Finite Expression Method for Solving High-Dimensional Partial Differential Equations

Senwei Liang[1] **and Haizhao Yang**[2,*]

[2]Department of Mathematics, Purdue University, West Lafayette, USA
[4]Department of Mathematics, University of Maryland College Park, College Park, USA
[*]Correspondence should be addressed to: hzyang@umd.edu

ABSTRACT

Designing efficient and accurate numerical solvers for high-dimensional partial differential equations (PDEs) remains a challenging and important topic in computational science and engineering, mainly due to the "curse of dimensionality" in designing numerical schemes that scale in dimension. This paper introduces a new methodology that seeks an approximate PDE solution in the space of functions with finitely many analytic expressions and, hence, this methodology is named the finite expression method (FEX). It is proved in approximation theory that FEX can avoid the curse of dimensionality. As a proof of concept, a deep reinforcement learning method is proposed to implement FEX for various high-dimensional PDEs in different dimensions, achieving high and even machine accuracy with a memory complexity polynomial in dimension and an amenable time complexity. An approximate solution with finite analytic expressions also provides interpretable insights into the ground truth PDE solution, which can further help to advance the understanding of physical systems and design postprocessing techniques for a refined solution.

1 Introduction

Partial differential equations (PDEs) are fundamental in scientific fields modeling many physical phenomena such as diffusion[1,2], fluid dynamics[3,4], and quantum mechanics[5,6]. Developing efficient and accurate solvers for numerical solutions to high-dimensional PDEs remains an important and challenging topic[7]. Many traditional solvers, such as finite element method (FEM)[8] and finite difference[9], are usually limited to low-dimensional domains since the computational cost increases exponentially in the dimension[7,10]. Recently, neural networks (NNs) as mesh-free parameterization are widely employed in solving high-dimensional PDEs[11–16] and control problems[17]. In theory, NNs have the capability of approximating various functions well and lessening the curse of dimensionality[18–22]. Yet the highly non-convex objective function and the spectral bias towards fitting a smooth function in NN optimization make it difficult to achieve high accuracy[23–25]. In practice, NN-based solvers can hardly achieve a highly accurate solution even when the true solution is a simple function, especially for high-dimensional problems[16,26]. Besides, NN parametrization may still require large memory and high computation cost for high-dimensional problems[27]. Finally, numerical solutions provided by traditional solvers and NN-based solvers are not interpretable, e.g., the dependence of the solution on variables cannot be readily seen from numerical solutions.

In this paper, we propose the finite expression method (FEX), a methodology that aims to find a solution in the function space of mathematical expressions with finitely many operators. Compared with the NN and FEM methods, our FEX enjoys the following advantages (summarized in Fig. 1): (1) The expression may reproduce the true solution and achieve a solution with high and even machine accuracy. (2) The expression requires a low cost on memory (a line of string) for solution storage and computation for solution evaluation. (3) The expression has good interpretability with an explicit and legible form. Besides, from the approximation perspective illustrated in Sec. 2.3, the expression in FEX is capable of avoiding the curse of dimensionality in theory.

While many techniques can be used to implement the proposed FEX, we provide a numerically effective implementation based on reinforcement learning (RL). We formulate the problem of searching for mathematical expressions as a combinatorial optimization (CO) over both discrete and continuous variables. Many traditional algorithms (e.g., genetic algorithms and simulated annealing[28]) solve the CO based on hand-crafted heuristics, which highly rely on the problem statement and domain-specific knowledge[29,30]. Recently, RL becomes a popular and general tool to learn to construct the CO solution based on reward feedback, without much heuristic design. The success of RL applications involving CO, such as automatic algorithm design[31–33] and symbolic regression[34,35], also inspires us to seek the mathematical expression solution with RL. Specifically, in our implementation, the mathematical expression is represented by a binary tree, where each node is an operator along with parameters (scaling and bias) as shown in Fig. 3b and elaborated in Sec. 3.1. The objective function is a functional whose

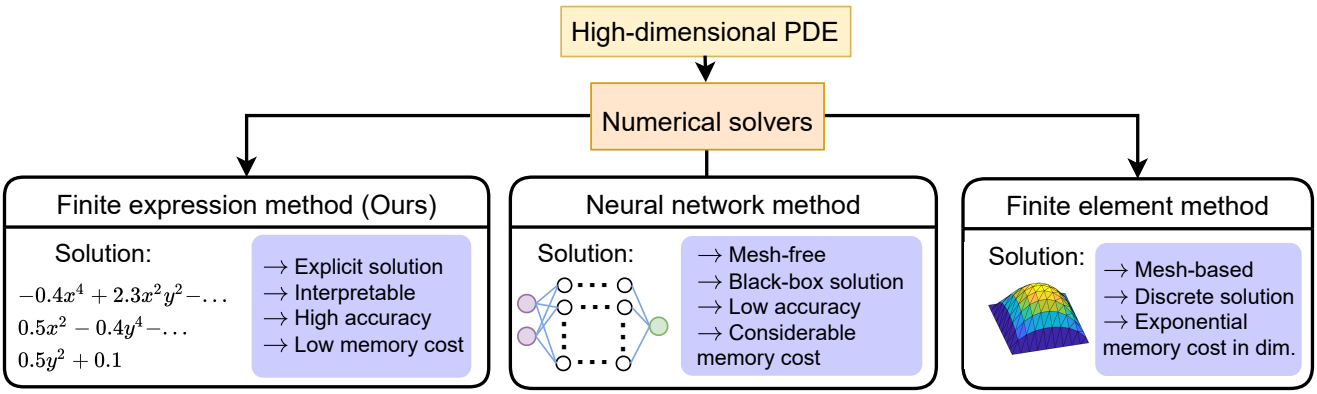

Figure 1. Overview of various numerical solvers for PDEs. In our finite expression method, we aim to find a PDE solution as a mathematical expression with finitely many operators. The resulting solution may reproduce the true solution, and achieve high and even machine accuracy. Furthermore, the mathematical expression costs a memory complexity polynomial in dimension and advances the understanding of physical systems and the design of postprocessing techniques for a refined solution.

minimizer is the PDE solution (Sec. 2.2) so our problem becomes minimization over the discrete variables (operators) and continuous variables (parameters) of the tree. Optimizing both discrete and continuous variables simultaneously is essentially difficult. We propose a searching loop for this CO as shown in Fig. 2. Our idea is to first determine good operator selection that has a high possibility of recovering the structure of the true solution and then optimize over the parameters. The proposal of operator selection is drawn from a controller, which will be updated via the policy gradient[34] of RL. In Sec. 4, we numerically demonstrate the ability of this RL-based implementation to find the mathematical expressions that solve high-dimensional PDEs with high and even machine accuracy. Furthermore, FEX provides interpretable insights into the ground truth PDE solution, which can further help to advance the understanding of physical systems and design postprocessing techniques for a refined solution.

2 An Abstract Methodology of Finite Expression Method

The goal of FEX is to find a mathematical expression to solve a PDE. This section will formally define the function space of mathematical expressions and formulate the problem of FEX as a CO.

2.1 Function Space with Finite Expressions in FEX

Mathematical expressions will be introduced to form the function space of FEX.

Definition 2.1 (Mathematical expression)**.** A mathematical expression is a combination of symbols, which is well-formed by syntax and rules and forms a valid function. The symbols include operands (variables and numbers), operators (e.g.,"+", "sin", integral, derivative), brackets, punctuation.

In the definition of mathematical expressions, we only consider the expression that forms a valid function. In our context, "$\sin(x \times y) + 1$" is a mathematical expression, but, for example, "$5 > x$" and "$\sin(x \times y) +$" are not a mathematical expression as they do not form a valid function. The operands and operators comprise the structure of an expression. The parentheses play a role in clarifying the operation order.

Definition 2.2 (k-finite expression)**.** A mathematical expression is called a k-finite expression if the number of operators in this expression is k.

"$\sin(x \times y) + 1$" is a 3-finite expression since there are three operators in it ("$\times$", "sin", and "+"). The series, such as "$1 + x^1 + \frac{x^2}{2} + \frac{x^3}{6} + \cdots$", belongs to a mathematical expression, but it is not a finite expression since the amount of the operators is infinite. Formally, with the concept of finite expression, we can define FEX as follows,

Definition 2.3 (Finite expression method)**.** The finite expression method is a methodology to solve a PDE numerically by seeking a finite expression such that the resulting function solves the PDE approximately.

We denote $\mathbb{S}_k$ as a set of functions that are formed by finite expressions with the number of operators not larger than k. $\mathbb{S}_k$ forms the function space of FEX. Clearly, $\mathbb{S}_1 \subset \mathbb{S}_2 \subset \mathbb{S}_3 \cdots$.

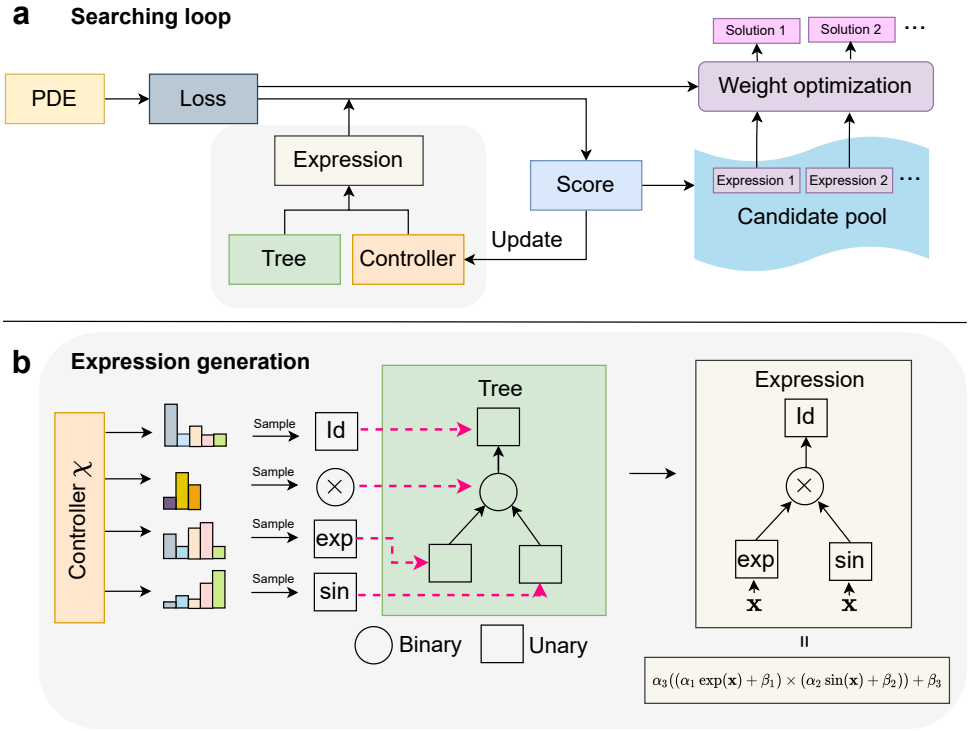

Figure 2. Representation of the components of our FEX implementation. (a) The searching loop for the symbolic solution consists of expression generation, score computation, controller update, and candidate optimization. (b) Depiction of the expression generation with a binary tree and a controller χ. The controller outputs the probability mass functions for each node of the tree, from which the node values are sampled. The expression with learnable scaling and bias parameters is generated based on the predefined tree and the sampled node values.

