Engineering with Computers
https://doi.org/10.1007/s00366-024-01955-7

ORIGINAL ARTICLE

NeuFENet: neural finite element solutions with theoretical bounds for parametric PDEs

Biswajit Khara[1] · Aditya Balu[1] · Ameya Joshi[2] · Soumik Sarkar[1] · Chinmay Hegde[2] · Adarsh Krishnamurthy[1] · Baskar Ganapathysubramanian[1]

Received: 13 April 2023 / Accepted: 4 February 2024

Abstract
We consider a mesh-based approach for training a neural network to produce field predictions of solutions to parametric partial differential equations (PDEs). This approach contrasts current approaches for "neural PDE solvers" that employ collocation-based methods to make pointwise predictions of solutions to PDEs. This approach has the advantage of naturally enforcing different boundary conditions as well as ease of invoking well-developed PDE theory—including analysis of numerical stability and convergence—to obtain capacity bounds for our proposed neural networks in discretized domains. We explore our mesh-based strategy, called *NeuFENet*, using a weighted Galerkin loss function based on the Finite Element Method (FEM) on a parametric elliptic PDE. The weighted Galerkin loss (FEM loss) is similar to an energy functional that produces improved solutions, satisfies *a priori* mesh convergence, and can model Dirichlet and Neumann boundary conditions. We prove theoretically, and illustrate with experiments, convergence results analogous to mesh convergence analysis deployed in finite element solutions to PDEs. These results suggest that a mesh-based neural network approach serves as a promising approach for solving parametric PDEs with theoretical bounds.

Keywords Neural solvers · Deep learning · Physics informed learning · Parametric PDE · Data-free modeling

1 Introduction

Scientific machine learning is an emerging field combining machine learning developments with scientific computation. This field has witnessed a variety of approaches that deploy neural networks to solve partial differential equations (PDE). Such *neural PDE solvers* provide a very different strategy for solving differential equations than traditional numerical methods; they primarily rely on *optimization* techniques rather than the exact solution of systems of equations. The seminal paper on Physics Informed Neural Networks (PINNs) [1] initiated this recent explosion in this line of work.

Neural PDE solvers span a wide spectrum in terms of the amount of data usage. Some methods are "data-driven" [2–5], where the solution to a given PDE is constructed from available experimental data or the underlying PDE is inferred from available data (commonly termed as the discovery of hidden physics). In contrast, at the other end of the spectrum are the so-called "data-free" methods that do not rely on input–output pairs but solely use the PDE and the boundary conditions to obtain the solution. In the past few years, many such methods have been proposed [5–17]. Our work in this paper follows the latter data-free approach.

The core of neural PDE solvers is deep neural networks, which can represent arbitrarily complicated functions from the input to the output domain and, therefore, can approximate the PDE solution. Most neural methods use a pointwise prediction framework (also known as implicit neural networks [18]). These pointwise prediction frameworks take $\mathbf{x} \in D$ (the spatial coordinates of the field) as input and produce an output solution value of $u(\mathbf{x})$ (the solution field value at $\mathbf{x}$) as shown in Fig. 1a. Thus, the neural PDE solvers create a mapping between the input domain D to the range of the solution. Due to a pointwise prediction framework,

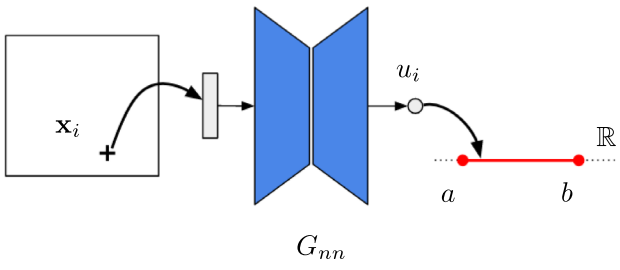
✉ Adarsh Krishnamurthy
adarsh@iastate.edu

✉ Baskar Ganapathysubramanian
baskarg@iastate.edu

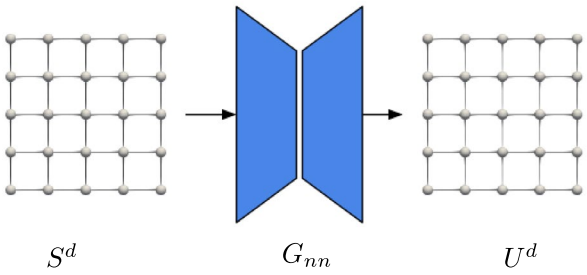
[1] Iowa State University, Ames, USA

[2] New York University, New York, USA

Published online: 10 April 2024



(a) *Conventional neural view: domain-to-range mapping*



(b) *A more complex view: field-to-field mapping*

Fig. 1 **a** Several neural methods (like [1, 45]) are trained to produce point predictions: $G_{nn} : D \to \mathbb{R}$, which are easier to train, but more difficult to analyze and converge [54, 55], **b** In the *NeuFENet* approach, we train a neural network to produce a discretized field solution over a mesh. Such an approach directly links powerful PDE analysis techniques at the cost of a larger network. The terms S^d and U^d can be considered the discrete version of any meaningful pair of input and output relevant to the PDE; these will be made precise in the next section

these methods do not require a mesh and thus rely on collocating points from the domain. To take advantage of the modern stochastic gradient descent (SGD) based methods, this set of collocation points are often selected in a random or quasi-random manner [19]. The trained network approximates the discrete solution via a complicated and nonlinear mapping. This approach contrasts classical numerical PDE approaches, which usually rely on a linear combination of local functions with limited differentiability (even when the exact solution may be analytic). However, such pointwise prediction neural methods do not naturally account for the domain topology. In particular, the "local" nature of the solution and the sparsity of matrices that emerge naturally in classical methods are missing in these neural methods.

Some researchers have explored the idea of using classical methods such as finite difference methods (FDM) and finite volume methods (FVM) to construct neural architectures for solving PDEs [20–22]. Inspired by traditional numerical techniques, these frameworks construct a mapping between an input field and the solution field while using the discretization techniques associated with conventional numerical methods. These methods take advantage of the "local" nature of the solution and sparsity of matrices, similar to traditional numerical methods. In particular, mathematical concepts from finite element methods (FEM) are naturally translatable to neural networks (quadrature can be represented as convolutions) and provide interesting possibilities, including variational arguments (monotone convergence to the solution), mesh convergence, basis order-based convergence, and natural incorporation of boundary conditions. The current work builds upon these ideas.

Neural architecture: In this paper, we develop a finite element (FEM)-based neural architecture for solving PDEs. Figure 1b shows an abstract outline of this idea where the mapping is obtained with the use of convolutional neural networks, a specific class of network architectures specialized in learning from discrete domains, such as input field S^d and the output field U^d, as shown in Fig. 1b. The nature of the input field S^d and the output field U^d (in Fig. 1b) would depend on the actual PDE under consideration and will be made more concrete in later sections. There are several benefits in developing a finite element method (FEM)-based neural architecture. FEM-based numerical methods are often backed by a well-developed and elegant theory that connects the discretization of the domain (in terms of element/cell dimension, h) and the properties of the basis functions used to approximate the field (in terms of polynomial order, α) with the quality of the ensuing numerical solution to the PDE.

In particular, numerical stability arguments and *a priori* error estimates allow users to reason about the accuracy, robustness, and convergence [23, 24]. Such theoretical arguments rely on the spatial discretization of the domain and properties of the basis functions in finite elements.[1] ReLU activation functions are continuous piecewise functions and, therefore, have been the subject of study in relation to finite element basis functions [28]. FEM has been used in conjunction with neural networks for solving both forward and inverse problems, primarily with the help of mesh-based discretizations [29–32]. These methods construct the neural network in such a way that imitates the action of the stiffness matrix on the gradient of the unknowns, i.e., the neural architecture is designed to mimic FEM. Our present work differs from these works, because we do not introduce the structure of FEM in the network architecture. However, our

[1] In contrast, state-of-art neural methods allow us to use basis functions beyond polynomials or Fourier bases and approximate much more complicated mappings. Although such methods *can* be analyzed theoretically, the estimates are often impractical [25–27] This is a very active area of research, and we expect tighter estimates in the future.

method does share some similarities with these methods in mapping material properties to solution and using an energy-based loss function [31, 32].

Among the more recent works, VarNet [33] introduces a neural method that uses the weak form of the PDE and uses randomly sampled training points for training the model; it also uses separate test functions for the weak form. A similar method is found in graph convolutional network (GCN) [34], which also utilizes the weak formulation and separate test function space as compared to the solution space. However, GCN imposes the boundary conditions exactly. FEA-Net [35], on the other hand, is a data-based method that does use FEM approximations but not the PDE itself. In comparison, our method is data-free (i.e., we do not require {input, solution} pair to solve the PDE). However, we share a similarity with FEA-Net in calculating the derivatives of the solution function over the domain using convolution kernels (as described in Sect. 4.3).

Loss functions: Having decided on borrowing the discretization scheme from FEM, multiple avenues exist to define the loss function. First, we need to define the spatial derivatives at each quadrature point. In FEM, this is done by directly differentiating the basis functions. Pointwise prediction methods perform this by differentiating the neural network with respect to the input variable. The differentiation process in the numerical method is straightforward and interpretable, while that is not necessarily the case in pointwise neural methods. Once the spatial derivatives are defined via the basis functions, we can either compare the weak form against the predefined basis functions from the test function space and perform a residual minimization or use an energy minimization approach. There have been some efforts in introducing weak formulation in physics-based neural solvers such as [6, 33–35], but most of them are still collocation-based, and satisfy the Dirichlet boundary conditions approximately. In the present work, we choose the Rayleigh–Ritz (RR) method [36, 37]. The RR method states that the solution to a PDE must be the stationary point of some functional (i.e., "energy") under certain conditions. We note that the RR method has been used in a neural network for solving PDEs before [13, 38–43]. However, the approach used in these methods closely matches the pointwise prediction approach outlined in Fig. 1a, in contrast to our proposed approach.

Boundary conditions: The imposition of boundary conditions can also be challenging in neural methods. Very few neural methods satisfy/apply the boundary conditions exactly [34, 44–47], with most methods relying on approximate approaches [1, 7, 48] usually by including an additional loss function corresponding to the imposed boundary conditions. It has been shown by Van der Meer et al. [49] and Wang et al. [50] that these losses have to be carefully weighed, making this a non-trivial exercise in hyperparameter tuning. This hyperparameter sensitivity underlines the difficulty of applying the boundary conditions in a neural network-based method (or simply a neural method). Also, note that the method by Yu et al. [38] (using RR method) is unable to apply the Dirichlet boundary conditions precisely and consequently makes use of a penalty-based approach for imposing a Dirichlet boundary condition, which we avoid altogether.

Parametric PDEs: Going beyond a single PDE, there is growing interest in neural approaches that solve parametric PDEs (i.e., PDEs defined by a family of parametrized boundary conditions or coefficient fields). Most neural PDE methods have been limited to solving for a single instance of the PDE than a class of parametric solutions. Extending an instance PDE solver into a parametric PDE solver can greatly augment rapid design exploration, as alluded to in SimNet [51] and Wang and Perdikaris [52], where the authors build a conventional implicit neural solver for parametric PDEs.

In this paper, we build upon recent efforts that train networks to predict the full-field solution [15, 20, 53] on parametric PDEs. Our contributions are as follows:

1. We present an algorithm that bridges traditional numerical and neural methods. The neural network is designed to map inputs to the discretized field solution u. However, the neural network is *not responsible* for ensuring the spatial differentiability of the solution. Rather, the discrete field solution relies on traditional numerical methods (and associated numerical differentiation and quadrature) to construct the loss function. Such an approach allows the natural incorporation of different boundary conditions and allows *a priori* error estimates.
2. We define the loss functions based on the Rayleigh–Ritz method coupled with the approximation scheme provided by a continuous Galerkin FEM. By defining such loss functions, we utilize function spaces with appropriate differentiability. This also accounts for the "local" nature of the solution, resulting in computationally efficient loss evaluations.
3. We prove error convergence (similar to conventional mesh convergence) for a particular class of PDEs.
4. We demonstrate *NeuFENet*'s performance on linear Poisson equation in 2D and 3D with both Dirichlet and Neumann boundary conditions. Further, we test the parametric capability of this method on Poisson's equation by considering a case involving stochastic diffusivity, which requires access to a parametric PDE solver.

The rest of the paper is arranged as follows: the definitions and terminologies regarding the parametric Poisson's equation are introduced in Sect. 2 and the mathematical formulations are described in detail in Sect. 3. The implementation

aspects of *NeuFENet* are described in Sect. 4. Section 5 presents a theoretical analysis of the errors, and finally, computational results are presented in Sect. 6.

2 Mathematical preliminaries

Consider a bounded open (spatial) domain $D \in \mathbb{R}^n, n \geq 2$ with a Lipschitz continuous boundary $\Gamma = \partial D$. We will denote the domain variable as $\mathbf{x}$, where the boldface denotes a vector or tuple of real numbers. In $\mathbb{R}^n$, we have $\mathbf{x} = (x_1, x_2, \ldots, x_n)$; but for 2D and 3D domains, we will frequently use the notation $\mathbf{x} = (x, y)$ and $\mathbf{x} = (x, y, z)$, respectively. Consider also a probability space (Ω, F, P), where Ω is the sample space, F is the σ-algebra of the subsets of Ω and P, a probability measure. We consider an abstract PDE on the function $u : D \times \Omega \to \mathbb{R}$ as

$$\mathcal{N}[u; s(\mathbf{x}, \omega)] = f(\mathbf{x}), \quad \mathbf{x} \in D, \ \omega \in \Omega \tag{1a}$$

$$\mathcal{B}[u] = g(\mathbf{x}), \quad \mathbf{x} \in \Gamma. \tag{1b}$$

Here, $\mathcal{N}$ is a differential operator (possibly nonlinear) operating on a function u. The differential equation depends on the input-data (e.g., material property) s which in turn is a function of the domain variable $\mathbf{x}$ and parameter $\omega \in \Omega$. Thus, $\mathcal{N}$ is essentially a family of PDE's parameterized by ω.[2] $\mathcal{B}$ is a boundary operator on u. In general, there can be multiple boundary operators for different part of the boundary Γ.

