International Journal of Numerical Modelling: Electronic Networks, Devices and Fields

WILEY

RESEARCH ARTICLE

Multilayer Neural Networks Enhanced With Hybrid Methods for Solving Fractional Partial Differential Equations

Amina Hassan Ali[1,2] | Norazak Senu[1,3] | Ali Ahmadian[4,5]

[1]Department of Mathematics and Statistics, Universiti Putra Malaysia, Serdang, Malaysia | [2]Department of Mathematics, College of Education, University of Sulaimani, Sulaymaniyah, Iraq | [3]Institute for Mathematical Research, Universiti Putra Malaysia, Serdang, Malaysia | [4]Faculty of Engineering and Natural Sciences, Istanbul Okan University, Istanbul, Turkey | [5]Jadara University Research Center, Jadara University, Irbid, Jordan

Correspondence: Norazak Senu (norazak@upm.edu.my) | Ali Ahmadian (ahmadian.hosseini@gmail.com)

Received: 19 March 2025 | **Revised:** 3 June 2025 | **Accepted:** 13 June 2025

Funding: The authors are very thankful to Malaysia Ministry of Education for awarding the Fundamental Research Grant Scheme (Ref. No. FRGS/1/2022/STG06/UPM/02/2) for supporting this work.

Keywords: Adam algorithm | deep neural network | fractional partial differential equations | Laplace transform method | limited-memory Broyden-Fletcher-Goldfarb-Shanno

ABSTRACT

This paper introduces a novel multilayer neural network technique to solve partial differential equations with non-integer derivatives (FPDEs). The proposed model is a deep feed-forward multiple layer neural network (DFMLNN) that is trained using advanced optimization approaches, namely adaptive moment estimation (Adam) and limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS), which integrate neural networks. First, the Adam method is employed for training, and then the model is further improved using L-BFGS. The Laplace transform is used, concentrating on the Caputo fractional derivative, to approximate the FPDE. The efficacy of this strategy is confirmed through rigorous testing, which involves making predictions and comparing the outcomes with exact solutions. The results illustrate that this combined approach greatly improves both precision and effectiveness. This proposed multilayer neural network offers a robust and reliable framework for solving FPDEs.

1 | Introduction

Over the past several years, Fractional Calculus (FC), which encompasses the examination of fractional derivatives and integrals, has had a substantial influence on various academic disciplines [1, 2]. Over time, the scope of FC applications has significantly broadened, covering diverse domains in both science and engineering, as outlined in a comprehensive review [3]. One significant benefit of FC is its capacity to illustrate how contemporary phenomena are influenced by their complete history, known as the memory effect. FC is particularly valuable for representing intricate nonlinear events and higher-order dynamics, offering more flexibility compared to conventional Leibniz derivatives [4]. Researchers can enhance the flexibility in analyzing specific mathematical models by altering the order of the fractional operator from whole number values to fractional values.

Recently, academics have shown considerable interest in FPDEs because of their wide range of applications in science and engineering [5]. The increasing interest in models incorporating fractional order derivatives, whether in time, space, or both, stems from their ability to more precisely and effectively reflect a wide range of natural events. Conventional integer-order partial differential equations (PDEs) forecast the future states of physical systems exclusively on their present states, without considering past data. However, when it comes to effectively representing materials and processes that possess memory and hereditary qualities, where past states have a substantial influence

on future behavior, integer-order models are inadequate. FPDEs are particularly useful in situations like this, as they offer more efficient methods for modeling systems.

The scarcity of explicit analytic solutions for fractional order PDEs in existing literature emphasizes the necessity for effective and dependable numerical approaches. Several sophisticated numerical methods have been devised to estimate solutions to these equations. As an illustration, [6] utilized a modified Adomian decomposition method to solve fractional diffusion-wave equations. In a similar manner, [7] employed the homotopy perturbation method to get approximate analytical solutions for Korteweg-de Vries (KdV) equations with fractional orders. Additional research, such as the studies conducted by [8, 9], utilized the generalized differential transform method to solve linear fractional PDEs. In contrast, [10] suggested a finite difference scheme to address equations involving time- and space-fractional derivatives.

Recent studies have employed the reproducing kernel algorithm to solve various classes of time-fractional FPDEs. Reference [11] addressed Robin boundary conditions in heat and fluid flow problems, while [12] extended the approach to general time-FPDEs in fluid dynamics. The method was also applied to Dirichlet-type diffusion-Gordon equations in porous media [13]. A computational technique equipped with error estimates has been proposed to address singular Fredholm time-fractional partial integro-differential equations [14].