2.2 Identifying PDE Solutions in FEX

We denote a functional $\mathcal{L} : \mathbb{S} \to \mathbb{R}$ associated with a given PDE, where $\mathbb{S}$ is a function space and the minimizer of $\mathcal{L}$ is the best solution to solve the PDE in $\mathbb{S}$. In FEX, given the number of operators k, the problem of seeking a finite expression solution is formulated as a CO over $\mathbb{S}_k$ via

$$\min\{\mathcal{L}(u)|u \in \mathbb{S}_k\}. \tag{1}$$

The choice of the functional $\mathcal{L}$ is problem-dependent and one may conceive a better functional for a specific PDE with a specific constraint or domain. Some popular choices include a least-square method[15,36,37], a variation formulation[16,38], and a weak formulation[39,40] (see Appx. 7.1).

2.3 Approximation Theory of Elementary Expressions in FEX

The most important theoretical question in solving high-dimensional problems is whether or not a solver suffers from the curse of dimensionality. It will be shown that the function space of k-finite expressions, i.e., $\mathbb{S}_k$ in (1), is a powerful function space that avoids the curse of dimensionality in approximating high-dimensional continuous functions, leveraging the recent development of advanced approximation theory of deep neural networks[21,22]. First of all, it can be proved that $\mathbb{S}_k$ is dense in $C([0,1]^d)$ for an arbitrary $d \in \mathbb{N}$ in the following theorem.

Theorem 2.1. The function space $\mathbb{S}_k$, generated with operators including "+", "−", "×", "/", "$|\cdot|$", "sign$(\cdot)$", and "$\lfloor\cdot\rfloor$", is dense in $C([a,b]^d)$ for arbitrary a, $b \in \mathbb{R}$ and $d \in \mathbb{N}$ if $k \geq \mathcal{O}(d^4)$.

Here "$|\cdot|$", "sign$(\cdot)$", and "$\lfloor\cdot\rfloor$" denote the absolute, sign and floor functions[21], respectively. The proof of Thm. 2.1 can be found in Appx. 7.2. The denseness of $\mathbb{S}_k$ means that the function space of k-finite expressions can approximate any d-dimensional continuous functions to any accuracy, while k is only required to be $\mathcal{O}(d^4)$ independent of the approximation accuracy. The proof of Thm. 2.1 takes the advantage of operators "sign$(\cdot)$" and "$\lfloor\cdot\rfloor$", which might not be frequently used in mathematical expressions. If it is more practical to restrict the operator list to regular

operators like "+", "×", "sin(·)", exponential functions, and the rectified linear unit (ReLU), then it can be proved that $\mathbb{S}_k$ can approximate Hölder functions without the curse of dimensionality in the following theorem.

Theorem 2.2. Suppose the function space $\mathbb{S}_k$ is generated with operators including "+", "−", "×", "÷", "$\max\{0,x\}$", "$\sin(x)$", and "2^x". Let $p \in [1,+\infty)$. For any f in the Hölder function class $\mathcal{H}^{\alpha}_{\mu}([0,1]^d)$ and $\varepsilon > 0$, there exists a k-finite expression ϕ in $\mathbb{S}_k$ such that $\|f-\phi\|_{L^p} \le \varepsilon$, if $k \ge \mathcal{O}(d^2(\log d + \log \frac{1}{\varepsilon})^2)$.

The proof of Thm. 2.2 can be found in Appx. 7.2. Although finite expressions have a powerful approximation capacity for high-dimensional functions, it is challenging to theoretically prove that our FEX solver to be introduced in Sec. 3 can identify the desired finite expressions with this power. However, Sec. 4 shows numerically that our FEX solver can identify desired finite expressions up to machine accuracy for several classes of high-dimensional PDEs.

3 An Implementation of FEX

After the introduction of the abstract methodology of FEX in Sec. 2, numerical implementation of FEX is proposed here for various PDEs. First, binary trees are applied to construct finite expressions in Sec. 3.1. Next, our CO (1) is formulated in terms of the parameter and operator selection to seek the expression that approximates a PDE solution in Sec. 3.2. To resolve this CO, a searching loop is proposed to search for good operators such that the expression with this operator selection will potentially recover the true solution.

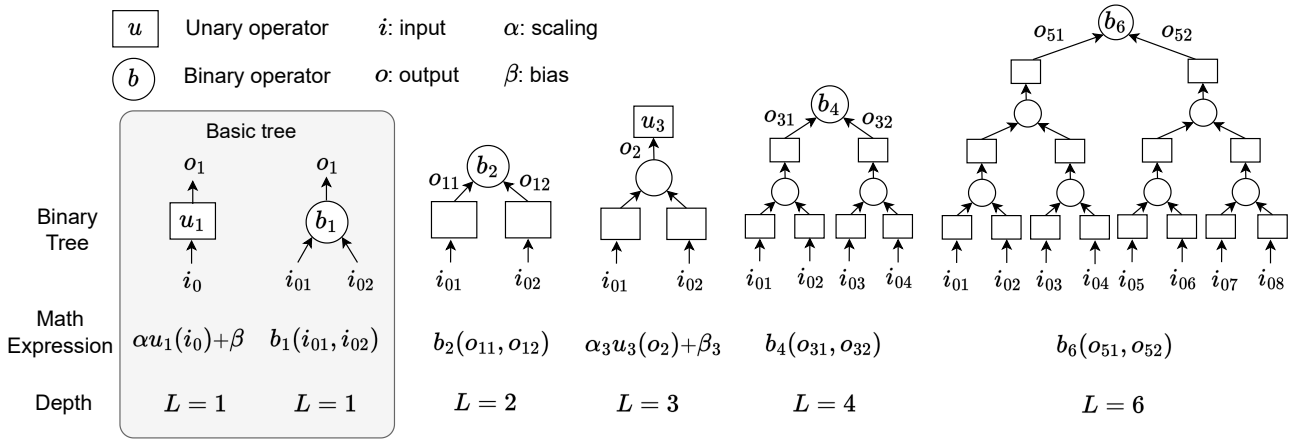

Figure 3. Computational rule of a binary tree. For a binary tree, the value of each node is either a unary or a binary operator. We first define the computation flow of a depth-1 tree that contains only one operator. Then for the binary tree with more than one layer, the computation is conducted by recursion.

3.1 Finite Expressions with Binary Trees

A finite expression can be reformulated as a binary tree $\mathcal{T}$ as illustrated in Fig. 3. Each tree node takes a value from an operator set and all the node values form an operator sequence $\boldsymbol{e}$ according to a fixed order to traverse the tree. In each node of the unary operator, a scaling parameter α and a bias parameter β are added to the operator to enhance expressiveness. All these parameters are denoted by $\boldsymbol{\theta}$. Therefore, a finite expression is denoted by $u(\boldsymbol{x};\mathcal{T},\boldsymbol{e},\boldsymbol{\theta})$ as a function in $\boldsymbol{x}$. Given a fixed $\mathcal{T}$, then the maximal number of operators is upper bounded by a constant denoted as $k_{\mathcal{T}}$. In FEX, $\mathbb{S}_{k_{\mathcal{T}}} = \{u(\boldsymbol{x};\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}) | \boldsymbol{e},\boldsymbol{\theta}\}$ is the function space in the CO to solve a PDE, which is also a subset of functions with at most $k_{\mathcal{T}}$-finite expressions.

The configuration of the tree of various depths can be designed as in Fig. 3. Each tree node is either a binary operator or a unary operator that takes value from the corresponding binary or unary set. The binary set can be $\mathbb{B} := \{+,-,\times,\div,\cdots\}$. The unary set can be $\mathbb{U} := \{\sin,\exp,\log,\mathrm{Id},(\cdot)^2,\int \cdot \mathrm{d}x_i, \frac{\partial \cdot}{\partial x_i},\cdots\}$, which contains elementary functions (e.g., polynomial and trigonometric function) and integral or differentiation operators. Here "Id" denotes the identity map. Notice that if an integration or a derivative is used in the expression, the operator can be applied numerically. Each entry of the operator sequence $\boldsymbol{e}$ is one-to-one associated with the tree traversal. The length of $\boldsymbol{e}$ is the total number of tree nodes. For example, Fig. 2b shows a tree with 4 nodes and $\boldsymbol{e} = (\mathrm{Id},\times,\exp,\sin)$.

The computation flow of the binary tree is conducted recursively. The operator of a node is granted higher precedence than that of its parent node. First, as in Fig. 3, we present the computation flow of the basic trees (a

tree has a depth of 1 with only 1 operator). For a basic tree with a unary operator u_1, when the input is i_0, then the output $o_1 = \alpha u_1(i_0) + \beta$, where α and β are scaling and bias parameters, respectively. For a basic tree with a binary operator b_1, when the input is i_{01} and i_{02}, the output becomes $o_1 = b_1(i_{01}, i_{02})$. With these two basic trees, we are ready to define the computation for arbitrary depth by recursion, as the examples shown in Fig. 3. Specifically, the input of a parent node is the output of the child node(s). In our implementation, the tree input at the bottom layer is a d-dimensional variable $\boldsymbol{x}$ (Fig. 2b). The unary operator directly linked to $\boldsymbol{x}$ is applied element-wisely to each entry of $\boldsymbol{x}$, and then the scaling α becomes a d-dimensional vector used for a linear transformation from $\mathbb{R}^d$ to $\mathbb{R}^1$.