Given a PDE along with some boundary conditions, such as the one presented in Eq. 1, the goal is to find a solution u that satisfies Eq. 1 as accurately as possible. Previous works such as [1, 7, 45] seek to find a pointwise mapping $u : D \to \mathbb{R}$. Here (see next section), by coupling deep neural networks with numerical methods, we explore other mappings to retrieve a discrete field solution.

In this work, we focus on the Poisson's equation considering both Dirichlet and Neumann boundary conditions, along with a heterogeneous and stochastic diffusivity

$$-\boldsymbol{\nabla} \cdot (\nu(\mathbf{x}, \omega)\boldsymbol{\nabla} u) = f(\mathbf{x}) \text{ in } D \tag{2}$$

along with the boundary conditions

$$u = g \text{ on } \Gamma_D \tag{3}$$

$$\frac{\partial u}{\partial n} = h \text{ on } \Gamma_N, \tag{4}$$

where ν is the *permeability* (or *diffusivity*) which depends on both $\mathbf{x}$ and the random variable ω; and f is the forcing. In relation to Eq. 1, ν plays the role of the data s. Γ_D and Γ_N are the boundaries of the domain D where Dirichlet and Neumann conditions are specified respectively. We will assume that $\partial D = \Gamma = \Gamma_D \cup \Gamma_N$.

2.1 Poisson's equation in heterogeneous media

We are mostly interested in the problem of a steady state mass (or heat) transfer through an inhomogeneous medium (material). This essentially means that the material has different properties at different points. The only material property appearing in the Poisson's equation (Eq. 2) is $\nu(\mathbf{x})$, and thus, the inhomogeneity can be modeled by a spatially varying ν, i.e., $\nu = \nu(\mathbf{x})$. The equation and the BC's are given by

$$-\boldsymbol{\nabla} \cdot (\tilde{\nu}(\mathbf{x})\boldsymbol{\nabla} u) = 0 \text{ in } D \tag{5a}$$

$$u(0, y) = 1 \tag{5b}$$

$$u(1, y) = 0 \tag{5c}$$

$$\frac{\partial u}{\partial y}(x, 0) = 0 \tag{5d}$$

$$\frac{\partial u}{\partial y}(x, 1) = 0, \tag{5e}$$

where D is a hypercube domain in $\mathbb{R}^n$, $n = 2, 3$. The diffusivity/permeability $\tilde{\nu}$ is heterogeneous with respect to $\mathbf{x}$ and is also parameterized by $\omega \in \Omega$. The specific form of $\tilde{\nu}$ is given in Eq. 51.

3 Formulations

3.1 Neural approximation of the solution

Instead of seeking a mapping between the domain and an interval on the real line (Fig. 1a), we seek a mapping between the input s and the full-field solution u in the discrete spaces (Fig. 1b). S^d denotes the discrete representation of the known quantity s. S^d could be either available only at discrete points (perhaps from some experimental data) or, in many cases, s might be known in a functional form, and thus, S^d will be simply the values of s evaluated on the discrete points. Therefore, if we denote a *NeuFENet* (see

[2] While a probability-based definition of ω is not needed for defining a parameteric PDE, we choose this definition for two reasons. First, such a formulation allows easy extension to the stochastic PDE case. Second, such a formulation will allow using expectation-based arguments in the analysis of convergence.

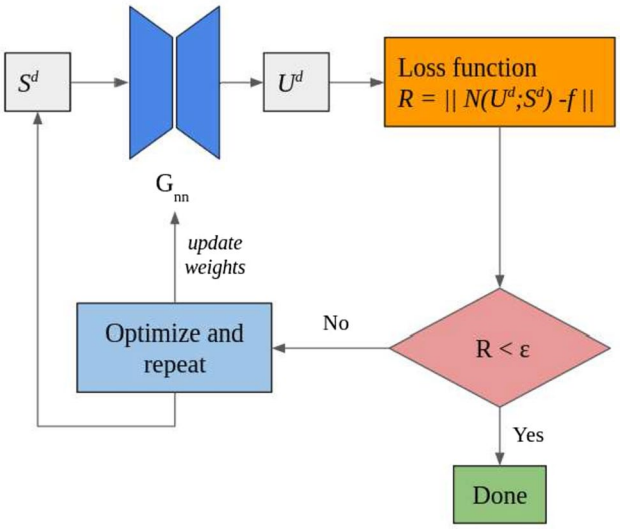


Fig. 2 *NeuFENet* flowchart. The approximation is discussed in Sect. 3.1, the loss function is discussed in Sect. 3.2, and the training algorithm is discussed in Sect. 3.3

Fig. 1b) network by G_{nn}, then G_{nn} takes as input a discrete or functional representation of s and outputs a discrete solution field U_θ^d as

$$U_\theta^d = G_{nn}\left(S^d;\theta\right), \tag{6}$$

where θ denotes the network parameters. The mathematical formulations presented in this section only assume a suitable neural network that provides the mapping mentioned above between S^d and U^d. The network architecture is discussed in Sect. 4.1. A full flowchart of the method is shown in Fig. 2.

An untrained network, as expected, will produce a mapping that does not satisfy the discrete PDE and will possess a large error. We aim to bring this error down to an acceptable level, thereby reaching a solution that is "close enough" to the exact solution. As explained below, we do this by designing the loss function based on major ideas from the classical numerical methods.

3.2 Loss functions inspired by variational methods

The design of the loss function, along with the choice of the neural mapping, forms the central part of our approach. The finite element-based loss function is inspired by the Galerkin formulation of an elliptic PDE as well as the Rayleigh–Ritz method. In this case, we actually construct a function of certain regularity in the domain variable $\mathbf{x}$, as opposed to just assuming a function of certain differentiability at the collocation points.

Suppose $H^1(D) = W^{1,2}(D)$ denotes the Sobolev space of functions whose first derivatives are square integrable. Define the space V as

$$V = \left\{v \in H^1(D) \,:\, v(0,y) = 1, v(1,y) = 0, \|v\|_V < \infty\right\}, \tag{7}$$

where $\|v\|_V$ is defined as

$$\|v\|_V = \int_D v(\mathbf{x})|\boldsymbol{\nabla} v|^2 \mathrm{d}\mathbf{x}. \tag{8}$$

Then, the Galerkin formulation for the Poisson's equation presented in Eq. 5 is to find $u \in V$, such that

$$B(u,w) = L(w) \;\; \forall w \in V, \tag{9}$$

where

$$B(u,w) = \int_D v(\boldsymbol{\nabla} w \cdot \boldsymbol{\nabla} u)\mathrm{d}\mathbf{x} \tag{10a}$$

$$L(w) = \int_D wf\,\mathrm{d}\mathbf{x}. \tag{10b}$$

Equation 9 actually represents a number of equations each for a different test function w taken from the space V. Thus, if V is discretized such that it can be represented by a finite basis, then the Galerkin formulation (Eq. 9) yields a finite system of algebraic equations.

From the theory of variational calculus [36, 56], it is also known that Eq. 9 is the Euler–Lagrange equation of the following functional $J(u)$ of $u \in V$:

$$J(u) = \frac{1}{2}B(u,u) - L(u). \tag{11}$$

Therefore, the solution u can also be written as the minimizer of the cost function J

$$u = \underset{u \in V}{\arg\min}\, J(u). \tag{12}$$

In *NeuFENet*, we make use of this functional J, but instead of minimizing it against the solution u, we minimize it against the network parameters θ. Since V is an infinite dimensional space, we need to discretize it to a finite subspace where we can evaluate J. For this, let $\mathcal{K}^h$ be a discretization of D into n_{el} finite elements K_i such that $\cup_{n_{el}} K_i = D$. Then, the discrete function space V^h is defined as

$$V^h = \left\{v^h \in V \,:\, v^h|_K \in P_m(K),\; K \in \mathcal{K}^h\right\}, \tag{13}$$

where $P_m(K)$ denotes the set of polynomial functions of degree m define on K. Let the dimension of V^h be N, which essentially means that the number of unknowns in the domain is also N. Suppose $\{\phi_i(\mathbf{x})\}_{i=1}^N$ is a suitable basis that span V^h. Then, any function $u^h \in V^h$ can be written as

$$u^h(\mathbf{x}) = \sum_{i=1}^{N} \phi_i(\mathbf{x}) U_i, \tag{14}$$

where U_i are the function values at the nodal points in the mesh $\mathcal{K}^h$. Then, Eq. 12 can be rewritten for u^h as

$$u^h = \underset{u^h \in V^h}{\arg\min}\, J(u^h). \tag{15}$$

This minimization of the energy functional in a finite-dimensional space is commonly known as the Rayleigh–Ritz method. However, in the presence of a neural network, we minimize J with respect to the network parameter θ instead of u^h. For this, we need to first explicitly state the θ dependence of u^h. This can be done by a slight modification to Eq. 14 as below

$$u^h(\mathbf{x};\theta) = \sum_{i=1}^{N} \phi_i(\mathbf{x}) U_i(\theta), \tag{16}$$

along with a generalization of the function space V^h as

$$V_\theta^{h'} = \left\{ v(\theta) \in V^h \,:\, v(x=0,y;\theta) = 1, v(x=1,y;\theta) = 0 \right\}. \tag{17}$$

Then, we can finally write down the *NeuFENet* solution $u^h_{\theta^*}$ in two steps

$$\theta^* = \underset{\theta \in \Theta}{\arg\min}\, J(u^h(\mathbf{x};\theta)) \tag{18a}$$

$$u^h_{\theta^*} = u^h(\mathbf{x};\theta^*), \tag{18b}$$

where $u^h_{\theta^*} \in V_\theta^{h'}$.

The discussion leading to Eq. 18 is based on a non-parametric diffusivity, i.e., $\tilde{\nu} = \nu(\mathbf{x})$. Extension of this formulation to the parametric PDE case is straightforward. Specifically, for a parameterized ω, i.e., $\tilde{\nu} = \nu(\mathbf{x}, \omega)$, the function space $V_\theta^{h'}$ is modified as

$$\begin{aligned} V_\theta^h = \Big\{ v \,:\, \|v(\mathbf{x},\omega;\theta)\|_{V_\theta^h} < \infty,\ v(x=0,y,\omega;\theta) = 1, \\ v(x=1,y,\omega;\theta) = 0, \} \end{aligned} \tag{19}$$

where $\|\cdot\|_{V_\theta^h}$ is the energy norm

$$\|v\|^2_{V_\theta^h} = \mathbb{E}_{\omega\sim\Omega}\left[\int_D \nu(\mathbf{x},\omega;\theta) |\nabla v|^2 \mathrm{d}\mathbf{x} \right]. \tag{20}$$

With this choice of function space, the *NeuFENet* loss function can be written as

$$L(\theta) = \mathbb{E}_{\omega\in\Omega} J\left(u^h(\mathbf{x},\omega;\theta)\right). \tag{21}$$

The right-hand side of Eq. 21 involves two integrations: one over the spatial domain D and the other an expectation over Ω. The integration over D is evaluated numerically using Gaussian quadratures rules. And the expectation over Ω is evaluated approximately by a summation over a finite number of samples, that is

$$\hat{L}(\theta) = \frac{1}{N_s} \sum_{i=1}^{N_s} J\left(s(\omega_i), u^h(\mathbf{x},\omega_i);\theta\right). \tag{22}$$

The loss function $\hat{L}(\theta)$ is now just a function of θ and we can minimize it with respect to θ

$$\theta^* = \underset{\theta\in\Theta}{\arg\min}\, \hat{L}(\theta) \tag{23a}$$

$$u^h_{\theta^*} = u^h(\mathbf{x},\omega;\theta^*). \tag{23b}$$

Remark 1 To simplify notations when we analyze errors in Sect. 5, we make a distinction between two representations of θ^*: one of them is the theoretical minimum (denoted by $\tilde{\theta}$ in Sect. 5) and the other is the *actual* set of parameters (denoted by θ in Sect. 5) obtained by optimizing Eq. 23a with an optimization algorithm. This terminology then spawns two variants for $u^h_{\theta^*}$: $u_{\tilde{\theta}}$ and u_θ, respectively.

3.3 Training algorithm for *NeuFENet*

We provide two versions of the training algorithm. (i) an algorithm for computing the solution for an instance of a PDE and (ii) an algorithm for approximating the solution for a parametric PDE. The model architecture and the loss function remain the same for both. For the instance version, we use a simple approach as explained in Algorithm 1. While sampling from a distribution of coefficients/forcing field for a parametric PDE, we employ the mini-batch-based optimization approach as explained in Algorithm 2. The sampling of the known quantities can be performed using any random or qseudo-random sequence (see Sect. 6.3 for an example). For training the neural network, we predict the solution field using sampled inputs and compute the loss using the loss function derived above. We employ gradient descent-based optimizers such as Adam [57] to perform the numerical optimization.

