Journal of Machine Learning Research 26 (2025) 1-31 Submitted 10/23; Revised 1/25; Published 8/25

Finite Expression Method for Solving High-Dimensional Partial Differential Equations

Senwei Liang SENWEI.LIANG@TTU.EDU
Department of Mathematics and Statistics
Texas Tech University
Lubbock, TX 79409, USA

Haizhao Yang HZYANG@UMD EDU
Department of Mathematics and Department of Computer Science
University of Maryland, College Park
College Park, MD 20742, USA

Editor: Pradeep Ravikumar

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.

Keywords: high-dimensional PDEs, deep neural networks, mathematical expressions, finite expression method, reinforcement learning

1. Introduction

Partial differential equations (PDEs) play a fundamental role in scientific fields for modeling diverse physical phenomena, including diffusion (Philibert, 2005; Kirkwood et al., 1960), fluid dynamics (Acheson, 1991; Shinbrot, 2012) and quantum mechanics (Feynman et al., 1965; Landau and Lifshitz, 2013). Developing efficient and accurate solvers for numerical solutions to high-dimensional PDEs remains an important and challenging topic (Weinan et al., 2021). Many traditional solvers, such as finite element method (FEM) (Reddy, 2004) and finite difference (Grossmann et al., 2007), are usually limited to low-dimensional domains since the computational cost increases exponentially in the dimension (Weinan et al., 2021; Li et al., 2022). Recently, neural networks (NNs) as mesh-free parameterization are widely employed in solving high-dimensional PDEs (E et al., 2017; Han et al., 2018; Khoo et al., 2017; Raissi et al., 2019; Sirignano and Spiliopoulos, 2018; E and Yu, 2018)

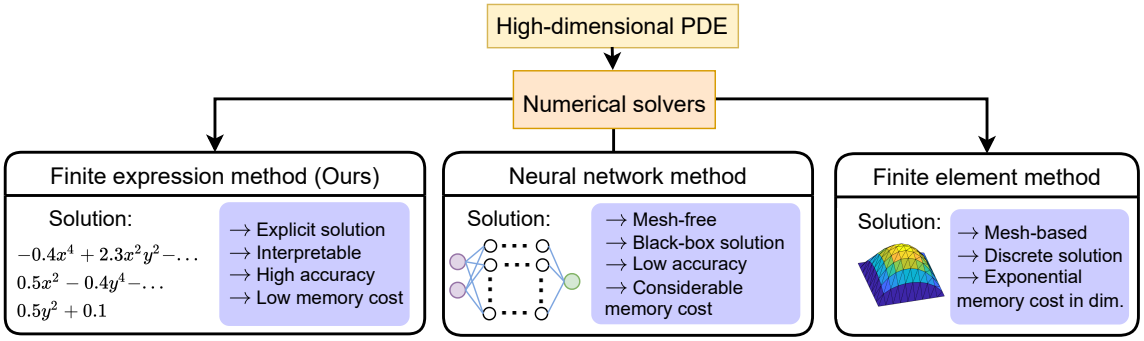

Figure 1: Overview of various numerical solvers for PDEs. In our finite expression method, we aim to find a PDE solution as a mathematical expression with finitely many operators. The resulting solution can reproduce the true solution and achieve high, even machine-level, accuracy. Furthermore, the mathematical expression has memory complexity polynomial in dimension and provides interpretable insights to advance understanding of physical systems and aid in designing postprocessing techniques for solution refinement.

and control problems (Han and Weinan, 2016). In theory, NNs have the capability of approximating various functions well and lessening the curse of dimensionality (Shen et al., 2021a,b; Yarotsky, 2021; Zhang et al., 2022; Jiao et al., 2023). Yet the highly non-convex objective function and the spectral bias toward fitting a smooth function in NN optimization make it difficult to achieve high accuracy (Xu et al., 2020; Cao et al., 2021; Ronen et al., 2019). 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 (E and Yu, 2018; Liu et al., 2020). In addition, NN parameterization may still require large memory and high computational cost for high-dimensional problems (Bianco et al., 2018). Finally, numerical solutions from both traditional solvers and NN-based solvers lack interpretability, e.g., the dependence of the solution on variables is not readily apparent from the 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 Figure 1): (1) The expression can reproduce the true solution and achieve high, even machine-level, accuracy. (2) The expression requires low memory (a line of string) for solution storage and low computational cost for evaluation on new points. (3) The expression has good interpretability in an explicit and readable form. Moreover, from an approximation theory perspective detailed in Section 2.3, the expression in FEX is capable of avoiding the curse of dimensionality in theory.

In FEX, we formulate the search for mathematical expressions as a combinatorial optimization (CO) involving both discrete and continuous variables. While many techniques can be used to solve the CO in FEX, we provide a numerically effective implementation based on reinforcement learning (RL). Traditional algorithms (e.g., genetic programming and simulated annealing (Murray-Smith, 2012)) address CO problems by employing handcrafted heuristics that are highly dependent on specific problem formulations and domain

knowledge (Bello et al., 2016; Cheung et al., 2019; Mazyavkina et al., 2021). However, RL has emerged as a popular and versatile tool for learning how to construct CO solutions based on reward feedback, without the need for extensive heuristic design. The success of RL applications in CO, such as automatic algorithm design (Ramachandran et al., 2018; Bello et al., 2017; Co-Reyes et al., 2021) and symbolic regression (Petersen et al., 2021; Landajuela et al., 2021), has inspired us to seek mathematical expression solutions 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 Figure 3 and further explained in Section 3.1. The objective function we aim to minimize is a functional, and its minimization leads to the solution of the PDE, as described in Section 2.2. Consequently, our problem involves the minimization of both discrete operators and continuous parameters embedded within the tree structure. Optimizing both discrete and continuous variables simultaneously is inherently difficult. We propose a search loop for this CO as depicted in Figure 2. Our idea is to first identify good operator selections that have a high possibility of recovering the true solution structure, and then optimize the parameters. The proposals of operator selections are drawn from a controller which will be updated via the policy gradient (Petersen et al., 2021) iteratively. In Section 4, we numerically demonstrate the ability of this RL-based implementation to find mathematical expressions that solve high-dimensional PDEs with high, even machine-level accuracy. Furthermore, FEX provides interpretable insights into the ground-truth PDE solutions, which can further advance understanding of physical systems and aid in designing postprocessing techniques to refine solutions.

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 in FEX.

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

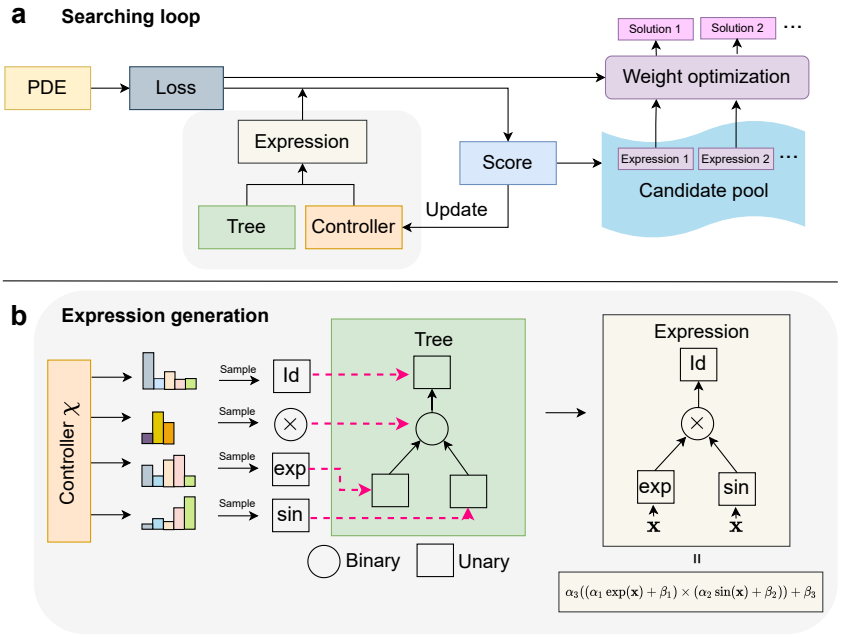

Figure 2: Representation of the components of our FEX implementation. (a) The searching loop for the finite expression solution consists of expression generation, score computation, controller update, and candidate optimization. (b) Depiction of the expression generation with a binary tree and a controller $\boldsymbol{\chi}$. The controller outputs probability mass functions for each node in the tree, and these functions are used to sample node values. The final expression, which incorporates learnable scaling and bias parameters, is constructed based on the predefined tree structure and the sampled node values.