3.2 Implementation of FEX

Given a tree $\mathcal{T}$, we aim to seek the PDE solution from the function space $\mathbb{S}_{k_\mathcal{T}}$. The mathematical expression can be identified via the minimization of the associated functional $\mathcal{L}$, i.e.,

$$\min\{\mathcal{L}(u(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}))|\boldsymbol{e},\boldsymbol{\theta}\}. \tag{2}$$

We introduce the framework for implementing FEX, as displayed in Fig. 2a, to seek a minimizer of (2). The basic idea is to find a good operator sequence $\boldsymbol{e}$ that may uncover the structure of the true solution, and then optimize the parameter $\boldsymbol{\theta}$ to minimize the functional (2). In our framework, the searching loop consists of four parts (see Sec. 6 for details): 1) Score computation (i.e., rewards in RL). A mix-order optimization algorithm is proposed to efficiently assess the score of the operator sequence $\boldsymbol{e}$ to uncover the true structure. A higher score suggests a higher possibility to help to identify the true solution. 2) Operator sequence generation (i.e., taking actions in RL). A controller is proposed to generate operator sequences with high scores (see Fig. 2b). 3) Controller update (i.e., policy optimization in RL). The controller is updated to increase the probability of producing a good operator sequence via the score feedback of the generated ones. While the controller can be modeled in many ways (e.g., heuristic algorithm), we introduce the policy gradient in RL to optimize the controller. 4) Candidate optimization (i.e., a non-greedy strategy). During searching, we maintain a candidate pool to store the operator sequence with a high score. After searching, the parameters $\boldsymbol{\theta}$ of high-score operator sequences are optimized to approximate the PDE solution.

4 Numerical Examples

Numerical results will be provided to demonstrate the effectiveness of our FEX implementation introduced in Sec. 3.2 using two classical PDE problems: high-dimensional PDEs with constraints (such as Dirichlet boundary conditions and integration constraints) and eigenvalue problems. The computational tools for high-dimensional problems are very limited and NNs are probably the most popular ones. Therefore, FEX will be compared with NN-based solvers. Through our examples, the goal is to numerically demonstrate that:

- **Accuracy.** FEX can achieve high and even machine accuracy for high-dimensional problems, while NN-based solvers can only achieve the accuracy of $\mathcal{O}(10^{-4})$ to $\mathcal{O}(10^{-2})$.
- **Scalability.** FEX is scalable in the problem dimension with an almost constant accuracy and a low memory requirement, i.e., the accuracy of FEX remains essentially the same when the dimension grows, while NN-based solvers have a worse accuracy when the dimension becomes larger.
- **Interpretability.** FEX provides interpretable insights of the ground truth PDE solution and helps to design postprocessing techniques for a refined solution.

In particular, to show the benefit of interpretability, we will provide examples to show that the explicit formulas of FEX solutions help to design better NN-parametrization in NN-based solvers to achieve higher accuracy. The FEX-aided NN-based solvers are referred to as FEX-NN in this paper. Finally, we will show the convergence of FEX with the growth of the tree size when the true solution can not be exactly reproduced by finite expressions using the available operators and a binary tree. All results of this section are obtained with 6 independent experiments to achieve their statistics.

4.1 Experimental Setting

This part provides the setting of FEX and NN-based solvers. Specific hyper-parameters for each experiment are reported in Sec. 6.5. The accuracy of a numerical solution $\tilde{u}$ compared with the true solution u is measured by a *relative* L^2 *error*, i.e., $\|\tilde{u}-u\|_{L^2(\Omega)}/\|u\|_{L^2(\Omega)}$. The integral in the L^2 norm is estimated by the Monte Carlo integral for high-dimensional problems.

Implements of FEX. The depth-3 binary tree (Fig. 2b) with 3 unary operators and 1 binary operator is used to generate mathematical expressions. The binary set is $\mathbb{B} = \{+, -, \times\}$ and the unary set is $\mathbb{U} = \{0, 1, \mathrm{Id}, (\cdot)^2, (\cdot)^3, (\cdot)^4, \exp, \sin, \cos\}$. A fully connected NN is used as a controller $\boldsymbol{\chi}_\Phi$ with constant input. The output size of the controller NN is $n_1|\mathbb{B}| + n_2|\mathbb{U}|$, where $n_1 = 1$ and $n_2 = 3$ represent the number of binary and unary operators, respectively, and $|\cdot|$ denotes the cardinality of a set.

Implements of NN-based Solvers. Residual networks (ResNets)[16,41] $u(\boldsymbol{x}; \boldsymbol{\Theta})$ parameterized by $\boldsymbol{\Theta}$ are used to approximate a solution and a minimization problem $\min_{\boldsymbol{\Theta}} \mathcal{L}(u(\cdot; \boldsymbol{\Theta}))$ is solved to identify a numerical solution[7,15,16].

4.2 High-dimensional PDEs

Several numerical examples for high-dimensional PDEs including linear and nonlinear cases are provided here. In these tests, the true solutions of PDEs have explicit formulas that can be reproduced by the binary tree defined in Sec. 3.1 and $\mathbb{B}, \mathbb{U}$ defined in Sec. 4.1.

4.2.1 Poisson Equation

We consider a Poisson equation[16] with a Dirichlet boundary condition on a d-dimensional domain $\Omega = [-1, 1]^d$,

$$-\Delta u = f \text{ for } \boldsymbol{x} \in \Omega, \quad u = g \text{ for } \boldsymbol{x} \in \partial\Omega. \tag{3}$$

Let the true solution be $\frac{1}{2}\sum_{i=1}^d x_i^2$, and then f becomes a constant function $-d$. The functional $\mathcal{L}$ is defined by least-square error (21) is applied in the NN-based solver and FEX to seek the PDE solution for various dimensions ($d = 10$, 20, 30, 40 and 50).

4.2.2 Linear Conservation Law

The linear conservation law[42] is considered here with a domain $T \times \Omega = [0, 1] \times [-1, 1]^d$ and an initial value problem

$$\frac{\pi d}{4} u_t - \sum_{i=1}^d u_{x_i} = 0 \text{ for } \boldsymbol{x} = (x_1, \cdots, x_d) \in \Omega, t \in [0, 1], \quad u(0, \boldsymbol{x}) = \sin(\frac{\pi}{4}\sum_{i=1}^d x_i) \text{ for } \boldsymbol{x} \in \Omega, \tag{4}$$

where the true solution is $u(t, \boldsymbol{x}) = \sin(t + \frac{\pi}{4}\sum_{i=1}^d x_i)$, and $d = 5$, 11, 17, 23, 29, 35, 41, 47 and 53 in our tests. The functional $\mathcal{L}$ in the NN-based solver and FEX to identify the solution is defined by

$$\mathcal{L}(u) := \|u_t - \sum_{i=1}^d u_{x_i}\|^2_{L^2(T\times\Omega)} + \lambda\|u(0, \boldsymbol{x}) - \sin(\frac{\pi}{4}\sum_{i=1}^d x_i)\|^2_{L^2(\Omega)}. \tag{5}$$

4.2.3 Nonlinear Schrödinger Equation

We consider a nonlinear Schrödinger equation[43] with a cubic term on a d-dimensional domain $\Omega = [-1, 1]^d$,

$$-\Delta u + u^3 + Vu = 0 \text{ for } \boldsymbol{x} \in \Omega, \tag{6}$$

where $V(\boldsymbol{x}) = -\frac{1}{9}\exp(\frac{2}{d}\sum_{i=1}^d \cos x_i) + \sum_{i=1}^d (\frac{\sin^2 x_i}{d^2} - \frac{\cos x_i}{d})$ for $\boldsymbol{x} = (x_1, \cdots, x_d)$. $\hat{u}(\boldsymbol{x}) = \exp(\frac{1}{d}\sum_{j=1}^d \cos(x_j))/3$ is the solution of the PDE (6). To avoid the trivial zero solution, we apply different strategies during the score computation and candidate optimization phases. During the score computation phase, the norm of the test function u, i.e., $\|u\|_{L_2(\Omega)}$, is used as a penalty for the function that is close to zero by

$$\mathcal{L}_1(u) := \|-\Delta u + u^3 + Vu\|^2_{L_2(\Omega)} / \|u\|^3_{L_2(\Omega)}. \tag{7}$$

During the candidate optimization phase, an integration constraint is imposed to (6), i.e., $\int_\Omega u(\boldsymbol{x})d\boldsymbol{x} = \int_\Omega \hat{u}(\boldsymbol{x})d\boldsymbol{x}$. The functional $\mathcal{L}$ in the NN-based solver and FEX to identify the solution is defined by

$$\mathcal{L}_2(u) := \|-\Delta u + u^3 + Vu\|^2_{L_2(\Omega)} + \lambda\Big(\int_\Omega u(\boldsymbol{x})d\boldsymbol{x} - \int_\Omega \hat{u}(\boldsymbol{x})d\boldsymbol{x}\Big)^2, \tag{8}$$

where the second term imposes the integration constraint. Various dimensions are tested in the numerical results, e.g., $d = 6$, 12, 18, 24, 30, 36, 42 and 48.

4.3 Results

Three main sets of numerical results for the PDE problems above will be presented. First, the errors of the numerical solutions by NN-based solvers and FEX are compared. Second, a convergence test is analyzed when the tree size of FEX increases. Finally, FEX is applied to design special NN parametrization to solve PDEs in NN-based solvers.

