Journal of Computational Mathematics
Vol.44, No.1, 2026, 1–34.

http://www.global-sci.org/jcm
doi:10.4208/jcm.2407-m2024-0047

ELEMENT LEARNING: A SYSTEMATIC APPROACH OF ACCELERATING FINITE ELEMENT-TYPE METHODS VIA MACHINE LEARNING, WITH APPLICATIONS TO RADIATIVE TRANSFER*

Shukai Du[1)]
Department of Mathematics, University of Wisconsin–Madison, USA
Email: sdu49@wisc.edu
Samuel N. Stechmann
Department of Mathematics, Department of Atmospheric and Oceanic Sciences, University of Wisconsin–Madison, USA

Abstract

In this paper, we propose a systematic approach for accelerating finite element-type methods by machine learning for the numerical solution of partial differential equations (PDEs). The main idea is to use a neural network to learn the solution map of the PDEs and to do so in an element-wise fashion. This map takes input of the element geometry and the PDE's parameters on that element, and gives output of two operators: (1) the in2out operator for inter-element communication, and (2) the in2sol operator (Green's function) for element-wise solution recovery. A significant advantage of this approach is that, once trained, this network can be used for the numerical solution of the PDE for any domain geometry and any parameter distribution without retraining. Also, the training is significantly simpler since it is done on the element level instead on the entire domain. We call this approach element learning. This method is closely related to hybridizable discontinuous Galerkin (HDG) methods in the sense that the local solvers of HDG are replaced by machine learning approaches. Numerical tests are presented for an example PDE, the radiative transfer or radiation transport equation, in a variety of scenarios with idealized or realistic cloud fields, with smooth or sharp gradient in the cloud boundary transition. Under a fixed accuracy level of 10^{-3} in the relative L^2 error, and polynomial degree $p = 6$ in each element, we observe an approximately 5 to 10 times speed-up by element learning compared to a classical finite element-type method.

Mathematics subject classification: 65N30, 65N55, 68T07.
Key words: Scientific machine learning, Spectral element, Discontinuous Galerkin, Hybridization, Hybridizable discontinuous Galerkin, Radiation transport, Radiative transfer.

1. Introduction

1.1. Background and motivation

In the past decade, (artificial) neural networks and machine learning tools have surfaced as game-changing technologies across numerous fields, resolving an array of challenging problems.

* Received March 5, 2024 / Revised version received June 1, 2024 / Accepted July 15, 2024 / Published online October 8, 2024 /
1) Corresponding author

Examples include image recognition [61,66], playing the game go [97], protein folding [56], and large language models such as GPT3 [9].

Given these impressive results, it is reasonable to envision the potential of neural networks (NNs) for the numerical solution of partial differential equations or other scientific computing problems. There has been significant work in this direction [6, 10, 11, 15–17, 25, 27, 29, 40, 41, 44, 45, 48, 53, 55, 57–59, 71, 81, 83, 86, 94, 100, 103, 106, 108, 109]. As examples, we mention two major groups – (1) neural networks as function approximators, (2) neural networks as operator approximators.

The first type of these methods uses neural networks to approximate the solution of PDEs. Examples include physics-informed neural networks (PINNs) [24,73,75,87] and Deep Ritz methods [107]. These methods have demonstrated promising results, especially in the realm of high-dimensional problems or inverse problems, since they seem to bypass the notorious "curse of dimensionality", potentially attributable to the neural network's efficient approximation capabilities in high-dimensional spaces [36, 47, 78].

While the potential advantages are promising, it remains uncertain if the aforementioned methods have a real advantage over traditional algorithms, such as finite element-type methods, when addressing many classic PDEs. A crucial factor contributing to this uncertainty stems from the complex optimization problems and error landscapes-often non-convex-introduced by neural networks due to their hidden layers and non-linear activation functions.

The second type of these methods employs neural networks to approximate operators, with methods like Neural Operator [60,76] and DeepOnet [72] serving as leading examples. A primary advantage of these methods hinges on their speed – once trained, solving a problem is simply a forward propagation of the network. For instance, in [82], it was reported that the neural operator based method can be orders of magnitude faster than classical finite volume/difference based methods in weather simulation. This suggests great potential of using machine learning to accelerate computation.

However, this type of approach is essentially data-driven and its performance can depend on the training samples and how well it generalizes. Consequently, the reliability of the result may falter during extreme events, as indicated in [82]. In addition to this, it is common for these approaches to be constrained by the geometry of the domain and boundary conditions. For instance, the Fourier neural operator method that relies on Fourier expansion for solution representation is specifically tailored for rectangular/cubic domains with periodic boundary conditions [60]. However, there are recent attempts to extend operator learning to more complex geometries [70, 95].

On the other hand, for many years, traditional methods like finite element-type methods have played a critical role in the advancement of various scientific and engineering disciplines. These methodologies, honed and well-understood over more than five decades, have given us a plethora of reliable and robust techniques successfully employed across an array of problems. Examples include, but are definitely not limited to, conforming and non-conforming finite element [8, 23], mixed finite element [80, 89], discontinuous Galerkin [2, 21], along with effective techniques such as slope limiter [21, 69], multi-scale finite elements [38], finite element exterior calculus [3, 43], *hp*-adaptivity [5, 54] to tailor these methods to different application scenarios [22, 33–35, 42, 50, 74, 77, 101].

Despite the undeniable utility of these classical approaches, they also come with constraints. A primary limitation is on their speed, especially when facing high-dimensional problems, such as those found in radiative transfer, or scenarios when fast forward solvers are necessary, such

as in inverse problems. Furthermore, they may lack efficiency in dealing with data or geometric structures characterized by multi-scale variations. Considering these factors, it is desirable to devise ways of reducing the computational cost of these classical approaches.

In this paper, we propose a novel approach (which we call element learning), aimed at accelerating finite element-type methods via machine learning. This effort seeks to leverage the extensive knowledge gained from decades of development in classical finite element methods, with the objective of using the modern computational power of machine learning to enhance the computational speed.

1.2. Main idea

To describe the main ideas of the methods here, we will utilize the contexts of both a general class of partial differential equations and also the specific example of the radiative transfer or radiation transport equation. To proceed with the discussion, we note that a wide variety of PDEs, including radiative transfer, can be written in the following abstract formulation:

$$\mathcal{L}(\sigma)[u] = f \quad \text{in } \Omega. \tag{1.1}$$

In this representation, $\mathcal{L}(\sigma)$ represents a combination of differential and integral operators, which also depend on the coefficients σ. We are in pursuit of a solution denoted by u, while f is given as the data, comprising both the external force term and the prescribed boundary conditions. For example, in the context of atmospheric radiation, σ represents the optical properties of the atmosphere (extinction and scattering coefficients), u denotes the radiant intensity, and f symbolizes the inflow and black body radiation.

The specific PDE example here is chosen to be the radiative transfer equation for a number of reasons. Radiative transfer is important in many fields, including medical imaging [4, 91], neutron transport [68, 90], and climate and weather prediction [51, 52]. Furthermore, the equation offers intriguing mathematical characteristics as it exhibits both hyperbolic properties in weak scattering scenarios and elliptic properties in instances of strong scattering. Finally, the equation is computational challenging because of its high or moderately-high dimensionality so a faster solver is still highly demanded.

For finite element-type methods, the domain Ω is partitioned into a collection of simple geometric objects (e.g. triangles, tetrahedron, rectangles, cubes), and a key aspect for the success of these methods is the effective incorporation of the solution operators on each element to construct the solution for the entire domain. The study of this has engendered a plethora of diverse finite element methods, including conforming and non-conforming, mixed finite element, and discontinuous Galerkin methods. Recently, it was realised that many of these methods can be implemented within the framework of hybridizable discontinuous Galerkin methods in a unified way [19, 20, 31].

To elaborate on hybridizable finite element-type methods and their key aspects for element learning, we would need to find the discretization of the following two operators on an element-wise basis:

- The in2out operator responsible for inter-element communication

$$u_{\partial K}^{\text{in}} \;\rightarrow\; u_{\partial K}^{\text{out}}. \tag{1.2}$$

  For radiative transfer, it is the inflow-radiation-to-outflow-radiation operator. For elliptic-type equations, this equates to the Dirichlet-to-Neumann operator (the DtN map).

- The in2sol operator responsible for recovering the solution (or desired properties of the solution) on each element

$$u^{\text{in}}_{\partial K} \;\rightarrow\; u_K. \tag{1.3}$$

  For radiative transfer, it is the inflow-radiation-to-solution operator. For elliptic-type equations, this is the Dirichlet-to-solution operator (i.e. the Green's function). Note that we can also replace u_K by its partial statistics if desired. For instance, we can replace the radiative intensity u_K by its mean intensity or radiative heating rate, if desired.

Once these two types of operators or their approximations are obtained, we can recover the solution on the whole domain following the standard HDG routine: (1) use the in2out operators to assemble a global system (involving only the variables on the skeleton of the mesh), (2) solve the global system, (3) use in2sol operator to recover the solution of interest in an element-wise fashion. See Figs. 1.1 and 1.2 for a visual demonstration. (Further details will be elaborated in Section 2.2.)

We now introduce the primary concept of element learning (see Fig. 1.3 for a visual demonstration). In this approach, instead of directly computing the approximations for the in2out

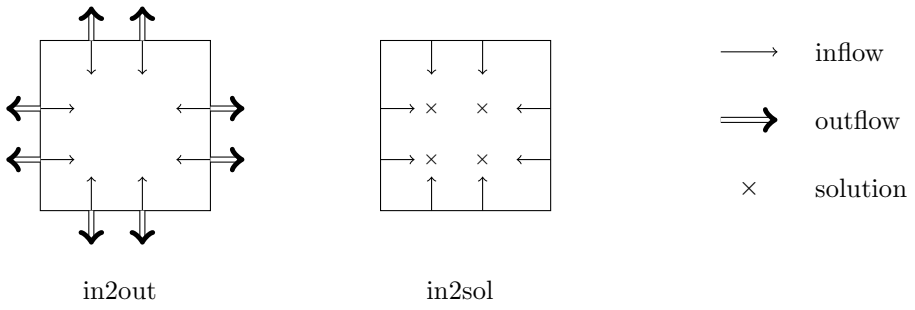

Fig. 1.1. Symbolic representation of the in2out and in2sol operators, from (1.2) and (1.3), on a square element. When these operators from one element are coupled with the operators from neighboring elements, they can be used to find a global solution (see Fig. 1.2).

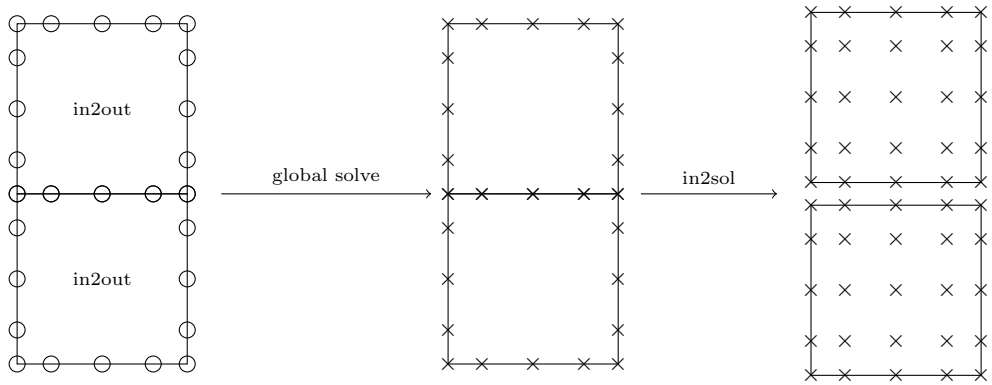

Fig. 1.2. Solution strategy of both element learning and the standard HDG method. To create a global solver, the local solvers (in2out operators, from Fig. 1.1) from neighboring elements are coupled together. In this schematic diagram, we use "○" for undetermined DOFs and "×" for determined DOFs (solved solution).

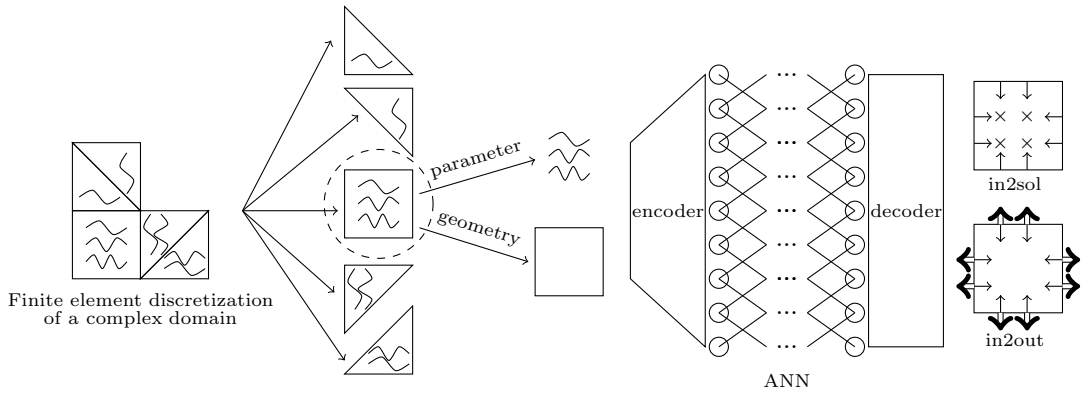

Fig. 1.3. Element learning - architecture. A finite element mesh is broken down element by element. For each element, a neural network takes the element geometry and PDE parameters as inputs, and returns the in2sol and in2out operators, (1.2) and (1.3), as outputs. Then, beyond this schematic diagram, the in2out operators of neighboring elements are systematically coupled together to assemble a global solver (see Fig. 1.2).

and in2sol operators by inverting a local system on each element, we leverage a neural network to provide the approximations to the in2out and in2sol operators. Note that the neural network will not itself be the approximation of the in2out nor in2sol operator; instead, here, the neural network will provide the in2out and in2sol operators at outputs. That is, the neural network approximates the following operators:

$$(\sigma, K) \;\to\; \left[u^{\text{in}}_{\partial K} \;\to\; u^{\text{out}}_{\partial K}\right], \tag{1.4a}$$

$$(\sigma, K) \;\to\; \left[u^{\text{in}}_{\partial K} \;\to\; u_K\right]. \tag{1.4b}$$

The above operators take the coefficients σ and the element geometry K as inputs. The outputs, in turn, are the in2out and in2sol operators. Thus, once this network is trained, it only needs a simple forward propagation of the network to obtain the approximations for the in2out and in2sol operators. It is important to highlight that although the PDEs might be linear (the in2out and in2sol are linear operators), the operators (1.4) are non-linear. This justifies the use of neural networks with non-linear activation functions and hidden layers.

Certain benefits arise from defining the operators in (1.2) and (1.3) and the nonlinear mapping in (1.4) on an element, rather than over the entire domain. For instance, in working on an individual element rather than the entire domain, there is a significant reduction in the variations from the parameter σ, and in the complexity of geometry. Hence it is reasonable to anticipate that the neural network can provide a more reliable and accurate approximation to this non-linear mapping (1.4) on an element, compared to a neural-network-based approximation of this mapping on the whole domain.

Element learning can be regarded as a framework that bridges the data-driven machine learning approach and the traditional finite element approach. To see this more clearly, note that in the extreme cases, such as when we have only a single element covering the whole domain, the methodology simplifies into a purely data-driven approach where the data are generated by a finite element method with a single element (with potentially high-order approximations leading it to a spectral-like method). Conversely, if we minimize the mesh size or decrease the polynomial degree, causing a significant reduction of the variations of the discretization of the coefficients distribution, the methodology approximates a traditional finite element method.