Algorithm 1 Algorithm for instance PDE solver

Require: S^d, α, TOL and `max_epoch` ▷ α = learning rate

Initialize G_{nn}
for `epoch` $\leftarrow$ 1 to `max_epoch` **do**
  $U_\theta^d \leftarrow G_{nn}(S^d)$
  Apply Dirichlet boundary conditions to U_θ^d
  $loss = L(S^d, U_\theta^d)$
  $\theta \leftarrow optimizer(\theta, \alpha, \nabla_\theta(loss))$
  if $loss <$ TOL **then**
    `break`
  end if
end for

Algorithm 2 Algorithm for parametric PDE solver

Require: $\{S_i^d\}_{i=1}^{N_s}$, α, TOL and `max_epoch` ▷ α = learning rate

Initialize G_{nn}
for `epoch` $\leftarrow$ 1 to `max_epoch` **do**
  for `mb` $\leftarrow$ 1 to `max_mini_batches` **do**
    Sample $(S^d)_{mb}$ from the set $\{S_i^d\}_{i=1}^{N_s}$
    $(U_\theta^d)_{mb} \leftarrow G_{nn}\left((S^d)_{mb}\right)$
    Apply Dirichlet boundary conditions on $(U_\theta^d)_{mb}$
    $loss_{mb} = L((S_\theta^d)_{mb}, (U_\theta^d)_{mb})$
    $\theta \leftarrow optimizer(\theta, \alpha, \nabla_\theta(loss_{mb}))$
  end for
end for

4 Implementations

4.1 Model architecture for *NeuFENet*

Due to the structured grid representation of $\mathcal{S}^d$ and similarly structured representation of U_θ^d, deep convolutional neural networks are a natural choice of network architecture. The spatial localization of convolutional neural networks helps learn the local interaction between the discrete points. Since the network takes an input of a discrete grid representation (similar to an image, possibly with multiple channels) and predicts an output of the solution field of a discrete grid representation (similar to an image, possibly with multiple channels), this is considered to be similar to an image segmentation or image-to-image translation task in computer vision. U-Nets [58, 59] have been known to be effective for applications such as semantic segmentation and image reconstruction. Due to its success in diverse applications, we choose U-Net architecture for *NeuFENet*. The architecture of the network is shown in Fig. 5. First, a block of convolution and instance normalization is applied. Then, the output is saved for later use via skip-connection. This intermediate output is then down-sampled at a lower resolution for a subsequent block of convolution, instance normalization layers. This process is repeated two times. Now, the upsampling starts where the saved outputs of similar dimensions are concatenated with the output of upsampling for creating the skip connections, followed by a convolution layer. LeakyReLU activation was used for all the intermediate layers, with Sigmoid activation for the final layer.

4.2 Applying boundary conditions

In *NeuFENet*, the Dirichlet boundary conditions are applied exactly. The output U_θ^d does not contain the boundary conditions. Thus, a small post-processing step is done to the network output to force the Dirichlet boundary conditions onto the respective boundaries. This can be done in a differentiable manner in modern machine learning software libraries such as PyTorch [60]. This exact imposition of Dirichlet boundary conditions allows the training process to be much smoother and interpretable, because there is no penalty term involved

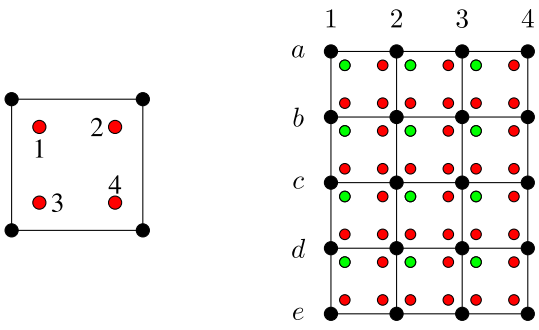


Fig. 3 (Left) A single 2D element in FEM, with black dots denoting "nodes" and red dots denoting 2×2 Gauss quadrature points. (Right) A finite element mesh, with 4×3 linear elements and 5×4 nodes. Each of these elements contains Gauss points for integration to be performed within that element. Within each element, the "first" quadrature point (marked "1" on left) is marked green, and others red

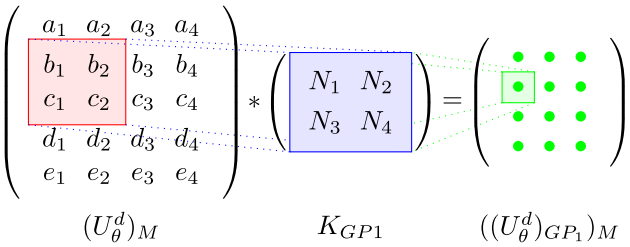


Fig. 4 Quadrature quantity evaluation in FEM context. $(U^d_\theta)_M$ is the matrix view of the nodal values.K_{GP1} is kernel containing the basis function values at "gauss point - 1" (top left corner). This convolution results in the function values evaluated at the Gauss point "1" of each element (marked green). $(U^d_{\theta GP1})_M$ is the matrix of this result. Function values (or their derivatives) evaluated at Gauss points can then be used in any integral evaluation. For example, $\int u^h dD = |J| \sum_{l \in M} \left[\sum_{i=1}^{4} (w_i (U^d_\theta)_{GPi})_M \right]$, where $|J|$ is the transformation Jacobian for integration and w are the quadrature weights

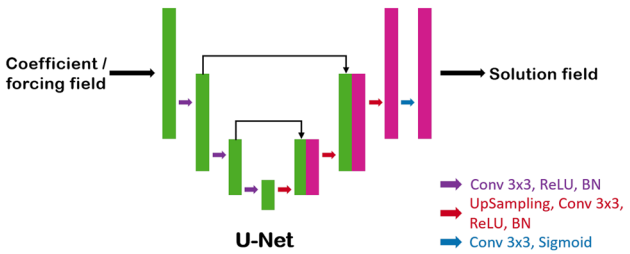


Fig. 5 UNet architecture used for training *NeuFENet*

in the loss function. Thus, the loss function retains its convex nature with respect to the solution u^h. On the other hand, Neumann conditions are included in the variational form of the PDE right at the continuous level. Especially, zero-Neumann conditions are exactly satisfied at the discrete level without requiring us to do anything ("do-nothing" conditions).

4.3 Calculation of derivatives and integration

The full domain integration (i.e., $\int D$) is nothing but the simple sum of the integration over the individual elements (i.e., $\sum_{i=1}^{N_{el}} \int D_i$). This integration over an individual element is, in turn, the simple weighted sum of the integrand evaluated at the Gauss quadrature points. This evaluation at a single Gauss point can be represented as convolution. Thus, if there are 4 Gauss points in each element, then 4 convolution operations will evaluate the integrand at those points for each element. After that, we only need to sum across Gauss points first, followed by a sum across elements. See Figs. 3 and 4 for visualized representation of this process.

Remark 2 Since the result of the integration process is a scalar loss value, there is no requirement to compute, store, or assemble a matrix.

5 Error analysis

5.1 Error analysis for the instance case

We provide estimates on the errors incurred by *NeuFENet* in approximating the solution. Suppose the exact solution of Eq. 5 is u and the solution obtained at the end of the training process is given by u_θ (see Remark 1). Let us define $u_{\tilde{\theta}}$ as the best possible function in V_θ. Note that this function may or may not be able to match u^h, but it represents the best possible function that the neural network function class can produce. Note also that $u_{\tilde{\theta}}$ might be different than u_θ, especially if the optimization algorithm cannot reach the optimum $\tilde{\theta}$. We try to bound the error $u_\theta - u$ by first breaking down the total error into errors from different sources. Theorem 2 is our main result for the single instance PDE version, while Theorem 3 is our main result for the parametric PDE version.

Lemma 1 *Let u_θ be the solution of Eq. 11 when it is optimized by Algorithm 1. Then, the optimization error $\|e_\theta\|_{V^h} = \sqrt{2\|\mathcal{E}_J\|_1}$, where $e_\theta = u_\theta - u_{\tilde{\theta}}$ and $\mathcal{E}_J = J(u_\theta) - J(u_{\tilde{\theta}})$.*

Proof

$$
\begin{aligned}
J(u_\theta) &= \int_D \left[\frac{1}{2}\nu|\boldsymbol{\nabla} u_\theta|^2 - f u_\theta\right]\mathrm{d}\mathbf{x} \\
&= \int_D \left[\frac{1}{2}\nu|\boldsymbol{\nabla} u_{\tilde{\theta}} + \boldsymbol{\nabla} e_\theta|^2 - f(u_{\tilde{\theta}} + e_\theta)\right] d\mathbf{x} \ \ (\text{using } e_\theta = u_\theta - u_{\tilde{\theta}}) \\
&= \int_D \left[\frac{1}{2}\nu|\boldsymbol{\nabla} u_{\tilde{\theta}}|^2 + \frac{1}{2}\nu|\boldsymbol{\nabla} e_\theta|^2 + \nu\boldsymbol{\nabla} u_{\tilde{\theta}}\cdot\boldsymbol{\nabla} e_\theta - f(u_{\tilde{\theta}} + e_\theta)\right]\mathrm{d}\mathbf{x} \\
&= \int_D \left[\frac{1}{2}\nu|\boldsymbol{\nabla} u_{\tilde{\theta}}|^2 - f u_{\tilde{\theta}}\right]\mathrm{d}\mathbf{x} + \frac{1}{2}\int_D \nu|\boldsymbol{\nabla} e_\theta|^2 d\mathbf{x} + \int_D \left[\nu\boldsymbol{\nabla} e_\theta\cdot\boldsymbol{\nabla} u_{\tilde{\theta}} - f e_\theta\right]\mathrm{d}\mathbf{x} \\
&= J(u_{\tilde{\theta}}) + \frac{1}{2}\|e_\theta\|^2_{V^h},
\end{aligned}
\tag{24}
$$

where in the final step, we have used the definitions of J and $\|e_\theta\|_{V^h}$ along with the fact that

$\int_D \left[\nu\boldsymbol{\nabla} e_\theta\cdot\boldsymbol{\nabla} u_{\tilde{\theta}} - f e_\theta\right] d\mathbf{x} = 0$, since $e_\theta \in V^h$ (see Eqs. 9 and 10). Therefore, we have

$$
\|e_\theta\|^2_{V^h} = 2\left[J(u_\theta) - J(u_{\tilde{\theta}})\right] = 2\mathcal{E}_J = 2\|\mathcal{E}_J\|_1, \tag{25}
$$

since $\mathcal{E}_J = J(u_\theta) - J(u_{\tilde{\theta}}) > 0$. □

We next describe a theorem that provides an estimate on the network capacity.

Theorem 1 *Fix $p \in \Omega$ and consider a PDE as defined in Eq. 1 over a compact domain D which is uniformly discretized with resolution h. Let u_h^* be the true solution evaluated at the grid points. Consider the hypothesis class*

$$
\mathcal{H} := \left\{ u_\Theta : p \mapsto \sum_{l=1}^{k} a_l^i \mathrm{ReLU}\left(\left\langle w_l^i, p\right\rangle + b_l^i\right), i = 1, \ldots, n. \right\} \tag{26}
$$

defined as the set of all two-layer neural networks with k hidden neurons equipped with ReLU activation. Then, as long as $k = \Omega(1/h)$, there exists a network in $\mathcal{H}$ for which $\mathrm{Err}_{\mathcal{H}} = 0$.

Proof The proof follows a recent result by Bubeck et al. [61]. Let the output resolution of the network be n. For a fixed p, consider the (linear) vector space spanned by all possible (perhaps uncountably many) basis functions of the form

$$
f(\cdot) = a_l^i \mathrm{ReLU}\left(\left\langle w_l^i, \cdot\right\rangle + b_l\right),
$$

where a, b, w are arbitrary real-valued weights. Since $i = 1, \ldots, n$, the span of this space is no more than n-dimensional and isomorphic to $\mathbb{R}^m$ for some $m \le n$. Therefore, there is a set of no more than n basis functions (i.e., n neurons) that can be used to represent u_p^* any fixed p. Assuming the dimension of p is small, we have $n \lesssim 1/h$. Therefore, $k = \Omega(1/h)$ neurons are sufficient to reproduce u_p^*. □

Notice that the above theorem shows that there exist *NeuFENet* architectures that exactly drive the modeling error down to zero. However, the proof is *non-constructive*, and there is no obvious algorithm to find the basis functions that reproduce the solution at the evaluation points. Theorem 1 essentially allows us to choose the neural network parameter family Θ, such that the modeling error $e_{\mathcal{H}}$ is low. Since we are free to choose the network architecture, we can always assume (and *a posteriori* confirm) that

$$
\|e_{\mathcal{H}}\| = \|u_{\tilde{\theta}} - u^h\| \le \epsilon, \ \ \epsilon > 0. \tag{27}
$$

Remark 3 *NeuFENet* is designed to be agnostic to a neural network. Therefore, either a fully connected neural network or a convolutional neural network can be used for the network approximation. Since a convolutional neural network can be interpreted as a special case of a fully connected network with sparse weights [62], the above estimate still holds.

For the third source of error, i.e., the error due to discretization using finite element method can be estimated from standard finite element analysis literature. We start with the following assumption:

Assumption 1 Assume that the spatial domain D is discretized by a mesh $\mathcal{K}^h$ that consists of hyperrectangular elements. Each element $K \in \mathcal{K}^h$ has a bounded radius, i.e., $0 < h_{\min} \le r(K) \le h_{\max} < \infty$. We define the mesh length $h = \min\{r(K)\}_{i=1}^{n_{el}}$

Lemma 2 *The exact solution to Eq. 9, $u \in H^2(D)$.*

Proof In Eq. 9, the diffusivity $\nu(\mathbf{x};\omega) \in C(D)$ for any fixed $\omega \in \Omega$. Furthermore, the forcing function $f = 0 \in L^2(D)$. Using results from regularity theory (such as [36], Sec. 6.3 Theorem 1), we conclude $u \in H^2(D)$. □

Lemma 3 *Let Assumption 1 hold. Further assume that the basis functions $\phi_i(\mathbf{x})$ in Eq. 14 are chosen, such that*

$\phi_i(\mathbf{x}) \in C^\alpha(K)$ $(\alpha \geq 1)$ locally within each element $K \in \mathcal{K}^h$. Then

$$\|u^h - u\|_{L^2(D)} \leq C_d h^{\alpha+1}, \tag{28}$$

where $C_d = C_d(D, |u|_{H^2})$ is a constant.

Proof We refer to standard texts such as Oden and Reddy [63](Sec. 8.6, Theorem 8.5) or Brenner and Scott [23](Sec. 5.7) for the proof. □

Finally we can write down the estimate for the generalization error in *NeuFENet* in the form of the following theorem.

Theorem 2 *Consider any NeuFENet architecture producing fields with grid spacing h. Let $\mathcal{H}$ denote the hypothesis class of all networks obeying that architecture. Suppose that $\Theta \in \mathcal{H}$ is a NeuFENet trained using the loss function J defined in Eq. 12. Then, its generalization error obeys*

$$\|e_G\|_{V_\theta} \leq \mathrm{Err}_\Theta + \mathrm{Err}_\mathcal{H} + O\big(h^{\alpha+1}\big), \tag{29}$$

where Err_Θ is a term that only depends on the NeuFENet optimization procedure and $\mathrm{Err}_\mathcal{H}$ only depends on the choice of hypothesis class $\mathcal{H}$. The α in the third term is the local degree of the basis functions as in Lemma 3.