Estimated Solution Error. The depth-3 binary tree (Fig. 3) is used in FEX with four nodes (a root node (R), a middle node (M), and two leave nodes (L1 and L2)). Fig. 4 shows the operator distribution obtained by FEX and the error comparison between the NN method and our FEX. The results show that NN solutions have numerical errors between $\mathcal{O}(10^{-4})$ and $\mathcal{O}(10^{-2})$ and the errors grow in the problem dimension d, agreeing with the numerical observation in the literature. Meanwhile, FEX can identify the true solution structure for the Poisson equation and the linear conservation law with errors of order 10^{-7}, reaching the machine accuracy since the single-float precision is used. In the results of the nonlinear Schrödinger equation, FEX identifies the solutions of the form $\exp(\cos(\cdot))$ but achieves errors of order 10^{-5}. Note that $\int_\Omega \hat{u}(\boldsymbol{x})d\boldsymbol{x}$ in (8) is estimated by the Monte-Carol integration with millions of points as an accurate and precomputed constant, but $\int_\Omega u(\boldsymbol{x})d\boldsymbol{x}$ can only be estimated with fixed and small batch size, typically less than 10,000, in the optimization iterations. As the dimension grows, the estimation error of $\int_\Omega u(\boldsymbol{x})d\boldsymbol{x}$ increases, and, hence, even the ground true solution has an increasingly large error according to (8). Therefore, the optimization solver may return an approximate solution without machine accuracy. Designing a functional $\mathcal{L}$ free of the Monte-Carol error (e.g., Eqn. (21) and (5)) for the nonlinear Schödinger equation could ensure machine accuracy.

Numerical Convergence. The numerical convergence analysis is performed using the Poisson equation as an example. Binary trees of depths 2, 3, 4, 6 and 8 are used (see Fig. 3). The square operator $(\cdot)^2$ is excluded in $\mathbb{U}$ so that the binary tree defined in Sec. 3.1 can not reproduce the true solution (sum of the square of coordinates) exactly. This setting can mimic the case of a complicated solution while a small binary tree was used. Fig. 5 shows the error distribution of FEX with the growth of dimensions and the change of tree depths. FEX obtains smaller errors with increasing tree size. Notice that, compared with the errors of NN-based solvers reported in Fig. 4, FEX gets a higher accuracy when a larger tree is used.

FEX-NN. FEX provides interpretable insights of the ground truth PDE solution by the operator distribution obtained from the searching loop. It may be beneficial to design NN models with a special structure to increase the accuracy of NN-based solvers. In the results of the Poisson equation, we observe that the square operator has a high probability to appear at the leave nodes, which suggests that the true solution may contain the structure $\boldsymbol{x}^2 := (x_1^2, \cdots, x_d^2)$ at the input $\boldsymbol{x}$. As a result, we define the FEX-NN by $v(\boldsymbol{x}^2; \boldsymbol{\Theta})$ for the Poisson equation. Similarly, we use FEX-NNs $\sin(v(\boldsymbol{x}; \boldsymbol{\Theta}))$ for the linear conservation law and $\exp(v(\boldsymbol{x}^2; \boldsymbol{\Theta}))$ for the nonlinear Schrödinger equation. Fig. 4 shows the errors of FEX-NN with the growth of dimensions, and it is clear that FEX-NN outperforms the vanilla NN-based method by a significant margin.

4.4 Eigenvalue Problem

Consider identifying the smallest eigenvalue γ and the associated eigenfunction u of the eigenvalue problem[16],

$$-\Delta u + w \cdot u = \gamma u, \quad \boldsymbol{x} \in \Omega, \quad \text{for } u = 0 \text{ on } \partial\Omega. \tag{9}$$

The minimization of the Rayleigh quotient $\mathcal{I}(u) = \frac{\int_\Omega \|\nabla u\|_2^2 dx + \int_\Omega w \cdot u^2 dx}{\int_\Omega u^2 dx}$, s.t., $u|_{\partial\Omega} = 0$, gives the smallest eigenvalue and the corresponding eigenfunction. In the NN-based solver[16], the following functional is defined

$$\mathcal{L}(u) := \mathcal{I}(u) + \lambda_1 \int_{\partial\Omega} u^2 dx + \lambda_2 \Big(\int_\Omega u^2 dx - 1 \Big)^2 \tag{10}$$

to seek an NN solution. Considering an example of $w = \|\boldsymbol{x}\|_2^2$ and $\Omega = \mathbb{R}^d$, the smallest eigenvalue of (9) is d and the associated eigenfunction is $\exp(-\frac{\|\boldsymbol{x}\|_2^2}{0.5})$. The domain Ω is truncated from $\mathbb{R}^d$ to $[-3,3]^d$ for simplification[16].

In FEX, the functional (10) is also used to estimate a solution. FEX discovers a high probability to have the "exp" operator at the tree root (100% for $d = 2$, 4, 6, 8, and 93.3% for $d = 10$ as shown in Fig. 6). Therefore, it is reasonable to assume that the eigenfunction is of the form $\exp(v(\boldsymbol{x}))$. Let $u(\boldsymbol{x})$ be $\exp(v(\boldsymbol{x}))$ and then Eqn. (9) is simplified to

$$-\Delta v - \|\nabla v\|_2^2 + \|\boldsymbol{x}\|_2^2 = \gamma. \tag{11}$$

Eqn. 11 does not have a trivial zero solution so we can avoid the integration constraint used in Eqns. (8) and (10), which leads to Monte-Carol errors. Utilizing Eqn. (11) and the Rayleigh quotient $\mathcal{I}$, v and γ are alternatively updated

until convergence. The detail of this iterative algorithm is presented in Appx. 7.3. Fig. 6 shows the relative absolute error of the estimated eigenvalues with the growth of the dimensions. We can see that directly optimizing (10) with the NN method produces a large error on the eigenvalue estimation, especially when the dimension is high (e.g., the relative error is up to 20% when $d = 10$). With the postprocessing algorithm with FEX, we can identify the eigenvalue with an error close to zero.

5 Discussion

This paper proposed the finite expression method, a methodology to find a PDE solution in the form of a simple mathematical expression. Our theory shows that mathematical expression can overcome the curse of dimensionality. We provided one implementation of FEX based on reinforcement learning. Our results demonstrated the effectiveness of FEX to achieve a solution with high and even machine accuracy for solving high-dimensional PDEs, while existing solvers suffer from low accuracy.

6 Methods

We present the pseudo-code of FEX in Alg. 1 for searching for a solution with a fixed tree. Algorithm 2 shows the process of searching the solution from an expanding tree set $\{\mathcal{T}_1, \mathcal{T}_2, \cdots\}$ (e.g., the trees grow with depth). For the reader's convenience, we summarize the key notations used in this section in Table 1.

Notation	Explanation
$\mathcal{T}$	a binary tree
$\boldsymbol{e}$	operator sequence
$\boldsymbol{\theta}$	trainable scaling and bias parameters
$\mathcal{L}$	the functional associated with the PDE solution
S	the score function mapping from a operator sequence to $[0,1]$
$\boldsymbol{\chi}_\Phi$	the controller parameterized by Φ
$\mathcal{J}$	the objective function for the policy-gradient approach

Table 1. A summary of notations in the FEX implementation.

6.1 Score Computation

The score of an operator sequence $\boldsymbol{e}$ is critical to guide the controller toward generating good operator sequences and help to maintain a candidate pool of high scores. Intuitively, the score of $\boldsymbol{e}$ is defined in the range $[0,1]$, namely $S(\boldsymbol{e})$, by

$$S(\boldsymbol{e}) := \big(1 + L(\boldsymbol{e})\big)^{-1}, \tag{12}$$

where $L(\boldsymbol{e}) := \min\{\mathcal{L}(u(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}))|\boldsymbol{\theta}\})$. When $L(\boldsymbol{e})$ tends to 0, the expression represented by $\boldsymbol{e}$ is close to the true solution, and the score $S(\boldsymbol{e})$ goes to 1. Otherwise, $S(\boldsymbol{e})$ goes to 0. The global minimizer of $\mathcal{L}(u(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}))$ over $\boldsymbol{\theta}$ is difficult and expensive to obtain. Instead of exhaustively searching for a global minimizer, a first-order optimization algorithm and a second-order one are combined to accelerate the evaluation of $S(\boldsymbol{e})$.

First-order algorithms (e.g., the stochastic gradient descent[44] and Adam[45]) that utilize gradient to update are popular in machine learning. They usually require a small learning rate and a large number of iterations for convergence. It may be time-consuming to optimize $L(\boldsymbol{e})$ using a first-order algorithm. Alternatively, second-order algorithms (e.g., the Newton method[46] and the Broyden-Fletcher-Goldfarb-Shanno method (BFGS)[47]) use the Hessian matrix for faster convergence, but obtaining a good minimizer requires a good initial guess. To reduce the time of optimizing $L(\boldsymbol{e})$ in our implementation, a first-order algorithm is used for T_1 steps to obtain a good initial guess and then a second-order algorithm is applied for T_2 steps. Let $\boldsymbol{\theta}_0^{\boldsymbol{e}}$ be an initialization and $\boldsymbol{\theta}_{T_1+T_2}^{\boldsymbol{e}}$ be the parameter set after $T_1 + T_2$ steps of this hybrid optimization. Then $\boldsymbol{\theta}_{T_1+T_2}^{\boldsymbol{e}}$ serves as an approximate $\arg\min_{\boldsymbol{\theta}} \mathcal{L}(u(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}))$. Finally, $S(\boldsymbol{e})$ is estimated by

$$S(\boldsymbol{e}) \approx \big(1 + \mathcal{L}(u(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}_{T_1+T_2}^{\boldsymbol{e}}))\big)^{-1}. \tag{13}$$

6.2 Operator Sequence Generation

The role of the controller is to generate operator sequences with high scores during the searching loop. Let $\boldsymbol{\chi}_\Phi$ be a controller with model parameter Φ, and Φ is updated to increase the probability for good operator sequences during the searching loop. $\boldsymbol{e} \sim \boldsymbol{\chi}_\Phi$ is used to denote the process to sample an $\boldsymbol{e}$ according to the controller $\boldsymbol{\chi}_\Phi$.

Treating tree node values of $\mathcal{T}$ as random variables, the controller $\boldsymbol{\chi}_\Phi$ outputs probability mass functions $\boldsymbol{p}_\Phi^1, \boldsymbol{p}_\Phi^2, \cdots, \boldsymbol{p}_\Phi^s$ to characterize their distributions, where s is the total number of nodes. Each tree node value e_j is sampled from $\boldsymbol{p}_\Phi^j$ to obtain an operator. Then $\boldsymbol{e} := (e_1, e_2, \cdots, e_s)$ is the operator sequence sampled from $\boldsymbol{\chi}_\Phi$. See Fig. 2b for an illustration. Besides, we adopt the ϵ-greedy strategy[48] to enhance exploration of a potentially high-score $\boldsymbol{e}$. With probability $\epsilon < 1$, e_i is sampled from a uniform distribution of the operator set. With probability $1-\epsilon$, $e_i \sim \boldsymbol{p}_\Phi^i$. A larger ϵ leads to a higher probability to explore new sequences.