"$\sin(x \times y) + 1$" is a 3-finite expression since there are three operators in it (i.e., "$\times$", "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 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. This $\mathbb{S}_k$ forms the function space in FEX. Clearly, $\mathbb{S}_1 \subset \mathbb{S}_2 \subset \mathbb{S}_3 \cdots$.

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 least-square methods (Lagaris et al., 1998; Sirignano and Spiliopoulos, 2018; Dissanayake and Phan-Thien, 1994), variation formulations (E and Yu, 2018; Yulei et al., 2021; Chen et al., 2023; Zang et al., 2020) and so on.

2.2.1 Least Square Method

Suppose that the PDE is given by

$$\mathcal{D}u(\boldsymbol{x}) = f(u(\boldsymbol{x}), \boldsymbol{x}),\ \boldsymbol{x} \in \Omega, \qquad \mathcal{B}u(\boldsymbol{x}) = g(\boldsymbol{x}),\ \boldsymbol{x} \in \partial\Omega, \tag{2}$$

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 (Evans, 2010)). The least square method (Lagaris et al., 1998; Sirignano and Spiliopoulos, 2018; Dissanayake and Phan-Thien, 1994) defines a straightforward functional to characterize the error of the estimated solution by

$$\mathcal{L}(u) := \|\mathcal{D}u(\boldsymbol{x}) - f(u, \boldsymbol{x})\|^2_{L^2(\Omega)} + \lambda\|\mathcal{B}u(\boldsymbol{x}) - g(\boldsymbol{x})\|^2_{L^2(\partial\Omega)}, \tag{3}$$

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

2.2.2 Variation Formulation

We next introduce the variation formulation, which is commonly used to identify numerical PDE solutions (E and Yu, 2018; Yulei et al., 2021). As an example, consider an elliptic PDE with homogeneous Dirichlet boundary conditions. This PDE is:

$$-\Delta u(\boldsymbol{x}) + c(\boldsymbol{x})u(\boldsymbol{x}) = f(\boldsymbol{x}),\ \boldsymbol{x} \in \Omega, \qquad u(\boldsymbol{x}) = 0,\ \boldsymbol{x} \in \partial\Omega, \tag{4}$$

where c is a bounded function and $f \in L^2$. The solution u to PDE (4) minimizes the variation formulation $\frac{1}{2}\int_\Omega \|\nabla u\|^2 + cu^2 \mathrm{d}\boldsymbol{x} - \int_\Omega fu\mathrm{d}\boldsymbol{x}$. By incorporating the boundary condition penalty into this variation, we obtain the functional:

$$\mathcal{L}(u) := \frac{1}{2}\int_\Omega \|\nabla u\|^2 + cu^2\mathrm{d}\boldsymbol{x} - \int_\Omega fu\mathrm{d}\boldsymbol{x} + \lambda\int_{\partial\Omega} u^2\mathrm{d}\boldsymbol{x}. \tag{5}$$

An alternative variation formulation can be defined using test functions. 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. (4) is defined as the function that satisfies the bilinear equations:

$$a(u, v) := \int_\Omega \nabla u\nabla v + cuv - fv\mathrm{d}\boldsymbol{x} = 0, \quad \forall v \in H^1_0(\Omega), \quad u(\boldsymbol{x}) = 0,\ \boldsymbol{x} \in \partial\Omega, \tag{6}$$

where $a(u, v)$ is constructed by multiplying (4) and v, and integration by parts. All derivatives of the solution function can be transferred to the test function through repeated integration by parts, yielding another bilinear forms (Chen et al., 2023). The weak solution can be reformulated as the solution to a saddle-point problem (Zang et al., 2020):

$$\min_{u\in H_0^1(\Omega)} \max_{v\in H_0^1(\Omega)} |a(u,v)|^2/\|v\|^2_{L^2(\Omega)}. \tag{7}$$

Then, the functional $\mathcal{L}$ identifying the PDE solution is:

$$\mathcal{L}(u) := \max_{v\in H_0^1(\Omega)} |a(u,v)|^2/\|v\|^2_{L^2(\Omega)} + \lambda \int_{\partial\Omega} u^2 \mathrm{d}\boldsymbol{x}. \tag{8}$$

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 (Zhang et al., 2022; Jiao et al., 2023). 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 4 *The function space $\mathbb{S}_k$, generated with operators including "$+$", "$-$", "$\times$", "$/$", "$|\cdot|$", "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 (Zhang et al., 2022), respectively. The proof of Theorem 4 can be found in Appendix A. 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 Theorem 4 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 "$+$", "$\times$", "sin$(\cdot)$", 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 5 *Suppose the function space $\mathbb{S}_k$ is generated with operators including "$+$", "$-$", "$\times$", "$\div$", "$\max\{0, x\}$", "$\sin(x)$", 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 $\|f - \phi\|_{L^p} \leq \varepsilon$, if $k \geq \mathcal{O}(d^2(\log d + \log \frac{1}{\varepsilon})^2)$.*

The proof of Theorem 5 can be found in Appendix A. 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 Section 3 can identify the desired finite expressions with this power. Furthermore, the information-theoretic limitations discussed in (Yarotsky and Zhevnerchuk, 2020) highlight that deep models with fewer weights

would require significantly higher weight precision to achieve their theoretical approximation power. Similarly, for FEX, while the method may require only a few operators to form an explicit expression to approximate a high-dimensional function, it would require high precision to learn the parameters of these expressions. Given the finite precision in practical implementations, achieving the full theoretical approximation power of FEX for general high-dimensional functions can be challenging. Nevertheless, the numerical experiments in Section 4 demonstrate that our FEX solver can identify the desired finite expressions to machine precision for several classes of high-dimensional PDEs. Therefore, FEX would be an appealling alternative of existing tools for high-dimensional problems. More importantly, it is intuitive to understand when FEX can find solutions of high accuracy, solutions with explicit expressions, which is a class of functions widely adopted in benchmark comparisons in the literature, while deep models struggle to achieve high accuracy.

3. An Implementation of FEX

Following the introduction of the abstract FEX methodology in Section 2, this section proposes a numerical implementation. First, binary trees are applied to construct finite expressions in Section 3.1. Next, our CO problem (1) is formulated in terms of parameter and operator selection to find expressions that approximate PDE solutions in Section 3.2. To resolve this CO, we propose implementing a search loop to identify effective operators that have the potential to recover the true solution when selected for expression. In Appendix B, we provide pseudo-code for the FEX algorithm, which employs expanding trees to search for a solution. For the reader's convenience, we have summarized 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 scoring function that maps an operator sequence to $[0, 1]$
χ_Φ	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.

3.1 Finite Expressions with Binary Trees

A finite expression can be represented as a binary tree $\mathcal{T}$, as depicted in Figure 3. Each node in the tree is assigned a value from a set of operators, and all these node values collectively form an operator sequence $\boldsymbol{e}$, following a predefined order to traverse the tree (e.g., inorder traversal).

