element mesh adaptation. Neurocomputing. 2005;63:447–63. https://doi.org/10.1016/j.neucom.2004.06.009.
24. Nadeem Anjam Y, Ali A. On singularities of solution of the elasticity system in a bounded domain with angular corner points. Math Method Appl Sci. 2022;45(5):3124–43. https://doi.org/10.1002/mma.7980.
25. Patel AA, Safdari M. Smart adaptive mesh refinement with NEMoSys. In: Aiaa scitech 2021 forum. Reston, Virginia: American Institute of Aeronautics and Astronautics; 2021. p. 1–12 https://doi.org/10.2514/6.2021-1239. arXiv:2108.09304
26. Pfaff T, et al. Learning Mesh-Based Simulation with Graph Networks. In: *The ninth international conference on learning representations*. 2020. p. 1–18. arXiv: 2010.03409
27. Ronneberger O, Fischer P, Brox T. U-Net: Convolutional Networks for Biomedical Image Segmentation. 2015. p. 1–8. arXiv:1505.04597
28. Schubert S, et al. Circular convolutional neural networks for panoramic images and laser data. In: IEEE intelligent vehicles symposium, proceedings, 2019-June(Iv). 2019. p. 653–660. https://doi.org/10.1109/IVS.2019.8813862
29. Srivastava, et al. Dropout: a simple way to prevent neural networks from overfitting. J Mach Learn Res. 2014;15:1929–58.
30. Vartziotis D, et al. Mesh smoothing using the geometric element transformation method. Comput Methods Appl Mech Eng. 2008;197(45–48):3760–7. https://doi.org/10.1016/j.cma.2008.02.028.
31. Virtanen P, et al. SciPy 1.0: fundamental algorithms for scientific computing in Python. Nature Methods. 2020;17(3):261–72. https://doi.org/10.1038/s41592-019-0686-2.
32. Williams ML. Stress singularities resulting from various boundary conditions in angular corners of plates in extension. J Appl Mech. 1952;19(4):526–8. https://doi.org/10.1115/1.4010553.
33. Yang J, et al. Reinforcement learning for adaptive mesh refinement. 2021. arXiv:2103.01342
34. Zhang Z, Jimack PK, Wang H. MeshingNet3D: efficient generation of adapted tetrahedral meshes for computational mechanics. 2021.
35. Zhang, Z., et al. MeshingNet: a new mesh generation method based on deep learning. In: International Conference on Computational Science. 2020. p. 1–14. arXiv:2004.07016
36. Zhu JZ, Zienkiewicz OC. Adaptive techniques in the finite element method. Commun Appl Numeric Method. 1988;4(2):197–204. https://doi.org/10.1002/cnm.1630040210.
37. Zienkiewicz OC, Zhu JZ. A simple error estimator and adaptive procedure for practical engineering analysis. Int J Numer Meth Eng. 1987;24(2):337–57. https://doi.org/10.1002/nme.1620240206.
38. Zienkiewicz OC, Zhu JZ. Adaptivity and mesh generation. Int J Numer Meth Eng. 1991;32(4):783–810. https://doi.org/10.1002/nme.1620320409.
39. Zienkiewicz OC, Zhu JZ. The superconvergent patch recovery anda posteriori error estimates. Part: 1 the recovery technique. Int J Num Methods Eng. 1992;33(7):1331–64. https://doi.org/10.1002/nme.1620330702.