6.3 Controller Update

The goal of the controller update is to guide the controller toward generating high-score operator sequences $\boldsymbol{e}$. The updating rule of a controller can be designed based on heuristics (e.g., genetic and simulated annealing algorithms) and gradient-based methods (e.g., policy gradient and darts[49]). As proof of concept, we introduce a policy-gradient-based updating rule in RL. The policy gradient method aims to maximize the return by optimizing a parameterized policy and the controller in our problem plays the role of a policy.

In this paper, the controller $\boldsymbol{\chi}_\Phi$ is modeled as a neural network parameterized by Φ. The training objective of the controller is to maximize the expected score of a sampled $\boldsymbol{e}$, i.e.,

$$\mathcal{J}(\Phi) := \mathbb{E}_{\boldsymbol{e}\sim\boldsymbol{\chi}_\Phi} S(\boldsymbol{e}). \tag{14}$$

Taking the derivative of (14) with respect to Φ, we have

$$\nabla_\Phi \mathcal{J}(\Phi) = \mathbb{E}_{\boldsymbol{e}\sim\boldsymbol{\chi}_\Phi}\big\{S(\boldsymbol{e})\sum_{i=1}^{s}\nabla_\Phi \log(\boldsymbol{p}_\Phi^i(e_i))\big\}, \tag{15}$$

where $\boldsymbol{p}_\Phi^i(e_i)$ is the probability corresponding to the sampled e_i. When the batch size is N and $\{\boldsymbol{e}^{(1)}, \boldsymbol{e}^{(2)}, \cdots, \boldsymbol{e}^{(N)}\}$ are sampled under $\boldsymbol{\chi}_\Phi$ each time, the expectation can be approximated by

$$\nabla_\Phi \mathcal{J}(\Phi) \approx \frac{1}{N}\sum_{k=1}^{N}\big\{S(\boldsymbol{e}^{(k)})\sum_{i=1}^{s}\nabla_\Phi \log(\boldsymbol{p}_\Phi^i(e_i^{(k)}))\big\}. \tag{16}$$

Next, the model parameter Φ is updated via the gradient ascent with a learning rate η, i.e.,

$$\Phi \leftarrow \Phi + \eta\nabla_\Phi \mathcal{J}(\Phi). \tag{17}$$

The objective in (14) helps to improve the average score of generated sequences. In our problem, the goal is to find $\boldsymbol{e}$ with the best score. To increase the probability of obtaining the best case, the objective function proposed in[34] is applied to seek the optimal solution via

$$\mathcal{J}(\Phi) = \mathbb{E}_{\boldsymbol{e}\sim\boldsymbol{\chi}_\Phi}\{S(\boldsymbol{e})|S(\boldsymbol{e}) \geq S_{\nu,\Phi}\}, \tag{18}$$

where $S_{\nu,\Phi}$ represents the $(1-\nu)\times 100\%$-quantile of the score distribution generated by $\boldsymbol{\chi}_\Phi$. In a discrete form, the gradient computation becomes

$$\nabla_\Phi \mathcal{J}(\Phi) \approx \frac{1}{N}\sum_{k=1}^{N}\big\{(S(\boldsymbol{e}^{(k)}) - \hat{S}_{\nu,\Phi})\mathbb{1}_{\{S(\boldsymbol{e}^{(k)})\geq\hat{S}_{\nu,\Phi}\}}\sum_{i=1}^{s}\nabla_\Phi \log(\boldsymbol{p}_\Phi^i(e_i^{(k)}))\big\}. \tag{19}$$

where $\mathbb{1}$ is an indicator function that takes value 1 if the condition is true otherwise 0, and $\hat{S}_{\nu,\Phi}$ is the $(1-\nu)$-quantile of the scores $\{S(\boldsymbol{e}^{(i)})\}_{i=1}^N$.

6.4 Candidate Optimization

As introduced in Sec. 6.1, the score of $\boldsymbol{e}$ is based on the optimization of a nonconvex function at a random initialization. Therefore, the optimization may get stuck at poor local minimizers and the score sometimes may not reflect whether $\boldsymbol{e}$ reveals the structure of the true solution. The operator sequence $\boldsymbol{e}$ corresponding to the true

solution (or approximately) may not have the best score. For the purpose of not missing good operator sequences, a candidate pool $\mathbb{P}$ with capacity K is maintained to store several $\boldsymbol{e}$'s of a high score.

During the search loop, if the size of $\mathbb{P}$ is less than the capacity K, $\boldsymbol{e}$ will be put in $\mathbb{P}$. If the size of $\mathbb{P}$ reaches K and $S(\boldsymbol{e})$ is larger than the smallest score in $\mathbb{P}$, then $\boldsymbol{e}$ will be appended to $\mathbb{P}$ and the one with the least score will be removed. After the searching loop, for every $\boldsymbol{e} \in \mathbb{P}$, the objective function $\mathcal{L}(u(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}))$ is optimized over $\boldsymbol{\theta}$ using a first-order algorithm with a small learning rate for T_3 iterations.

6.5 Experimental Details

This part provides the setting of FEX and NN-based solvers. Specific hyper-parameters for each experiment are reported in Appx. 6.5.

Implements of FEX. There are four main parts in the implementation of FEX as introduced in Section 3.2. We will only briefly describe the key numerical choices here. (1) *Score computation.* The score is updated first by Adam with a learning rate 0.001 for $T_1 = 20$ iterations and then by BFGS with a learning rate 1 for maximum $T_2 = 20$ iterations. (2) *Operator sequence generation.* The depth-3 binary tree (Fig. 2b) with 3 unary operators and 1 binary operator is used to generate mathematical expressions. The binary set is $\mathbb{B} = \{+,-,\times\}$ and the unary set is $\mathbb{U} = \{0,1,\mathrm{Id},(\cdot)^2,(\cdot)^3,(\cdot)^4,\exp,\sin,\cos\}$. A fully connected NN is used as a controller $\boldsymbol{\chi}_\Phi$ with constant input. The output size of the controller NN is $n_1|\mathbb{B}|+n_2|\mathbb{U}|$, where $n_1 = 1$ and $n_2 = 3$ represent the number of binary and unary operators, respectively, and $|\cdot|$ denotes the cardinality of a set. (3) *Controller update.* The batch size for the policy gradient update is $N = 10$ and the controller is trained for 1000 iterations using Adam with a fixed learning rate 0.002. We adopt the ϵ-greedy strategy to increase the exploration of new $\boldsymbol{e}$. The probability ϵ of sampling an e_i by random is 0.1. (4) *Weight optimization.* The candidate pool capacity is set to be $K = 10$. For any $\boldsymbol{e} \in \mathbb{P}$, the parameter $\boldsymbol{\theta}$ is optimized using Adam with an initial learning rate 0.01 for $T_3 = 20{,}000$ iterations. The learning rate decays according to the cosine decay schedule[50].

Implements of NN-based Solvers. The ResNet comprises 7 fully connected layers and 3 skip connections. The skip connection is placed between every two layers. The number of neurons in each hidden layer is 50. The NN is optimized by Adam with an initial learning rate 0.001 for 15,000 iterations. The learning rate decays following the cosine decay schedule.

Poisson equation. The coefficient λ in the functional (21) is 100. The batch size for the interior and boundary is 5,000 and 1,000, respectively. In the NN method, we use a ResNet with ReLU^2, i.e., $(\max\{x,0\})^2$, to approximate the solution. Especially in Numerical Convergence, since the deeper trees are used, the controller is updated for 5000 iterations for trees with different depth.

Linear conservative law. The coefficient λ in the functional (5) is 100. In the NN method, we use ReLU ($\max\{x,0\}$) activation. The batch size for the interior and boundary is 5,000 and 1,000, respectively. We use the same batch size as the NN method except that we increase the batch size to 20,000 for the interior and 4,000 for the boundary when the dimension is not smaller than 36.

Schrödinger equation. The coefficient λ in the functional (8) is 1. The batch size for estimating the first term and second term of (8) is 2,000 and 10,000, respectively. In the NN method, ReLU^2 is used in ResNet.

Data and Code Availability

Source codes for reproducing the results in this paper are available at the online data warehouse: https://github.com/LeungSamWai/Finite-expression-method. The source codes are released under MIT license.

References

1. Philibert, J. One and a half century of diffusion: Fick, einstein. *Diffusion Fundamentals: Leipzig 2005* **1**, 8 (2005).
2. Kirkwood, J. G., Baldwin, R. L., Dunlop, P. J., Gosting, L. J. & Kegeles, G. Flow equations and frames of reference for isothermal diffusion in liquids. *The J. Chem. Phys.* **33**, 1505–1513 (1960).
3. Acheson, D. J. Elementary fluid dynamics (1991).
4. Shinbrot, M. *Lectures on fluid mechanics* (Courier Corporation, 2012).
5. Feynman, R. P., Leighton, R. B. & Sands, M. The feynman lectures on physics; vol. i. *Am. J. Phys.* **33**, 750–752 (1965).
6. Landau, L. D. & Lifshitz, E. M. *Quantum mechanics: non-relativistic theory*, vol. 3 (Elsevier, 2013).

7. Weinan, E., Han, J. & Jentzen, A. Algorithms for solving high dimensional pdes: From nonlinear monte carlo to machine learning. *Nonlinearity* **35**, 278 (2021).

8. Reddy, J. *An introduction to the finite element method*, vol. 1221 (McGraw-Hill New York, 2004).

9. Grossmann, C., Roos, H.-G. & Stynes, M. *Numerical treatment of partial differential equations*, vol. 154 (Springer, 2007).

10. Li, F., Villa, U., Park, S. & Anastasio, M. A. 3-d stochastic numerical breast phantoms for enabling virtual imaging trials of ultrasound computed tomography. *IEEE Transactions on Ultrason. Ferroelectr. Freq. Control.* **69**, 135–146, DOI: 10.1109/TUFFC.2021.3112544 (2022).