Within each node featuring a unary operator, we incorporate two additional parameters, a scaling parameter α and a bias parameter β, to enhance expressiveness. All these parameters are denoted by $\boldsymbol{\theta}$. Therefore, a finite expression can be denoted as $u(\boldsymbol{x}; \mathcal{T}, \mathbf{e}, \boldsymbol{\theta})$, a function of $\boldsymbol{x}$. For a fixed $\mathcal{T}$, the maximal number of operators is upper bounded by a constant denoted $k_{\mathcal{T}}$. In this implementation, $\{u(\boldsymbol{x}; \mathcal{T}, \mathbf{e}, \boldsymbol{\theta}) | \mathbf{e}, \boldsymbol{\theta}\}$ forms the function space

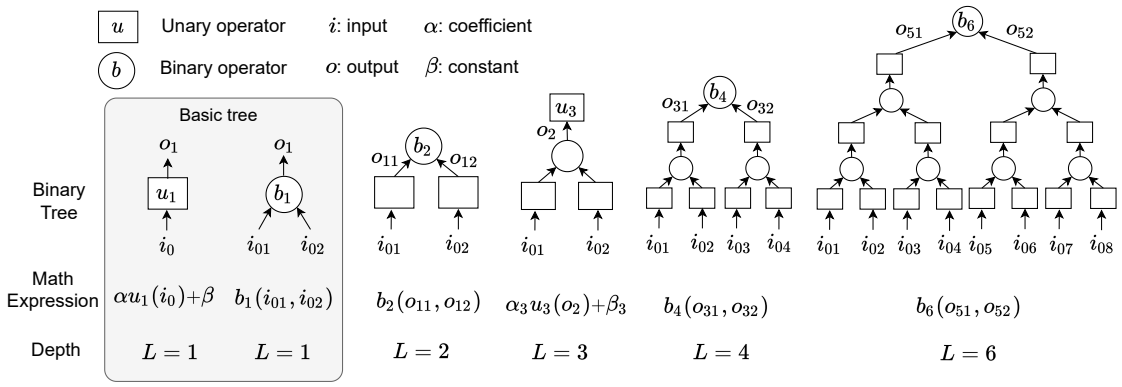

Figure 3: Computational rule of a binary tree. In a binary tree, each node holds either a unary or a binary operator. We begin by defining the computational process for a depth-1 tree (i.e., $L = 1$), which contains only a single operator. For binary trees with more than depth one (i.e., $L > 1$), the computation is performed recursively.

in the CO to solve a PDE. This is a subset of functions expressible with at most $k_{\mathcal{T}}$ finite expressions.

The configuration of the tree of various depths can be designed as in Figure 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), antiderivative and differentiation operators. Here "Id" denotes the identity map. Note that if an integration or a derivative is used in the expression, the operator can be applied numerically. Each entry in the operator sequence $\boldsymbol{e}$ has a one-to-one correspondence with the traversal of the nodes of the tree. The length of $\boldsymbol{e}$ equals the total number of tree nodes. For example, when inorder traversal is used, Figure 2b depicts 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 Figure 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 Figure 3. Specifically, the input of a parent node is the output of the child node(s). When the tree input at the bottom layer is a d-dimensional variable $\boldsymbol{x}$ (Figure 2b), the unary operator directly linked to $\boldsymbol{x}$ is applied element-wisely to each entry of $\boldsymbol{x}$ and the scaling α becomes a d-dimensional vector to perform a linear transformation from $\mathbb{R}^d$ to $\mathbb{R}^1$.

3.2 Solving a CO in FEX

Given a tree $\mathcal{T}$, we aim to seek the PDE solution from the function space $\{u(\boldsymbol{x};\mathcal{T},\mathbf{e},\boldsymbol{\theta})|\mathbf{e},\boldsymbol{\theta}\} \subset \mathbb{S}_{k_\mathcal{T}}$. The mathematical expression can be identified via the minimization of the functional $\mathcal{L}$ associated with a PDE, i.e.,

$$\min\{\mathcal{L}(u(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}))|\boldsymbol{e},\boldsymbol{\theta}\}. \tag{9}$$

We introduce the framework for implementing FEX, as displayed in Figure 2a, to seek a minimizer of (9). 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 (9). In our framework, the searching loop consists of four parts: 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 Figure 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.

3.2.1 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

$$S(\boldsymbol{e}) := \big(1 + L(\boldsymbol{e})\big)^{-1}, \tag{10}$$

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 (Rumelhart et al., 1986) and Adam (Kingma and Ba, 2014)) that utilize gradient to update are popular in machine learning. Typically, they demand a small learning rate and a substantial number of iterations to converge effectively. It can become time-consuming to optimize $L(\boldsymbol{e})$ using a first-order algorithm. Alternatively, second-order algorithms (e.g., the Newton method (Avriel, 2003) and the Broyden-Fletcher-Goldfarb-Shanno method (BFGS) (Fletcher, 2013)) use the (approximated) Hessian matrix for faster convergence, but obtaining a good minimizer requires a good initial guess. To expedite the optimization process of $L(\boldsymbol{e})$ in our implementation, we employ a two-step approach. Initially, a first-order algorithm is utilized for T_1 steps to

obtain a well-informed initial guess. Subsequently, a second-order algorithm is applied for an additional T_2 steps to further refine the solution. 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 two-step optimization. Then $\boldsymbol{\theta}_{T_1+T_2}^{\boldsymbol{e}}$ serves as an approximate of $\arg\min_{\boldsymbol{\theta}} \mathcal{L}(u(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}))$. Finally, $S(\boldsymbol{e})$ is estimated by

$$S(\boldsymbol{e}) \approx \big(1+\mathcal{L}(u(\cdot;\mathcal{T},\boldsymbol{e},\boldsymbol{\theta}_{T_1+T_2}^{\boldsymbol{e}}))\big)^{-1}. \tag{11}$$

Remark that the approximation may exhibit significant variation due to the randomness associated with the initialization of $\boldsymbol{\theta}_0^{\boldsymbol{e}}$.

3.2.2 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. We use $\boldsymbol{e}\sim\boldsymbol{\chi}_\Phi$ 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 Figure 2b for an illustration. Besides, we adopt the ϵ-greedy strategy (Sutton and Barto, 2018) 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 of exploring new sequences.

3.2.3 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 (Liu et al., 2019)). 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.,

$$\mathcal{J}(\Phi):=\mathbb{E}_{\boldsymbol{e}\sim\boldsymbol{\chi}_\Phi} S(\boldsymbol{e}). \tag{12}$$

Taking the derivative of (12) with respect to Φ, we have

$$\nabla_\Phi \mathcal{J}(\Phi)=\mathbb{E}_{\boldsymbol{e}\sim\boldsymbol{\chi}_\Phi}\big\{S(\boldsymbol{e})\sum_{i=1}^{s}\nabla_\Phi\log(\boldsymbol{p}_\Phi^i(e_i))\big\}, \tag{13}$$

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

$$\nabla_\Phi \mathcal{J}(\Phi) \approx \frac{1}{N}\sum_{k=1}^{N}\big\{S(\boldsymbol{e}^{(k)})\sum_{i=1}^{s}\nabla_\Phi \log(\boldsymbol{p}_\Phi^i(e_i^{(k)}))\big\}. \tag{14}$$

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

$$\Phi \leftarrow \Phi + \eta\nabla_\Phi \mathcal{J}(\Phi). \tag{15}$$

The objective in (12) 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 (Petersen et al., 2021) is applied to seek the optimal solution via

$$\mathcal{J}(\Phi) = \mathbb{E}_{\boldsymbol{e}\sim\boldsymbol{\chi}_\Phi}\{S(\boldsymbol{e})|S(\boldsymbol{e}) \geq S_{\nu,\Phi}\}, \tag{16}$$

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

$$\nabla_\Phi \mathcal{J}(\Phi) \approx \frac{1}{N}\sum_{k=1}^{N}\big\{(S(\boldsymbol{e}^{(k)}) - \hat{S}_{\nu,\Phi})\mathbb{1}_{\{S(\boldsymbol{e}^{(k)})\geq \hat{S}_{\nu,\Phi}\}}\sum_{i=1}^{s}\nabla_\Phi \log(\boldsymbol{p}_\Phi^i(e_i^{(k)}))\big\}. \tag{17}$$

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 $\{S(\boldsymbol{e}^{(i)})\}_{i=1}^N$.

3.2.4 Candidate Optimization

As introduced in Section 3.2.1, 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.

4. Numerical Examples

Numerical results will be provided to demonstrate the effectiveness of our FEX implementation introduced in Section 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 $\mathcal{O}(10^{-2})$. Furthermore, we demonstrate the effectiveness of our RL-based approach by comparing its performance to a solution developed using genetic programming (GP) (Murray-Smith, 2012).
- **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. 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 (Figure 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 (see Appendix C for more details). 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.

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 (Figure 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. Especially in Section of "Numerical Convergence", since the deeper trees are used, the controller is updated for 5000 iterations for trees with different depth. 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) *Candidate 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 (He et al., 2019).

Implements of NN-based Solvers. Residual networks (ResNets) (He et al., 2016; E and Yu, 2018) $u(\boldsymbol{x}; \boldsymbol{\Theta})$ 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 (Sirignano and Spiliopoulos, 2018; Weinan et al., 2021; E and Yu, 2018).