Artificial neural networks (ANNs) have been increasingly popular for solving differential equations (DEs) in recent years, particularly FPDEs. The field has made significant progress thanks to the development of deep learning approaches that employ deep neural networks [15–17]. These networks are highly proficient in extracting implicit information of different kinds and have achieved exceptional success in various scientific and engineering fields, including image classification [18], natural language processing [19], and fault detection [20]. The universal approximation theorem states that a multilayer feed-forward network may estimate any continuous function with arbitrary precision, given that it has a sufficient number of hidden neurons [21, 22]. Therefore, neural networks provide significant benefits in the process of fitting functions.

Multiple researches have shown the successful application of ANNs in solving FPDEs. As an illustration, a particular study utilized neural networks with radial basis functions to tackle the problem of fractional heat conduction and wave equations [23]. A recent study implemented a neural network that utilizes sine and cosine functions to effectively address FPDEs [24]. In addition, researchers built a neural network that utilizes Legendre polynomials to solve equations related to fractional diffusion in space and time [25]. The study introduced a neural network that deals with the spatiotemporal variable-order fractional advection–diffusion equation, incorporating a nonlinear source term. The network employs shifted Legendre orthogonal polynomials with changeable coefficients as a computational tool [26]. The authors suggested a Monte Carlo fractional physics-informed neural network (fPINNs) to solve forward and inverse FPDEs using a machine learning approach based on sampling. A recent study presented a novel NN methodology to solve the time-fractional Fokker–Planck equation [27]. The study utilized the SLeNN-ELM approach, which is a modified version of the Legendre neural network method, to address fractional DEs that involve constant and proportional delays [28]. The study conducted more research on a neural network model for the spatio-temporal fractional-order nonlinear reaction-advection-diffusion equation. This model utilized shifted Legendre orthogonal polynomials with variable coefficients as a mathematical tool [29]. The time difference PINN was created to estimate solutions for fractional water wave models [29]. An innovative approach for solving FPDEs utilizing ANNs has been recently presented [30]. A deep learning approach for solving FPDEs is developed in [31].

This article introduces a novel approach for solving FPDEs in the Caputo sense by employing a deep feed-forward ANN architecture. The proposed method stands out by incorporating a hybrid optimization strategy that combines the fast convergence of the Adam optimizer with the fine-tuning capabilities of the L-BFGS algorithm. This innovative fusion ensures both precision and effectiveness in solving FPDEs. Furthermore, by utilizing the Laplace transform methodology, our approach accurately approximates the Caputo-type space-fractional derivative, resulting in robust and reliable solutions. Multiple examples demonstrate the effectiveness of our method, highlighting its exceptional precision and efficiency. The key contributions of this work, which highlight these advancements, are as follows:

- Introduction of a novel deep feed-forward ANN architecture with two hidden layers specifically designed to approximate solutions of FPDEs in the Caputo sense.
- Development of an innovative hybrid optimization technique that combines the rapid convergence of the Adam method with the precise adjustments of the L-BFGS algorithm. To our knowledge, this represents the first use of such a combined approach for solving FPDEs.
- Application of the Laplace transform technique to accurately approximate the Caputo-type space-fractional derivative.
- Demonstration of the proposed DFMLNN model and hybrid optimization method's effectiveness and superiority through various examples, including comparisons with exact solutions to illustrate its accuracy.

The paper's structure is arranged as follows: Section 2 offers a concise overview of the fundamental definitions that are pertinent to our research. Section 3 provides a comprehensive explanation of the building of the DFMLNN model. This includes the establishment of its architecture, the framework for FPDE construction, and the hybrid training approach. Section 4 provides multiple illustrative solutions along with their respective outcomes. Section 5 provides a concise overview of the paper's contributions and findings.

2 | Preliminary Concepts

This section outlines the definitions of the fractional derivatives used in this study and includes the Laplace transform applied to the Caputo fractional derivative.

Definition 2.1. For an order $\alpha > 0$, the Caputo derivative is defined as follows, according to [32, 33]:

$$ {}_a^C D_x^\alpha \phi(x) = \begin{cases} \dfrac{1}{\Gamma(s-\alpha)} \displaystyle\int_a^x \phi^{(s)}(x)(x-l)^{s-\alpha-1} dl, & for\ s-1<\alpha<s, \\ \dfrac{d^s}{dx^s}\phi(x), & for\ \alpha = s. \end{cases} \tag{1} $$

where s is a natural number.