Some advantageous features of element learning are the following:

- The methodology can inherit numerous beneficial properties from classical finite element methods, such as proficient handling of complex geometry and boundary conditions, as it operates within the traditional finite element framework.

- A computational speedup is expected for element learning in comparison to traditional finite element-type methods, since element learning brings a memory reduction via hybridization (as in, e.g. HDG) and a faster creation of the in2out and in2sol operators via machine learning.

- The methodology does not require retraining when geometry changes or mesh size is refined or coefficients are modified. Training is only necessary for a single element with various geometrical and coefficient variations. Subsequently, this element can be used freely to solve any domain geometry and coefficient distribution.

- Compared to many data-driven approaches, element learning is expected to be more reliable. This is due to the considerable reduction in geometric complexity and coefficient variations within each element. As the mesh is further refined, the reliability of the element learning approach increases.

- The methodology facilitates a natural coupling between data-driven and classical finite element approaches. For instance, one could use the element learning solution as a preconditioner/initial condition to accelerate the convergence of a classical finite element method.

- Element learning has a good potential to be used for massive-scale parallel computing on GPUs since the forward propagation of the neural network for the in2sol and in2out operators on each element are completely independent processes.

The organization of the remainder of the paper is as follows. In Section 2, we introduce the element learning method for radiative transfer equations. Then, in Section 3, we present numerical experiments to test the performance of the element learning methods in various scenarios, by comparing it with classical finite element-type methods such as discontinuous Galerkin (DG) and HDG methods. A concluding discussion is in Section 4.

2. Element Learning for Radiative Transfer

In this section, we begin with Section 2.1, in which we introduce the radiative transfer or radiation transport equation that is considered in this paper.

Then, in the Section 2.2, we show how to obtain a discretization of the in2out and in2sol operators by hybridizable discontinuous Galerkin method, which lays the foundation for element learning. This subsection include three parts: discontinuous Galerkin, hybridizable discontinuous Galerkin, and the calculation for the in2out and in2sol operators.

Finally, in the Section 2.4 we introduce the element learning accelerating method, including the neural network structure and the data generation algorithm.

2.1. Radiative transfer

Let Ω be a spatial domain and S be the unit sphere in $\mathbb{R}^d$. We first introduce the inflow and outflow boundaries

$$\Gamma^- := \{(\mathbf{x}, \mathbf{s}) \in \partial\Omega \times S : \mathbf{n}(\mathbf{x}) \cdot \mathbf{s} \leq 0\},$$
$$\Gamma^+ := \{(\mathbf{x}, \mathbf{s}) \in \partial\Omega \times S : \mathbf{n}(\mathbf{x}) \cdot \mathbf{s} \geq 0\},$$

where $\mathbf{n}(\mathbf{x})$ is the outward-pointing normal vector at $\mathbf{x} \in \partial\Omega$. We consider the following radiative transfer equation:

$$\mathbf{s} \cdot \nabla u + \sigma_e u - \sigma_s \int_S p(\mathbf{s}, \mathbf{s}') u(\mathbf{s}') ds' = f \quad \text{in } \Omega, \tag{2.1a}$$
$$u = g \quad \text{on } \Gamma^-, \tag{2.1b}$$

where the solution $u(\mathbf{x}, \mathbf{s})$ represents the radiant intensity at $\mathbf{x}$ in the direction $\mathbf{s} \in S, \sigma_e(\mathbf{x})$ and $\sigma_s(\mathbf{x})$ are the extinction and scattering coefficients, respectively, $f(\mathbf{x}, \mathbf{s})$ and $g(\mathbf{x}, \mathbf{s})$ are the source term and the inflow radiation, respectively. Here $p(\mathbf{s}, \mathbf{s}')$ is the scattering phase function, for which we consider the Henyey-Greenstein phase function

$$p(\mathbf{s}, \mathbf{s}') = \frac{1 - g_{\text{asym}}^2}{c\big(1 + g_{\text{asym}}^2 - 2g_{\text{asym}} \cos \text{ang}(\mathbf{s}, \mathbf{s}')\big)^{3/2}},$$

where $\text{ang}(\mathbf{s}, \mathbf{s}')$ denotes the angle between $\mathbf{s}$ and $\mathbf{s}'$, g_{asym} is the asymmetric parameter taking values in $[0, 1]$, and the constant c is chosen such that $\int_S p(\mathbf{s}, \mathbf{s}') ds' = 1$. For isotropic scattering we have $g = 0$, while for the scattering of short-wave (solar) radiation in water clouds, g can take values from 0.8 to 0.9 [98].

2.2. Discretizations of the in2out and int2sol operators

In this subsection, we show how to obtain a discretization for the in2out and the in2sol operators by hybridizable discontinuous Galerkin methods. Since this is likely the first paper that introduces HDG methods for radiative transfer, we shall introduce this method with some details in the following three parts: Discontinuous Galerkin methods, HDG methods, and the implementation of HDG for the discretization of the in2out and in2sol operators.

2.2.1. Discontinuous Galerkin methods

We begin by partitioning the spatial domain Ω into a collection of polyhedral elements $\mathcal{T}_h$, and partitioning the unit sphere S into a collection of angular elements $\mathcal{T}_h^a$. We shall assume the partitions are conforming and shape-regular, these are standard assumptions for finite element-type methods and we refer to [8] for more details.

Let $\mathcal{E}_h$ be the collection of all faces of $\mathcal{T}_h$, and let $\mathcal{E}_h^0$ and $\mathcal{E}_h^\Gamma$ be the collection of the interior faces ($F \not\subset \partial\Omega$) and boundary faces ($F \subset \partial\Omega$), respectively. We also define the skeleton of the mesh as $\mathcal{E} := \cup_{F \in \mathcal{E}_h} F$. For each element $K \in \mathcal{T}_h$, we denote by $\mathcal{E}_K$ the collection of the faces of K. We require that the angular partition $\mathcal{T}_h^a$ respects the spatial partition $\mathcal{T}_h$ in the sense that for any $K \times K^a \in \mathcal{T}_h \times \mathcal{T}_h^a$ and a given face $F \in \mathcal{E}_K$, the angular element K^a is either entirely inflow or entirely outflow with respect to the face F and the element K.

We denote by $\mathcal{P}_k(K)$ the polynomial space with degree k supported on the element K. Now we can introduce the following DG approximation spaces:

$$V_h := \left\{ u_h \in L^2(\Omega \times S) : \, u_h\big|_{K\times K^a} \in \mathcal{P}_k(K) \otimes \mathcal{P}_{k^a}(K^a), \;\; \forall\, K \times K^a \in \mathcal{T}_h \times \mathcal{T}_h^a \right\}.$$

Namely, u_h is piece-wise polynomial on each spatial-angular element $K \times K^a \in \mathcal{T}_h \times \mathcal{T}_h^a$.

For the DG methods, we seek for $u_h \in V_h$ such that

$$\begin{aligned} &\sum_{K\in\mathcal{T}_h} \sum_{K^a\in\mathcal{T}_h^a} \bigg(\int_{K^a}\int_{\partial K} (\mathbf{s}\cdot\mathbf{n})\widehat{u}_h v - \int_{K^a}\int_K u_h \mathbf{s}\cdot\nabla v + \int_{K^a}\int_K \sigma_e u_h v \\ &\qquad\qquad - \int_{K^a}\int_K \sigma_s(\mathbf{x}) \int_S p(\mathbf{s},\mathbf{s}') u_h(\mathbf{x},\mathbf{s}') ds' v(\mathbf{x},\mathbf{s}) dx ds \bigg) \\ &= \sum_{K\in\mathcal{T}_h} \sum_{K^a\in\mathcal{T}_h^a} \int_{K^a}\int_K f v \end{aligned} \tag{2.2a}$$

hold for all $v \in V_h$, where the numerical flux quantity $\widehat{u}_h$ is defined as follows:

$$\widehat{u}_h(\mathbf{x},\mathbf{s}) = \begin{cases} u_h(\mathbf{x},\mathbf{s}), & \text{if} \quad \mathbf{n}_F\cdot\mathbf{s} \geq 0, \\ u_h^{\text{nbr}}(\mathbf{x},\mathbf{s}), & \text{if} \quad \mathbf{n}_F\cdot\mathbf{s} \leq 0, \quad F \in \mathcal{E}_h^0, \\ g(\mathbf{x},\mathbf{s}), & \text{if} \quad \mathbf{n}_F\cdot\mathbf{s} \leq 0, \quad F \in \mathcal{E}_h^\Gamma, \end{cases} \tag{2.2b}$$

where u_h^{nbr} is the value of u_h from the neighbouring elements. Namely, we choose the upwind flux for $\widehat{u}_h$. We refer to [33,34] and references therein for more details on the DG discretization of the radiative transfer equation.

2.2.2. Hybridization

For hybridizable DG (HDG) methods, we introduce another variable $\widehat{u}_h$ living on the skeleton $\mathcal{E}$. This variable (also known as hybrid unknown) can be regarded as a Lagrange multiplier of enforcing the consistency condition. To begin with, let us introduce its corresponding approximation space

$$\widehat{V}_h := \left\{ \widehat{u}_h \in L^2\big((\cup_{F\in\mathcal{E}_h} F)\times S\big) : \, \widehat{u}_h\big|_{F\times K^a} \in \mathcal{P}_k(F)\otimes\mathcal{P}_{k^a}(K^a), \;\; \forall\, F\times K^a \in \mathcal{E}_h \times \mathcal{T}_h^a \right\}.$$

Namely, $\widehat{u}_h$ is a piece-wise polynomial on the skeleton $\mathcal{E}$, where each piece of the skeleton is a face $F \in \mathcal{E}_h$. We shall also introduce

$$\widehat{V}_h^g := \left\{ \widehat{u}_h \in \widehat{V}_h : \, \widehat{u}_h = \mathrm{P}_M g, \;\; \forall\, F\times K^a \in \mathcal{E}_h \times \mathcal{T}_h^a \;\text{ s.t. }\; F\subset\partial\Omega,\, \mathbf{n}_F\cdot K^a \leq 0 \right\}, \tag{2.3}$$

where P_M is the orthogonal projection from the space $L^2(F\times K^a)$ to the space $\mathcal{P}_k(F)\otimes P_{k^a}(K^a)$. This space can be regarded as projecting the boundary data g to the hybrid unknown space $\widehat{V}_h$.

For HDG methods, we seek $(u_h, \widehat{u}_h) \in V_h \times \widehat{V}_h^g$ such that

$$\begin{aligned} &\sum_{F\in\mathcal{E}_K} \sum_{K^a\in\mathcal{T}_h^a\cap(K^a\cdot\mathbf{n}_F\geq 0)} \int_{K^a}\int_F (\mathbf{s}\cdot\mathbf{n}_F) u_h v \\ &+ \sum_{F\in\mathcal{E}_K} \sum_{K^a\in\mathcal{T}_h^a\cap(K^a\cdot\mathbf{n}_F\leq 0)} \int_{K^a}\int_F (\mathbf{s}\cdot\mathbf{n}_F) \widehat{u}_h v \end{aligned}$$

$$
\begin{aligned}
&+\sum_{K^a\in\mathcal{T}_h^a}\left(-\int_{K^a}\int_K u_h\mathbf{s}\cdot\nabla v+\int_{K^a}\int_K \sigma_e u_h v\right)\\
&-\sum_{K^a\in\mathcal{T}_h^a}\int_{K^a}\int_K \sigma_s(\mathbf{x})\int_S p(\mathbf{s},\mathbf{s}')u_h(\mathbf{x},\mathbf{s}')ds'v(\mathbf{x},\mathbf{s})dxds\\
=&\sum_{K^a\in\mathcal{T}_h^a}\int_{K^a}\int_K fv,\quad \forall\, v\in V_h
\end{aligned}
\tag{2.4a}
$$

holds for all $K\in\mathcal{T}_h$. Note that (2.4a) can be solved element-wise for each $K\in\mathcal{T}_h$, if given $\widehat{u}_h$ and f as the input data. Namely, by solving (2.4a), we can element-wise express the solution u_h as a function of $\widehat{u}_h$ and f by $u_h=u_h(\widehat{u}_h,f)$. The global solver is nothing but enforcing the following consistency condition:

$$
\sum_{K\in\mathcal{T}_h}\sum_{F\in\mathcal{E}_K}\sum_{K^a\in\mathcal{T}_h^a\cap(K^a\cdot\mathbf{n}_F\geq 0)}\int_F\int_{K^a}\big(\widehat{u}_h-u_h(\widehat{u}_h,f)\big)\eta=0,\quad\forall\,\eta\in\widehat{V}_h^0. \tag{2.4b}
$$

2.2.3. The in2out and in2sol operators

In this part, we show how to derive a discretization for the in2out and in2sol operators by the HDG formulation (2.4).

Let $\varphi_j^{K\times K^a}$ be the basis of $\mathcal{P}_k(K)\otimes P_{k^a}(K^a)$, and $\psi_j^{F\times K^a}$ be the basis of $\mathcal{P}_k(F)\otimes\mathcal{P}_{K^a}(K^a)$. Then, the solution u_h and the hybrid unknown $\widehat{u}_h$ can be represented as the following linear combinations:

$$
\begin{aligned}
u_h&=\sum_{K\in\mathcal{T}_h}\sum_{K^a\in\mathcal{T}_h^a}\sum_j u_j^{K\times K^a}\varphi_j^{K\times K^a},\\
\widehat{u}_h&=\sum_{F\in\mathcal{E}_h}\sum_{K^a\in\mathcal{T}_h^a}\sum_j \widehat{u}_j^{F\times K^a}\psi_j^{F\times K^a}.
\end{aligned}
$$

We can rewrite (2.4a) into a matrix form as follows:

$$
B^K[u_h]+\widehat{B}^K[\widehat{u}_h]-C^K[u_h]+M^K[u_h]-S^K[u_h]=[f], \tag{2.5}
$$

where $[u_h],[\widehat{u}_h]$ and $[f]$ represents the DOFs of $u_h,\widehat{u}_h$ and f, respectively. Namely,

$$
[u_h]_{K,K^a,i}=u_i^{K\times K^a},\quad [\widehat{u}_h]_{F,K^a,i}=\widehat{u}_i^{F\times K^a},\quad [f]_{K,K^a,i}=\int_K\int_{K^a} f\varphi_i^{K\times K^a}.
$$

The matrix terms of (2.5) are assembled as follows (let the test function $v=\varphi_i^{K\times K^a}$):

$$
\big(B^K[u_h]\big)_{K,K^a,i}=\sum_j u_j^{K\times K^a}\left(\sum_{F\in\mathcal{E}_K}\mathbb{1}_{K^a\cdot\mathbf{n}_F\geq 0}\int_{K^a}\int_F(\mathbf{s}\cdot\mathbf{n}_F)\,\varphi_j^{K\times K^a}\varphi_i^{K\times K^a}\right), \tag{2.6a}
$$

$$
\big(\widehat{B}^K[\widehat{u}_h]\big)_{K,K^a,i}=\sum_{F\in\mathcal{E}_K}\sum_j \widehat{u}_j^{F\times K^a}\left(\mathbb{1}_{K^a\cdot\mathbf{n}_F\leq 0}\int_{K^a}\int_F(\mathbf{s}\cdot\mathbf{n}_F)\psi_j^{F\times K^a}\varphi_i^{K\times K^a}\right), \tag{2.6b}
$$

$$
\big(C^K[u_h]\big)_{K,K^a,i}=\sum_j u_j^{K\times K^a}\left(\int_{K^a}\int_K\varphi_j^{K\times K^a}\mathbf{s}\cdot\nabla\varphi_i^{K\times K^a}\right), \tag{2.6c}
$$

$$
\big(M^K[u_h]\big)_{K,K^a,i}=\sum_j u_j^{K\times K^a}\left(\int_{K^a}\int_K\sigma_e\varphi_j^{K\times K^a}\varphi_i^{K\times K^a}\right), \tag{2.6d}
$$

$$\left(S^K[u_h]\right)_{K,K^a,i} = \sum_{K_*^a \in \mathcal{T}_h^a} \sum_j u_j^{K\times K_*^a} \Big(\int_K \sigma_s(\mathbf{x}) \int_{K^a} \int_{K_*^a} p(\mathbf{s},\mathbf{s}') \varphi_j^{K\times K_*^a} \times (\mathbf{x},\mathbf{s}') \varphi_i^{K\times K^a}(\mathbf{x},\mathbf{s}) ds' ds dx \Big). \quad (2.6e)$$

We leave the details on how the above formulations are derived in the Appendix A.