The ResNet maps from $\mathbb{R}^d$ to $\mathbb{R}^1$ and consists of seven fully connected layers with three skip connections. Each hidden layer contains 50 neurons. The neural network is optimized using the Adam optimizer with an initial learning rate of 0.001 for 15,000 iterations. The learning rate is decayed following a cosine decay schedule.

Poisson equation. The coefficient λ in the functional (3) is 100. The batch size for the interior and boundary is 5,000 and 1,000, respectively. In the NN method, we use the ReLU^2 activation, i.e., $(\max\{x, 0\})^2$, in ResNet to approximate the solution.

Linear conservative law. The coefficient λ in the functional (20) 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 (23) is 1. The batch size for estimating the first term and second term of (23) is 2,000 and 10,000, respectively. Besides, ReLU^2 is used in ResNet.

Implements of genetic programming (GP). GP is an evolutionary algorithm used for automated symbolic regression and the evolution of computer programs to solve complex problems. Through a process of selection, crossover, and mutation, GP evolves and refines these programs over generations to optimize a specific fitness or objective function. We utilize GP to optimize the CO (1) directly using the Python package GPlearn (Stephens, 2017).

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 Section 3.1 and $\mathbb{B}, \mathbb{U}$ defined in Section 4.1.

4.2.1 Poisson Equation

We consider a Poisson equation (E and Yu, 2018) with a Dirichlet boundary condition on a d-dimensional domain $\Omega = [-1, 1]^d$,

$$-\Delta u = f \text{ for } \boldsymbol{x} \in \Omega, \quad u = g \text{ for } \boldsymbol{x} \in \partial\Omega. \tag{18}$$

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 $\mathcal{L}$ defined by least-square error (3) 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 (Jingrun et al., 2023) is considered here with a domain $T \times \Omega = [0,1] \times [-1,1]^d$ and an initial value:

$$\frac{\pi d}{4} u_t - \sum_{i=1}^{d} u_{x_i} = 0 \text{ for } \boldsymbol{x} = (x_1, \cdots, x_d) \in \Omega, t \in [0,1], \quad u(0, \boldsymbol{x}) = \sin(\frac{\pi}{4} \sum_{i=1}^{d} x_i) \text{ for } \boldsymbol{x} \in \Omega, \tag{19}$$

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 $\mathcal{L}$ used in the NN-based solver and FEX to identify the solution is defined by

$$\mathcal{L}(u) := \| \frac{\pi d}{4} u_t - \sum_{i=1}^{d} u_{x_i} \|^2_{L^2(T \times \Omega)} + \lambda \| u(0, \boldsymbol{x}) - \sin(\frac{\pi}{4} \sum_{i=1}^{d} x_i) \|^2_{L^2(\Omega)}. \tag{20}$$

4.2.3 Nonlinear Schrödinger Equation

We consider a nonlinear Schrödinger equation (Han et al., 2020) with a cubic term on a d-dimensional domain $\Omega = [-1,1]^d$,

$$-\Delta u + u^3 + Vu = 0 \text{ for } \boldsymbol{x} \in \Omega, \tag{21}$$

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 $\boldsymbol{x} = (x_1, \cdots, x_d)$. And we let $\hat{u}(\boldsymbol{x}) = \exp(\frac{1}{d} \sum_{j=1}^{d} \cos(x_j))/3$ be the solution of the PDE (21). 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

$$\mathcal{L}_1(u) := \| -\Delta u + u^3 + Vu \|^2_{L_2(\Omega)} / \|u\|^3_{L_2(\Omega)}. \tag{22}$$

During the candidate optimization phase, an integration constraint is imposed to (21), i.e., $\int_\Omega u(\boldsymbol{x}) d\boldsymbol{x} = \int_\Omega \hat{u}(\boldsymbol{x}) d\boldsymbol{x}$. The functional $\mathcal{L}$ used in FEX to fine-tune the identified operator sequence is defined by

$$\mathcal{L}_2(u) := \| -\Delta u + u^3 + Vu \|^2_{L_2(\Omega)} + \lambda \Big(\int_\Omega u(\boldsymbol{x}) d\boldsymbol{x} - \int_\Omega \hat{u}(\boldsymbol{x}) d\boldsymbol{x} \Big)^2, \tag{23}$$

where the second term imposes the integration constraint. The functional (23) is also used in the NN-based solver to approximate the PDE solution. Various dimensions are tested in the numerical results, e.g., $d = 6$, 12, 18, 24, 30, 36, 42 and 48. Remark that we avoid using (23) in the score computation because the Monte-Carlo error tends to be significant in the second term of (23) when the batch size is small, but using a large batch size can lead to inefficient computations. Hence, for the computational efficiency, we will utilize (22) instead in the score computation, rather than (23).

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 (Figure 3) is used in FEX with four nodes (a root node (R), a middle node (M), and two leave nodes (L1 and L2)). Figure 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 (23) is estimated by the Monte-Carlo 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 (23). Therefore, the optimization solver may return an approximate solution without machine accuracy. Designing a functional $\mathcal{L}$ free of the Monte-Carlo error (e.g., Eqns. (3) and (20)) for the nonlinear Schödinger equation could ensure machine accuracy. Furthermore, as shown in Figure 4, GP tends to identify solutions of lower quality that fail to achieve a high level of 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 Figure 3). The square operator $(\cdot)^2$ is excluded in $\mathbb{U}$ so that the binary tree defined in Section 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. Figure 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 Figure 4, FEX gets a higher accuracy when a larger tree is used. These results underscore FEX's ability to identify better solutions without relying on any special construction and provide evidence that its performance is not simply due to biases in the construction, such as tree structure or operator set.

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. Figure 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 (E and Yu, 2018),

$$-\Delta u + wu = \gamma u, \quad \boldsymbol{x} \in \Omega, \quad \text{for } u = 0 \text{ on } \partial\Omega. \tag{24}$$

The minimization of the Rayleigh quotient $\mathcal{I}(u) = \frac{\int_\Omega \|\nabla u\|_2^2 d\boldsymbol{x} + \int_\Omega w u^2 d\boldsymbol{x}}{\int_\Omega u^2 d\boldsymbol{x}}$, s.t., $u|_{\partial\Omega} = 0$, gives the smallest eigenvalue and the corresponding eigenfunction. In the NN-based solver (E and Yu, 2018), the following functional is defined

$$\mathcal{L}(u) := \mathcal{I}(u) + \lambda_1 \int_{\partial\Omega} u^2 d\boldsymbol{x} + \lambda_2 \big(\int_\Omega u^2 d\boldsymbol{x} - 1 \big)^2 \tag{25}$$

to seek an NN solution.

4.4.1 FEX vs NN for non-analytic eigenfunctions

We consider a potential $w(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2 + \delta \sum_{i=1}^d x_i^4$ and $\Omega = \mathbb{R}^d$, which incorporates a quartic perturbation term $\delta \sum_{i=1}^d x_i^4$ with a small $\delta \geq 0$. The domain Ω is truncated from $\mathbb{R}^d$ to $[-3, 3]^d$ for simplification as done in (E and Yu, 2018). When $\delta = 0$, the smallest eigenvalue of (24) is d and the associated eigenfunction is $\exp(-\frac{\|\boldsymbol{x}\|_2^2}{2})$. Potentials with quartic terms (e.g., $\delta > 0$) can lead to eigenfunctions in non-analytic form. While the eigenfunction corresponding to the smallest eigenvalue can be non-analytic, perturbation theory suggests that the true smallest eigenvalue can be approximated by $n + 3/4n\delta$ (with details in Appendix D), which we use as a reference for the true smallest eigenvalue.

We consider a small $\delta = 0.1$. Both the FEX and NN methods use the functional (25) to estimate the eigenvalue. We used the same batch size across different dimensions to discretize the integration over the domain, as well as the integration over the boundary. Specifically, the batch size for estimating the first term and third term of the objective function (25) is 500,000 while that of the second term (boundary) is 100,000. Figure 6 compares FEX and the NN method to approximate the smallest eigenvalue for $d = 10, 12, 14, 16$ and 18 using the Rayleigh quotient (25). As the dimension d increases, the smallest eigenvalues identified by the NN method deviate significantly from the reference value, whereas those obtained using FEX remain much closer to it. Furthermore, the FEX method demonstrates greater stability, with substantially smaller standard deviation in the identified smallest eigenvalues.