11. E, W., Han, J. & Jentzen, A. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Stat.* **5**, 349–380, DOI: 10.1007/s40304-017-0117-6 (2017).

12. Han, J., Jentzen, A. & E, W. Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci.* **115**, 8505–8510 (2018).

13. Khoo, Y., Lu, J. & Ying, L. Solving parametric pde problems with artificial neural networks. *arXiv: Numer. Analysis* (2017).

14. Raissi, M., Perdikaris, P. & Karniadakis, G. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *J. Comput. Phys.* **378**, 686 – 707, DOI: https://doi.org/10.1016/j.jcp.2018.10.045 (2019).

15. Sirignano, J. & Spiliopoulos, K. Dgm: A deep learning algorithm for solving partial differential equations. *J. computational physics* **375**, 1339–1364 (2018).

16. E, W. & Yu, B. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.* **6**, 1–12 (2018).

17. Han, J. & Weinan, E. Deep learning approximation for stochastic control problems. *ArXiv* **abs/1611.07422** (2016).

18. Shen, Z., Yang, H. & Zhang, S. Deep network with approximation error being reciprocal of width to power of square root of depth. *Neural Comput.* (2021).

19. Shen, Z., Yang, H. & Zhang, S. Neural network approximation: Three hidden layers are enough. *arXiv:2010.14075* (2020).

20. Yarotsky, D. Elementary superexpressive activations. *ArXiv* **abs/2102.10911** (2021).

21. Shen, Z., Yang, H. & Zhang, S. Deep network approximation: Achieving arbitrary accuracy with fixed number of neurons. *ArXiv* **abs/2107.02397** (2021).

22. Jiao, Y. *et al.* Deep neural networks with relu-sine-exponential activations break curse of dimensionality on h\" older class. *arXiv preprint arXiv:2103.00542* (2021).

23. Xu, Z.-Q. J., Zhang, Y., Luo, T., Xiao, Y. & Ma, Z. Frequency principle: Fourier analysis sheds light on deep neural networks. *Commun. Comput. Phys.* **28**, 1746–1767, DOI: https://doi.org/10.4208/cicp.OA-2020-0085 (2020).

24. Cao, Y., Fang, Z., Wu, Y., Zhou, D.-X. & Gu, Q. Towards understanding the spectral bias of deep learning. *arXiv preprint arXiv:1912.01198* (2019).

25. Ronen, B., Jacobs, D., Kasten, Y. & Kritchman, S. The convergence rate of neural networks for learned functions of different frequencies. *Adv. Neural Inf. Process. Syst.* **32** (2019).

26. Liu, Z., Cai, W. & Xu, Z.-Q. J. Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains. *Commun. Comput. Phys.* **28**, 1970–2001, DOI: 10.4208/cicp.oa-2020-0179 (2020).

27. Bianco, S., Cadene, R., Celona, L. & Napoletano, P. Benchmark analysis of representative deep neural network architectures. *IEEE Access* **6**, 64270–64277 (2018).

28. Murray-Smith, D. J. *Modelling and simulation of integrated systems in engineering: issues of methodology, quality, testing and application* (Elsevier, 2012).

29. Bello, I., Pham, H., Le, Q. V., Norouzi, M. & Bengio, S. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940* (2016).

30. Mazyavkina, N., Sviridov, S., Ivanov, S. & Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Comput. & Oper. Res.* **134**, 105400 (2021).

31. Ramachandran, P., Zoph, B. & Le, Q. V. Searching for activation functions (2018).

32. Bello, I., Zoph, B., Vasudevan, V. & Le, Q. V. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, 459–468 (PMLR, 2017).

33. Co-Reyes, J. D. *et al.* Evolving reinforcement learning algorithms. In *International Conference on Learning Representations* (2021).

34. Petersen, B. K. *et al.* Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations* (2021).

35. Landajuela, M. *et al.* Discovering symbolic policies with deep reinforcement learning. In Meila, M. & Zhang, T. (eds.) *Proceedings of the 38th International Conference on Machine Learning*, vol. 139 of *Proceedings of Machine Learning Research*, 5979–5989 (PMLR, 2021).

36. Lagaris, I. E., Likas, A. & Fotiadis, D. I. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks* **9**, 987–1000 (1998).

37. Dissanayake, M. W. M. G. & Phan-Thien, N. Neural-network-based Approximations for Solving Partial Differential Equations. *Comm. Numer. Methods Engrg.* **10**, 195–201 (1994).

38. Yulei *et al.* Deep nitsche method: Deep ritz method with essential boundary conditions. *Commun. Comput. Phys.* **29**, 1365–1384, DOI: https://doi.org/10.4208/cicp.OA-2020-0219 (2021).

39. Chen, F., Huang, J., Wang, C. & Yang, H. Friedrichs learning: Weak solutions of partial differential equations via deep learning. *arXiv preprint arXiv:2012.08023* (2020).

40. Zang, Y., Bao, G., Ye, X. & Zhou, H. Weak adversarial networks for high-dimensional partial differential equations. *J. Comput. Phys.* **411**, 109409 (2020).

41. He, K., Zhang, X., Ren, S. & Sun, J. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778 (2016).

42. Chen, J., Jin, S. & Lyu, L. A deep learning based discontinuous galerkin method for hyperbolic equations with discontinuous solutions and random uncertainties. *arXiv preprint arXiv:2107.01127* (2021).

43. Han, J., Lu, J. & Zhou, M. Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion monte carlo like approach. *J. Comput. Phys.* **423**, 109792 (2020).

44. Rumelhart, D. E., Hinton, G. E. & Williams, R. J. Learning representations by back-propagating errors. *nature* **323**, 533–536 (1986).

45. Kingma, D. P. & Ba, J. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

46. Avriel, M. *Nonlinear programming: analysis and methods* (Courier Corporation, 2003).

47. Fletcher, R. *Practical methods of optimization* (John Wiley & Sons, 2013).

48. Sutton, R. S. & Barto, A. G. *Reinforcement learning: An introduction* (MIT press, 2018).

49. Liu, H., Simonyan, K. & Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations* (2019).

50. He, T. *et al.* Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 558–567 (2019).

51. Evans, L. *Partial Differential Equations*. Graduate studies in mathematics (American Mathematical Society, 2010).

7 Appendix

7.1 Functionals for solving PDEs

The choice of functionals to solve a PDE is problem-dependent. We present three popular choices (a least-square method, a variation formulation, and a weak formulation) and discuss examples of the boundary value problems without loss of generality, which can be generalized to other kinds of PDEs in a similar manner.

7.1.1 Least Square Method

Suppose that the PDE is given by

$$\mathcal{D}u(\boldsymbol{x}) = f(u(\boldsymbol{x}),\boldsymbol{x}),\ \boldsymbol{x}\in\Omega, \qquad \mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}),\ \boldsymbol{x}\in\partial\Omega, \tag{20}$$

where $\mathcal{D}$ is a differential operator, $f(u(\boldsymbol{x}),\boldsymbol{x})$ can be a nonlinear function in u, Ω is a bounded domain in $\mathbb{R}^d$, and $\mathcal{B}u = g$ characterizes the boundary condition (e.g., Dirichlet, Neumann and Robin[51]). The least square method[15,36,37] defines a straightforward functional to characterize the error of the estimated solution by

$$\mathcal{L}(u) := \|\mathcal{D}u(\boldsymbol{x}) - f(u,\boldsymbol{x})\|^2_{L^2(\Omega)} + \lambda\|\mathcal{B}u(\boldsymbol{x}) - g(\boldsymbol{x})\|^2_{L^2(\partial\Omega)}, \tag{21}$$

where λ is a positive coefficient to enforce the boundary constraint.

7.1.2 Variation Formulation

We next introduce the variation formulation that is widely used in identifying a numerical solution[16,38]. Consider an example of an elliptic PDE with a homogeneous Dirichlet boundary condition. The PDE is given by

$$-\Delta u(\boldsymbol{x}) + c(\boldsymbol{x})u(\boldsymbol{x}) = f(\boldsymbol{x}),\ \boldsymbol{x}\in\Omega, \qquad u(\boldsymbol{x}) = 0,\ \boldsymbol{x}\in\partial\Omega, \tag{22}$$

where c is a bounded function and $f\in L^2$. The solution u of PDE (22) minimizes a variation formulation $\frac{1}{2}\int_\Omega \|\nabla u\|^2 + cu^2 \mathrm{d}\boldsymbol{x} - \int_\Omega fu\mathrm{d}\boldsymbol{x}$. By imposing the boundary condition penalty in the variation form, one may obtain the functional

$$\mathcal{L}(u) := \frac{1}{2}\int_\Omega \|\nabla u\|^2 + cu^2\mathrm{d}\boldsymbol{x} - \int_\Omega fu\mathrm{d}\boldsymbol{x} + \lambda\int_{\partial\Omega} u^2\mathrm{d}\boldsymbol{x}. \tag{23}$$

7.1.3 Weak Formulation

Another alternative method is to find the weak solution of the PDE based on the weak formulation. Let $v\in H^1_0(\Omega)$ be a test function, where $H^1_0(\Omega)$ denotes the Sobolev space whose weak derivative is L^2 integrable with zero boundary values. The weak solution u of Eqn. (22) is defined as the function that satisfies the bilinear equations:

$$\begin{aligned} a(u,v) :&= \int_\Omega \nabla u\nabla v + cuv - fv\mathrm{d}\boldsymbol{x} = 0, \quad \forall v\in H^1_0(\Omega),\\ u(\boldsymbol{x}) &= 0,\ \boldsymbol{x}\in\partial\Omega, \end{aligned} \tag{24}$$

where $a(u,v)$ is constructed by multiplying (22) and v, and integration by parts. One can transfer all the derivatives of the solution function to the test function by conducting integration by parts several times[39]. The weak solution is rewritten as the solution of a saddle-point problem[40],

$$\min_{u\in H^1_0(\Omega)}\max_{v\in H^1_0(\Omega)} |a(u,v)|^2/\|v\|^2_{L^2(\Omega)}. \tag{25}$$