By inverting (2.5), we obtain

$$[u_h] = A_{i2u}^K [\widehat{u}_h] + f_u^K, \quad (2.7)$$

where

$$A_{i2u}^K := -\left(B^K - C^K + M^K - S^K\right)^{-1} \widehat{B}^K, \quad (2.8a)$$

$$f_u^K := \left(B^K - C^K + M^K - S^K\right)^{-1} [f]. \quad (2.8b)$$

In the above formulation, the matrix A_{i2u}^K maps the inflow radiation $\widehat{u}_h$ to the corresponding part of the numerical solution u_h, this represents a discretization of the in2sol operator, and f_u^K is the part of the solution u_h that corresponds to the forcing f. Based on them, we immediately obtain

$$A_{i2o}^K = R_K^o A_{i2u}^K \quad \text{(inflow to outflow radiation)}, \quad (2.9a)$$

$$\widehat{f}_u^K = R_K^o f_u^K \quad \text{(forcing on skeleton)}, \quad (2.9b)$$

where R_K^o denotes a restriction of the DOFs of u_h (on the element K) to the outflow DOFs of $\widehat{u}_h$ on K, see Fig. 2.1 for a visualization of the DOFs on a cubic element. Here A_{i2o}^K represents a discretization of the in2out operator.

By (2.4b) and (2.9), we have

$$\sum_{K\in\mathcal{T}_h} \left(\widehat{R}_K^o [\widehat{u}_h] - A_{i2o}^K [\widehat{u}_h] \right) = \sum_{K\in\mathcal{T}_h} \widehat{f}_u^K. \quad (2.10)$$

Here, the matrix $\widehat{R}_K^o$ restricts the DOFs of $\widehat{u}_h$ to the outflow DOFs of $\widehat{u}_h$ on element K. Note that the system (2.10) only involves the hybrid unknown $\widehat{u}_h$.

Let us conclude this subsection by the Algorithm 2.1.

Algorithm 2.1: HDG Implementation.

1 **Creation of local solver**: For each element $K \in \mathcal{T}_h$, solve the local system (2.8) to obtain A_{i2u}^K (discretization of the Green's function) and f_u^K.

2 Assemble the global system (2.10) using A_{i2o}^K and $\widehat{f}_u^K$.

3 **Global solve**: Solve the global system (2.10) to obtain $\widehat{u}_h$.

4 For each element $K \in \mathcal{T}_h$, recover the solution u_h by using (2.7), which needs the hybrid solution $\widehat{u}_h$ obtained in Step 3, and the matrix A_{i2u}^K and the term f_u^K obtained in Step 1.

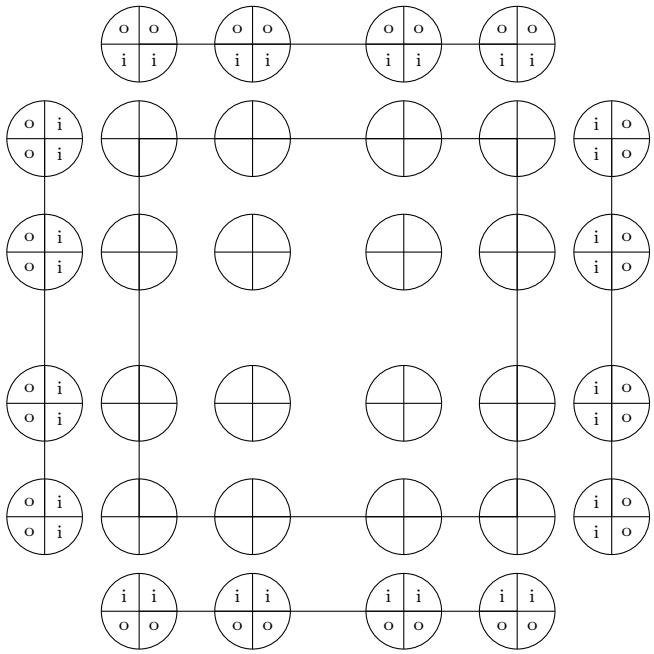

Fig. 2.1. A visual representation of the DOFs of the solution u_h and $\widehat{u}_h$. A single Q_3 spectral element ($p_x = p_y = 3$) is shown with a circle at each of the element's $4 \times 4 = 16$ spatial quadrature points. Each circle also represents the angular coordinate, as 4 uniformly-partitioned P_0 angular elements ($N_a = 4$ and $p_a = 0$) at each spatial quadrature point. On the skeleton along the boundary of the spatial element, the letters "i" and "o" represent the inflow and the outflow DOFs of $\widehat{u}_h$, respectively.

2.3. Solvers for the linear algebraic systems

Many solvers are in use for radiative transfer, such as source iteration, sweeps, diffusion synthetic acceleration, preconditioned GMRES, multigrid, or combinations among them, we refer to [1,65] for a review on this topic and [13,30,79,104] for more. Performance of the solver can vary depending on the problem itself (e.g. strong or weak scattering), the numerical discretizations (e.g. discrete ordinate, spherical harmonic, finite volume, discontinuous Galerkin), serial or parallel implementations, software being used (e.g. Matlab, C, Python, Fortran), and the computer architecture (CPU or GPU, parallelization support, memory speed).

Here, for the DG method, we invert the system (2.2a) using a preconditioned GMRES method, similar to [84, 92], where the preconditioner is obtained by inverting the advection-extinction subsystem with UMFPACK [26]. We have examined several test cases and observed in [33] that this solver scales near optimally as $\mathcal{O}(N^{1.1})$ where N is the total number of degrees of freedom, and it outperforms solver choices such as source iteration or directly applying UMFPACK to the whole system. For the HDG method, note that we have two systems to invert, namely the local solver and the global solver, see Algorithm 2.1. This differs from the DG method, for which we only need to invert the global system (2.2a). The local systems of HDG are inverted by UMFPACK. Then, the global system is inverted by a GMRES method [93]. We remark that GMRES is a Krylov subspace method and supports matrix-free implementation, which would be an interesting future direction of research.

2.4. Neural network

The HDG method introduced in the previous subsection allows a reduction of the DOFs of u_h, which is defined on the whole DG space V_h, to the hybrid unknown $\widehat{u}_h$, which is defined only on the skeleton space $\widehat{V}_h$. In addition, the higher order of the approximation we use, the more reduction that can be achieved, and so one can expect a faster convergence.

However, a significant difficulty that hinders the use of very high order HDG methods is that the local system becomes much more expensive to invert as the polynomial degree increases. To see this, note that the local solver (2.8a) requires one to solve for all possible combinations of the inflow boundary conditions. As a result, as the polynomial degree increases, the dimension of the inflow radiation space (the number of columns of $\widehat{B}^K$ in (2.8a)) increases dramatically. Considering that the system size (the number of rows of $\widehat{B}^K$ in (2.8a)) also increases, it very quickly becomes difficult and potentially infeasible to apply HDG methods as the polynomial degree becomes large. We also refer to [105], where it is shown that a majority of the computational cost comes from inverting the local systems to assemble the global matrix.

To mitigate this issue, in this section, we aim to leverage a neural network to accelerate the local system inversion (2.8). Note that (2.8b) requires the inversion for only one instance $[f]$, so a major computational cost comes from (2.8a), which requires the inversion for multiple instances where each instance is a column vector of $\widehat{B}^K$. So here we focus on using neural network to accelerate the calculation for A^K_{i2u} in (2.8a).

We remark that A^K_{i2u} depends on the parameter (σ, K), and what we aim to approximate is not A^K_{i2u} for some particular (σ, K). Instead, we aim to recover the full mapping $S_{DG} : (\sigma, K) \to A^K_{i2u}$, which takes (σ, K) as input, and returns the corresponding Green's function discretization A^K_{i2u} as output. Since S_{DG} is non-linear, we cannot use linear regression. This justifies the use of neural networks with non-linear activation functions. Numerical evidences will be presented in Section 3.1.

One way is to use a neural network to directly approximate the operator $S_{\text{DG}} : (\sigma, K) \to A^K_{i2u}$ with A^K_{i2u} as its output. However, in many cases, the full solution u_h from A^K_{i2u} is not needed, and only some partial statistics of the solution u_h (e.g. mean intensity, radiative heating rate) are needed. For instance, here we consider the case of recovering the mean intensity

$$u^m_h(\mathbf{x}) = \frac{1}{|S|}\int_S u_h(\mathbf{x}, \mathbf{s})ds. \tag{2.11}$$

In this case, we do not need to calculate the full matrix A^K_{i2u} but only a small fraction of the statistics of this matrix. To be more specific, we only need the following two matrices: (1) A^K_{i2o} (defined in (2.9a)) for the assembling of the global system (2.10), and (2) A^K_{i2m} for recovering the mean intensity of u_h. Here $A^K_{i2m} := R^K_{u2m}A^K_{i2u}$ and R^K_{u2m} is the matrix form of the above angular-averaging operator defined in (2.11). Considering this, here we aim to construct a neural network (NN) to approximate the following operator:

$$\widetilde{S}_{\text{DG}} : (\sigma, K) \;\; \to \;\; \big(A^K_{i2o}, A^K_{i2m}\big), \tag{2.12}$$

where A_{i2o} is for inter-element communication and for assembling the global system (2.10) and A_{i2m} is for local solution recovery for the mean intensity u^m_h.

Note that with this NN, the local solver creation process in Algorithm 2.1 can be replaced by machine learning approaches for acceleration. As a result, for element learning, the only system we need to invert is the global system which shares the same structure as the global system of a standard HDG method.

In the next subsection, we explain how to construct neural networks to the approximation of $\widetilde{S}_{\mathrm{DG}}$ in more details.

2.4.1. Networks architecture

To use a NN to approximate the operator $\widetilde{S}_{\mathrm{DG}}$ in (2.12), we begin by considering the parameterization of the optical coefficient σ and the element geometry K. For the conciseness of the presentation and the convenience of the implementation, we consider the case when $d = 2$ and K is a rectangular element. The generalization to $d = 3$ and other types of element (e.g. triangular elements) follow the same ideas. We consider the case when the angular space S is a unit circle, so the radiation direction $\mathbf{s}$ can be characterized by a single parameter, namely the angle $\theta \in [0, 2\pi]$, by $\mathbf{s} = (\cos\theta, \sin\theta)$. Also, we consider a fixed angular discretization of $[0, 2\pi]$ by a uniform partition $\mathcal{T}_h^a$ with N_a elements and polynomial degree p_a.

To consider the parameterization of element geometry K, let $\widehat{K} = [-1, 1]^2$ be the reference element, in which case the geometry of K is determined by the following push-forward map:

$$\begin{aligned} F_K : \widehat{K} &\to K, \\ (\hat{x}, \hat{y}) &\mapsto \left(a_{11}^K \hat{x} + a_{12}^K \hat{y}, a_{21}^K \hat{x} + a_{22}^K \hat{y}\right) + \left(b_1^K, b_2^K\right). \end{aligned}$$

Namely, the geometry of K is parameterized by the parameters a_{ij}^K and b_i^K with $i, j = 1, 2$. One could then proceed by including parameters a_{ij}^K and b_i^K into the input layer of the NN. However, in many cases, it is straightforward to derive the change of the solution under a push-forward map. So in these cases, we do not need to insert a_{ij}^K and b_i^K to the input layer of the neural network. For instance, if K is a square element, then the push forward map F_K becomes

$$F_K(\hat{x}, \hat{y}) = h(\hat{x}, \hat{y}) + \left(b_1^K, b_2^K\right),$$

where h represents the size of the element K. In this case, it is straightforward to derive that solving (2.1) on K is equivalent to solving the same equation on $\widehat{K}$ with re-scaled coefficients $h\sigma_e/2$, $h\sigma_s/2$, and forcing term $hf/2$. Thus, it suffices to consider the parameterization on the reference element $\widehat{K}$. For the following discussion, we focus on this case for simplicity.

We remark that, if one were to use a mesh containing multiple element types (e.g. triangles, quadrilateral), two approaches could be considered. For the first approach, one could use multiple neural networks so that each network learns the operator (2.12) for each element type. For the second approach, one could use a neural network to learn the operator (2.12) on a general polyhedral element, in which case the simple-geometry element can be regarded as a special case of the complex-geometry element, see, for instance, [28, 32].

To parameterize σ, we consider a nodal representation by spectral element

$$\sigma(x, y) \approx \sigma_h(x, y) = \sum_{i=1}^{p_x+1} \sum_{j=1}^{p_y+1} \sigma_{ij}\, \varphi_i(x)\varphi_j(y), \tag{2.13}$$

where φ_i is the Lagrange polynomial associated with the i-th Legendre-Gauss-Lobatto (LGL) quadrature point on $[-1, 1]$. Note that we need to parameterize both the extinction σ_e and the scattering σ_s. As a result, the input layer of the network is a vector

$$\mathbf{x}_0 := ([\sigma_e], [\sigma_s]) \in \mathbb{R}^{N_{\mathrm{in}}}, \tag{2.14}$$

where $N_{\mathrm{in}} = 2(p_x+1)(p_y+1)$, and $[\sigma_e]$ and $[\sigma_s]$ are the two row vectors with their entries as the nodal representation of σ_e and σ_s, respectively. For applications when σ_e and σ_s are correlated

by $\sigma_s = \widetilde{\omega}\sigma_e$ where $\widetilde{\omega}$ is a given constant (e.g. short-wave radiation by cloud), we only need to parameterize one of these two coefficients so the input size becomes $N_{\text{in}} = (p_x+1)(p_y+1)$.

We next consider the output layer. Note the matrix A^K_{i2o} has the size of $2(p_x+p_y+2)(N_a/2)$ $\times(p_a+1)\times 2(p_x+p_y+2)(N_a/2)(p_a+1)$, and the matrix A^K_{i2m} has the size $(p_x+1)(p_y+1)\times$ $2(p_x+p_y+2)(N_a/2)(p_a+1)$. As a result, the output layer is a vector

$$\mathbf{x}_{N_l} := \big([A^K_{i2o}], [A^K_{i2m}]\big) \in \mathbb{R}^{N_{\text{out}}}, \tag{2.15}$$

where

$$N_{\text{out}} = (p_x+p_y+2)N_a(p_a+1)\big((p_x+p_y+2)N_a(p_a+1)+(p_x+1)(p_y+1)\big),$$

and $[A^K_{i2o}]$ and $[A^K_{i2m}]$ are the two row vectors collecting the entries of the matrices A^K_{i2o} and A^K_{i2m}, respectively.