4.4.2 Development of postprocessing techniques with FEX

Here, we demonstrate how FEX can facilitate the development of postprocessing techniques to achieve more refined results. For instance, consider the case where of $w = \|\boldsymbol{x}\|_2^2$ and $\Omega = \mathbb{R}^d$. In this scenario, the smallest eigenvalue of (24) is d, and the corresponding eigenfunction is $\exp(-\frac{\|\boldsymbol{x}\|_2^2}{2})$. We examine dimensions $d = 2, 4, 6, 8, 10$. Given the lower

dimensionality compared to the previous section, we can use smaller batch sizes: specifically, the batch size for estimating the first and third terms of the objective function (25) is 10,000, while that for the second term (boundary) is 2,000.

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 Figure 7). 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. (24) is simplified to

$$-\Delta v - \|\nabla v\|_2^2 + \|\boldsymbol{x}\|_2^2 = \gamma. \tag{26}$$

Eqn. 26 does not have a trivial zero solution so we can avoid the integration constraint used in Eqns. (23) and (25), which leads to Monte-Carlo errors. Using Eqn. (26) and the Rayleigh quotient $\mathcal{I}$, the values of v and γ are alternatively updated until they reach convergence. The detail of this iterative algorithm is presented in Appendix E. Figure 7 shows the relative absolute error of the estimated eigenvalues with the growth of the dimensions. We can see that directly optimizing (25) 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 and Perspectives

In this paper, we proposed the finite expression method - a methodology to find PDE solutions as simple mathematical expressions. Our theory showed that mathematical expressions can overcome the curse of dimensionality. We provided one implementation of representing mathematical expressions using trees and solving the formulated combinatorial optimization in FEM using reinforcement learning. Our results demonstrated effectiveness of FEM at achieving high, even machine-level accuracy on various high-dimensional PDEs while existing solvers suffered from low accuracy in comparison.

While the proposed FEX solver is capable of identifying finite expressions with machine-level accuracy for several classes of high-dimensional PDEs, its computational cost remains significant. This cost is primarily divided into two phases: the searching phase and the fine-tuning phase.

- While the fine-tuning phase in FEX resembles the procedure used in NN methods for optimizing the parameters of surrogate models, it is significantly more efficient. This efficiency arises from the substantially smaller number of parameters in the FEX method compared to NN models. Specifically, the number of parameters in FEX is approximately $d \times 2^{L/2-1} + 2^{L/2+1}$ where d is the input dimension and L is the depth of the tree. In most of the experiments presented in Section 4, we used $L \leq 4$. In contrast, the number of parameters for the NN model is about $dm + (\ell - 2)m^2 + m$ where m represents the size of the hidden layer and ℓ is the number of hidden layers. For the NN models used in Section 4, we set $m = 50$ and $\ell = 7$.

- The searching phase in the FEX method introduces a significant computational overhead. This phase involves RL-based optimization of the controller and the evaluation

of operator sequence scores, both of which are computationally intensive. However, the method's flexibility in configuring the searching process allows for balancing cost and performance.

Users can adjust various hyperparameters to tailor the computational effort to specific problem requirements or resource constraints. These hyperparameters include the number of iterations for the RL-based optimization of the controller, the number of iterations used to estimate the score function for an operator sequence, the depth of the binary tree structure, and the batch size.

Furthermore, certain components of the FEX algorithm can be parallelized to enhance efficiency. For instance, the score estimation for each operator sequence within a batch can be performed in parallel, and the final candidate fine-tuning process is also amenable to parallelization.

- If a complete binary tree is used (consisting of alternating layers of unary and binary nodes), the number of operators grows approximately as $2^{L/2}$, where L is the tree depth. Empirically, we observed an exponential increase in optimization time with increasing tree depth, which aligns with the exponential growth in the number of operators. However, in practice, it is often unnecessary to use extremely deep or complete binary trees. Instead, shallower or pruned trees can provide a good balance.

While CO is inherently a complex challenge, continued exploration can be highly advantageous in achieving improved performance, particularly when dealing with more intricate PDE problems within the framework of FEX.

- **Efficient computation of operator sequence score.** We employed Formulation (11) as a proxy for assessing the quality of an operator sequence. However, it's important to note that this formulation necessitates optimization over multiple steps, resulting in non-negligible computational costs that can impact the overall expense of solving the CO problem in FEX. Therefore, it is crucial to define a more efficient scoring method that reduces computational expenses and simplifies the identification of favorable operator sequences.

- **Design of the controller.** As an illustrative example of CO solving in this work, we employed a straightforward fully connected network to model the distribution responsible for proposing operator selections. It's worth noting that more sophisticated techniques can also be applied, such as recurrent neural networks (Petersen et al., 2021), which take prior decisions as input and generate new decisions as output. These advanced methods can offer enhanced modeling capabilities and adaptability to the context of the problem.

Data and Code Availability

No date is generated in this work. Source codes for reproducing the results in this paper are available online at: `https://github.com/LeungSamWai/Finite-expression-method`. The source codes are released under MIT license.

Acknowledgments

H. Yang was partially supported by the US National Science Foundation under awards DMS-2244988, DMS-2206333, the Office of Naval Research Award N00014-23-1-2007, and the DARPA D24AP00325-00. S. Liang acknowledges partial support from a startup grant by Texas Tech University. S. Liang acknowledges the helpful suggestion from Dr. Yuanran Zhu from Lawrence Berkeley National Laboratory on the eigenvalue problems.

Appendix A. Proofs of Theorems

Proof [Proof of Theorem 4] By Theorem 1.1 of Zhang et al. (2022), 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, $\sigma(x) = \frac{\text{sign}(x)+1}{2}\sigma_1(x) - \frac{\text{sign}(x)-1}{2}\sigma_2(x)$. 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 "$\times$", $Nd - 1 + (L-1)N(N-1)$ 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 [Proof of Theorem 5] By Cor. 3.8 of Jiao et al. (2023), 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

$$N = \max\{2d \left\lceil \log_2 \left(\sqrt{d} \left(\frac{3\mu}{\varepsilon} \right)^{1/\alpha} \right) \right\rceil, 2 \left\lceil \log_2 \frac{3\mu d^{\alpha/2}}{2\varepsilon} \right\rceil + 2\}$$

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 Theorem 4, there are $N(d+1) + (L-1)N^2$ operators "$\times$", $Nd - 1 + (L-1)N(N-1)$ 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. ■

Appendix B. Algorithm to Solve CO with Reinforcement Learning

We have included the pseudo-code for the proposed FEX implementation in Algorithm 1 and Algorithm 2. Specifically, Algorithm 1 outlines the procedure for solving the CO problem using a predefined fixed tree. On the other hand, Algorithm 2 is designed to iteratively expand a tree, thereby enhancing its expressiveness in the pursuit of identifying an improved solution based on the approach described in Algorithm 1.

Appendix C. Using a Fully Connected Network to Model a Controller

We illustrate the use of a fully connected neural network to model the controller, which generates probability mass functions for each node within a tree, in Figure 8. As an example, assume that we have a tree with three nodes: $n_1 = 2$ nodes for the binary set with $|\mathbb{B}| = 2$ operators, and $n_2 = 1$ node for the unary set with $|\mathbb{U}| = 3$ operators. Consequently,

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 $\mathcal{L}(u(\boldsymbol{x}; \mathcal{T}, \boldsymbol{e}^{(n)}, \boldsymbol{\theta}))$ through coarse-tuning over $T_1 + T_2$ iterations to get score $S(\boldsymbol{e}^{(n)})$
        if $\boldsymbol{e}^{(n)}$ belongs to the top-K of all scorings in $\mathbb{P}$ **then**
            $\mathbb{P}$.append($\boldsymbol{e}^{(n)}$)
            $\mathbb{P}$ pops some $\boldsymbol{e}$ of the smallest score when $|\mathbb{P}| > K$
        end if
    end for
    Update $\boldsymbol{\chi}$ using (17)
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 $\mathcal{L}(u(\cdot; \mathcal{T}, \hat{\boldsymbol{e}}, \hat{\boldsymbol{\theta}})) \leq \epsilon$ **then**
        Break
    end if
end for
return the expression with the smallest functional value.

the output size of the neural network is $n_1|\mathbb{B}| + n_2|\mathbb{U}| = 7$. This output is divided into $n_1 + n_2$ parts, each corresponding to the probability mass function for its respective node. In addition, the input to the neural network is simply a constant vector.