Proof The result is simply an application of the triangle inequality. Dropping the subscript V_θ

$$\begin{aligned}\|e_G\| &= \|u_\theta - u_{\tilde{\theta}} + u_{\tilde{\theta}} - u^h + u^h - u\| \\ &\leq \|u_\theta - u_{\tilde{\theta}}\| + \|u_{\tilde{\theta}} - u^h\| + \|u^h - u\| \\ &= \|e_\theta\| + \|e_\mathcal{H}\| + \|e_h\| \\ &= \mathrm{Err}_\Theta + \mathrm{Err}_\mathcal{H} + O(h^{\alpha+1}).\end{aligned} \tag{30}$$

Using Lemma 1, the optimization error Err_Θ is nothing but $\sqrt{2\|\mathcal{E}_J\|_1}$. □

5.2 Extending the error analysis for the parametric version

The above theorem is for a single parameter choice $p \in \Omega$. An identical argument can be extended to the loss constructed by sampling a *finite* number (say m) of parameters from a distribution over Ω. We obtain the following corollary from Theorem 1:

Lemma 4 (Barron [64]) *Consider a finite-sample version of the loss $\hat{L}$ constructed by taking the average over m parameter choices sampled from Ω. Consider the hypothesis class*

$$\mathcal{H} := \left\{ u_\Theta : p \mapsto \sum_{l=1}^{k} a_l^i \mathrm{ReLU}\big(\langle w_l^i, p\rangle + b_l^i\big), i = 1, \ldots, n \right\} \tag{31}$$

defined as the set of all two-layer neural networks with k hidden neurons equipped with ReLU activation. Then, as long as $k = \Omega(m/h)$, there exists a network in $\mathcal{H}$ for which $\mathrm{Err}_\mathcal{H} = 0$.

This lemma shows that a wide two-layer network exists that can reproduce any (finite) set of field solutions to a PDE system as long as the width scales linearly in the cardinality of the set. Getting bounds independent of the cardinality is an interesting open question. We have the following result for the optimization error:

Lemma 5 *Let u_θ be the solution of the optimization problem in Eq. 23 when it is optimized by Algorithm 2. Then, the optimization error $\|e_\theta\|_{V_\theta^h} = \sqrt{2\|\mathcal{E}_{\hat{L}}\|_1}$, where $e_\theta = u_\theta - u_{\tilde{\theta}}$, $\mathcal{E}_{\hat{L}} = \hat{L}(\theta) - \hat{L}(\tilde{\theta})$.*

Proof Starting with the definition of $\hat{L}$ from Eq. 22 and denoting $\nu_i = \nu(\mathbf{x}, \omega_i)$ and $u_{\theta,i} = u(\mathbf{x}, \omega_i; \theta)$

$$\begin{aligned}\hat{L}(\theta) &= \frac{1}{N_s}\sum_{i=1}^{N_s} J(\nu_i, u_{\theta,i}) \\ &= \frac{1}{N_s}\sum_{i=1}^{N_s}\left(J(\nu_i, u_{\tilde{\theta},i}) + \frac{1}{2}\|e_{\theta,i}\|_{V^h}^2\right) \\ &= \hat{L}(\tilde{\theta}) + \frac{1}{2}\|e_\theta\|_{V_\theta^h}^2,\end{aligned} \tag{32}$$

where we have used $\|e_\theta\|_{V_\theta^h}^2 = \mathbb{E}\|e_\theta\|_{V^h} = \frac{1}{N_s}\sum_{i=1}^{N_s}\|e_{\theta,i}\|_{V^h}^2$. Denoting $\mathcal{E}_{\hat{L}} = \hat{L}(\theta) - \hat{L}(\tilde{\theta}) > 0$, we have the result. □

With this, an analog of Theorem 2 can be stated for the parametric training with finite data as below.

Theorem 3 *Consider any NeuFENetarchitecture producing fields with grid spacing h. Let $\mathcal{H}$ denote the hypothesis class of all networks obeying that architecture. Suppose that $\Theta \in \mathcal{H}$ is a NeuFENet trained using the loss function $\hat{L}$ defined in Eq. 22. Then, its generalization error obeys*:

$$\|e_G\|_{V_\theta^h} \leq \mathrm{Err}_\Theta + \mathrm{Err}_\mathcal{H} + O\big(h^{\alpha+1}\big), \tag{33}$$

where Err_Θ is a term that only depends on the NeuFENet optimization procedure; *$\mathrm{Err}_\mathcal{H}$ only depends on the choice of hypothesis class $\mathcal{H}$ and α is the local degree of the basis function as in Lemma 3.*

Proof The result is simply an application of the triangle inequality. Dropping the subscript V_θ

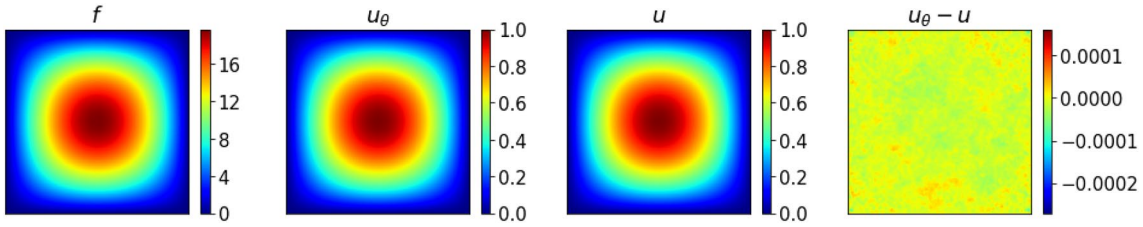


Fig. 6 Solution to Eq. 35 with the forcing term as shown in Eq. 36 on a 256×256 grid. (Left) the discrete forcing F^d, (middle-left) the solution U^d obtained from a *NeuFENet* with a U-net architecture, (middle-right) the exact solution $u_{ex} = \sin(\pi x)\sin(\pi y)$ evaluated on the mesh, denoted U^d_{ex}, (right) the error $(U^d - U^d_{ex})$. From left to right: f, u^h_θ, u_{num} and $(u^h_\theta - u_{num})$. Note: $\|u_\theta\| = 0.499979$, $\|u\| = 0.5$, $\|u_\theta - u\| = 2.4 \times 10^{-5}$

$$\begin{aligned}\|e_G\|_{V^h_\theta} &= \|u_\theta - u_{\tilde{\theta}} + u_{\tilde{\theta}} - u^h + u^h - u\|_{V^h_\theta} \\ &\leq \|u_\theta - u_{\tilde{\theta}}\|_{V^h_\theta} + \|u_{\tilde{\theta}} - u^h\|_{V^h_\theta} + \|u^h - u\|_{V^h_\theta} \\ &\leq \|e_\theta\|_{V^h_\theta} + \|e_{\mathcal{H}}\|_{V^h_\theta} + \|e_h\|_{V^h_\theta}.\end{aligned} \tag{34}$$

The third norm can be estimated as

$$\|e_h\|_{V^h_\theta} = \|u^h - u\|_{V^h_\theta} = \sqrt{\mathbb{E}_{\omega\sim\Omega}\|u^h_i(\omega) - u(\omega)\|^2_V} = \sqrt{\frac{1}{N_s}\sum_{i=1}^{N_s}\|u^h_i - u_i\|^2_V} \leq \sqrt{\frac{1}{N_s}\sum_{i=1}^{N_s}\|v_i\|^2_{L^2(D)}\|\nabla(u^h_i - u_i)\|^2_{L^2(D)}}$$

$$\leq \sqrt{\frac{1}{N_s}\sum_{i=1}^{N_s}\|v_i\|^2_{L^2(D)}\left(\sum_{K\in\mathcal{K}^h}\|\nabla(u^h_i - u_i)\|^2_{L^2(K)}\right)} \leq \sqrt{\frac{1}{N_s}\sum_{i=1}^{N_s}\|v_i\|^2_{L^2(D)}C^h_I\left(\sum_{K\in\mathcal{K}^h}\|(u^h_i - u_i)\|^2_{L^2(K)}\right)}$$

$$= \sqrt{\frac{1}{N_s}\sum_{i=1}^{N_s}C^h_I\|v_i\|^2_{L^2(D)}\|(u^h_i - u_i)\|^2_{L^2(D)}} \leq \sqrt{\frac{1}{N_s}\sum_{i=1}^{N_s}C_{di}h^{2(\alpha+1)}} \leq C_e h^{\alpha+1},$$

where $C_e = C_e\left(\hat{\Omega}_{N_s}, D, C^h_I, A, B\right)$. Here, C^h_I is the discrete inverse Poincaré constant, such that $\|\nabla u\|_{L^2(K)} \leq C^h_I\|u\|_{L^2(K)}$ for all $K \in \mathcal{K}^h$; $A = \max\{\|v_i\|_{L^2(D)}\}_{i=1}^{N_s}$ and $B = \max\{|u_i|_{H^2(D)}\}_{i=1}^{N_s}$. □

6 Results

6.1 Error convergence for a Poisson problem

At the outset, we would like to validate the error bounds stated in Sect. 5. To this end, we solve the following non-parametric Poisson's equation:

$$-\Delta u = f \quad \text{in } D \subset \mathbb{R}^2 \tag{35a}$$

$$u = 0 \quad \text{on } \partial D, \tag{35b}$$

where $D = [0, 1]^2$ is a two-dimensional square domain. The forcing is chosen as

$$f = f(\mathbf{x}) = f(x, y) = 2\pi^2 \sin(\pi x)\sin(\pi y). \tag{36}$$

The exact solution to Eq. 35 with the forcing function in Eq. 36 is given by $u_{ex}(x, y) = \sin(\pi x)\sin(\pi y)$. For solving this problem, we seek to train a *NeuFENet* that can predict the solution u given the forcing f. At the discrete level, the

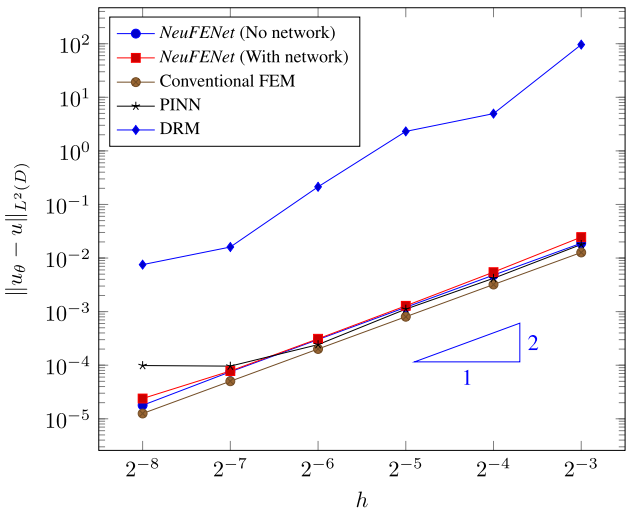


Fig. 7 Convergence of the error in L^2 norm for the Poisson's equation with analytical solution $u = \sin(\pi x)\sin(\pi y)$

network takes input F^d and outputs U^d. The loss function Eq. 11 takes the concrete form

$$J(u) = \frac{1}{2}\int |\nabla u|^2 \mathrm{d}\mathbf{x} - \int uf \mathrm{d}\mathbf{x}. \tag{37}$$

Figure 6 shows the contours of the network input F^d, output U^d, the exact solution evaluated on the mesh $\mathcal{K}^h$ (a 256×256 grid) and the error $(U^d - U_{ex})$.

To study convergence behavior, we repeat this procedure with varying mesh length h, starting from $h = \frac{1}{8}$ and gradually decreasing h till $h = \frac{1}{256}$. The calculated errors for each resolution are reported in Fig. 7.

Theorem 2 estimates the total error by the contributions from three individual sources of errors, namely, the error due to finite element discretization (e_h), the error due to network approximation capacity ($e_{\mathcal{H}}$), and the error due to the optimization process (e_θ). Of these three errors, e_h maintains a particular relation to the discretization parameter h (i.e., the mesh size). The other two errors do not hold a straightforward relation with the mesh size or the network parameters. Thus, we put the estimate in Eq. 29 to test by keeping both $e_{\mathcal{H}}$ and e_θ sufficiently low, so that the dominating error is the discretization error e_h.

We illustrate this by running two sets of *NeuFENet* simulations.

- The first set is where we completely remove the neural network and optimize the loss directly with respect to the function u^h, that is

$$u^h = \underset{u^h \in V^h}{\arg\min}\, J\left(u^h\right), \tag{38}$$

  which is nothing but classic Rayleigh–Ritz optimization. The absence of a neural network eliminates the error $e_{\mathcal{H}}$. Therefore, the total error is a combination of only e_h and e_θ. The total errors from this algorithm are plotted against the mesh size h in Fig. 7 under the legend "*NeuFENet* (No network)". We see that the error plot has a slope of 2.
- The second set uses a neural network. Thus, the optimization statement is the same as presented in Eq. 18. Therefore, as discussed in Theorem 2, the total error combines all three errors indicated in Eq. 29. To keep $e_{\mathcal{H}}$ negligible, we need to be aware that the spatial degrees of freedom (i.e., the number of bases in V^h) are inversely proportional to h^2 (for a 2D domain). Thus, as we decrease h, the "size" of V^h increases. Therefore, the function space V_θ^h must also get bigger; in particular, it should be big enough to satisfy $V_\theta^h \supset V^h$. One way to accommodate this is to use a high-capacity network to solve the equation at all h-levels. However, the UNet architecture described in Sect. 4.1 does not allow very high depth when the input/output size is low. Thus, we must gradually enhance the network capacity at different h levels by increasing the network depth. Using this strategy, we solve Eq. 35 at various h-levels and plot the errors in Fig. 7. The slope of 2 of the error curve confirms that both $e_{\mathcal{H}} \approx 0$ and $e_\theta \approx 0$. Interestingly, if we do not keep $e_{\mathcal{H}}$ negligible by increasing the depth of the network for smaller h, both $e_{\mathcal{H}}$ and e_θ can dominate (see Appendix 2). This suggests that a gradual increase in network complexity is warranted as the discretization becomes finer. This is made computationally efficient using multi-grid-like approaches [65].