We may define the functional $\mathcal{L}$ to identify the PDE solution by

$$\mathcal{L}(u) := \max_{v\in H^1_0(\Omega)} |a(u,v)|^2/\|v\|^2_{L^2(\Omega)} + \lambda\int_{\partial\Omega} u^2\mathrm{d}\boldsymbol{x}. \tag{26}$$

7.2 Proofs of Theorems

Proof of Thm. 2.1. By Thm. 1.1 of Shen et al (2021)[21], for any $f\in C([a,b]^d)$ as a continuous function on $[a,b]^d$, there exists a fully connected neural network (FNN) ϕ with width $N = 36d(2d+1)$ and depth $L = 11$ (i.e., 11 hidden layers) such that, for an arbitrary $\varepsilon > 0$, $\|\phi - f\|_{L^\infty([a,b]^d)} < \varepsilon$. This FNN is constructed via an activation function with an explicit formula $\sigma(x) = \sigma_1(x) := \left|x - 2\lfloor\frac{x+1}{2}\rfloor\right|$ for $x\in[0,\infty)$ and $\sigma(x) = \sigma_2(x) := \frac{x}{|x|+1}$ for $x\in(-\infty,0)$. Therefore, $\sigma(x) = \frac{\mathrm{sign}(x)+1}{2}\sigma_1(x) - \frac{\mathrm{sign}(x)-1}{2}\sigma_2(x)$. Hence, it requires at most 18 operators to evaluate $\sigma(x)$. For an FNN of width N and depth L, there are $N(d+1)+(L-1)N^2$ operators "×", $Nd-1+(L-1)N(N-1)$ operators "+", and NL evaluations of $\sigma(x)$ to evaluate an output of the FNN. Therefore, the FNN ϕ is a mathematical expression with at most $k_d := 103680d^4 + 103824d^3 + 39600d^2 + 6804d - 1 = \mathcal{O}(d^4)$ operators. Therefore, for any $\varepsilon > 0$, any continuous function f on $[a,b]^d$, there is a k_d-finite expression that can approximate f uniformly well on $[a,b]^d$ within ε accuracy. Since k_d is independent of ε, it is clear that the function space of k_d-finite expressions is dense in $C([a,b]^d)$. □

Proof of Thm. 2.2. By Cor. 3.8 of Jiao et al (2021)[22], let $p \in [1,+\infty)$, for any $f \in \mathcal{H}^{\alpha}_{\mu}([0,1]^d)$ and $\varepsilon > 0$, there exists an FNN ϕ with width $N = \max\{2d\left\lfloor \log_2\left(\sqrt{d}\left(\frac{3\mu}{\varepsilon}\right)^{1/\alpha}\right)\right\rfloor, 2\left\lfloor \log_2 \frac{3\mu d^{\alpha/2}}{2\varepsilon}\right\rfloor + 2\}$ and depth $L = 6$ such that $\|\phi - f\|_{L^p([0,1]^d)} < \varepsilon$. This FNN is constructed via activation functions chosen from the set $\{\sin(x), \max\{0,x\}, 2^x\}$. Similar to the proof for Thm. 2.1, there are $N(d+1)+(L-1)N^2$ operators "×", $Nd-1+(L-1)N(N-1)$ operators "+", and NL evaluations of activation functions to evaluate an output of the FNN. Therefore, the total number of operators in ϕ as a mathematical expression is $\mathcal{O}(d^2(\log d + \log\frac{1}{\varepsilon})^2)$, which completes the proof. □

Algorithm 1 FEX with a fixed tree

Input: PDE and the associated functional $\mathcal{L}$; A tree $\mathcal{T}$; Searching loop iteration T; Coarse-tune iteration T_1 with Adam; Coarse-tune iteration T_2 with BFGS; Fine-tune iteration T_3 with Adam; Pool size K; Batch size N.
Output: The solution $u(\boldsymbol{x}; \mathcal{T}, \hat{\boldsymbol{e}}, \hat{\boldsymbol{\theta}})$.

Initialize the agent $\boldsymbol{\chi}$ for the tree $\mathcal{T}$
$\mathbb{P} \leftarrow \{\}$
for _ from 1 to T **do**
  Sample N sequences $\{\boldsymbol{e}^{(1)}, \boldsymbol{e}^{(2)}, \cdots, \boldsymbol{e}^{(N)}\}$ from $\boldsymbol{\chi}$
  for n from 1 to N **do**
    Optimize $\mathcal{L}(u(\boldsymbol{x}; \mathcal{T}, \boldsymbol{e}^{(n)}, \boldsymbol{\theta}))$ by coarse-tune with $T_1 + T_2$ iterations.
    Compute the reward $R(\boldsymbol{e}^{(n)})$ of $\boldsymbol{e}^{(2)}$
    if $\boldsymbol{e}^{(n)}$ belongs to the top-K of S **then**
      $\mathbb{P}$.append($\boldsymbol{e}^{(n)}$)
      $\mathbb{P}$ pops some $\boldsymbol{e}$ with the smallest reward when overloading
    end if
  end for
  Update $\boldsymbol{\chi}$ using (19)
end for
for $\boldsymbol{e}$ in $\mathbb{P}$ **do**
  Fine-tune $\mathcal{L}(u(\boldsymbol{x}; \mathcal{T}, \boldsymbol{e}, \boldsymbol{\theta}))$ with T_3 iterations.
end for
return the expression with the smallest fine-tune error.

Algorithm 2 FEX with progressively expanding trees

Input: Tree set $\{\mathcal{T}_1, \mathcal{T}_2, \cdots\}$; Error tolerance ϵ;
Output: the solution $u(\boldsymbol{x}; \hat{\mathcal{T}}, \tilde{\boldsymbol{e}}, \tilde{\boldsymbol{\theta}})$.

for $\mathcal{T}$ in $\{\mathcal{T}_1, \mathcal{T}_2, \cdots\}$ **do**
  Initialize the agent $\boldsymbol{\chi}$ for the tree $\mathcal{T}$
  Obtain $u(\boldsymbol{x}; \mathcal{T}, \hat{\boldsymbol{e}}, \hat{\boldsymbol{\theta}})$ from Algorithm 1
  if $\mathcal{L}(u(\cdot; \mathcal{T}, \hat{\boldsymbol{e}}, \hat{\boldsymbol{\theta}})) \le \epsilon$ **then**
    Break
  end if
end for
return the expression with the smallest functional value.

7.3 Iterative method for eigenpairs

First, by solving Eqn. (10) with our FEX, we obtain an estimated eigenfunction $u(\boldsymbol{x}; \mathcal{T}, \hat{\boldsymbol{e}}, \hat{\boldsymbol{\theta}})$ and get the initial estimation of the eigenvalue through the Rayleigh quotient $\gamma_0 = \mathcal{I}(u(\cdot; \mathcal{T}, \hat{\boldsymbol{e}}, \hat{\boldsymbol{\theta}}))$. Then we can utilize Eqn. (11) to iteratively find the eigenpair. We define the function for Eqn. (11) by

$$\mathcal{L}_2(v, \gamma) := \left\| -\Delta v - \|\nabla v\|_2^2 + \|\boldsymbol{x}\|_2^2 - \gamma \right\|_{L_2(\Omega)}^2. \tag{27}$$

Given γ_i, we aim to find v that is expressed by mathematical expression and minimizes $\mathcal{L}_2(v, \gamma_i)$. Assume v is expressed by a binary tree ($v := v(\cdot; \mathcal{T}, \boldsymbol{e}, \boldsymbol{\theta})$), and then we can search the solution using our FEX with the following

optimization,

$$\boldsymbol{e}_i^*, \boldsymbol{\theta}_i^* \approx \arg\min_{\boldsymbol{e},\boldsymbol{\theta}} \mathcal{L}_2(v(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}),\gamma_i). \tag{28}$$

Next, we can compute the current estimated eigenvalue by $\gamma_{i+1} = \mathcal{I}(\exp(v(\cdot;\mathcal{T},\boldsymbol{e}_i^*,\boldsymbol{\theta}_i^*)))$.

If continuing this loop for G times, we will obtain the eigenpair γ_G and $\exp(v(\cdot;\mathcal{T},\boldsymbol{e}_G^*,\boldsymbol{\theta}_G^*))$.

Implementation. In our iteration method, the number of the iterative loop is $G = 10$. $\lambda_1 = \lambda_2 = 500$ in (10). The batch size for estimating the first term and third term of (10) is 10,000 while that of the second term (boundary) is 2,000. ReLU^2 is used in ResNet, following[16].