Appendix D. Eigenvalue Reference

When $w(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2$, the smallest eigenvalue of (24) is d and the associated eigenfunction is $\phi(\boldsymbol{x}) := \exp(-\frac{\|\boldsymbol{x}\|_2^2}{2})$. When considering $w(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2 + \delta \sum_{i=1}^d x_i^4$, the Rayleigh quotient of ϕ gives

$$\begin{aligned}
\mathcal{I}(\phi) &= \frac{\int_\Omega \|\nabla\phi\|_2^2 dx + \int_\Omega (\|\boldsymbol{x}\|_2^2 + \delta\sum_{i=1}^d x_i^4)\phi^2 dx}{\int_\Omega \phi^2 dx} \\
&= d + \delta \frac{\int_\Omega \sum_{i=1}^d x_i^4 \phi^2 dx}{\int_\Omega \phi^2 dx} \\
&= d + \delta \frac{\int_\Omega \sum_{i=1}^d x_i^4 \exp(-\|\boldsymbol{x}\|_2^2) dx}{\int_\Omega \exp(-\|\boldsymbol{x}\|_2^2) dx} \\
&= d + \delta d \frac{\int_{-\infty}^{\infty} x_i^4 \exp(-x_i^2) dx}{\int_{-\infty}^{\infty} \exp(-x_i^2) dx} = d + \delta d \frac{\Gamma(5/2)}{\Gamma(1/2)} = d + 3/4\delta d,
\end{aligned} \tag{27}$$

where Γ represents the Gamma function. Therefore, the smallest eigenvalue should be less than $\mathcal{I}(\phi) = d + 3/4\delta d$.

Appendix E. FEX-inspired Iterative Method for Eigenpairs

First, by solving Eqn. (25) with our FEX, we obtain an estimated eigenfunction $u(\boldsymbol{x}; \mathcal{T}, \hat{\boldsymbol{e}}, \hat{\boldsymbol{\theta}})$ and get the 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. (26) to iteratively find the eigenpair. We define the function for Eqn. (26) by

$$\mathcal{L}_2(v, \gamma) := \big\| -\Delta v - \|\nabla v\|_2^2 + \|\boldsymbol{x}\|_2^2 - \gamma \big\|_{L_2(\Omega)}^2. \tag{28}$$

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,

$$\boldsymbol{e}_i^*, \boldsymbol{\theta}_i^* \approx \arg\min_{\boldsymbol{e}, \boldsymbol{\theta}} \mathcal{L}_2(v(\cdot; \mathcal{T}, \boldsymbol{e}, \boldsymbol{\theta}), \gamma_i). \tag{29}$$

Next, we can calculate 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 $\exp(v(\cdot; \mathcal{T}, \boldsymbol{e}_G^*, \boldsymbol{\theta}_G^*))$.

For implementation, the number of the iterative loop is $G = 10$. $\lambda_1 = \lambda_2 = 500$ in (25). The batch size for estimating the first term and third term of (25) is 10,000 while that of the second term (boundary) is 2,000. ReLU^2 is used in ResNet, following E and Yu (2018).

References

David J Acheson. Elementary fluid dynamics, 1991.

Mordecai Avriel. *Nonlinear programming: analysis and methods.* Courier Corporation, 2003.

Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

Irwan Bello, Barret Zoph, Vijay Vasudevan, and Quoc V Le. Neural optimizer search with reinforcement learning. In *International Conference on Machine Learning*, pages 459–468. PMLR, 2017.

Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

Yuan Cao, Zhiying Fang, Yue Wu, Ding-Xuan Zhou, and Quanquan Gu. Towards understanding the spectral bias of deep learning. In *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, pages 2205–2211. International Joint Conferences on Artificial Intelligence Organization, 2021.

Fan Chen, Jianguo Huang, Chunmei Wang, and Haizhao Yang. Friedrichs learning: Weak solutions of partial differential equations via deep learning. *SIAM Journal on Scientific Computing*, 45(3):A1271–A1299, 2023.

Wang Chi Cheung, Vincent Tan, and Zixin Zhong. A thompson sampling algorithm for cascading bandits. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 438–447. PMLR, 2019.

John D Co-Reyes, Yingjie Miao, Daiyi Peng, Esteban Real, Quoc V Le, Sergey Levine, Honglak Lee, and Aleksandra Faust. Evolving reinforcement learning algorithms. In *International Conference on Learning Representations*, 2021. URL `https://openreview.net/forum?id=0XXpJ4OtjW`.

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

Weinan E and Bing Yu. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Commun. Math. Stat.*, 6:1–12, 2018.

Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, 2017. doi: 10.1007/s40304-017-0117-6. URL `https://doi.org/10.1007/s40304-017-0117-6`.

L.C. Evans. *Partial Differential Equations*. Graduate studies in mathematics. American Mathematical Society, 2010. ISBN 9780821849743. URL `https://books.google.com/books?id=Xnu0o_EJrCQC`.

Richard P Feynman, Robert B Leighton, and Matthew Sands. The feynman lectures on physics; vol. i. *American Journal of Physics*, 33(9):750–752, 1965.

Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

Christian Grossmann, Hans-Görg Roos, and Martin Stynes. *Numerical treatment of partial differential equations*, volume 154. Springer, 2007.

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

Jiequn Han and E Weinan. Deep learning approximation for stochastic control problems. *ArXiv*, abs/1611.07422, 2016.

Jiequn Han, Jianfeng Lu, and Mo Zhou. Solving high-dimensional eigenvalue problems using deep neural networks: A diffusion monte carlo like approach. *Journal of Computational Physics*, 423:109792, 2020.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Tong He, Zhi Zhang, Hang Zhang, Zhongyue Zhang, Junyuan Xie, and Mu Li. Bag of tricks for image classification with convolutional neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 558–567, 2019.

Yuling Jiao, Yanming Lai, Xiliang Lu, Fengru Wang, Jerry Zhijian Yang, and Yuanyuan Yang. Deep neural networks with relu-sine-exponential activations break curse of dimensionality in approximation on hölder class. *SIAM Journal on Mathematical Analysis*, 55 (4):3635–3649, 2023.

Chen Jingrun, Jin Shi, and Liyao Lyu. A deep learning based discontinuous galerkin method for hyperbolic equations with discontinuous solutions and random uncertainties. *Journal of Computational Mathematics*, 41(6):1281–1304, 2023. ISSN 1991-7139. doi: https://doi.org/10.4208/jcm.2205-m2021-0277.

Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *arXiv: Numerical Analysis*, 2017.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

John G Kirkwood, Robert L Baldwin, Peter J Dunlop, Louis J Gosting, and Gerson Kegeles. Flow equations and frames of reference for isothermal diffusion in liquids. *The Journal of Chemical Physics*, 33(5):1505–1513, 1960.

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

Mikel Landajuela, Brenden K Petersen, Sookyung Kim, Claudio P Santiago, Ruben Glatt, Nathan Mundhenk, Jacob F Pettit, and Daniel Faissol. Discovering symbolic policies with deep reinforcement learning. In Marina Meila and Tong Zhang, editors, *Proceedings*

of the 38th International Conference on Machine Learning, volume 139 of *Proceedings of Machine Learning Research*, pages 5979–5989. PMLR, 18–24 Jul 2021. URL https://proceedings.mlr.press/v139/landajuela21a.html.

Lev Davidovich Landau and Evgenii Mikhailovich Lifshitz. *Quantum mechanics: non-relativistic theory*, volume 3. Elsevier, 2013.

Fu Li, Umberto Villa, Seonyeong Park, and Mark A. Anastasio. 3-d stochastic numerical breast phantoms for enabling virtual imaging trials of ultrasound computed tomography. *IEEE Transactions on Ultrasonics, Ferroelectrics, and Frequency Control*, 69(1):135–146, 2022. doi: 10.1109/TUFFC.2021.3112544.

Hanxiao Liu, Karen Simonyan, and Yiming Yang. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019. URL https://openreview.net/forum?id=S1eYHoC5FX.

Ziqi Liu, Wei Cai, and Zhi-Qin John Xu. Multi-scale deep neural network (mscalednn) for solving poisson-boltzmann equation in complex domains. *Communications in Computational Physics*, 28(5):1970–2001, Jun 2020. ISSN 1991-7120. doi: 10.4208/cicp.oa-2020-0179. URL http://dx.doi.org/10.4208/cicp.OA-2020-0179.