Finally, we construct a fully connected neural network from $\mathbf{x}_0$ to $\mathbf{x}_{N_l}$. Namely,

$$\mathbf{x}_i = f_i(A_i\mathbf{x}_{i-1}+\mathbf{b}_i), \quad i=1,2,\dots,N_l,$$

where A_i is a linear transformation, $\mathbf{b}_i$ is a vector sharing the same size of $\mathbf{x}_i$, and f_i is a nonlinear activation function for $i=1,2,\dots,N_l-1$ and $f_{N_l}(\mathbf{x})=\mathbf{x}$. We can also write the neural network in a compact form as follows:

$$\mathbf{x}_{N_l} = \text{NN}_{\boldsymbol{\theta}}(\mathbf{x}_0), \quad \boldsymbol{\theta} = (\boldsymbol{\theta}_1,\dots,\boldsymbol{\theta}_{N_l}) = (A_1,\mathbf{b}_1,\dots,A_{N_l},\mathbf{b}_{N_l}).$$

2.4.2. Data generation

Now we consider the data generation that will be used to train the neural network $\text{NN}_{\boldsymbol{\theta}}$. We remark that only the data generation on the reference element $\widehat{K}$ is considered here, since the operator on the physical element K (2.12) can be derived based on the operator on the reference element $\widehat{K}$, see the second paragraph of Section 2.4.1 for more details. For simplicity, we shall consider the case with a fixed single-scattering albedo $\tilde{\omega}$, which relates extinction and scattering by $\sigma_s = \tilde{\omega}\sigma_e$. In this case, it suffices to consider the data generation for only one optical coefficient instead of two, and we will use simply σ_h to represent this coefficient.

We shall generate σ_h according to a probability distribution μ_σ. Then, the corresponding matrices $A^{\widehat{K}}_{i2o}(\sigma_h)$ and $A^{\widehat{K}}_{i2m}(\sigma_h)$ will be calculated by using σ_h and solving the local system on the reference element $\widehat{K}$, see Section 2.2.3 for more details. We next explain how we define the probability distribution μ_σ.

In (2.13), we define σ_{ij} to be the nodal representation of σ_h by spectral element. We can also express σ_h in terms of modal representation

$$\sigma_h(x,y) = \sum_{m=0}^{p_x}\sum_{n=0}^{p_y}\widetilde{\sigma}_{mn}L_m(x)L_n(y),$$

where L_m is the m-th Legendre polynomial on $[-1,1]$. Note that L_m with a higher index m represents higher frequency signals. Using this fact, we shall generate the probability distribution μ_σ as described in Algorithm 2.2.

We remark that in the above data generation process, the only undetermined parameters are c_{sm} and A_σ, which determine the overall smoothness and amplitude of the optical coefficient σ_h which are sampled from the distribution μ_σ, respectively.

Algorithm 2.2: Optical Coefficient Data Generation.

1 Draw samples of the modal representation $\widetilde{\sigma}_{mn}$ with low-frequency bias

$$\widetilde{\sigma}_{mn} = \exp\left(-c_{\text{sm}}\left(\left(\frac{m}{p_x}\right)^2 - \left(\frac{n}{p_y}\right)^2\right)\right)\left(X_{mn} - \frac{1}{2}\right),$$

where X_{mn} are i.i.d. uniform distribution on $[0,1]$, and c_{sm} determines the decaying speed of the high-frequency signals in the distribution σ_h so a larger c_{sm} is likely to generate a smoother σ_h. This procedure is guided by the consideration that low-frequency bias exists universally in the physical world.

2 Perform transformation from modal to nodal representations

$$\sigma_{ij} = T_{\text{modal}\to\text{nodal}}(\widetilde{\sigma}_{mn}).$$

3 Post-processing of σ_{ij}:

1. Guarantee positivity: $\sigma_{ij} = \sigma_{ij} - \min_{ij}\sigma_{ij}$. This is because extinction and scattering coefficients are non-negative.
2. Normalization: $\sigma_{ij} = \sigma_{ij}/\max_{ij}\sigma_{ij}$.
3. Amplification: $\sigma_{ij} = X_{A_\sigma}\sigma_{ij}$, where X_{A_σ} is a uniform distribution on $[0, A_\sigma]$ and A_σ is a given constant.

3. Numerical Results

In this section, we conduct numerical experiments to test the performance of our proposed element learning methods. In the first subsection, we show how we perform the training for the neural network for element learning. Then in the following subsections, we test the performance of the trained element on a variety of numerical experiments.

3.1. Element training

As is discussed in Section 2.4.1, it suffices to consider the training on the reference element $\widehat{K} = [-1,1]^2$. We choose the (spatial) polynomial degrees $p_x = p_y = 6$, namely a Q_6 element for the spatial discretization. We use a fixed angular discretization of the domain $[0, 2\pi]$ into $N_a = 28$ uniform partitions with piece-wise constant approximation ($p_a = 0$), namely a finite volume angular discretization.

For the data generation (see Section 2.4.2 for details), we choose the single scattering albedo $\tilde{\omega} = 1$ and the asymmetric parameter $g_{\text{asym}} = 0.8$. These are standard values used for short-wave radiation of water clouds [98]. Following the procedures introduced in Section 2.4.2, we generate $N_{\text{samp}} = 1000$ samples of the scattering coefficient σ_s with the amplitude parameter $A_\sigma = 10$ and the smoothness parameter $c_{\text{sm}} = 2$. Among these samples, 800 of them constitute the training set and the remaining 200 samples constitute the testing set.

For the neural network structure, we construct a fully connected neural network and consider four cases for the number of layers: $N_{\text{layer}} = 1, 2, 3, 4$. The size of the hidden layer is chosen to be twice of the input size, namely $N_{\text{hidden}} = 2N_{\text{in}}$. We use the exponential linear unit (ELU) activation function for the hidden layers and linear activation for the output layer.

The training is done on the PyTorch platform. We use the mean absolute error (of the neural network output, i.e. the A^K_{i2o} and A^K_{i2m} matrices from (2.12)) as the loss function and the Adam optimization that is innately built into the platform. For many of these setup details for the neural network and training, we are following similar approaches as in other works such as [60, 72]. We choose the batch size to be 50. For the training, we choose the learning rate 10^{-3} for the first 3000 epochs. We then decrease the learning rate to 10^{-4} and 10^{-5} for the following 3000 epochs and the last 3000 epochs, respectively. After the training, the NN was evaluated on Matlab by transforming into the open neural network exchange (ONNX) format.

The NN was trained on a single GTX30 series GPU within 20 minutes. Note that the training is required only for one time on the reference element, then this NN can be used for all the subsequent calculation without the need to retrain. We also emphasize that it is only the training of the NN that is done on a GPU. Once the training is finished and the NN is transformed into an ONNX format file, only the CPU is used for the evaluation of this NN. This is to ensure that both the element learning and the traditional numerical solvers are implemented on a CPU so that a fair comparison can be made.

In Fig. 3.1, we observe that the randomly generated scattering coefficient σ_s presents a certain level of smoothness thanks to the low-frequency bias we introduced in the generation procedures, see Algorithm 2.2. For the approximation errors of the neural networks, we observe that the error decreases as we increase the depth of the networks. This decreasing of the error is significant when it goes from linear regression ($N_{\text{layer}} = 1$) to the case with one hidden layer ($N_{\text{layer}} = 2$). The decreasing of the error becomes marginal from $N_{\text{layer}} = 3$ to $N_{\text{layer}} = 4$.

The numerical results suggest the necessity of using neural networks with hidden layers and non-linear activation function. Also, the results show that a shallow network ($N_{\text{layer}} = 3$ or 4) can already provide a good approximation, and in the present setup it is unlikely that any significant improvement of the approximation can be achieved by using a deeper network.

To visualize more clearly the necessity of using neural networks with hidden layers and non-linear activation, we compare the solution (mean intensity) obtained by the 4-layer network and

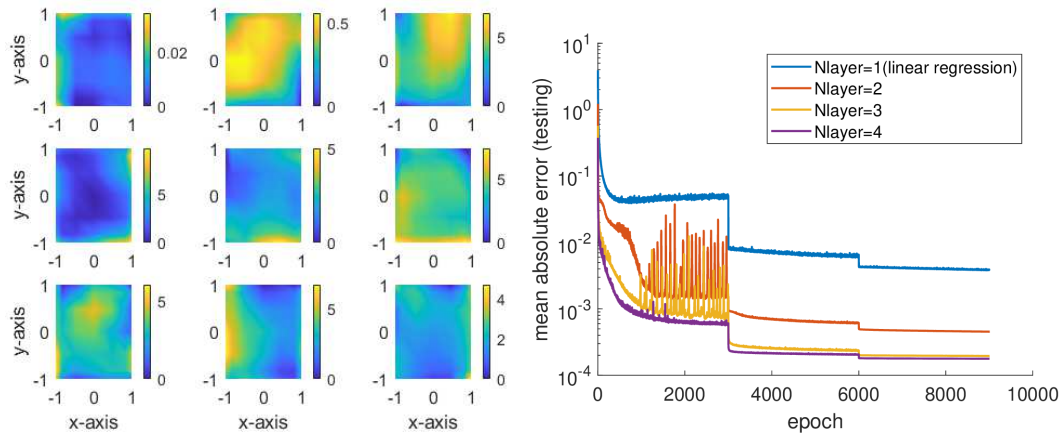

Fig. 3.1. Training the neural network. Left: 9 samples of the scattering coefficient σ_s generated from the Algorithm 2.2. Right: testing error for the networks with different number of layers. The testing error is lowest (and similar) for 3 or 4 layers. The neural network is able to learn the nonlinear operator from (2.12) with an error that is nearly as small as 10^{-4}.

the solution obtained by linear regression on one element. The results are collected in Fig. 3.2, where it is clearly shown that the neural network can provide a good approximation while the linear regression fails to produce a valid solution.

In addition to the depth of the NN, we also test how the width N_{hidden}, the number of samples N_{samp}, and the activation function would affect the error of the NN approximations. We also present the error with different polynomial degrees of the Q_4 and Q_8 elements. These results are collected in Fig. 3.3.

In the top-left panel of Fig. 3.3, we observe that by increasing the hidden layer width from N_{in} to $2N_{\text{in}}$, there is an obvious improvement in decreasing the error, whereas the improvement from $2N_{\text{in}}$ to $3N_{\text{in}}$ is small. A similar pattern appears in the test with different sample numbers (top-right panel): while there is a small improvement from $N_{\text{samp}} = 500$ to $N_{\text{samp}} = 2000$, the improvement from $N_{\text{samp}} = 1000$ to $N_{\text{samp}} = 2000$ is hard to recognize. This seems to suggest that 1000 samples is enough in the current setting of training an element.

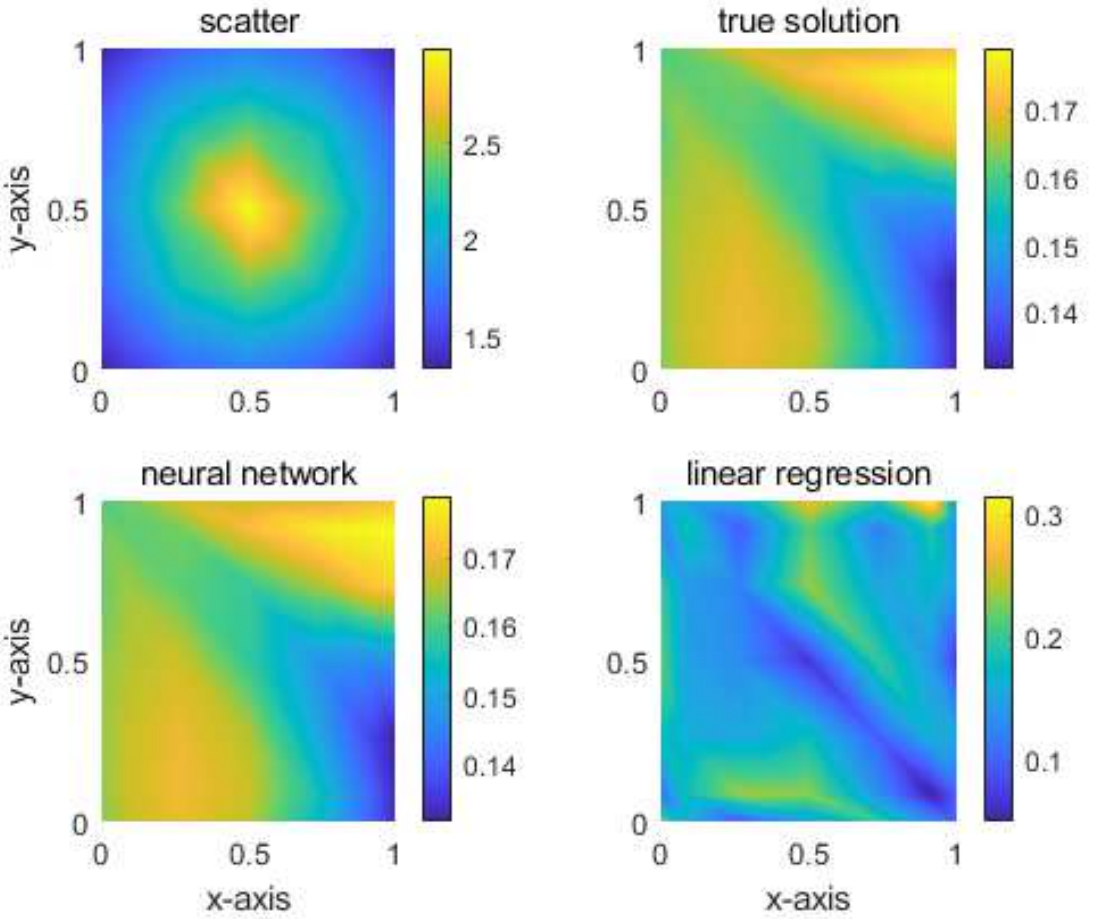

Fig. 3.2. A comparison between the radiative transfer solution (mean intensity) obtained based on a neural network ($N_{\text{layer}} = 4$) versus based on linear regression ($N_{\text{layer}} = 1$). In this example, the inflow radiation comes from the top and left boundaries and the forcing is zero. The scattering coefficient is shown in the top-left sub-figure. The neural network provides a good approximation to the "true solution" (which is obtained by inverting the HDG matrix system) while the linear regression fails to provide a satisfactory approximation.