Definition 2.2. Given the function p defined for t in the non-negative domain, the Laplace transform of p, denoted by $\mathscr{L}\{p\}$, is defined through the improper integral, as outlined in [34]:

$$ \mathscr{L}\{p(t)\} = P(s) = \int_0^\infty e^{-st} p(t) dt, \tag{2} $$

provided that the integral in (2) exists and is convergent. The inverse Laplace transform is defined as:

$$ \mathscr{L}^{-1}\{P(s)\} = \frac{1}{2\pi i} \int_{c-i\infty}^{c+i\infty} e^{st} P(s) ds. \tag{3} $$

Definition 2.3. The Laplace transform of the Caputo fractional derivative of order α is given in [35] as follows:

$$ \mathscr{L}\left\{ {}_0^C D_t^\alpha p(t) \right\} = s^\alpha \mathscr{L}\{p(t)\} - \sum_{i=0}^{l-1} s^{\alpha-i-1} \left({}_0 D_t^i p \right)(0). \tag{4} $$

3 | DFMLNN Model Construction

This section focuses on employing the DFMLNN model to address the FPDEs. We start with the framework for DFMLNN construction, detailing the essential framework needed to approach FPDEs with DFMLNN. Subsequently, we introduce a hybrid optimization method developed based on this framework.

3.1 | Framework for DFMLNN Construction

This subsection focuses on the utilization of a DFMLNN. The architecture of the DFMLNN model designed for solving FPDEs is illustrated in Figure 1. Below is a detailed description of each component within this architecture.

Input layer: The input layer consists of two nodes, labeled x and t.

First hidden layer (H1): This layer comprises k neurons, each labeled $N_n^{[1]}$, where n ranges from 1 to k. The neurons in this layer receive weighted inputs from the input layer. The weight from the input node x to the n-th neuron in the first hidden layer is denoted as $w_{n1}^{[1]}$, and the weight from t is denoted as $w_{n2}^{[1]}$. Each neuron also includes a bias term $b_n^{[1]}$.

Second hidden layer (H2): This layer contains k neurons, each denoted as $N_m^{[2]}$, where m ranges from 1 to k. Neurons in this layer are connected to those in the first hidden layer through weights $w_{mn}^{[2]}$, where $w_{mn}^{[2]}$ connects the n-th neuron of the first hidden layer to the m-th neuron of the second hidden layer. Each neuron in this layer also has a bias term $b_m^{[2]}$.

Output layer: The output layer consists of a single neuron, $N_1^{[3]}$, which aggregates the outputs of the second hidden layer using weights $w_{1m}^{[3]}$. This neuron sums the contributions from each neuron $N_m^{[2]}$ in the second hidden layer.

$N(x,t,p)$: The computation through the network layers is performed as follows:

$$ N(x,t,p) = \sum_{m=1}^{k} w_{1m}^{[3]} N_m^{[2]}, \tag{5} $$

where

$$ N_m^{[2]} = \phi\left(\sum_{n=1}^{k} w_{mn}^{[2]} N_n^{[1]} + b_m^{[2]} \right), \tag{6} $$

$$ N_n^{[1]} = \phi\left(w_{n1}^{[1]} x + w_{n2}^{[1]} t + b_n^{[1]} \right), \tag{7} $$

and the activation function ϕ is the hyperbolic tangent (tanh).

FPDEs processing: The output from $N_1^{[3]}$ is utilized alongside predetermined derivatives $(\hat{u}, \hat{u}_t, \hat{u}_{xx})$ to solve FPDEs, represented as:

$$ D_x^\alpha \hat{u} = H\left(x, t, \hat{u}_t, \hat{u}_{xx}\right). \tag{8} $$

Output decision and iterations: The solution derived from the FPDEs is evaluated using a cost function $C(p)$ and continues until the maximum number of iteration is reached. p denotes the parameters of the DFMLNN.

3.2 | Framework for FPDE Construction

Consider the following FPDEs:

$$ D_x^\alpha u(x,t) = H\left(x, t, u_t, u_{xx}\right), \quad 0 \le x \le 1, \quad 0 \le t \le 1, \quad 0 < \alpha \le 1, \tag{9} $$

$$ u(x,0) = g_0(x), u(0,t) = g_1(t), \quad u(1,t) = g_2(t) \tag{10} $$

here $D_x^\alpha u(x,t)$ denotes the Caputo fractional derivative. The function $H\left(x,t,u_t,u_{xx}\right)$ represents a nonlinear source term, while Equation (10) gives the necessary initial and boundary conditions. The initial condition $u(x,0) = g_0(x)$ defines the state of the solution at time $t = 0$, while the boundary conditions