Nina Mazyavkina, Sergey Sviridov, Sergei Ivanov, and Evgeny Burnaev. Reinforcement learning for combinatorial optimization: A survey. *Computers & Operations Research*, 134:105400, 2021.

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

Brenden K Petersen, Mikel Landajuela Larma, Terrell N. Mundhenk, Claudio Prata Santiago, Soo Kyung Kim, and Joanne Taery Kim. Deep symbolic regression: Recovering mathematical expressions from data via risk-seeking policy gradients. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=m5Qsh0kBQG.

Jean Philibert. One and a half century of diffusion: Fick, einstein. *Diffusion Fundamentals: Leipzig 2005*, 1:8, 2005.

Maziar Raissi, Paris Perdikaris, and George E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational physics*, 378:686–707, 2019.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2018. URL https://openreview.net/forum?id=SkBYYyZRZ.

JN Reddy. *An introduction to the finite element method*, volume 1221. McGraw-Hill New York, 2004.

Basri Ronen, David Jacobs, Yoni Kasten, and Shira Kritchman. The convergence rate of neural networks for learned functions of different frequencies. *Advances in Neural Information Processing Systems*, 32, 2019.

David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network with approximation error being reciprocal of width to power of square root of depth. *Neural Computation*, 2021a.

Zuowei Shen, Haizhao Yang, and Shijun Zhang. Neural network approximation: Three hidden layers are enough. *Neural Networks*, 141:160–173, 2021b.

Marvin Shinbrot. *Lectures on fluid mechanics*. Courier Corporation, 2012.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Trevor Stephens. gplearn: Genetic programming in python, 2017. URL `https://github.com/trevorstephens/gplearn`. Version 0.4.2.

Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

E Weinan, Jiequn Han, and Arnulf Jentzen. Algorithms for solving high dimensional pdes: From nonlinear monte carlo to machine learning. *Nonlinearity*, 35(1):278, 2021.

Zhi-Qin John Xu, Yaoyu Zhang, Tao Luo, Yanyang Xiao, and Zheng Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *Communications in Computational Physics*, 28(5):1746–1767, 2020. ISSN 1991-7120. doi: https://doi.org/10.4208/cicp.OA-2020-0085. URL `http://global-sci.org/intro/article_detail/cicp/18395.html`.

Dmitry Yarotsky. Elementary superexpressive activations. *ArXiv*, abs/2102.10911, 2021.

Dmitry Yarotsky and Anton Zhevnerchuk. The phase diagram of approximation rates for deep neural networks. *Advances in neural information processing systems*, 33:13005–13015, 2020.

Yulei, Liao, , 14603, , Yulei Liao, Pingbing, Ming, , 10035, , and Pingbing Ming. Deep nitsche method: Deep ritz method with essential boundary conditions. *Communications in Computational Physics*, 29(5):1365–1384, 2021. ISSN 1991-7120. doi: https://doi.org/10.4208/cicp.OA-2020-0219. URL `http://global-sci.org/intro/article_detail/cicp/18717.html`.

Yaohua Zang, Gang Bao, Xiaojing Ye, and Haomin Zhou. Weak adversarial networks for high-dimensional partial differential equations. *Journal of Computational Physics*, 411: 109409, 2020.

Shijun Zhang, Zuowei Shen, and Haizhao Yang. Deep network approximation: Achieving arbitrary accuracy with fixed number of neurons. *Journal of Machine Learning Research*, 23(276):1–60, 2022.

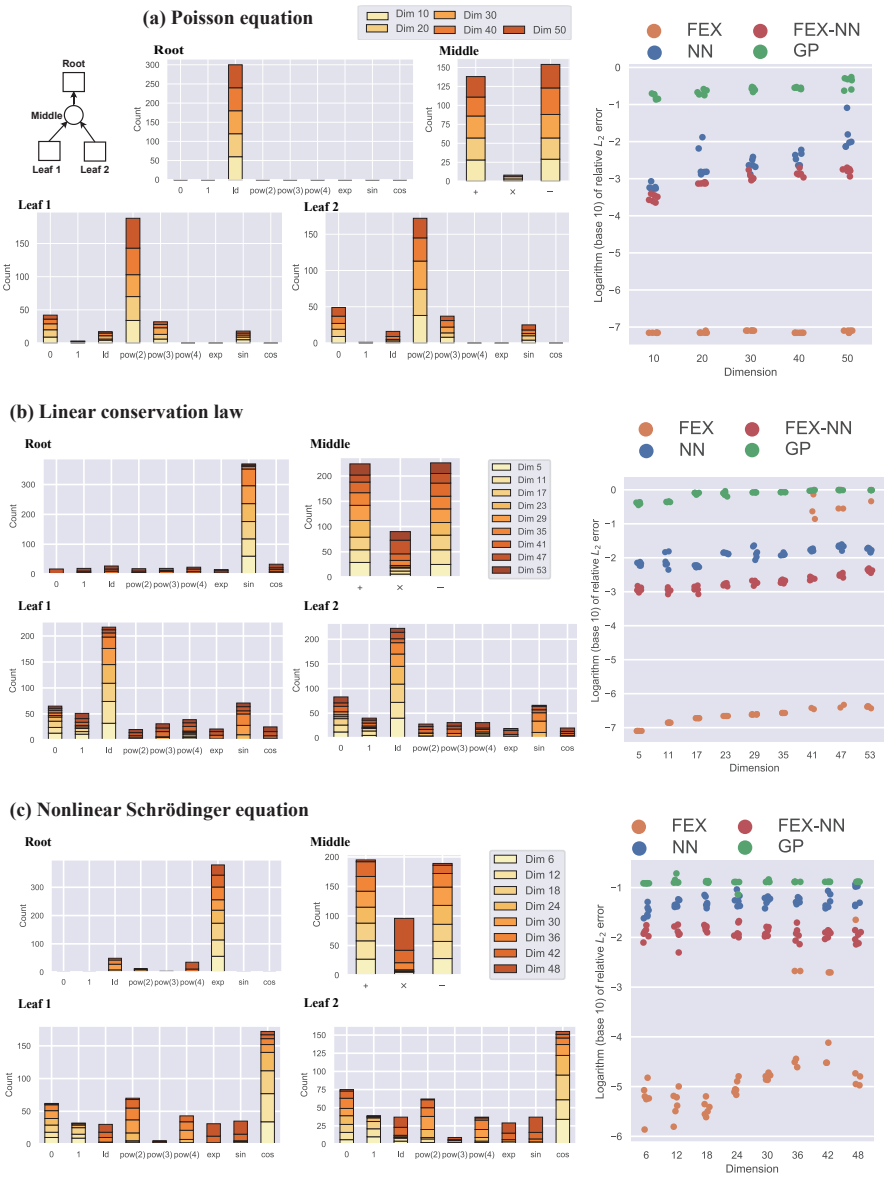

Figure 4: Distribution of the node values and the error comparison. In FEX, we search the optimal sequence of node values and show the frequency of the node values of the binary tree in the candidate pool, consisting of the root (Root), middle node (Middle), and two leaves (Leaf 1 and Leaf 2). Based on the observation of the distribution, we readily design the new NN parameterization (FEX-NN) to estimate the solution. The last column displays comparison of the relative L^2 error as the function of the dimension over 6 independent trials between FEX, NN, FEX-NN and Genetic Programming (GP) for various high-dimensional PDE problems. Rows (a), (b) and (c) represent the results for Poisson equation (18), Linear conservation law (19) and Nonlinear Schrödinger equation (21) respectively. For various dimensions, FEX identifies the true solution, approximating solutions of almost the machine accuracy.