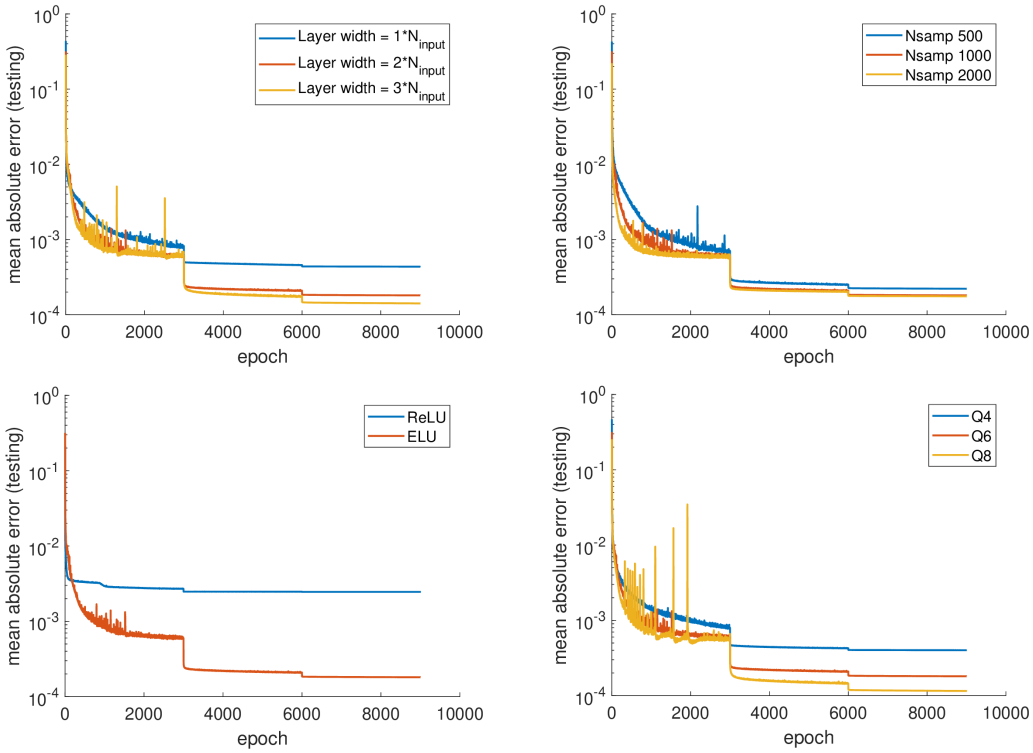

Fig. 3.3. Testing error for the networks with different hidden layer width (top-left), number of samples (top-right), activation function (bottom-left), and polynomial degree (bottom-right). These four tests are modified based on the baseline test of Fig. 3.1 which has the settings of (1) Q_6 element, (2) number of layers $N_{\text{layer}} = 4$, (3) hidden layer width $N_{\text{hidden}} = 2N_{\text{in}}$, (4) number of samples $N_{\text{samp}} = 1000$, and (5) the ELU activation function. Each test only changes one factor of the setting and keeps the remaining four factors unchanged.

We observe a significant improvement (more than 10 times error reduction) by using the ELU activation function compared to the popular ReLU activation function (bottom-left panel). In the bottom-right panel, we observe that the error decreases as we increase the polynomial degree. This is likely because the network width N_{hidden} increases as the polynomial degree increases, since $N_{\text{hidden}} = 2N_{\text{in}}$ and $N_{\text{in}} = (p_x + 1)(p_y + 1)$.

For the rest of the numerical tests, we shall use the trained neural network with the following setting: number of layers $N_{\text{layer}} = 4$, hidden layer width $N_{\text{hidden}} = 2N_{\text{in}}$, number of samples $N_{\text{samp}} = 1000$, and the ELU activation.

3.2. Testing the performance of element learning

Here we aim to test the performance of element learning method by comparing it with other classical numerical methods. Here we consider three methods: (1) DG – the discontinuous Galerkin method, (2) HDG – the hybridizable discontinuous Galerkin method, and (3) HDG-EL – the HDG method accelerated by element learning.

3.2.1. Test 1 – idealized clouds

In this first test, we consider the case in which the radiation is scattered by idealized clouds. We consider a rectangular domain $\Omega = [0, L_x] \times [0, L_y]$ with $L_x = 3$ and $L_y = 2$, and two round-shaped scatterers located at the center of the domain as idealized clouds, see Fig. 3.4. We choose the single scattering albedo of the scatters to be $\tilde{\omega} = 1$ and the asymmetric parameter $g_{\text{asym}} = 0.8$, to be consistent with the data samples we use for the element training, see Section 3.1.

For the boundary condition, we consider the inflow radiation g to be a collimated beam coming from the top and the left sides of the domain Ω, to represent solar radiation propagating towards the bottom-right direction

$$g_{\text{left, top}}(\theta) = \frac{28}{2\pi}\chi_{(2\pi\frac{22}{28}, 2\pi\frac{23}{28})}(\theta), \quad g_{\text{right, bottom}}(\theta) = 0,$$

where χ is the characteristic function. The forcing term f is chosen to be 0.

To compare, we use the same discretization setting for the DG, HDG, and HDG-EL methods. For each element $K \in \mathcal{T}_h$, we choose $p_x = p_y = 6, N_a = 28$, and $p_a = 0$ to be consistent with the setting we use for the element training, see Section 3.1. For a sequence of tests at different levels of refinement, we shall start with an initial mesh discretized by a uniform 6×4 partitioning (refinement level $l = 0$ with total 24 elements). Then, for each following refinement (refinement level $l = 1, 2, 3, 4$), we discretize the domain by a uniform $3(l + 2) \times 2(l + 2)$ partitioning.

For the iterative solvers, we use a stopping criterion for all the GMRES methods of 10^{-4} in the relative l^2 error. Refer to Section 2.3 for the details of the solvers. We choose this weak stopping criterion for a fair comparison between the classical approaches (DG and HDG) and the machine learning accelerated approach (HDG-EL), since the neural network's testing error is no smaller than 10^{-4} (see Fig. 3.1). All computation will be carried out in Matlab-R2023a with an Apple M1 CPU using a single thread. Despite this, we remark that for HDG and HDG-EL, their local solvers could be easily parallelized with Matlab's parfor command, and a faster convergence can be expected under parallelization. Here we deliberately do not to use any parallelization for a fair comparison between DG and HDG/HDG-EL.

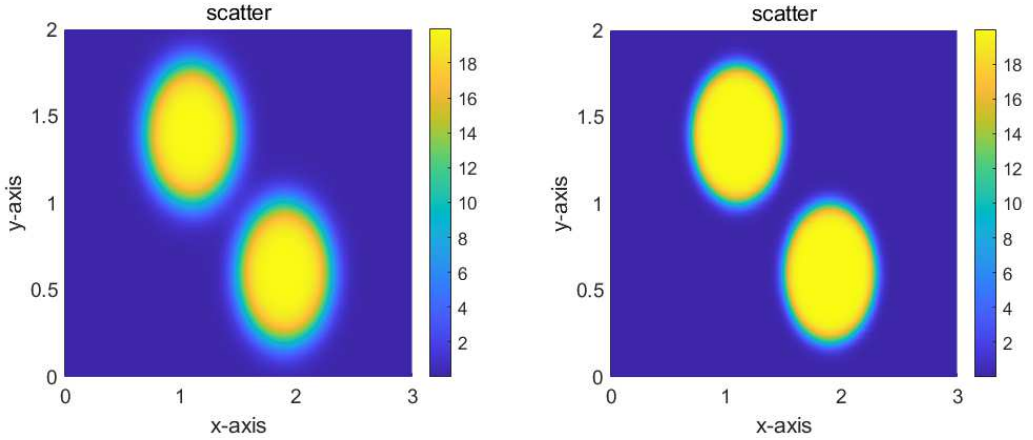

Fig. 3.4. The distribution of the scattering coefficient σ_s for two different test cases. Case 2 (right) has a sharper boundary transition at cloud edge compared to case 1 (left).

To estimate the error, we calculate a numerical solution by the DG method on an overrefined mesh with the refinement level $l = 10$, namely, on a 36×24 partition. We also increase the stopping criterion of the GMRES to 10^{-8} to match the high precision of this overrefined solution. We then compare the solution (mean intensity) obtained by DG, HDG, and HDG-EL methods against this overrefined solution to estimate the numerical error.

In Fig. 3.5, we plot the solution and the error obtained by HDG-EL, HDG, and DG method for the test case 1. We observe that HDG-EL provides a good approximation which reduces the error level to 10^{-3}, but not as small as DG and HDG, which reduces the error level to 10^{-4}. Fig. 3.6 shows the solution and the error for the test case 2, in which the scatterer has a sharper transition layer. We observe that the approximation by DG and HDG deteriorate to the error level 10^{-3}, but the HDG-EL can still provide a similar good approximation to the error level 10^{-3}.

For the following discussion, we present the solver time spent for DG, HDG, and HDG-EL methods. For DG, it is the time spent to invert the system (2.2a). For HDG-local, it is the time spent to invert all the HDG local systems (2.8a), while for HDG-EL-local, it is the time spent to infer the in2out and in2sol operators (see (2.12)) for all elements K by using the network. Finally, for HDG-global and HDG-EL-global, it is the time used to invert the global system (2.10).

In these comparisons, we do not account for the time required to assemble matrices. One reason is that the time it takes to assemble matrices can vary widely depending on a multitude of factors, which can complicate a comparison. For instance, the various influential factors include the software being used (e.g. Matlab based or Fortran based codes) and/or the hardware structure (stored in RAM or VRAM). In addition, we remark that there are matrix-free approaches available for both DG and HDG methods [62, 63], which could be considered in future work.

We show in Tables 3.1 (test case 1) and 3.2 (test case 2) the solver time spent for DG, HDG, and HDG-EL methods. Similar results are observed in both test cases. In the last refinement level, we observe that the local solvers for the HDG method take around 36 seconds, while the global solver only takes around 0.9 second. For the HDG-EL method, the global solver takes a time similar to the HDG method. However, for the local solver, we observe a significant decreasing of the time by using element learning, which reduces the time from 36 to only 0.7 seconds, which is around 50 times faster. The DG method only has the global system to invert, which takes around 26 seconds, which is faster than HDG but much slower than HDG-EL.

To better see the relations between the DOFs, the error, and the solver time for the three methods, we plot Figs. 3.7 and 3.8 for case 1 and case 2, respectively. By the left sub-figures

Table 3.1: Time (second) spent on DG solver, HDG local solver, HDG global solver, HDG-EL local solver, and HDG-EL global solver, for test case 1.

DOFs	DG	HDG-local	HDG-global	HDG-EL-local	HDG-EL-global
32928	2.24	4.27	0.05	0.09	0.04
74088	5.36	9.58	0.13	0.19	0.13
131712	11.21	16.45	0.35	0.31	0.34
205800	19.47	25.33	0.55	0.46	0.57
296352	25.70	36.28	0.88	0.69	0.87

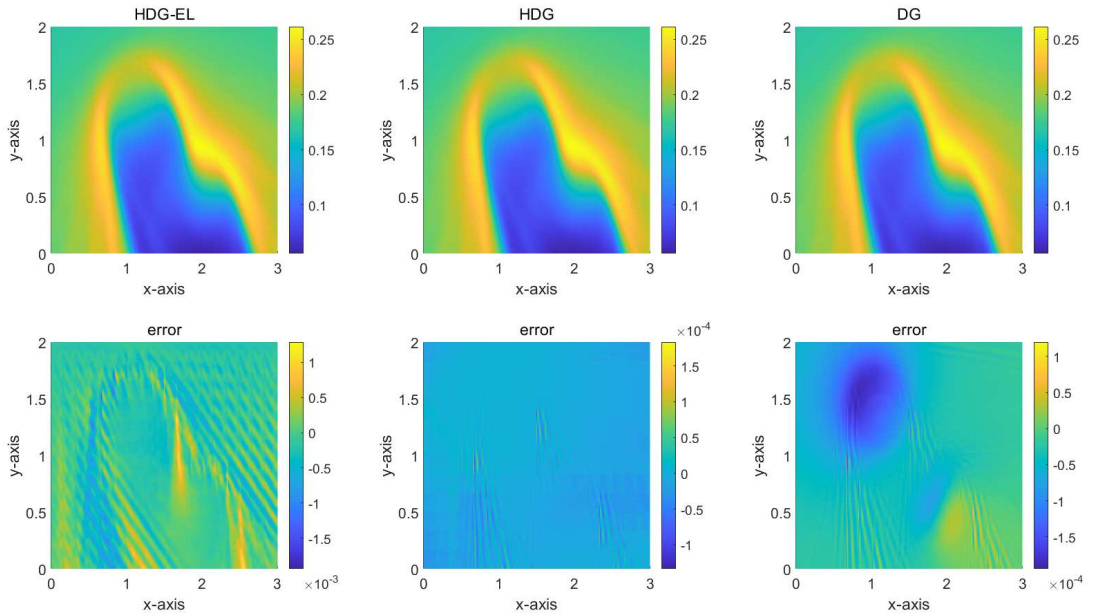

Fig. 3.5. Numerical solution (mean intensity) and error landscape for the test case 1 (left figure of Fig. 3.4), with smoother transition at cloud boundary. Top row: Numerical solution obtained by HDG-EL, HDG, and DG methods in the refinement level $l = 4$. Bottom row: Error landscape of the corresponding methods.

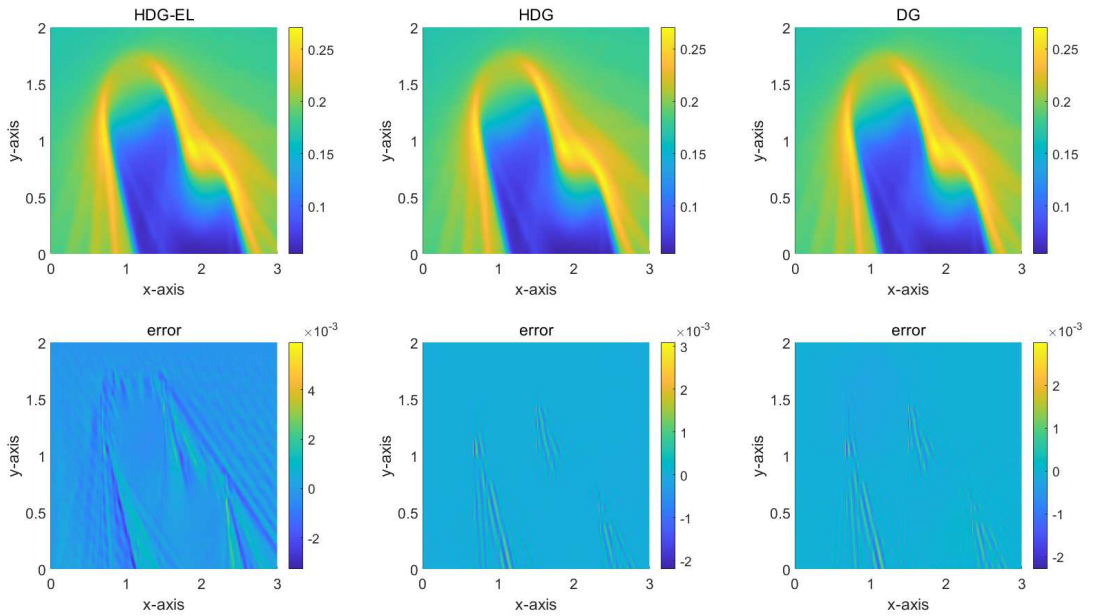

Fig. 3.6. Numerical solution and error landscape for the test case 2 (right figure of Fig. 3.4), with steeper gradient at cloud boundary. Top row: Numerical solution (mean intensity) obtained by HDG-EL, HDG, and DG methods in the refinement level $l = 4$. Bottom row: Error landscape of the corresponding methods. All methods achieve a similar error level of 10^{-3}.

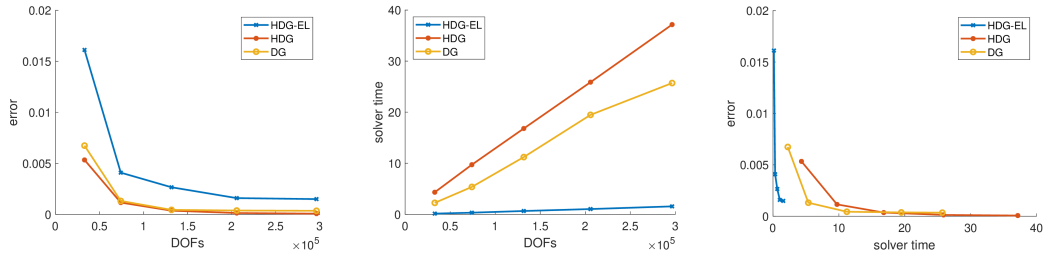

Fig. 3.7. Comparison of the different methods in terms of cost and accuracy, for test case 1. Left: DOFs vs relative L^2 error. Middle: DOFs vs solver time. Right: solver time vs relative L^2 error. The solver time for HDG and HDG-EL is the summation of the local solver and the global solver time. For an error level of approximately 2×10^{-3}, in the right panel, the HDG-EL method is around 5 to 10 times faster than the DG and HDG methods.