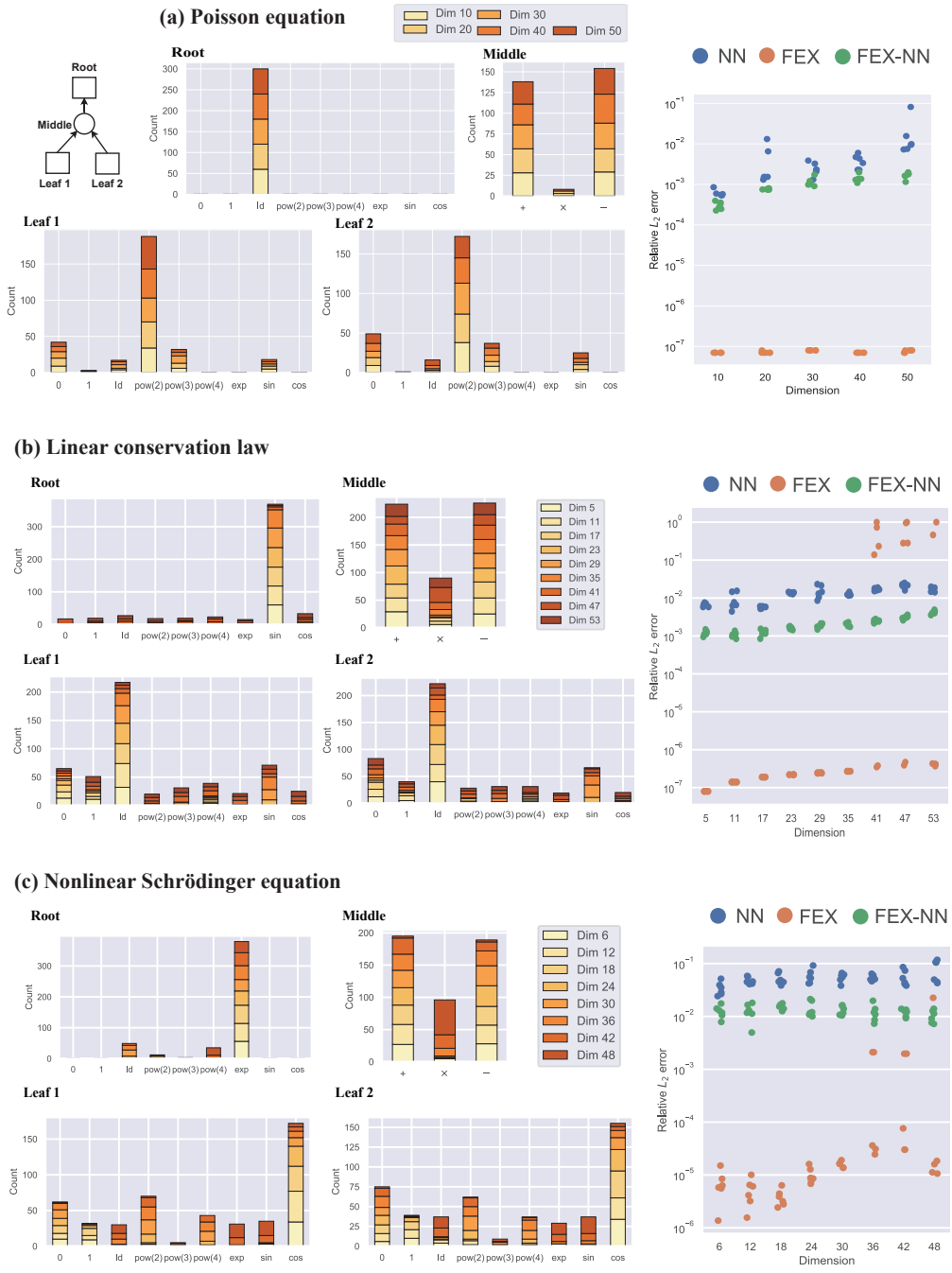

Figure 4. Distribution of the node values and the error comparison. We use Fex to search the optimal sequence of node values and show the frequency of the node values of the binary tree in the candidate pool, consisting of the root (Root), middle node (Middle), and two leaves (Leaf 1 and Leaf 2). Based on the observation of the distribution, we readily design the new NN parameterization (FEX-NN) to estimate the solution. The last column displays the relative L_2 error as the function of the dimension. Rows (a), (b) and (c) represent the results for Poisson equation (3), Linear conservation law (4) and Nonlinear Schrödinger equation (6) respectively. For various dimensions, FEX identifies the true solution, approximating solutions of almost the machine accuracy.

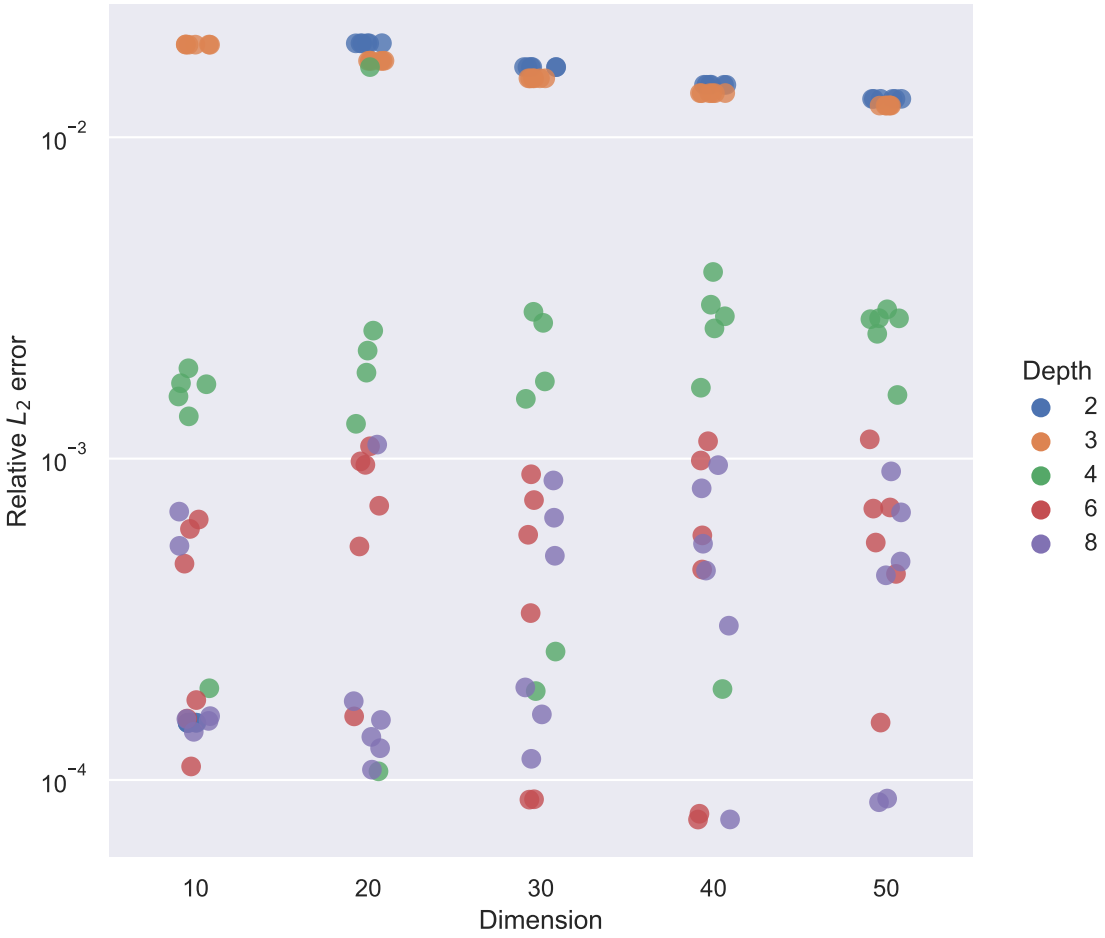

Figure 5. Relative L_2 error of solutions estimated by trees with increasing depth for the problems of various dimensions. We exclude the square operator $(\cdot)^2$ in the unary set $\mathbb{U}$, and the binary tree defined in Sec. 3.1 can not reproduce the true solution (sum of the square of coordinates) exactly in the example of the Poisson equation. We found that smaller errors could be obtained with larger tree sizes.

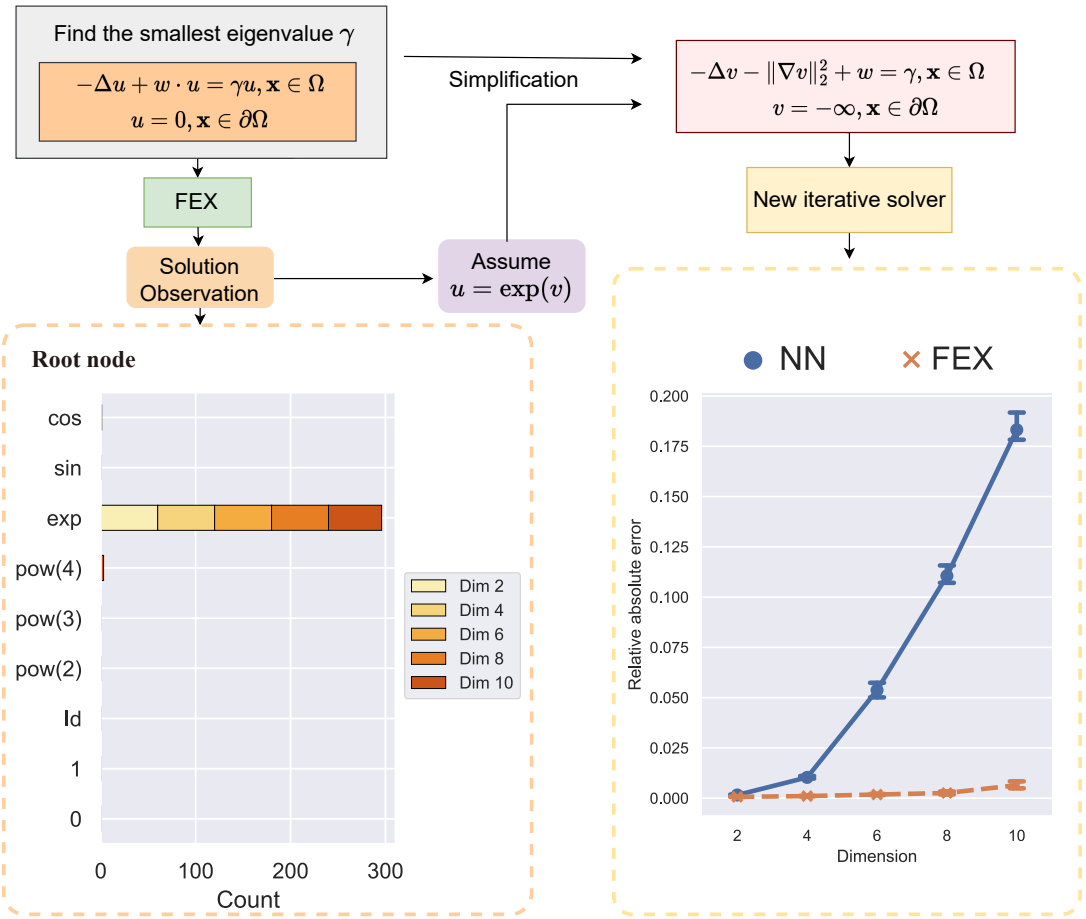

Figure 6. Eigenvalue problems and postprocessing algorithm design with FEX. Bottom left: We observed that the exponent operator "exp(·)" dominates the tree root in the FEX searching loop. Based on this observation, we assume the solution is $\exp(v(\boldsymbol{x}))$ and simplify the original PDE to a new PDE that avoids the trivial solution. Bottom right: The NN-based method produces a large error on the eigenvalue estimation, especially when the dimension is high ($d = 10$). With the postprocessing algorithm with FEX, we can identify the eigenvalue with an error close to zero.