Figure 7 also shows a plot of errors obtained from solving the same problem using three other methods for reference, namely, (i) conventional FEM, (ii) physics-informed neural networks (PINN), and (iii) the deep Ritz method (DRM). The conventional FEM solution was obtained using a GMRES solver with a 10^{-8} tolerance. A multilayer perceptron was used with 8 hidden layers, each with 100 neurons with tanh activation function for both the PINN and DRM solutions. The learning rate was kept at 10^{-3} for PINN, and 10^{-4} for DRM. In both PINN and DRM, the boundary conditions are applied approximately by minimizing the errors incurred on the boundary. This necessitates weighing the loss values in the interior (PDE) and at the boundary differently, i.e., the loss function has the form $\mathcal{L} = \phi(R_{PDE}) + \lambda\psi(R_{BC})$, where R stands for residual, and ϕ and ψ are functions that depend on the flavor of the method. In PINN, $\phi = \psi = \|\cdot\|_2^2$ [1, 66]; in DRM, ϕ takes the energy form of the residual, and $\psi = \|\cdot\|_2^2$ [38, 39]. In Fig. 7, PINN errors are comparable to that obtained from *NeuFENet*. On the other hand, DRM exhibits much higher errors compared to *NeuFENet*.

The high errors in DRM is not surprising, and can be understood as follows. If we take our model equations (Eq. 2 and Eq. 3) (and assume $\Gamma_D = \Gamma$), then the DRM loss can be written as

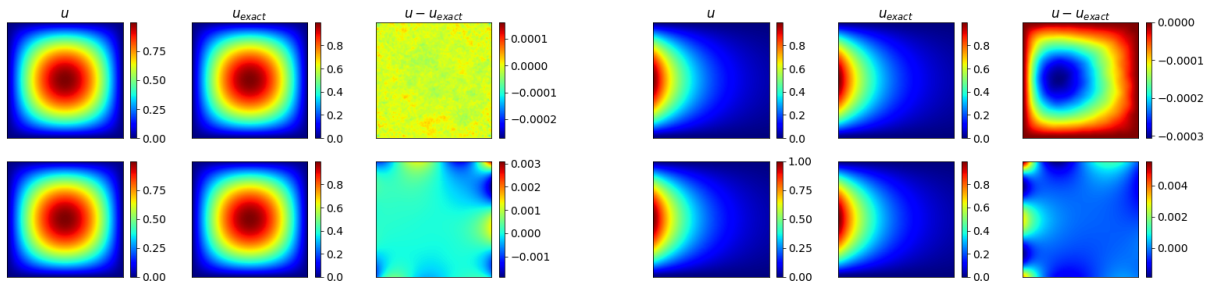
$$L_{\mathrm{DRM}} = \int_D \left[\frac{1}{2}\nu(\nabla w \cdot \nabla u) - wf\right]\mathrm{d}\mathbf{x} + \lambda \int_{\Gamma_D} (u-g)^2 \mathrm{d}\Gamma, \tag{39}$$

where $0 < \lambda \in \mathbb{R}$. The Euler–Lagrange equations for this loss function, however, turn out to be [40]:

$$-\nabla \cdot \nu\nabla u = f \ \text{ in } \ \Omega, \tag{40}$$

$$n \cdot \nu\nabla u + 2\lambda u = g \ \text{ on } \ \Gamma_D. \tag{41}$$

Because of this inconsistency, the exact solutions of Eqs. 2 and 3 do not satisfy Eq. 39. This leads to larger errors in DRM in Fig. 7. In *NeuFENet*, the boundary conditions are applied exactly; thus, it is a consistent method and obviates the need for a penalty parameter λ.



(a) *Solution to Equation 35 and 36:*
(top) NeuFENet, $\|u_\theta - u\|_{L^2} = 2.4\times10^{-5}$ *and* $\|u_\theta - u\|_{L^\infty} = 2.7\times10^{-4}$,
(bottom) PINN, $\|u_\theta - u\|_{L^2} = 3.6\times10^{-4}$ *and* $\|u_\theta - u\|_{L^\infty} = 3.1\times10^{-3}$

(b) *Solution to Equation 42:*
(top) NeuFENet, $\|u_\theta - u\|_{L^2} = 1.5\times10^{-4}$ *and* $\|u_\theta - u\|_{L^\infty} = 3.1\times10^{-4}$,
(bottom) PINN, $\|u_\theta - u\|_{L^2} = 5.6\times10^{-4}$ *and* $\|u_\theta - u\|_{L^\infty} = 5.5\times10^{-3}$

Fig. 8 Typical comparison between a *NeuFENet* solution and a PINN solution for two different boundary conditions

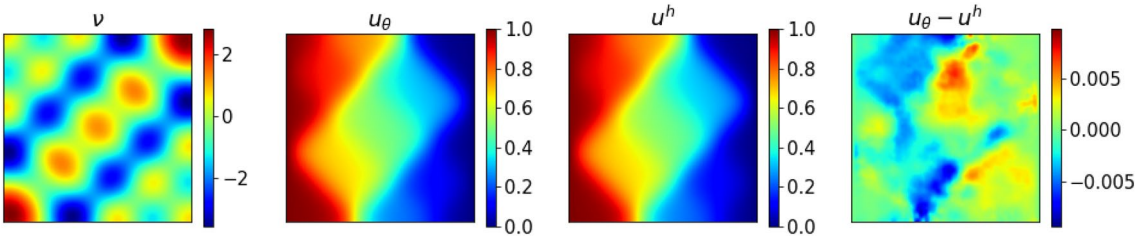


Fig. 9 (Left) contours of $\ln(\nu)$ corresponding to $\underline{a} = (-0.24, -0.17, 0.13, 0.07, -0.07, 0.14)$, (middle-left) *NeuFENet* prediction (u_θ), (middle-right) reference numerical solution using FEM (u^h), (right) contours of ($u_\theta - u^h$)

6.2 Exact imposition of Dirichlet boundary conditions

As mentioned in Sect. 4.2, Dirichlet boundary conditions can be applied exactly (subject to discretization) in *NeuFENet*. The exact imposition of boundary conditions is almost universal across numerical methods unless there is an explicit need to apply it weakly (or in an integral sense). Yet, this is not very common in neural methods. Most of the current neural methods apply only approximately the Dirichlet (as well as other) boundary conditions. We illustrate this with two simple examples below, the first with a homogeneous BC and the other with a non-homogeneous one.

- *Homogeneous BC:* We take the same equation and boundary conditions described in Sect. 6.1 (see Eq. 35 and Eq. 36). The boundary condition, in this case, is homogeneous throughout the entire boundary. We solve this problem using *NeuFENet* and also with a physics-informed neural network (PINN, [1]). In the case of *NeuFENet*, the FEM mesh $\mathcal{K}^h$ for *NeuFENet* is a 255×255 mesh of bilinear quadrilateral elements (i.e., 256×256 nodes), which is solved by optimizing Eq. 37 using Adam optimizer. The PINN solution is obtained by performing residual optimization using $65536 (= 256 \times 256)$ points randomly selected from $D = [0, 1]^2$. The contours of the solutions and their difference from the analytical solution are shown in Fig. 8(a).
- *Non-homogeneous BC:* Consider the following equation:

$$-\Delta u = 0 \quad \text{in } [0, 1]^2 \tag{42a}$$

$$u = \sin(\pi y) \quad \text{on } x = 0 \tag{42b}$$

$$u = 0 \quad \text{other boundaries.} \tag{42c}$$

The analytical solution to Eq. 42 is given by $u(x, y) = \exp(-\pi x)\sin(\pi y)$. We solve this set of equations with *NeuFENet* using a 32×32 bilinear quadrilateral mesh and PINN using $1024 (= 32 \times 32)$ randomly selected points. The contours are shown in Fig. 8b.

In both experiments, we notice that a non-exact application of boundary conditions can affect the ability of the network to approximate the solution in the interior, thus rendering a higher level of errors overall.

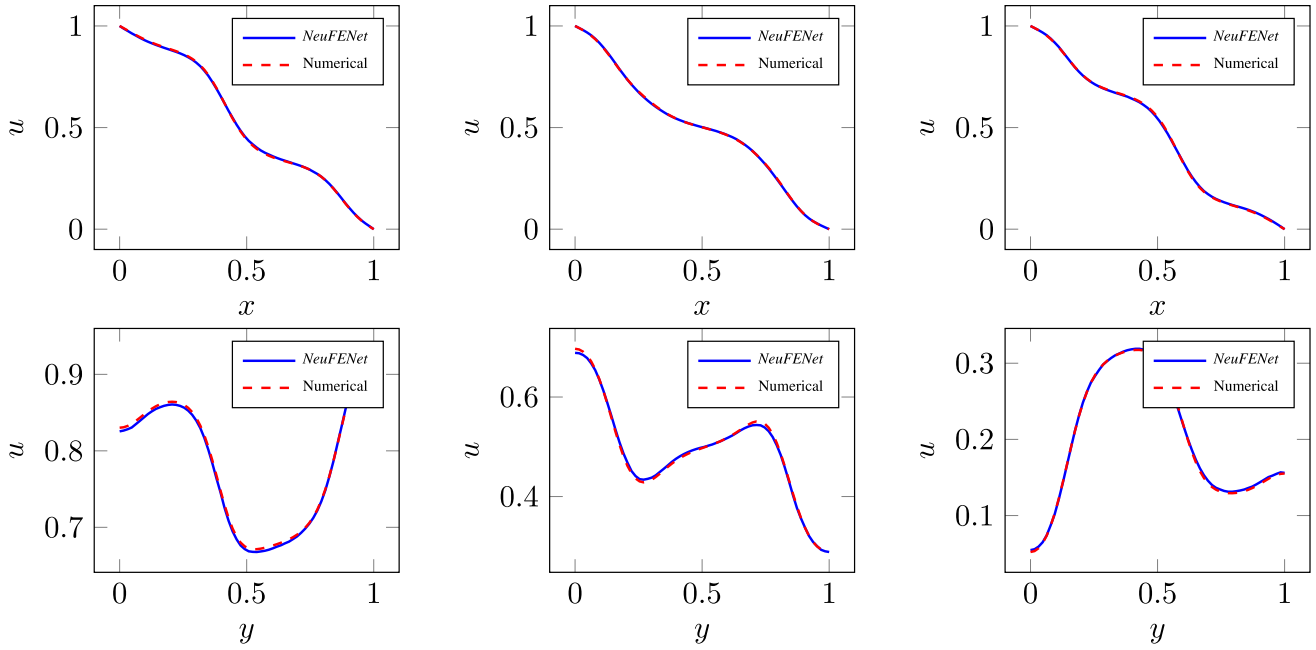


Fig. 10 Line cuts for the contours presented in Fig 9. (Top row) x-parallel line cuts at $y = 0.2, 0.5, 0.8$. (Bottom row) y-parallel line cuts at $x = 0.2, 0.5, 0.8$

6.3 Poisson's equation with parametric log permeability

Our second illustration is the solution of the PDE defined in Sect. 2.1 (Eq. 5), which is frequently used for simulating practical problems such as heat or mass transfer through an inhomogeneous media. We solve this problem in both 2D and 3D domains, i.e., $D = [0, 1]^2$ as well as $D = [0, 1]^3$; $\tilde{\nu}$ is now a function of $\mathbf{x} = (x, y)$ and is also parametric, as mentioned in Eq. 51.

We seek a mapping of the form $u = G_{nn}(\nu)$, where G_{nn} denotes the neural network. Therefore, S^d refers to the discrete version of ν. Following the principle of Karhunen–Loeve expansion as described in Appendix 1, the infinite-dimensional random space is truncated into a finite-dimensional space. Thereafter, a finite set of samples is prepared from this space for the training process. Suppose the number of samples is N_s. Then, the loss function takes the concrete form

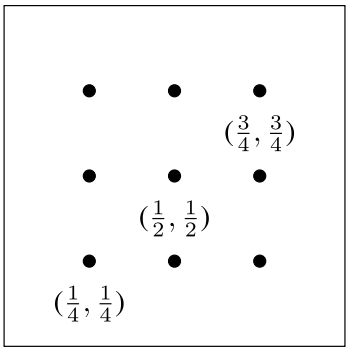
$$J = \frac{1}{N_s} \sum_{i=1}^{N_s} \int_D \tilde{\nu}_i(\mathbf{x}) |\nabla u_i(\mathbf{x})|^2 \mathrm{d}\mathbf{x}, \tag{43}$$

where, $u_i = G_{nn}(\nu_i, \theta)$. Both Dirichlet and Neumann conditions are present in this equation. In *NeuFENet*, the Dirichlet conditions are applied exactly. The zero-Neumann condition is also applied exactly at the continuous level, since the boundary integrals vanish at the continuous level (see Eq. 10).