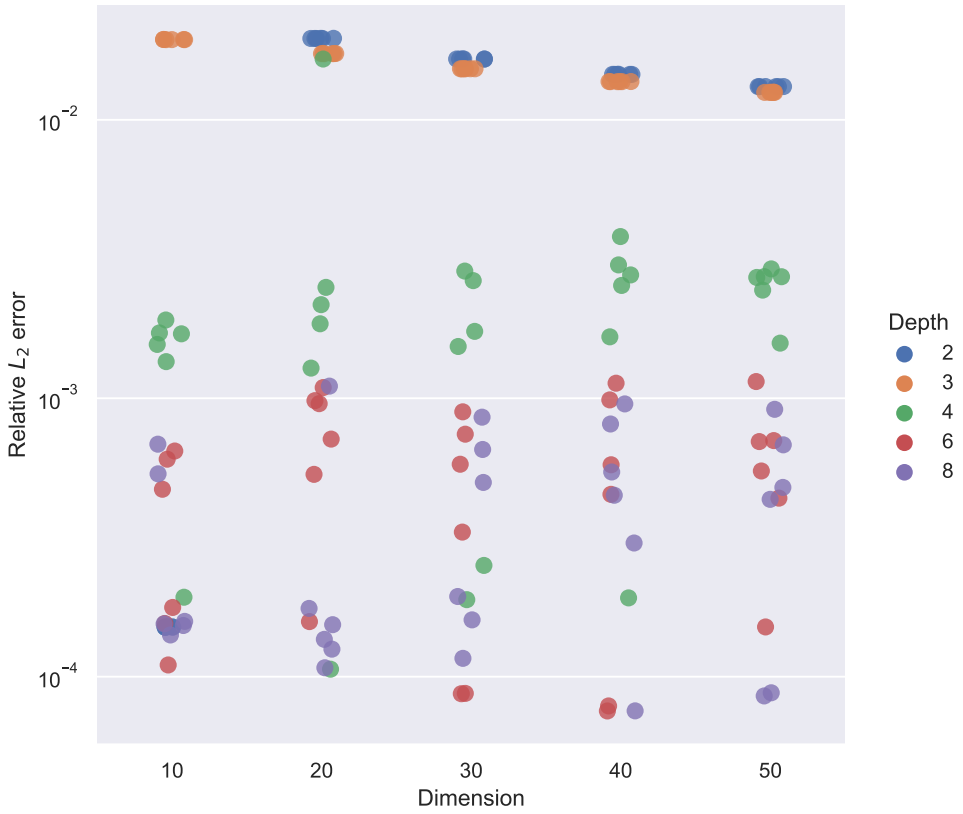

Figure 5: Relative L_2 error of solutions estimated by trees with increasing depth for the problems of various dimensions. We exclude the square operator $(\cdot)^2$ in the unary set $\mathbb{U}$, and the binary tree defined in Section 3.1 can not reproduce the true solution (sum of the square of coordinates)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

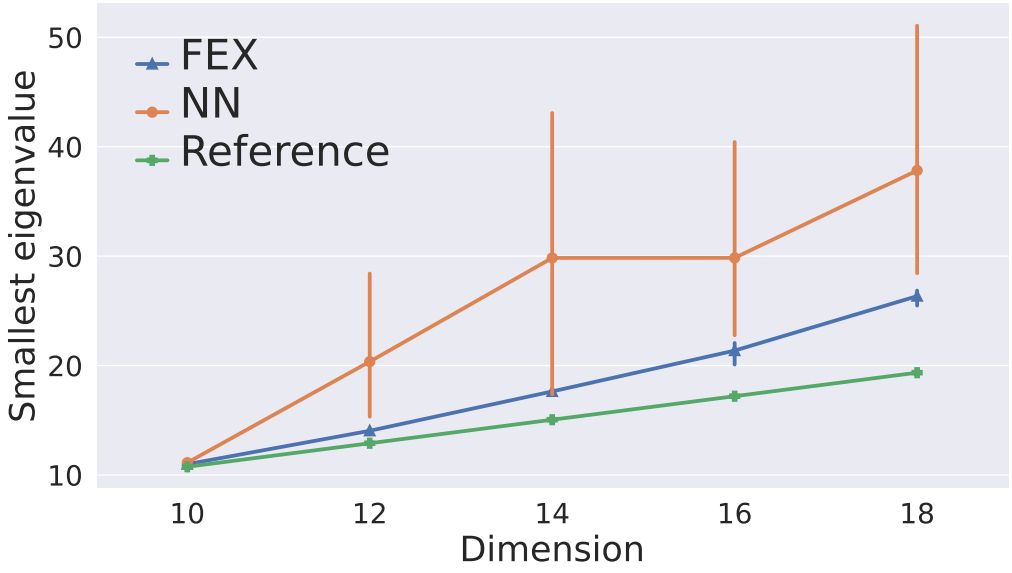

Figure 6: Comparison of the FEX method and the NN approach for approximating the smallest eigenvalue of (24) using the Rayleigh quotient with the given potential $w(\boldsymbol{x}) = \|\boldsymbol{x}\|_2^2 + \delta \sum_{i=1}^{d} x_i^4$ and $\delta = 0.1$ across different dimensions. The vertical line segments represent one standard deviation.

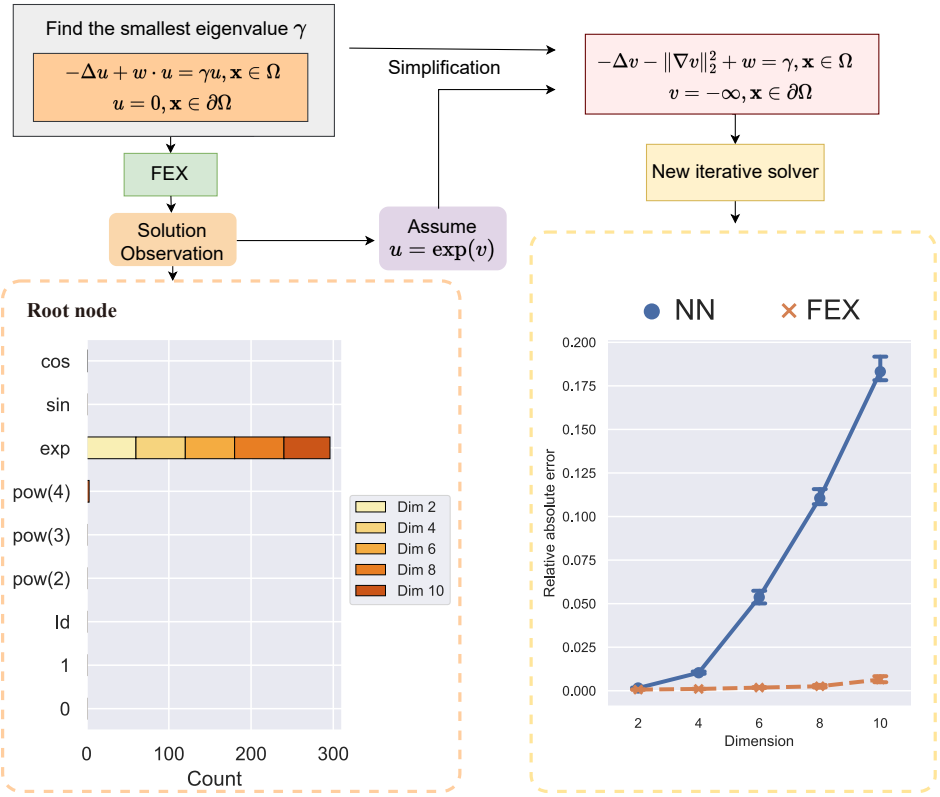

Figure 7: Eigenvalue problems and postprocessing algorithm design with FEX. Bottom left: We observed that the exponent operator "$\exp(\cdot)$" dominates the tree root in the FEX searching loop. Based on this observation, we assume the solution is $\exp(v(\boldsymbol{x}))$ and simplify the original PDE 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.

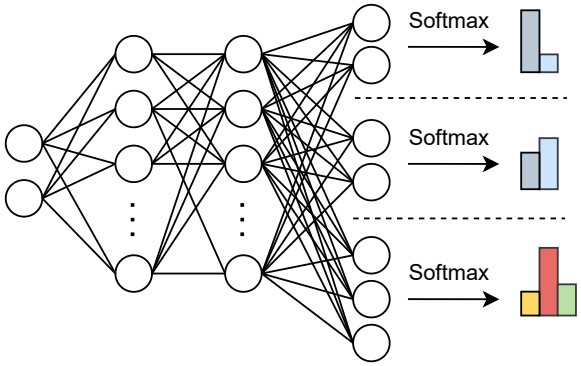

Figure 8: Illustration of using a fully connected neural network to model the controller that outputs the probability mass functions for each of the node within a tree. As an example, the tree contains three nodes: $n_1 = 2$ nodes for the binary set with $|\mathbb{B}| = 2$ operators and $n_2 = 1$ nodes for the unary set with $|\mathbb{U}| = 3$ operators.