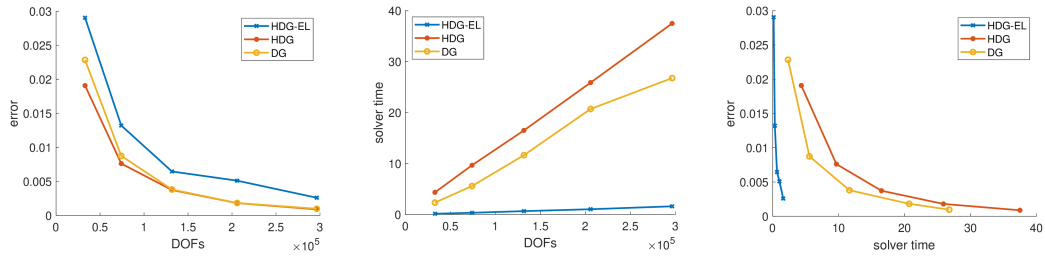

Fig. 3.8. Comparison of the different methods in terms of cost and accuracy, for test case 2. Left: DOFs vs relative L^2 error. Middle: DOFs vs solver time. Right: solver time vs relative L^2 error. The solver time for HDG and HDG-EL is the summation of the local solver and the global solver time. For an error level of approximately 2×10^{-3}, in the right panel, the HDG-EL method is around 10 times faster than the HDG and DG methods.

of Figs. 3.7 and 3.8, we observe that all the three methods can reduce the relative L^2 error to a satisfactory level of 10^{-3}. In addition, we observe that HDG and DG are more efficient in reducing the error (with respect to the DOFs) compared to the HDG-EL method. This is expected since HDG-EL is an approximation to the HDG method by using neural networks. We also observe that, for the test case 1 (which has a smoother transition layer for the scatterer, see Fig. 3.4), the errors are smaller for all the three methods, compared to the test case 2, which has a steeper transition layer for the scatterer.

In the middle figures of Figs. 3.7 and 3.8, we plot how the solver time changes according to the increase of the DOFs. For both test case 1 and test case 2, we observe that HDG-EL is significantly faster than HDG and DG methods, while the DG method is faster than the HDG method.

In the right figures of Figs. 3.7 and 3.8, we plot how the error changes according to the solver time. We observe that the HDG-EL methods are much faster than HDG and DG methods with a fixed error level. In the error level around 2×10^{-3}, we observe the HDG-EL method is about 5 to 10 times faster than the DG and HDG methods (around 5 to 10 times faster for test case 1, and around 10 times faster for test case 2). It is interesting to point out that despite the overall faster convergence of the HDG-EL method compared to HDG/DG in both the test cases, the speed-up of the HDG-EL method is more obvious in a more difficult test case (test case 2 with sharper transition at scatterer boundary). This seems to suggest the NN by element learning generalizes well (to relatively steep gradients in the PDE parameters and solutions).

We next show the numerical tests for Q_4 and Q_8 elements, to be compared with the results of the Q_6 element in Table 3.2 and Fig. 3.8. All the test settings will be kept the same except that the polynomial degree will be changed.

Table 3.3 and Fig. 3.9 show the test results for Q_4 element, while Table 3.4 and Fig. 3.10 show the results for Q_8 element. By comparing the left panel of the Figs. 3.9 and 3.10, we observe that HDG-EL is similar to HDG and DG in accuracy with fixed DOFs for Q_4 element. However, HDG-EL is much less accurate than DG and HDG with fixed DOFs for Q_8 element. An explanation for this observation is that the traditional high-order finite element solvers become highly accurate with only a few element. However, the element learning approach introduces machine learning error (or neural network approximation error) which does not decrease as fast as the high-order finite element methods.

When we compare the solver time while keeping the DOFs fixed (middle panel of Figs. 3.9 and 3.10), we observe that the speed-up from element learning is more significant in the case of the higher order Q_8 methods, compared to the lower order Q_4 methods. This is most obvious by comparing the local solver time of HDG and HDG-EL, see Tables 3.3 and 3.4. While the speed-up varies from 7 to 9 for Q_4 element, the speed-up for Q_8 element varies from 25 to 65.

Table 3.2: Time (second) spent on DG solver, HDG local solver, HDG global solver, HDG-EL local solver, and HDG-EL global solver, for test case 2.

DOFs	DG	HDG-local	HDG-global	HDG-EL-local	HDG-EL-global
32928	2.32	4.29	0.04	0.09	0.04
74088	5.57	9.52	0.14	0.19	0.13
131712	11.63	16.15	0.35	0.31	0.34
205800	20.71	25.33	0.56	0.48	0.54
296352	26.77	36.60	0.90	0.70	0.89

Table 3.3: Time (second) spent on DG solver, HDG local solver, HDG global solver, HDG-EL local solver, and HDG-EL global solver, for test case 2, for Q_4 element.

DOFs	DG	HDG-local	HDG-global	HDG-EL-local	HDG-EL-global
16800.00	0.83	0.81	0.04	0.07	0.03
37800.00	2.10	1.76	0.08	0.19	0.08
67200.00	3.86	3.04	0.20	0.24	0.19
105000.00	6.06	5.06	0.31	0.39	0.31
151200.00	9.03	7.16	0.46	0.55	0.48

Table 3.4: Time (second) spent on DG solver, HDG local solver, HDG global solver, HDG-EL local solver, and HDG-EL global solver, for test case 2, for Q_8 element.

DOFs	DG	HDG-local	HDG-global	HDG-EL-local	HDG-EL-global
54432.00	5.13	18.95	0.07	0.13	0.06
122472.00	15.08	40.85	0.20	0.26	0.21
217728.00	27.61	71.93	0.53	0.45	0.53
340200.00	43.06	110.43	0.90	0.84	0.89
489888.00	64.45	159.01	1.41	0.98	1.40

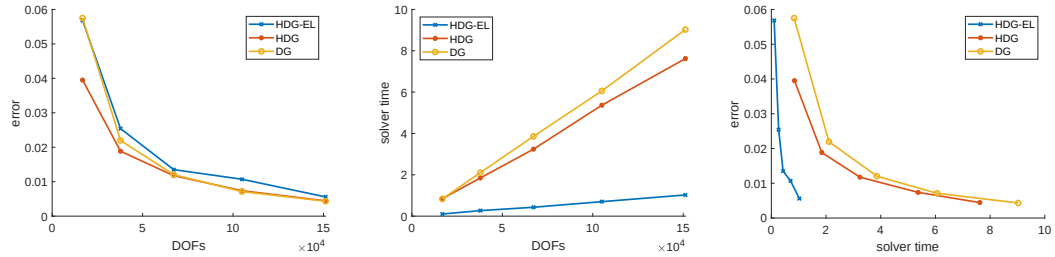

Fig. 3.9. Same test setting of Fig. 3.8 but with Q_4 element.

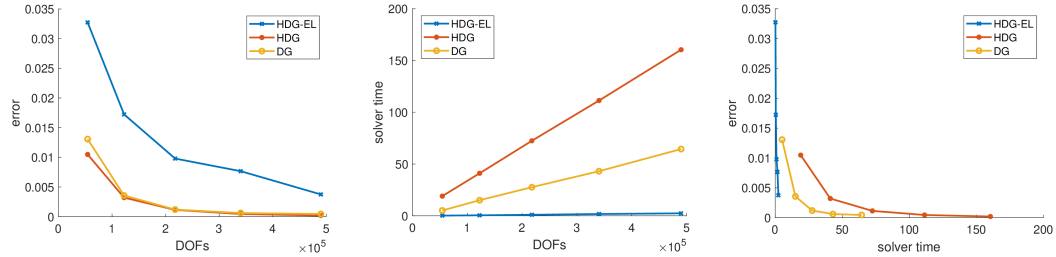

Fig. 3.10. Same test setting of Fig. 3.8 but with Q_8 element.

Finally, for the solver time versus the error (right panel of Figs. 3.9 and 3.10), we again observe an approximately 5 to 10 times speed-up similar to the case of Q_6. Also note that the error in element learning does not reach below $\mathcal{O}(10^{-3})$, due to the machine learning component of element learning. For even higher accuracy with errors below $\mathcal{O}(10^{-3})$, the corresponding machine learning part needs to provide a more precise approximation. The present results here are able to achieve low errors of $\mathcal{O}(10^{-1})$ to $\mathcal{O}(10^{-3})$ which are sufficiently accurate for many applications.

3.2.2. Test 2 – I3RC cloud fields

Here we consider a more realistic test case, for which the cloud field is retrieved from the International Comparison of 3-Dimensional Radiative Transfer Codes (I3RC) case 2 [12], see Fig. 3.11 for a visualization of the cloud field. We shall use the same experiment setting of the tests in Section 3.2.1, unless otherwise specified.

For the boundary condition, we take the inflow radiation g as follows:

$$g_{\text{left, top}}(\theta) = \frac{28}{2\pi}\chi_{(2\pi\frac{24}{28},2\pi\frac{25}{28})}(\theta), \quad g_{\text{right, bottom}}(\theta) = 0,$$

which mimics the solar radiation coming from the top and the left boundary and going towards the bottom-right direction.

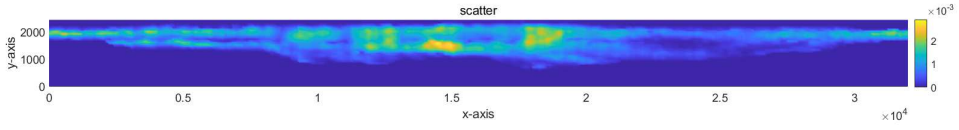

Fig. 3.11. Cloud field from I3RC case 2.

To compare, we use the same discretization setting for DG, HDG, and HDG-EL methods. We shall start with an initial mesh discretized by a uniform 26×2 partitioning (refinement level $l = 0$ with total 52 elements). Then, for each refinement (refinement level $l = 1, 2, 3, 4$), we discretize the domain by a uniform $13(l+2) \times 1(l+2)$ partitioning. To estimate the error, we calculate a numerical solution by the DG method on an overrefined mesh with the refinement level $l = 10$, namely, on a 156×12 mesh. We then compare the solution (mean intensity) obtained by HDG-EL, HDG, and DG methods against this over-refined solution to estimate the numerical error.

In Fig. 3.12, we plot the solution and the error obtained by the HDG-EL, HDG, and DG methods. We observe that HDG-EL provides a good approximation to the overrefined solution. The HDG-EL method reduces the error level to 2×10^{-3}, which is very close to HDG and is even slightly better than DG.

We show the solver time for the three methods in Table 3.5. We again observe that HDG-EL uses much less time (around 20 times faster) for the local solver compared to the HDG method. We observe slightly slower global solver with HDG-EL compared to HDG method. Again, we observe that the DG method is faster than HDG but much slower than HDG-EL.

We again show the relations between the DOFs, the error, and the solver time in Fig. 3.13. In the left sub-figure Fig. 3.13, we observe that all three methods behave similarly in reducing the error with respect to the DOFs. Unlike the test cases in Section 3.2.1, here we observe a similar performance of HDG-EL compared to DG and HDG.

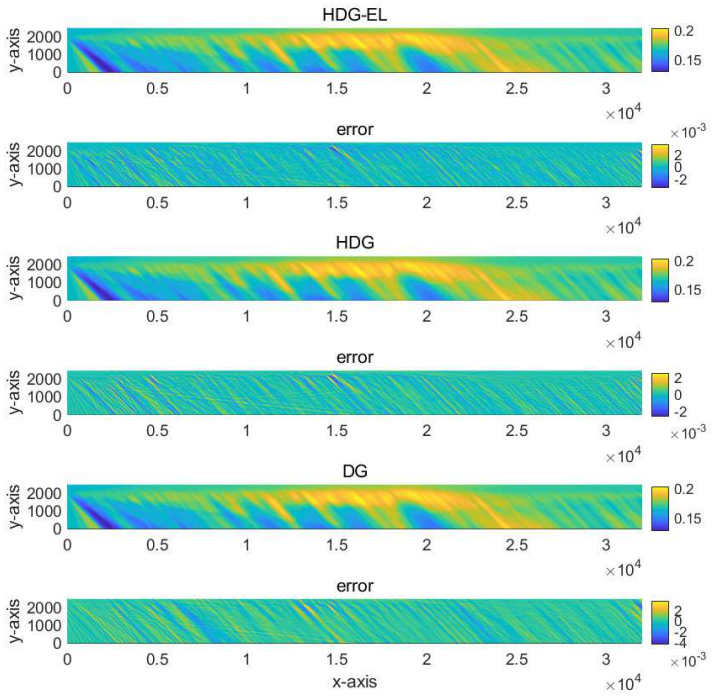

Fig. 3.12. Numerical solution and error landscape for I3RC test case (see Fig. 3.11). Odd row: numerical solution (mean intensity) obtained by HDG-EL, HDG, and DG methods in the refinement level $l = 4$. Even row: error landscape of the corresponding methods. All methods achieve a similar error level of approximately 2×10^{-3}.

Table 3.5: Time (second) spent on DG solver, HDG local solver, HDG global solver, HDG-EL local solver, HDG-EL global solver, for the I3RC test case.

DOFs	DG	HDG-local	HDG-global	HDG-EL-local	HDG-EL-global
71344	1.62	2.74	0.12	0.19	0.12
160524	5.18	6.64	0.33	0.37	0.41
285376	11.02	13.27	0.62	0.66	0.89
445900	15.86	21.31	1.11	0.99	1.52
642096	24.73	33.47	2.01	1.43	2.92

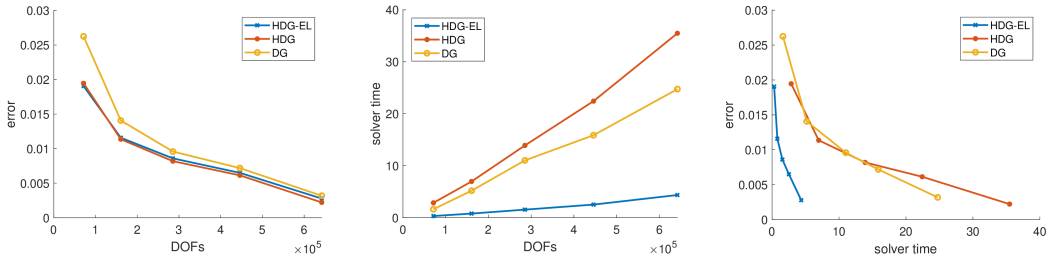

Fig. 3.13. Comparison of the different methods in terms of cost and accuracy, for the I3RC test case. Left: DOFs vs relative L^2 error. Middle: DOFs vs solver time. Right: solver time vs relative L^2 error. The solver time for HDG and HDG-EL is the summation of the local solver and the global solver time. For an error level of approximately 2×10^{-3}, in the right panel, the HDG-EL method is 5 to 10 times faster than the DG and HDG methods.