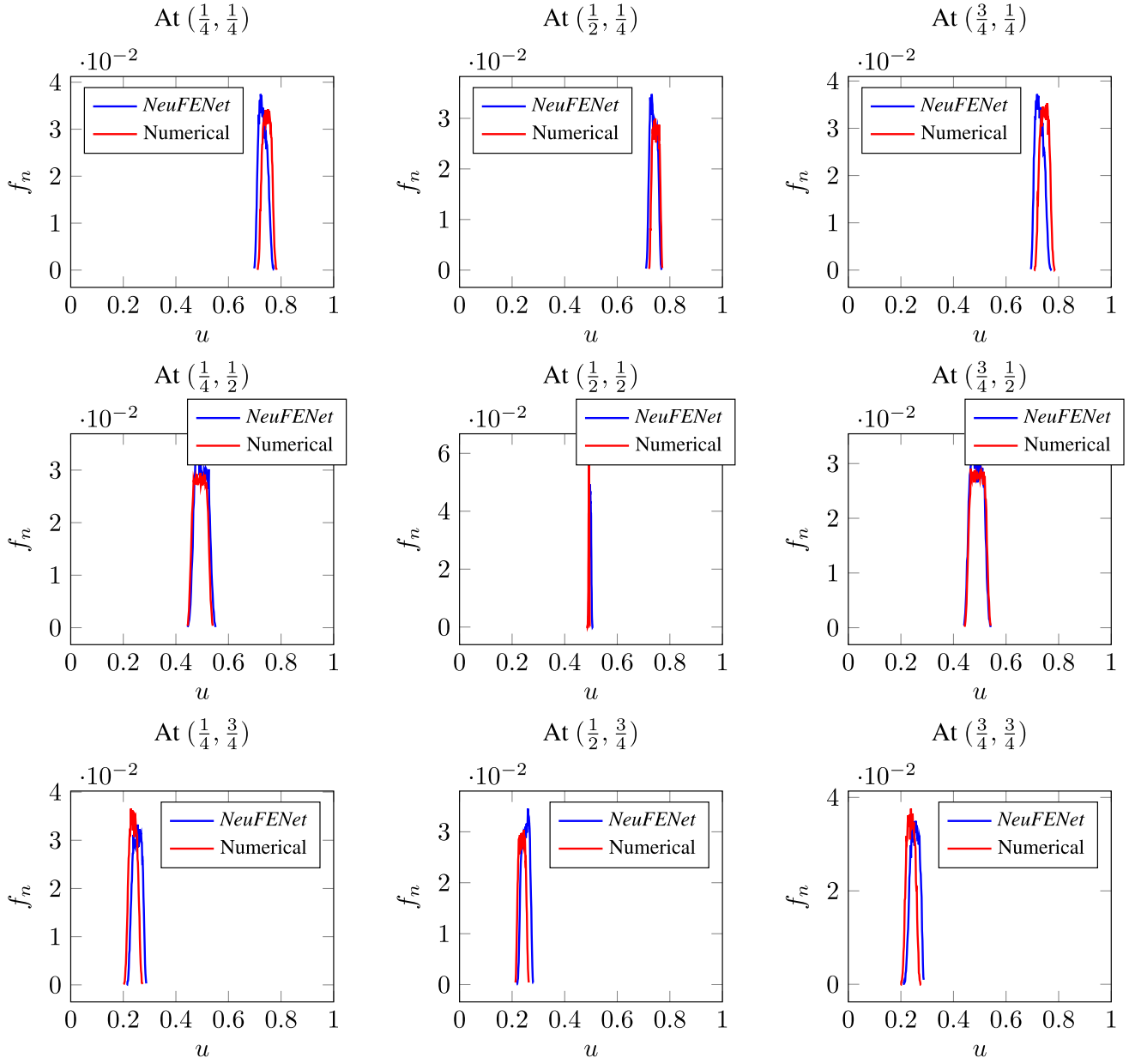
By optimizing the loss (Eq. 43), we attempt to learn the distribution of the stochastic solution, given that the coefficients in the log permeability K–L sum come from a known range of values that depends on the parameter space ω. We truncate the K–L sum after 6 terms. These six coefficients form a six-dimensional space from which the coefficient tuples $\{a_i\}_{i=1}^6$ can be drawn. The *NeuFENet* is trained by selecting a finite number (N_s) of pseudo-random samples from this 6-dimensional space, specifically $\mathbf{a} \in [-\sqrt{3}, \sqrt{3}]^6$ (see also Appendix 1). For the results shown below, we have taken $N_s = 65536$. We used the Adam optimization algorithm, with a learning rate of 10^{-4}. Once the network is trained, ***we can perform inference by evaluating the solution for any diffusivity ν taken from the sample space***. To illustrate the nature of the input (ν or S^d) and the solution (u or U^d), we present an anecdotal (i.e., a non-special and random) set of S^d and U^d in Fig. 9. A reference solution using a conventional FEM program is also presented therein. Furthermore, sectional line cuts for these contours are shown in Fig. 10. The line cuts display a close match between *NeuFENet* and the numerical solution.

6.3.1 Statistical distribution of solution

Since Eq. 5 is parametric, we can compare the quality of the solution from *NeuFENet* with a reference numerical solution in a statistical manner. We choose some points on the domain D (shown by black dots in Fig. 11a) and evaluate the



(a) *Query points for histogram analysis. Positions of points denoted by* (x, y) *tuple.*



(b) *Histograms.*

Fig. 11 Normalized histograms plotted for the query points shown in Fig. 11a

Table 1 Inference times of *NeuFENet* compared with solve time of a conventional FEM solver for the parametric problem

	NeuFENet	FEM (conventional)
64^2	4.76×10^{-5}	1×10^{-2}
128^2	9.8×10^{-5}	3.6×10^{-2}
256^2	2.74×10^{-4}	21.0×10^{0}

The 64^2, 128^2, and 256^2 results are obtained by averaging over 500 inferences/solves. All timing values are in seconds

solution values at those query points for 16, 384 samples of ν, where the $\{a_i\}_{i=1}^{6}$ tuples are taken from a smaller subset of the full sample space, namely, $\mathbf{a} \in [-\frac{1}{4}, \frac{1}{4}]^6$. Therefore, at a particular query point, those 16, 384 solution values approximately represent a distribution of the solution values at that point. This sample of solution values at each query point allows us to create histograms of the solution values at each query point.

The histograms are shown in Fig. 11. We notice a very close match between the histograms obtained from *NeuFENet* and a conventional FEM solver. This once again confirms that *NeuFENet* effectively provides the correct statistics of the parametric Poisson equation.

6.3.2 Comparisons with traditional FEM solver

Table 1 presents a comparison of the time taken for a single *NeuFENet* inference against the solve time for a single case using a conventional FEM solver. Note that the inference time does not include the "training time." The number of parameters present in the network is approximately 4.2 million. The *NeuFENet* model is queried on an NVIDIA A100-SXM4-80GB GPU. For the conventional solver, the reported timings are taken to be the lowest time obtained when run on 1, 2, 4, and 8 processors on a AMD EPYC 7543 32-Core compute node. It can be noted from Table 1 that the respective timings differ from each other by a few orders of magnitude.

This disparity in inference/solve times clearly demonstrate the benefit of using a neural solver for parametric

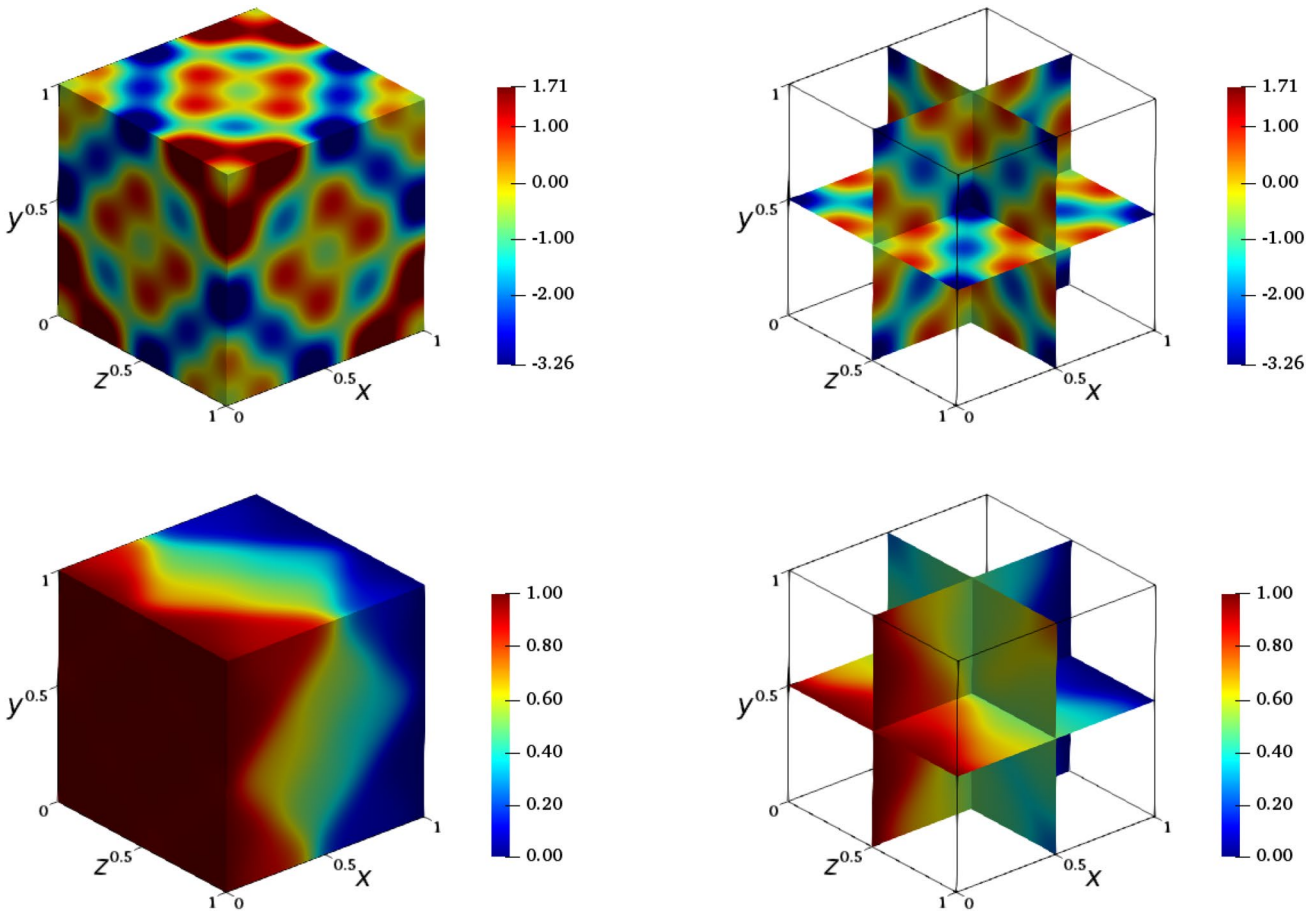


Fig. 12 Contours of $\ln(\nu(x, y, z))$ and the solution $u_\theta(x, y, z)$ to the 3D Poisson's problem (Eq. 44) on a $64 \times 64 \times 64$ mesh (for $\mathbf{a} = (-1, 1.4, 1.5, -1.3, -1.6, 0.3)$)

problems. Once the training is completed offline, inferences become very cheap compared to the conventional solver.

6.4 3D Poisson's equation

So far, we have focused on two-dimensional problems. Although 2D problems are useful in demonstrating the key features and properties of the *NeuFENet* method, a real test of neural PDE solvers lies in their ability to solve three-dimensional problems. *NeuFENet* can solve both 2D and 3D problems without much changes to the architecture. To provide an example of this, we solve the 3D counterpart of the PDE defined in Eq. 5, which is

$$-\boldsymbol{\nabla}\cdot(\tilde{\nu}(\mathbf{x})\boldsymbol{\nabla}u)=0 \text{ in } D=[0,1]^3, \tag{44a}$$

$$u(x=0,y,z)=1, \tag{44b}$$

$$u(x=1,y,z)=0, \tag{44c}$$

$$\hat{\mathbf{n}}\cdot\boldsymbol{\nabla}u=0 \quad \text{on all other boundaries}, \tag{44d}$$

where $\hat{\mathbf{n}}$ denotes the outward normal to the boundary. Once again, the functional form of $\nu(\tilde{\mathbf{x}})$ is described in Appendix 1. The loss function is a direct analog of Eq. 43

$$J=\int_D \tilde{\nu}(\mathbf{x})|\boldsymbol{\nabla}u(\mathbf{x})|^2\mathrm{d}\mathbf{x}, \tag{45}$$

In Fig. 12, we show one randomly selected pair of ν and u for a 3D problem obtained using *NeuFENet*. We optimize Eq. 45 for a randomly selected set of coefficients $\mathbf{a}=(-1,1.4,1.5,-1.3,-1.6,0.3)$. The plots of $\nu(x,y,z)$ and $u_\theta(x,y,z)$ are shown in Fig. 12.

7 Conclusions and future directions

In this paper, we develop a neural method *NeuFENet* for solving parametric PDEs where the discretization and the loss functions are inspired by the continuous Galerkin (cG) method and the Rayleigh–Ritz method, respectively. Due to the choice of discretization scheme, *NeuFENet* inherits the approximation properties of the cG method. This allows us to (i) calculate spatial derivatives in the same way as in finite element methods, (ii) perform spatial integration using simple Gaussian quadrature schemes, (iii) apply Dirichlet and (zero) Neumann boundary conditions exactly, and (iv) derive *a priori* error estimates.

The optimization problem is defined in terms of an energy functional derived from variational principles. This is in contrast to residual-based minimization, the more widely followed process across the current neural methods. There are some exceptions (see, for instance, [13, 38, 39]) that apply energy-based loss functions but do not apply the boundary conditions exactly, thereby incurring large errors. We showed examples of Poisson's equation solved using *NeuFENet*. Since Poisson's equation is a self-adjoint equation, its energy functional is convex and thus possesses a unique minima, which can be easily found by a gradient-based optimization method. We further illustrate that such a method can successfully solve stochastic PDEs and determine its statistical properties.

NeuFENet offers direct control over the regularity of the solution with respect to the spatial domain. For example, in the examples presented in this paper, the solutions belong to the $H^1(\Omega)$-space. Also, the Dirichlet boundary conditions are imposed exactly; thus, there is no need for a boundary loss function common in collocation-based neural methods. This leads to another advantage: the absence of any hyperparameter associated with the relative weight of the "interior" and "boundary" loss values. The only hyperparameter in this method is the learning rate for the optimization process.