In the middle sub-figures of Fig. 3.13, we again observe a significant reduction of the solver time of HDG-EL compared to HDG and DG methods. Because of this, in the right sub-figures of Fig. 3.13, we observe that HDG-EL is much faster in reducing the error to the level of 2×10^{-3}, compared to HDG and DG methods. We again observe a 5 to 10 times speed-up by using element learning.

4. Conclusion

In this paper, we propose a novel approach to accelerating finite element-type methods by machine learning. A main goal is for the numerous benefits of traditional finite element-type methods to be retained. Many of the advantageous features of the element learning approach were listed near the end of Section 1. Numerical experiments have demonstrated promising results that an approximately 5-10 times speed-up can be achieved by using the proposed element learning method.

As was pointed out above, the element learning method is closely related to hybridizable discontinuous Galerkin methods in the sense that the local solvers are replaced by machine learning approaches. As a result, all hybridizable discontinuous Galerkin methods, hybridized mixed method, and hybridized continuous Galerkin methods, can be potentially accelerated by element learning. This generality suggests the potential of the proposed method to be applied to more PDEs that have been studied with HDG discretizations. Examples include but are not limited to Maxwell's equations, linear elasticity, convection-diffusion, and Stokes/Navier-Stokes equations.

Despite the great potential of the proposed method, there still exist many topics that remains to be studied. Here we list a few of them:

1. Even greater speed-up can be expected if a higher order polynomial degree p is used. At the same time, for larger p, other factors may arise, as discussed in the next item.

2. As the polynomial degree p increases, a larger ratio of the DOFs can be saved by the techniques of hybridization and/or static condensation. So, a greater acceleration can be expected by the element learning approach. However, as we increase the polynomial degree, it becomes more difficult to train the neural network. This can leads to less reliable predictions by the network. Therefore, how to choose the polynomial degree for a good balance between the efficiency of the method (for which we aim to increase p), and the reliability of the method (for which we aim to decrease p), remains to be studied.

3. The current approach achieves a computational speed-up even with using a simple fully connected neural network without any a priori structures. It remains to be studied that whether better approximation of the networks or reduced computation cost can be achieved by introducing additional structures, such as dropout [99], convolutional layers (CNN) [67], attention mechanism (transformer) [102], or shortcut connections (ResNet) [49].

4. The testing error of the neural network for element learning did not reach below 10^{-4}. This limitation hampers the method to be used for obtaining solutions with high accuracy requirement. It remains to be explored if additional techniques can reduce the error to a lower value.

5. In some applications, a desirable property of data-driven approaches is the ability to learn a data-driven solver without knowing the underlying PDE or dynamical equations. Here, for element learning, note that the underlying PDE does not need to be known. The global solver is defined by the consistency condition in (2.4b), which makes no explicit reference to the PDE itself, and instead refers only to a local solver $u_h(\widehat{u}_h, f)$ in abstract form. In the HDG method, the local solver comes from the PDE itself. In element learning, on the other hand, the local solver is learned from data, which could come from synthetic data from a numerical PDE solution if the PDE is known, or could come from observational measurements or laboratory experiments. Knowledge of boundary conditions is still needed for an appropriate definition of the local in2out operators in element learning, in order to ensure a well-posed problem, and to allow the coupling of local solvers in creating the global solver. It would be interesting in the future to investigate the ideas of element learning in scenarios where the underlying PDE or dynamical equations are unknown due to the complexity of the system, or are computationally intractable due to the high dimensionality of the true system. For example, such a situation arises in physics parameterizations or subgrid-scale parameterizations in atmospheric, oceanic, and climate dynamics and other complex systems [7, 18, 46, 85, 88, 106].

6. It is possible to discretize the in2out and the in2sol operators on each element K by numerical approaches other than the HDG local solvers. Examples include low-order finite element methods (macro-element) and finite difference methods. In this case, each element K is more like a small domain instead of an element in the traditional sense. In the case of macro-element, each macro-element K is discretized by several low-order

elements. This setting can be related to the setting of multi-scale finite element methods; see, for instance, the work [39] where a multiscale HDG method was developed for second order elliptic equations. Considering the close connection between element learning and HDG methods, it would be interesting to explore the potential of element learning for solving multiscale problems.

7. The main idea of element learning is to approximate the map from element geometry and coefficients to the element-wise solution operator. A topic that remains to be studied is the theoretical aspects of the approximation abilities of neural networks [14,37,48,64,96] for this map. This includes three parts: (1) best approximation capability of the network, (2) the estimate of the optimization error, and (3) the estimate of the generalization error.

Appendix A Implementation of HDG: Further Details

Here we explain how the matrices in (2.6) are assembled in more detail. For the first term, we have

$$\begin{aligned}\left(B^K[u_h]\right)_{K,K^a,i} &= \sum_{F\in\mathcal{E}_K}\sum_{K_*^a\cdot\mathbf{n}_F\geq 0}\int_{K_*^a}\int_F (\mathbf{s}\cdot\mathbf{n}_F)u_h v \\ &= \sum_{F\in\mathcal{E}_K}\sum_{K_*^a\cdot\mathbf{n}_F\geq 0}\int_{K_*^a}\int_F (\mathbf{s}\cdot\mathbf{n}_F)\sum_j u_j^{K\times K_*^a}\,\varphi_j^{K\times K_*^a}\varphi_i^{K\times K^a} \\ &= \sum_j u_j^{K\times K^a}\left(\sum_{F\in\mathcal{E}_K}\mathbb{1}_{K^a\cdot\mathbf{n}_F\geq 0}\int_{K^a}\int_F (\mathbf{s}\cdot\mathbf{n}_F)\,\varphi_j^{K\times K^a}\varphi_i^{K\times K^a}\right).\end{aligned}$$

For the second term, we have

$$\begin{aligned}\left(\widehat{B}^K[\widehat{u}_h]\right)_{K,K^a,i} &= \sum_{F\in\mathcal{E}_K}\sum_{K_*^a\cdot\mathbf{n}_F\leq 0}\int_{K_*^a}\int_F (\mathbf{s}\cdot\mathbf{n}_F)\widehat{u}_h v \\ &= \sum_{F\in\mathcal{E}_K}\sum_{K_*^a\cdot\mathbf{n}_F\leq 0}\int_{K_*^a}\int_F (\mathbf{s}\cdot\mathbf{n}_F)\sum_j \widehat{u}_j^{F\times K_*^a}\,\psi_j^{F\times K_*^a}\varphi_i^{K\times K^a} \\ &= \sum_{F\in\mathcal{E}_K}\sum_j \widehat{u}_j^{F\times K^a}\left(\mathbb{1}_{K^a\cdot\mathbf{n}_F\leq 0}\int_{K^a}\int_F (\mathbf{s}\cdot\mathbf{n}_F)\psi_j^{F\times K^a}\varphi_i^{K\times K^a}\right).\end{aligned}$$

For the third term, we have

$$\left(C^K[u_h]\right)_{K,K^a,i} = \sum_{K^a\in\mathcal{T}_h^a}\int_{K^a}\int_K u_h(\mathbf{s}\cdot\nabla v) = \sum_j u_j^{K\times K^a}\left(\int_{K^a}\int_K \varphi_j^{K\times K^a}\mathbf{s}\cdot\nabla\varphi_i^{K\times K^a}\right).$$

For the fourth term, we have

$$\left(M^K[u_h]\right)_{K,K^a,i} = \sum_{K^a\in\mathcal{T}_h^a}\int_{K^a}\int_K \sigma_e u_h v = \sum_j u_j^{K\times K^a}\left(\int_{K^a}\int_K \sigma_e\varphi_j^{K\times K^a}\varphi_i^{K\times K^a}\right).$$

For the fifth term for scattering, we have

$$\begin{aligned}
&\left(S^K[u_h]\right)_{K,K^a,i} \\
&= \sum_{K^a\in\mathcal{T}_h^a}\int_{K^a}\int_K \sigma_s(\mathbf{x})\int_S p(\mathbf{s},\mathbf{s}')u_h(\mathbf{x},\mathbf{s}')ds'v_h(\mathbf{x},\mathbf{s})dxds \\
&= \int_{K^a}\int_K \sigma_s(\mathbf{x})\int_S p(\mathbf{s},\mathbf{s}')\sum_{K_*^a}\sum_j u_j^{K\times K_*^a}\varphi_j^{K\times K_*^a}(\mathbf{x},\mathbf{s}')ds'\varphi_i^{K\times K^a}(\mathbf{x},\mathbf{s})dxds \\
&= \sum_{K_*^a\in\mathcal{T}_h^a}\sum_j u_j^{K\times K_*^a}\left(\int_K \sigma_s(\mathbf{x})\int_{K^a}\int_{K_*^a} p(\mathbf{s},\mathbf{s}')\varphi_j^{K\times K_*^a}(\mathbf{x},\mathbf{s}')\varphi_i^{K\times K^a}(\mathbf{x},\mathbf{s})ds'dsdx\right).
\end{aligned}$$

Acknowledgments. The authors also thank the referees for their suggestions which have led to an improvement of the presentation of the paper.

The research of S. Du and S.N. Stechmann is partially supported by the NSF (Grant No. DMS 2324368) and by the Office of the Vice Chancellor for Research and Graduate Education at the University of Wisconsin-Madison with funding from the Wisconsin Alumni Research Foundation.

References

[1] M.L. Adams and E.W. Larsen, Fast iterative methods for discrete-ordinates particle transport calculations, *Prog. Nucl. Energy*, **40**:1 (2002), 3–159.

[2] D.N. Arnold, F. Brezzi, B. Cockburn, and L.D. Marini, Unified analysis of discontinuous Galerkin methods for elliptic problems, *SIAM J. Numer. Anal.*, **39**:5 (2002), 1749–1779.

[3] D.N. Arnold, R.S. Falk, and R. Winther, Finite element exterior calculus, homological techniques, and applications, *Acta Numer.*, **15** (2006), 1–155.

[4] S.R. Arridge and J.C. Schotland, Optical tomography: Forward and inverse problems, *Inverse Problems*, **25**:12 (2009), 123010.

[5] I. Babuška and M. Suri, The p and h-p versions of the finite element method, basic principles and properties, *SIAM Rev.*, **36**:4 (1994), 578–632.

[6] P. Batlle, M. Darcy, B. Hosseini, and H. Owhadi, Kernel methods are competitive for operator learning, *J. Comput. Phys.*, **496** (2024), 112549.

[7] A. Boral, Z.Y. Wan, L. Zepeda-Núñez, J. Lottes, Q. Wang, Y.-f. Chen, J.R. Anderson, and F. Sha, Neural ideal large eddy simulation: Modeling turbulence with neural stochastic differential equations, *arXiv:2306.01174*, 2023.

[8] S.C. Brenner and L.R. Scott, *The Mathematical Theory of Finite Element Methods*, in: *Texts in Applied Mathematics*, Vol. 15, Springer, 2008.

[9] T. Brown et al., Language models are few-shot learners, *Adv. Neural Inf. Process. Syst.*, **33** (2020), 1877–1901.

[10] J. Bruna, B. Peherstorfer, and E. Vanden-Eijnden, Neural Galerkin schemes with active learning for high-dimensional evolution equations, *J. Comput. Phys.*, **496** (2024), 112588.

[11] S.L. Brunton, J.L. Proctor, and J.N. Kutz, Discovering governing equations from data by sparse identification of nonlinear dynamical systems, *Proc. Natl. Acad. Sci.*, **113**:15 (2016), 3932–3937.

[12] R.F. Cahalan et al., The I3RC: Bringing together the most advanced radiative transfer tools for cloudy atmospheres, *Bull. Am. Meteorol. Soc.*, **86**:9 (2005), 1275–1294.

[13] L. Chacon, G. Chen, D.A. Knoll, C. Newman, H. Park, W. Taitano, J.A. Willert, and G. Womeldorff, Multiscale high-order/low-order (HOLO) algorithms and applications, *J. Comput. Phys.*, **330** (2017), 21–45.

[14] K. Chen, C. Wang, and H. Yang, Deep operator learning lessens the curse of dimensionality for PDEs, *arXiv:2301.12227*, 2023.

[15] N. Chen and Y. Li, BAMCAFE: A Bayesian machine learning advanced forecast ensemble method for complex turbulent systems with partial observations, *Chaos*, **31**:11 (2021), 113114.

[16] S. Chen, Z. Ding, Q. Li, and S.J. Wright, A reduced order Schwarz method for nonlinear multiscale elliptic equations based on two-layer neural networks, *J. Comput. Math.*, **42**:2 (2024), 570–596.

[17] Y. Chen, B. Hosseini, H. Owhadi, and A.M. Stuart, Solving and learning nonlinear PDEs with Gaussian processes, *J. Comput. Phys.*, **447** (2021), 110668.

[18] S.K. Clark, N.D. Brenowitz, B. Henn, A. Kwa, J. McGibbon, W.A. Perkins, O. Watt-Meyer, C.S. Bretherton, and L.M. Harris, Correcting a 200 km resolution climate model in multiple climates by machine learning from 25 km resolution simulations, *J. Adv. Model. Earth Syst.*, **14**:9 (2022), e2022MS003219.

[19] B. Cockburn, G. Fu, and F.J. Sayas, Superconvergence by M-decompositions. Part I: General theory for HDG methods for diffusion, *Math. Comp.*, **86**:306 (2017), 1609–1641.

[20] B. Cockburn, J. Gopalakrishnan, and R. Lazarov, Unified hybridization of discontinuous Galerkin, mixed, and continuous Galerkin methods for second order elliptic problems, *SIAM J. Numer. Anal.*, **47**:2 (2009), 1319–1365.

[21] B. Cockburn and C.-W. Shu, Runge-Kutta discontinuous Galerkin methods for convection-dominated problems, *J. Sci. Comput.*, **16**:3 (2001), 173–261.

[22] C.J. Cotter and J. Thuburn, A finite element exterior calculus framework for the rotating shallow-water equations, *J. Comput. Phys.*, **257** (2014), 1506–1526.

[23] M. Crouzeix and P.-A. Raviart, Conforming and nonconforming finite element methods for solving the stationary Stokes equations. I, *Analyse Numérique*, **7**:R-3 (1973), 33–75.

[24] S. Cuomo, V.S. Di Cola, F. Giampaolo, G. Rozza, M. Raissi, and F. Piccialli, Scientific machine learning through physics-informed neural networks: Where we are and what's next, *J. Sci. Comput.*, **92**:88 (2022).

[25] M. Darcy, B. Hamzi, G. Livieri, H. Owhadi, and P. Tavallali, One-shot learning of stochastic differential equations with data adapted kernels, *Phys. D: Nonlinear Phenom.*, **444** (2023), 133583.

[26] T.A. Davis, Algorithm 832: UMFPACK V4.3 – an unsymmetric-pattern multifrontal method, *ACM Trans. Math.*, **30**:2 (2004), 196–199.

[27] M.V. de Hoop, D.Z. Huang, E. Qian, and A.M. Stuart, The cost-accuracy trade-off in operator learning with neural networks, *J. Mach. Learn.*, **1**:3 (2022), 299–341.

[28] D.A. Di Pietro and R. Tittarelli, An introduction to hybrid high-order methods, in: *Numerical methods for PDEs*, *SEMA SIMAI Springer Ser.*, **15** (2018), 75–128.

[29] W. Ding, K. Ren, and L. Zhang, Coupling deep learning with full waveform inversion, *arXiv: 2203.01799*, 2022.

[30] J. Dölz, O. Palii, and M. Schlottbom, On robustly convergent and efficient iterative methods for anisotropic radiative transfer, *J. Sci. Comput.*, **90** (2022), 94.