7.1 Limitations and future work

Finally, we also identify some limitations of our work and opportunities for future work:

- *Non-self-adjoint equations:* Equation 21 is based on the Rayleigh–Ritz method (Eq. 11, 12), which is a well-known formalism for elliptic equations. It has the interpretation of minimization of "energy." However, when it comes to non-parabolic and hyperbolic equations, such "energy" functionals are not readily available. In the absence of an "energy" functional for non-self-adjoint equations, we need to resort to minimizing norms of the residuals. This gives rise to an interesting question as to what norm should be minimized and how to treat the loss function, such that the optimization problem is well conditioned and converges faster.
- *Inheritance from the continuous Galerkin methods:* The *NeuFENet* method, as discussed in the current work, relies on the continuous Galerkin formalism, which is shown to exhibit spurious oscillations convection-driven elliptic and parabolic equations. This issue can be addressed by the Petrov–Galerkin family of methods [67], where the test and trial functions may be taken from different spaces.
- *Dependence on a mesh: NeuFENet* is mesh based. For the examples shown in this paper, we did not need to store a mesh explicitly, because the meshes were structured and fully regular. However, for an arbitrary geometry, such a mesh will need to be saved to memory,

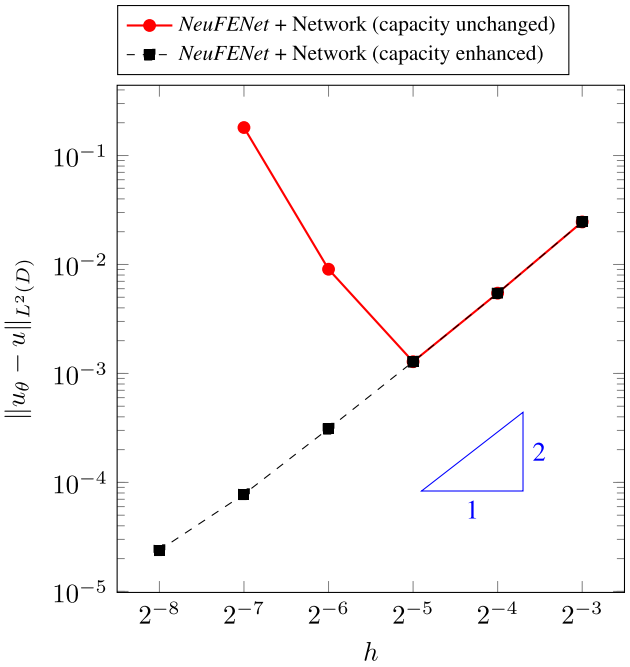


Fig. 13 Convergence of the error in L^2 norm for the Poisson equation with analytical solution $u = \sin(\pi x)\sin(\pi y)$

which can be memory-intensive, especially for 3D or higher dimensional problems. This issue of memory intensiveness can be alleviated by considering distributed frameworks (such as Balu et al. [65]).

- *Irregular geometry:* The current paper focuses only on axis-aligned meshes. However, this method can be easily extended to handle irregular geometries. A domain with an irregular boundary can be embedded in a regular axis-aligned domain, and the calculations can be carried out exactly as described in this paper. In this case, additional information is required to describe which nodes lie within the irregular boundary and which lie outside.

We anticipate that such approaches that tightly integrate neural architectures with well-developed scientific computing approaches will successfully achieve our goal of a (near) real-time neural PDE inference.

Appendix 1: Representation of random diffusivity

With ω taken from the sample space Ω, the diffusivity/permeability ν can be written as an exponential of a random quantity Z

$$\nu = \exp(Z(\mathbf{x};\omega)). \tag{46}$$

We assume that Z is square integrable, i.e., $\mathbb{E}\left[|Z(\mathbf{x};\omega)|^2\right] < \infty$. Then, we can write Z using the Karhunen–Loeve expansion [68], as

$$Z(\mathbf{x};\omega) = \bar{Z}(\mathbf{x}) + \sum_{i=1}^{\infty} \sqrt{\lambda_i}\phi_i(\mathbf{x})\psi_i(\omega), \tag{47}$$

where $\bar{Z}(\mathbf{x}) = \mathbb{E}(Z(\mathbf{x},\omega))$, and $\psi_i(\omega)$ are independent random variables with zero mean and unit variance. λ_i and $\phi_i(\mathbf{x})$ are the eigenvalues and eigenvectors corresponding to the Fredholm equation

$$\int_D C_Z(\mathbf{s},\mathbf{t})\phi(s)\mathrm{d}s = \lambda\phi(\mathbf{t}), \tag{48}$$

where $C_Z(s,t)$ is the covariance kernel given by,

$$C_Z(\mathbf{s},\mathbf{t}) = \sigma_z^2 \exp\left(-\left[\frac{s_1 - t_1}{\eta_1} + \frac{s_2 - t_2}{\eta_2} + \frac{s_3 - t_3}{\eta_3}\right]\right), \tag{49}$$

where η_i is the correlation length in the x_i coordinate. This particular form of the covariance kernel is separable in the three coordinates, thus the eigenvalues and the eigenfunctions of the multi-dimensional case can be obtained by combining the eigenvalues and eigenfunctions of the one-dimensional covariance kernel given by:

Table 2 Norm of solution fields for a few randomly selected

#	a_1	a_2	a_3	a_4	a_5	a_6	$\|u_\theta\|_{L^2(D)}$	$\|u^h\|_{L^2(D)}$	$\|u_\theta - u^h\|_{L^2(D)}$	$\frac{\|u_\theta - u^h\|_{L^2(D)}}{\|u^h\|_{L^2(D)}}$
1	0.223	0.134	−0.141	−0.215	0.124	−0.201	37.397	37.377	0.150	0.0040
2	0.002	−0.050	0.014	0.183	0.147	−0.096	38.673	38.657	0.269	0.0069
3	0.182	−0.246	−0.087	−0.132	0.024	−0.052	37.446	37.464	0.237	0.0063
4	0.143	0.048	0.098	−0.090	−0.101	−0.071	37.929	37.972	0.234	0.0062
5	−0.133	−0.020	−0.110	0.194	0.093	0.022	37.912	37.825	0.253	0.0067

$$C_Z(s,t) = \sigma_Z^2 \exp\left(-\frac{s-t}{\eta}\right), \tag{50}$$

where σ_Z is the variance and η is the correlation length in one-dimension.

Equation 46 can then be written as

$$\tilde{\nu}(\mathbf{x};\omega) = \exp\left(\sum_{i=1}^{m} a_i \sqrt{\lambda_{xi}\lambda_{yi}}\phi_i(x)\psi_i(y)\right), \tag{51}$$

where a_i is an m-dimensional parameter, λ_x and λ_y are vectors of real numbers arranged in the order of monotonically decreasing values; and ϕ and ψ are functions of x and y, respectively. λ_{xi} is calculated as

$$\lambda_{xi} = \frac{2\eta\sigma_x}{(1+\eta^2\omega_x^2)}, \tag{52}$$

where ω_x is the solution to the system of transcendental equations obtained after differentiating Eq. 48 with respect to $\mathbf{t}$. λ_{yi} are calculated similarly. $\phi_i(x)$ are given by

$$\phi_i(x) = \frac{a_i}{2}\cos(a_i x) + \sin(a_i x), \tag{53}$$

and $\psi_i(y)$ are calculated similarly. We take $m = 6$ and assume that each a_i is uniformly distributed in $[-\sqrt{3}, \sqrt{3}]$, and thus, $\mathbf{a} \in [-\sqrt{3}, \sqrt{3}]^6$. The input diffusivity ν in all the examples in Sects. 6.3 and 6.4 are calculated by choosing the 6-dimensional coefficient $\mathbf{a}$ from $[-\sqrt{3}, \sqrt{3}]^6$.

Appendix 2: Further discussion on convergence studies

Discussion on the role of keeping $e_{\mathcal{H}}$ and e_{θ} low

If we choose a fixed network architecture and use it to solve Eq. 35 across different h-levels, then the errors do not necessarily decrease with decreasing h. As shown in Fig. 13, the errors actually increase when $h > 2^{-5}$. This reason for this behavior is that, when h becomes low, the number of discrete unknowns in the mesh (i.e., U_i's in Eq. 14) increases. In fact, in this case, the number of basis functions/unknowns, N is exactly equal to $\frac{1}{h^2}$. As h decreases, the size of the space V^h increases. However, since the network remains the same, the discrete function space V^h does not remain a subspace of V_θ^h anymore. This network function class also needs to get bigger to accommodate all the possible functions at the lower values of h. Figure 13 also shows the errors obtained when the network is indeed enhanced to make $V_\theta^h \supset V$ (this is a clone of the errors plotted in Fig. 7).

Appendix 3: Solutions to the parametric Poisson's equation

Randomly selected examples

See Figs. 14, 15.

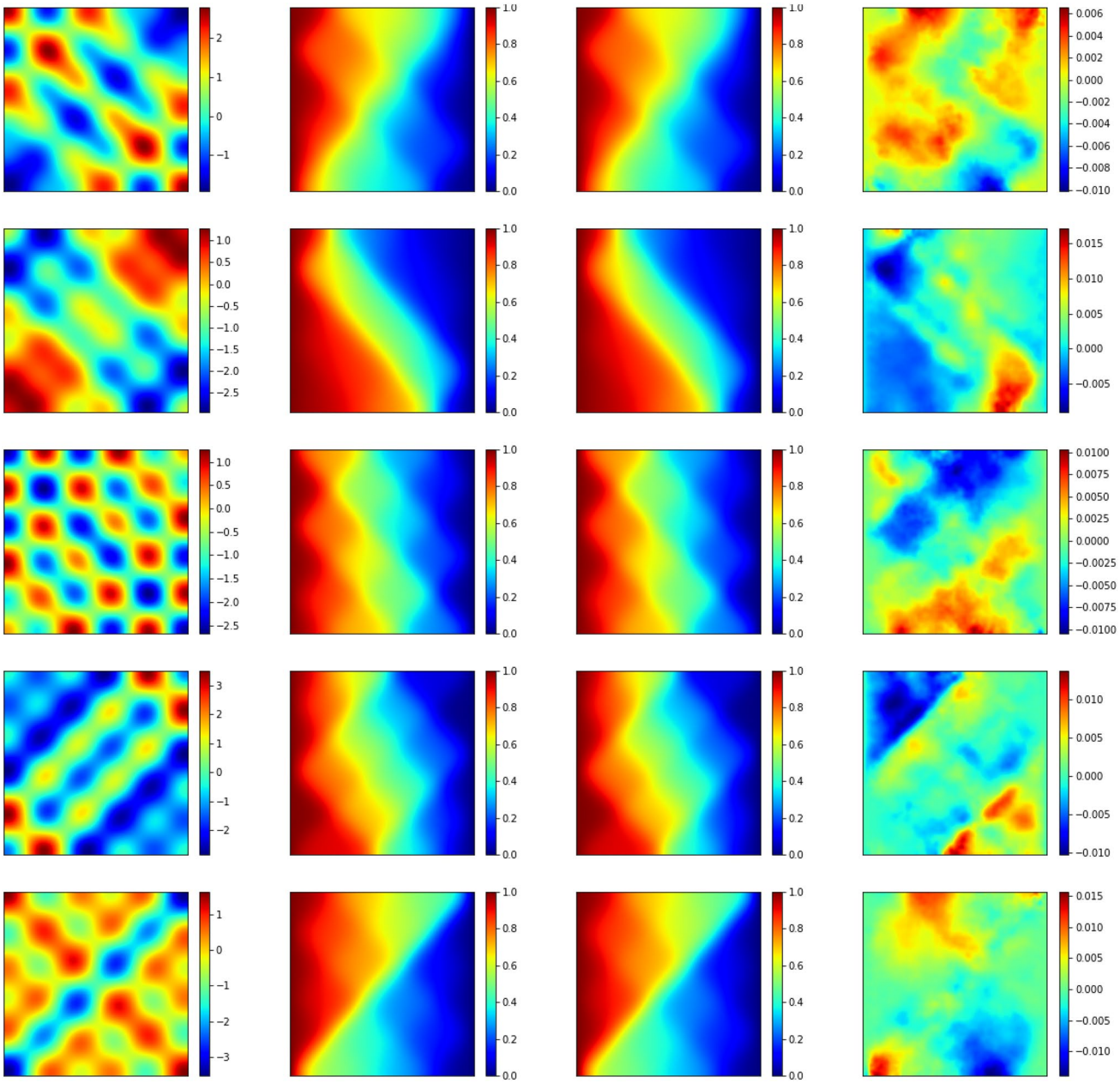


Fig. 14 Contours for the randomly selected examples presented in Table 2: (left) $\ln(\nu)$, (mid-left) u_θ, (mid-right) u^h and (right) $(u_\theta - u^h)$

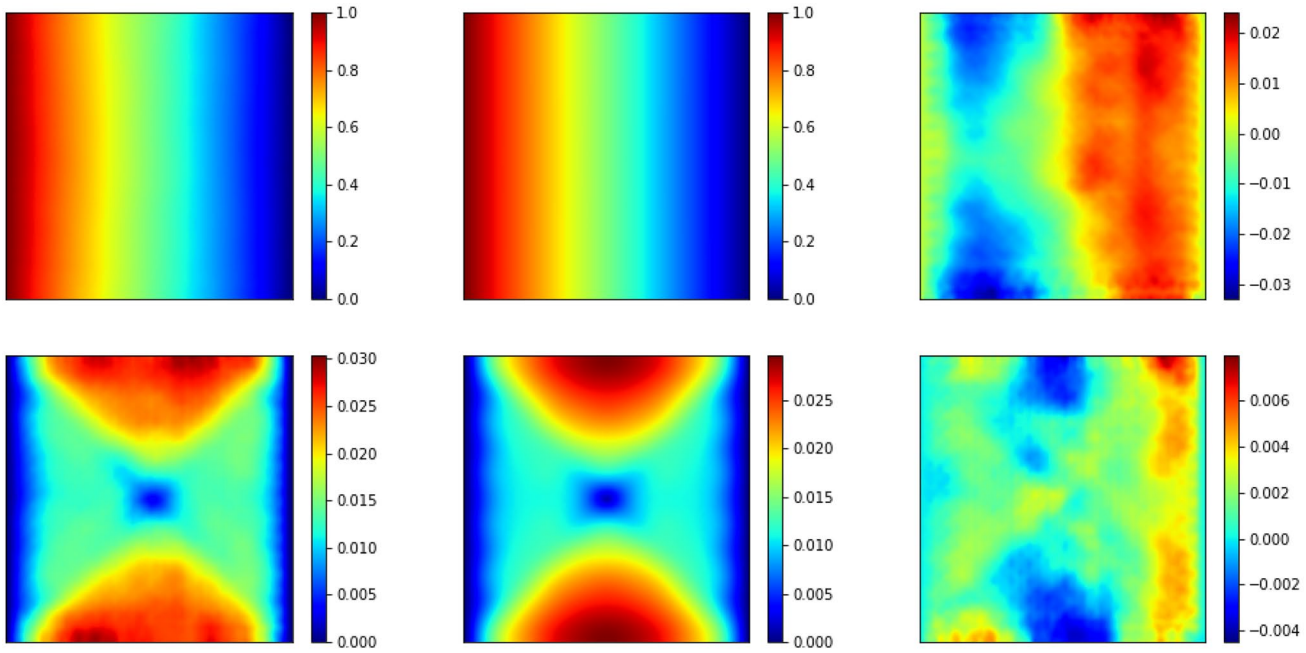


Fig. 15 (top) Mean and (bottom) standard-deviation fields: (left) *NeuFENet* (u_θ), (mid) Conventional FEM (u^h, (right) pointwise difference ($u_\theta - u^h$)

Mean and standard-deviation fields

See Table 3.