[31] S. Du and F.-J. Sayas, *An Invitation to the Theory of the Hybridizable Discontinuous Galerkin Method. Projections, Estimates, Tools*, Springer, 2019.

[32] S. Du and F.-J. Sayas, A note on devising HDG+ projections on polyhedral elements, *Math. Comp.*, **90**:327 (2021), 65–79.

[33] S. Du and S.N. Stechmann, Fast, low-memory numerical methods for radiative transfer via *hp*-adaptive mesh refinement, *J. Comput. Phys.*, **480** (2023), 112021.

[34] S. Du and S.N. Stechmann, Inverse radiative transfer with goal-oriented *hp*-adaptive mesh refinement: Adaptive-mesh inversion, *Inverse Problems*, **39**:11 (2023), 115002.

[35] P.D. Düben, P. Korn, and V. Aizinger, A discontinuous/continuous low order finite element shallow water model on the sphere, *J. Comput. Phys.*, **231**:6 (2012), 2396–2413.

[36] W. E, J. Han, and A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, *Commun. Math. Stat.*, **5**:4 (2017), 349–380.

[37] W. E, C. Ma, and L. Wu, The Barron space and the flow-induced function spaces for neural network models, *Constr. Approx.*, **55**:1 (2022), 369–406.

[38] Y. Efendiev and T.Y. Hou, *Multiscale Finite Element Methods: Theory and Applications*, in: *Surveys and Tutorials in the Applied Mathematical Sciences*, Vol. 4, Springer Science & Business Media, 2009.

[39] Y. Efendiev, R. Lazarov, and K. Shi, A multiscale HDG method for second order elliptic equations. Part I. Polynomial and homogenization-based multiscale spaces, *SIAM J. Numer. Anal.*, **53**:1 (2015), 342–369.

[40] B. Engquist, K. Ren, and Y. Yang, A generalized weighted optimization method for computational learning and inversion, in: *The Tenth International Conference on Learning Representations (ICLR 2022)*, OpenReview, 2022.

[41] Y. Fan, L. Lin, L. Ying, and L. Zepeda-Núnez, A multiscale neural network based on hierarchical matrices, *Multiscale Model. Simul.*, **17**:4 (2019), 1189–1213.

[42] I.M. Gamba and S. Rjasanow, Galerkin-Petrov approach for the Boltzmann equation, *J. Comput. Phys.*, **366** (2018), 341–365.

[43] A. Gillette, M. Holst, and Y. Zhu, Finite element exterior calculus for evolution problems, *J. Comput. Math.*, **35**:2 (2017), 187–212.

[44] G.A. Gottwald and S. Reich, Combining machine learning and data assimilation to forecast dynamical systems from noisy partial observations, *Chaos*, **31**:10 (2021), 101103.

[45] W. Gregory, M. Bushuk, A. Adcroft, Y. Zhang, and L. Zanna, Deep learning of systematic sea ice model errors from data assimilation increments, *J. Adv. Model. Earth Syst.*, **15**:10 (2023), e2023MS003757.

[46] Y. Guan, A. Chattopadhyay, A. Subel, and P. Hassanzadeh, Stable a posteriori LES of 2D turbulence using convolutional neural networks: Backscattering analysis and generalization to higher *Re* via transfer learning, *J. Comput. Phys.*, **458** (2022), 111090.

[47] J. Han, A. Jentzen, and W. E, Solving high-dimensional partial differential equations using deep learning, *Proc. Natl. Acad. Sci. USA*, **115**:34 (2018), 8505–8510.

[48] J. He, L. Li, J. Xu, and C. Zheng, Relu deep neural networks and linear finite elements, *J. Comput. Math.*, **38**:3 (2020), 502–527.

[49] K. He, X. Zhang, S. Ren, and J. Sun, Deep residual learning for image recognition, in: *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, 770–778, 2016.

[50] R.E. Heath, I.M. Gamba, P.J. Morrison, and C. Michler, A discontinuous Galerkin method for the Vlasov-Poisson system, *J. Comput. Phys.*, **231**:4 (2012), 1140–1174.

[51] R.J. Hogan and A. Bozzo, A flexible and efficient radiation scheme for the ECMWF model, *J. Adv. Model. Earth Syst.*, **10**:8 (2018), 1990–2008.

[52] R.J. Hogan et al., *Radiation in Numerical Weather Prediction*, Technical Report 816, European Centre for Medium-Range Weather Forecasts, 2017.

[53] I. Horenko, E. Vecchi, J. Kardoš, A. Wächter, O. Schenk, T.J. O'Kane, P. Gagliardini, and S. Gerber, On cheap entropy-sparsified regression learning, *Proc. Natl. Acad. Sci. USA*, **120**:1 (2023), e2214972120.

[54] P. Houston and E. Süli, A note on the design of *hp*-adaptive finite element methods for elliptic partial differential equations, *Comput. Methods Appl. Mech. Engrg.*, **194**:2-5 (2005), 229–243.

[55] J. Hu, J.G. Liu, Y. Xie, and Z. Zhou, A structure preserving numerical scheme for Fokker-Planck equations of neuron networks: Numerical analysis and exploration, *J. Comput. Phys.*, **433** (2021), 110195.

[56] J. Jumper et al., Highly accurate protein structure prediction with AlphaFold, *Nature*, **596**:7873 (2021), 583–589.

[57] A. Kaneda, O. Akar, J. Chen, V. Kala, D. Hyde, and J. Teran, A deep conjugate direction method for iteratively solving linear systems, *arXiv:2205.10763*, 2022.

[58] Y. Khoo, J. Lu, and L. Ying, Solving parametric PDE problems with artificial neural networks, *Eur. J. Appl. Math.*, **32**:3 (2021), 421–435.

[59] D. Kochkov, J.A. Smith, A. Alieva, Q. Wang, M.P. Brenner, and S. Hoyer, Machine learning-accelerated computational fluid dynamics, *Proc. Natl. Acad. Sci. USA*, **118**:21 (2021), e2101784118.

[60] N. Kovachki, Z. Li, B. Liu, K. Azizzadenesheli, K. Bhattacharya, A. Stuart, and A. Anandkumar, Neural operator: Learning maps between function spaces with applications to PDEs, *J. Mach. Learn. Res.*, **24** (2023), 4061–4157.

[61] A. Krizhevsky, I. Sutskever, and G.E. Hinton, Imagenet classification with deep convolutional neural networks, *Commun. ACM*, **60**:6 (2017), 84–90.

[62] M. Kronbichler and K. Kormann, Fast matrix-free evaluation of discontinuous Galerkin finite element operators, *ACM Trans. Math. Softw.*, **45**:3 (2019), 29.

[63] M. Kronbichler, K. Kormann, and W.A. Wall, Fast matrix-free evaluation of hybridizable discontinuous Galerkin operators, in: *Numerical Mathematics and Advanced Applications ENUMATH 2017*, Springer, 581–589, 2019.

[64] S. Lanthaler and A.M. Stuart, The curse of dimensionality in operator learning, *arXiv: 2306.15924*, 2023.

[65] E.W. Larsen and J.E. Morel, *Advances in Discrete-Ordinates Methodology*, in: *Nuclear Computational Science*, Springer, 2010.

[66] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning, *Nature*, **521**:7553 (2015), 436–444.

[67] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, Gradient-based learning applied to document recognition, *Proc. IEEE Inst. Electr. Electron. Eng.*, **86**:11 (1998), 2278–2324.

[68] P. Lesaint and P.-A. Raviart, On a finite element method for solving the neutron transport equation, *Publications Mathématiques et Informatique de Rennes*, S4 (1974), 8.

[69] R.J. LeVeque, *Numerical Methods for Conservation Laws*, in: *Lectures in Mathematics. ETH Zürich*, Birkhäuser Verlag, 1992.

[70] Z. Li et al., Geometry-informed neural operator for large-scale 3d PDEs, *arXiv:2309.00583*, 2023.

[71] A.J. Linot, J.W. Burby, Q. Tang, P. Balaprakash, M.D. Graham, and R. Maulik, Stabilized neural ordinary differential equations for long-time forecasting of dynamical systems, *J. Comput. Phys.*, **474** (2023), 111838.

[72] L. Lu, P. Jin, G. Pang, Z. Zhang, and G.E. Karniadakis, Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators, *Nat. Mach. Intell.*, **3**:3 (2021), 218–229.

[73] Y. Lu, L. Wang, and W. Xu, Solving multiscale steady radiative transfer equation using neural networks with uniform stability, *Res. Math. Sci.*, **9** (2022), 45.

[74] S. Marras, J.F. Kelly, M. Moragues, A. Müller, M.A. Kopera, M. Vázquez, F.X. Giraldo, G. Houzeaux, and O. Jorba, A review of element-based Galerkin methods for numerical weather prediction: Finite elements, spectral elements, and discontinuous Galerkin, *Arch. Comput. Methods Eng.*, **23**:4 (2016), 673–722.

[75] S. Mishra and R. Molinaro, Physics informed neural networks for simulating radiative transfer, *J. Quant. Spectrosc.*, **270** (2021), 107705.

[76] R. Molinaro, Y. Yang, B. Engquist, and S. Mishra, Neural inverse operators for solving PDE inverse problems, *arXiv:2301.11167*, 2023.

[77] P. Monk, *Finite Element Methods for Maxwell's Equations*, Oxford University Press, 2003.

[78] H. Montanelli, H. Yang, and Q. Du, Deep ReLU networks overcome the curse of dimensionality for generalized bandlimited functions, *J. Comput. Math.*, **39**:6 (2021), 801–815.

[79] J.E. Morel and T. Manteuffel, An angular multigrid acceleration technique for S_n equations with highly forward-peaked scattering, *Nucl. Sci. Eng.*, **107**:4 (1991), 330–342.

[80] J.-C. Nédélec, Mixed finite elements in $\mathbf{R}^3$, *Numer. Math.*, **35**:3 (1980), 315–341.
[81] H. Park, S.G. Srinivasan, M. Cong, D. Kim, B. Kim, J. Swartz, K. Museth, and E. Sifakis, Near-realtime facial animation by deep 3D simulation super-resolution, *arXiv:2305.03216*, 2023.
[82] J. Pathak et al., FourCastNet: A global data-driven high-resolution weather model using adaptive Fourier neural operators, *arXiv:2202.11214*, 2022.
[83] J. Pathak, B. Hunt, M. Girvan, Z. Lu, and E. Ott, Model-free prediction of large spatiotemporally chaotic systems from data: A reservoir computing approach, *Phys. Rev. Lett.*, **120**:2 (2018), 024102.
[84] B.W. Patton and J.P. Holloway, Application of preconditioned GMRES to the numerical solution of the neutron transport equation, *Ann. Nucl. Energy*, **29**:2 (2002), 109–136.
[85] P. Perezhogin, L. Zanna, and C. Fernandez-Granda, Generative data-driven approaches for stochastic subgrid parameterizations in an idealized ocean model, *arXiv:2302.07984*, 2023.
[86] D. Qi and A.J. Majda, Using machine learning to predict extreme events in complex systems, *Proc. Natl. Acad. Sci. USA*, **117**:1 (2020), 52–59.
[87] M. Raissi, P. Perdikaris, and G.E. Karniadakis, Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations, *J. Comput. Phys.*, **378** (2019), 686–707.
[88] S. Rasp, M.S. Pritchard, and P. Gentine, Deep learning to represent subgrid processes in climate models, *Proc. Natl. Acad. Sci. USA*, **115**:39 (2018), 9684–9689.
[89] P.-A. Raviart and J.M. Thomas, A mixed finite element method for 2-nd order elliptic problems, in: *Mathematical Aspects of Finite Element Methods. Lecture Notes in Mathematics*, Springer, Vol. 606, 292–315, 1977.
[90] W.H. Reed and T.R. Hill, *Triangular Mesh Methods for the Neutron Transport Equation*, Technical Report LA-UR-73-479, Los Alamos Scientific Laboratory, 1973.
[91] K. Ren, Recent developments in numerical techniques for transport-based medical imaging methods, *Commun. Comput. Phys.*, **8**:1 (2010), 1–50.
[92] K. Ren, G. Bal, and A.H. Hielscher, Frequency domain optical tomography based on the equation of radiative transfer, *SIAM J. Sci. Comput.*, **28**:4 (2006), 1463–1489.
[93] Y. Saad and M.H. Schultz, GMRES: A generalized minimal residual algorithm for solving nonsymmetric linear systems, *SIAM J. Sci. Comput.*, **7**:3 (1986), 856–869.
[94] T. Schneider, A.M. Stuart, and J.-L. Wu, Ensemble Kalman inversion for sparse learning of dynamical systems from time-averaged data, *J. Comput. Phys.*, **470** (2022), 111559.
[95] L. Serrano, L. Le Boudec, A.K. Koupaï, T.X. Wang, Y. Yin, J.-N. Vittaut, and P. Gallinari, Operator learning with neural fields: Tackling PDEs on general geometries, *arXiv:2306.07266*, 2023.
[96] Z. Shen, H. Yang, and S. Zhang, Optimal approximation rate of ReLU networks in terms of width and depth, *J. Math. Pures Appl.*, **157** (2022), 101–135.
[97] D. Silver et al., Mastering the game of Go with deep neural networks and tree search, *Nature*, **529**:7587 (2016), 484–489.
[98] A. Slingo, A GCM parameterization for the shortwave radiative properties of water clouds, *J. Atmos. Sci.*, **46**:10 (1989), 1419–1427.
[99] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, Dropout: A simple way to prevent neural networks from overfitting, *J. Mach. Learn. Res.*, **15**:1 (2014), 1929–1958.
[100] M.E. Tano and J.C. Ragusa, Sweep-Net: An artificial neural network for radiation transport solves, *J. Comput. Phys.*, **426** (2021), 109757.
[101] M. Taylor, J. Tribbia, and M. Iskandarani, The spectral element method for the shallow water equations on the sphere, *J. Comput. Phys.*, **130**:1 (1997), 92–108.
[102] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A.N. Gomez, L. Kaiser, and I. Polosukhin, Attention is all you need, *Adv. Neural Inf. Process. Syst.*, **30** (2017).
[103] P.R. Vlachas, J. Pathak, B.R. Hunt, T.P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos, Back-

propagation algorithms and reservoir computing in recurrent neural networks for the forecasting of complex spatiotemporal dynamics, *Neural Netw.*, **126** (2020), 191–217.

[104] J.S. Warsa, T.A. Wareing, and J.E. Morel, Krylov iterative methods and the degraded effectiveness of diffusion synthetic acceleration for multidimensional S_N calculations in problems with material discontinuities, *Nucl. Sci. Eng.*, **147**:3 (2004), 218–248.

[105] M. Woopen, A. Balan, G. May, and J. Schütz, A comparison of hybridized and standard DG methods for target-based *hp*-adaptive simulation of compressible flow, *Comput. Fluids*, **98** (2014), 3–16.

[106] J.-L. Wu et al., Enforcing statistical constraints in generative adversarial networks for modeling chaotic dynamical systems, *J. Comput. Phys.*, **406** (2020), 109209.

[107] B. Yu et al., The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems, *Commun. Math. Stat.*, **6**:1 (2018), 1–12.

[108] J. Yu and J.S. Hesthaven, A data-driven shock capturing approach for discontinuous Galerkin methods, *Comput. Fluids*, **245** (2022), 105592.

[109] L. Zepeda-Núñez, Y. Chen, J. Zhang, W. Jia, L. Zhang, and L. Lin, Deep Density: Circumventing the Kohn-Sham equations via symmetry preserving neural networks, *J. Comput. Phys.*, **443** (2021), 110523.