Table 3 Norm of the mean and standard-deviation fields

	$\|u_\theta\|_{L^2(D)}$	$\|u^h\|_{L^2(D)}$	$\|u_\theta - u^h\|_{L^2(D)}$	$\frac{\|u_\theta - u^h\|_{L^2(D)}}{\|u^h\|_{L^2(D)}}$
Mean	36.8186	37.1009	0.7919	0.0214
Std-dev	1.0933	1.0316	0.1407	0.1364

Acknowledgements This work was supported in part by the National Science Foundation under Grant Nos. CCF-2005804, LEAP-HI-2053760, OAC-1750865, CPS-FRONTIER-1954556, and USDA-NIFA-2021-67021-35329.

Data availability The data and code associated with this publication will be available on GitHub.

Declarations

Conflict of interest The authors have no conflict of interest to declare that are relevant to the content of this article.

References

1. Raissi M, Perdikaris P, Karniadakis GE (2019) Physics-informed neural networks: a deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. J Comput Phys 378:686–707
2. Rudy S, Alla A, Brunton SL, Nathan Kutz J (2019) Data-driven identification of parametric partial differential equations. SIAM J Appl Dyna Syst 18(2):643–660
3. Tompson J, Schlachter K, Sprechmann P, Perlin K (2017)Accelerating Eulerian fluid simulation with convolutional networks. In: International Conference on machine learning, pp 3424–3433. PMLR,
4. Raissi M, Karniadakis GE (2018) Hidden physics models: machine learning of nonlinear partial differential equations. J Comput Phys 357:125–141
5. Lu L, Jin P, Karniadakis GE (2019) Deeponet: learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. arXiv preprint arXiv:1910.03193,
6. Kharazmi E, Zhang Z, Karniadakis GE (2019) Variational physics-informed neural networks for solving partial differential equations. arXiv preprint arXiv:1912.00873,
7. Sirignano J, Spiliopoulos K (2018) Dgm: a deep learning algorithm for solving partial differential equations. J Comput Phys 375:1339–1364
8. Yang L, Dongkun Z, Karniadakis GE (2018) Physics-informed generative adversarial networks for stochastic differential equations. arXiv preprint arXiv:1811.02033,
9. Guofei Pang LL, Karniadakis GE (2019) fpinns: fractional physics-informed neural networks. SIAM J Sci Comput 41(4):A2603–A2626
10. Karumuri S, Tripathy R, Bilionis I, Panchal J (2020) Simulator-free solution of high-dimensional stochastic elliptic partial differential equations using deep neural networks. J Comput Phys 404:109120
11. Han J, Arnulf J, Weinan E (2018) Solving high-dimensional partial differential equations using deep learning. Proc Natl Acad Sci 115(34):8505–8510

12. Michoski C, Milosavljevic M, Oliver T, Hatch D (2019) Solving irregular and data-enriched differential equations using deep neural networks. arXiv preprint arXiv:1905.04351
13. Samaniego E, Anitescu C, Goswami S, Nguyen-Thanh VM, Hongwei G, Khader H, Zhuang X, Rabczuk T (2020) An energy approach to the solution of partial differential equations in computational mechanics via machine learning: Concepts, implementation and applications. Comput Methods Appl Mech Eng 362:112790
14. Ramabathiran Amuthan A, Prabhu R (2021) Spinn: sparse, physics-based, and partially interpretable neural networks for pdes. J Comput Phys 445:110600
15. Botelho S, Joshi A, Khara B, Sarkar S, Hegde C, Adavani S, Ganapathysubramanian B (2020) Deep generative models that solve pdes: distributed computing for training large data-free models. arXiv preprint arXiv:2007.12792,
16. Mitusch Sebastian K, Funke Simon W, Miroslav K (2021) Hybrid fem-nn models: combining artificial neural networks with the finite element method. J Comput Phys 446:110651
17. Jokar M, Semperlotti F (2021) Finite element network analysis: a machine learning based computational framework for the simulation of physical systems. Comput Struct 247:106484
18. Sitzmann V, Martel JNP, Bergman AW, Lindell DB, Wetzstein G (2020) Implicit neural representations with periodic activation functions. Adv Neural Inform Process Syst 33:7462–7473
19. Mishra S, Konstantin Rusch T (2021) Enhancing accuracy of deep learning algorithms by training with low-discrepancy sequences. SIAM J Numer Anal 59(3):1811–1834
20. Zhu Y, Zabaras N, Koutsourelakis P-S, Perdikaris P (2019) Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. J Comput Phys 394:56–81
21. Wen G, Li Z, Azizzadenesheli K, Anandkumar A, Benson Sally M (2021) U-fno–an enhanced fourier neural operator based-deep learning model for multiphase flow. arXiv preprint arXiv:2109.03697,
22. Ranade R, Hill C, Pathak J (2021) Discretizationnet: a machine-learning based solver for Navier-Stokes equations using finite volume discretization. Comput Methods Appl Mech Eng 378:113722
23. Brenner S, Scott R (2007) The mathematical theory of finite element methods, vol 15. Springer Science & Business Media
24. Larson MG, Bengzon F (2013) The finite element method: theory, implementation, and applications, vol 10. Springer Science & Business Media
25. Shin Y , Zhang Z, Karniadakis GE (2020) Error estimates of residual minimization using neural networks for linear pdes. arXiv preprint arXiv:2010.08019,
26. Mishra S, Molinaro R (2020) Estimates on the generalization error of physics informed neural networks (pinns) for approximating pdes. arXiv preprint arXiv:2006.16144
27. Jiao Y, Lai Y, Luo Y, Wang Y, Yang Y (2021) Error analysis of deep ritz methods for elliptic equations. arXiv preprint arXiv:2107.14478
28. He J, Li L, Xu J, Zheng C (2018) Relu deep neural networks and linear finite elements. arXiv preprint arXiv:1807.03973
29. Takeuchi J, Kosugi Y (1994) Neural network representation of finite element method. Neural Netw 7(2):389–395
30. Xu G, Littlefair G, Penson R, Callan R (1999) Application of fe-based neural networks to dynamic problems. In: ICONIP'99. ANZIIS'99 & ANNES'99 & ACNN'99. 6th International Conference on neural information processing. Proceedings (Cat. No. 99EX378), volume 3, pp 1039–1044. IEEE
31. Ramuhalli P, Udpa L, Udpa SS (2005) Finite-element neural networks for solving differential equations. IEEE Trans Neural Netw 16(6):1381–1392
32. Chao X, Wang C, Ji F, Yuan X (2012) Finite-element neural network-based solving 3-d differential equations in mfl. IEEE Trans Magn 48(12):4747–4756
33. Khodayi-Mehr R, Zavlanos M (2020) Varnet: variational neural networks for the solution of partial differential equations. In: Learning for dynamics and control, pp 98–307. PMLR,
34. Gao H, Zahr Matthew J, Wang J-X (2022) Physics-informed graph neural Galerkin networks: a unified framework for solving pde-governed forward and inverse problems. Comput Methods Appl Mech Eng 390:114502
35. Yao H, Gao Y, Liu Y (2020) Fea-net: a physics-guided data-driven model for efficient mechanical response prediction. Comput Methods Appl Mech Eng 363:112892
36. Evans LC (1998) Partial differential equations. Graduate Stud Math 19(4):7
37. Reddy JN (2010) An introduction to the finite element method, vol 1221. McGraw-Hill New York
38. Bing Y et al (2017) The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. arXiv preprint arXiv:1710.00211
39. Ming YL et al (2021) Deep Nitsche method: Deep ritz method with essential boundary conditions. Commun Comput Phys 29(5):1365–1384
40. Courte L, Zeinhofer M (2021) Robin pre-training for the deep ritz method. arXiv preprint arXiv:2106.06219
41. Müller J, Zeinhofer M (2022) Error estimates for the deep ritz method with boundary penalty. In: Mathematical and Scientific Machine Learning, pp 215–230. PMLR
42. Müller J, Zeinhofer M (2022) Notes on exact boundary values in residual minimisation. In: Mathematical and Scientific Machine Learning, pp 231–240. PMLR
43. Dondl P, Müller J, Zeinhofer M (2022) Uniform convergence guarantees for the deep ritz method for nonlinear problems. Adv Contin Discrete Models 2022(1):1–19
44. Hyuk Lee and In Seok Kang (1990) Neural algorithm for solving differential equations. J Comput Phys 91(1):110–131
45. Lagaris IE, Likas A, Fotiadis DI (1998) Artificial neural networks for solving ordinary and partial differential equations. IEEE Trans Neural Netw 9(5):987–1000
46. Malek A, Beidokhti Shekari R (2006) Numerical solution for high order differential equations using a hybrid neural network optimization method. Appl Math Comput 183(1):260–271
47. Sukumar N, Srivastava A (2021) Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. arXiv preprint arXiv:2104.08426
48. IsaacE L, AristidisC L, DimitrisG P (2000) Neural-network methods for boundary value problems with irregular boundaries. IEEE Trans Neural Netw 11(5):1041–1049
49. Remco Van der M, Cornelis O, Anastasia B (2020) Optimally weighted loss functions for solving pdes with neural networks. arXiv preprint arXiv:2002.06269
50. Sifan W, Yujun T, and Paris P (2020) Understanding and mitigating gradient pathologies in physics-informed neural networks. arXiv preprint arXiv:2001.04536,
51. Oliver H, Susheela N, Mohammad Amin N, Akshay S, Kaustubh T, Zhiwei F, Max R, Wonmin B, Sanjay C (2021) Nvidia simnet: an ai-accelerated multi-physics simulation framework. In: International Conference on computational science, pp 447–461. Springer
52. Wang S, Perdikaris P (2021) Long-time integration of parametric evolution equations with physics-informed deeponets. arXiv preprint arXiv:2106.05384
53. Paganini M, de Oliveira L, Nachman B (2018) Calogan: simulating 3d high energy particle showers in multilayer electromagnetic calorimeters with generative adversarial networks. Phys Rev D 97(1):014021

54. Krishnapriyan AS, Gholami A, Zhe S, Kirby RM, Mahoney MW (2021) Characterizing possible failure modes in physics-informed neural networks. arXiv preprint arXiv:2109.01050
55. Wang S, Teng Y, Perdikaris P (2021) Understanding and mitigating gradient flow pathologies in physics-informed neural networks. SIAM J Sci Comput 43(5):A3055–A3081
56. Fox C (1987) An introduction to the calculus of variations. Courier Corporation
57. Kingma DP, Jimmy B (2014) Adam: a method for stochastic optimization. arXiv preprint arXiv:1412.6980
58. Olaf R, Philipp F, Brox T (2015) U-net: convolutional networks for biomedical image segmentation. In: International Conference on medical image computing and computer-assisted intervention, pp 234–241. Springer
59. Özgün Ç, Abdulkadir A, Lienkamp SS, Brox T, Ronneberger O (2016) 3D U-Net: learning dense volumetric segmentation from sparse annotation. In: International Conference on medical image computing and computer-assisted intervention, pp 424–432. Springer
60. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L et al (2019) Pytorch: an imperative style, high-performance deep learning library. Adv Neural Inf Process Syst 32:8026–8037
61. Bubeck S, Eldan R, Lee YT, Mikulincer D (2020) Network size and weights size for memorization with two-layers neural networks. arXiv preprint arXiv:2006.02855
62. LeCun Y, Bottou L, Bengio Y, Haffner P (1998) Gradient-based learning applied to document recognition. Proc IEEE 86(11):2278–2324
63. Oden JT, Reddy JN (2012) An introduction to the mathematical theory of finite elements. Courier Corporation
64. Barron AR (1993) Universal approximation bounds for superpositions of a sigmoidal function. IEEE Trans Inf Theory 39(3):930–945
65. Balu A, Botelho S, Khara B, Rao V, Hegde C, Sarkar S, Adavani S, Krishnamurthy A, Ganapathysubramanian B (2021) Distributed multigrid neural solvers on megavoxel domains. arXiv preprint arXiv:2104.14538
66. Lu L, Meng X, Mao Z, Karniadakis GE (2021) Deepxde: a deep learning library for solving differential equations. SIAM Rev 63(1):208–228
67. Hughes TJR, Scovazzi G, Franca LP (2017) Multiscale and stabilized methods. In: Stein E, de Borst R, Hughes TJR (eds) Encyclopedia of computational mechanics second edition, vol 5, chap 2, pp 1–64. John Wiley & Sons, Ltd. https://doi.org/10.1002/9781119176817.ecm2051
68. Ghanem RG, Spanos PD (2003) Stochastic finite elements: a spectral approach. Courier Corporation