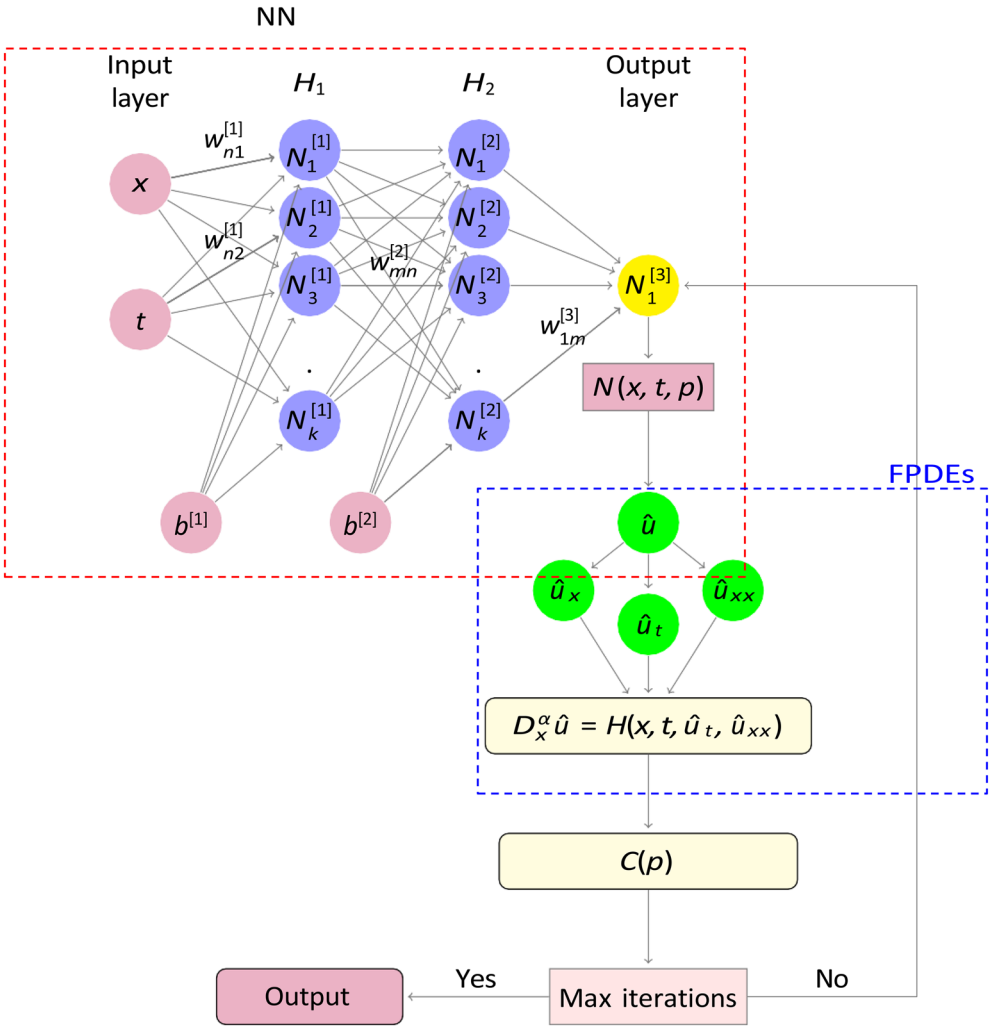


FIGURE 1 | Architecture of DFMLNN.

ALGORITHM 1 | L-BFGS two-loop recursion [38].

$g \leftarrow \nabla_p C(p_k)$
for $i = k-1, k-2, \ldots, k-m$ **do**
  $\alpha_i \leftarrow \rho_i s_i^T g$
  $g \leftarrow g - \alpha_i y_i$
end for
$r \leftarrow H_k^0 g$
for $i = k-m, k-m+1, \ldots, k-1$ **do**
  $\beta \leftarrow \rho_i y_i^T r$
  $r \leftarrow r + s_i(\alpha_i - \beta)$
end for
stop with result $H_k \nabla_p C(p_k) = r$

$u(0,t) = g_1(t)$ and $u(1,t) = g_2(t)$ prescribe the solution behavior at the spatial endpoints $x = 0$ and $x = 1$, respectively. These conditions together ensure that the problem is well-posed. Let's denote the DFMLNN of problem (9) by $\hat{u}(X,p)$. The DFMLNN framework consists of two main parts: The first part ensures the initial condition of problem (9) is met, while the second part comprises the output of the model (11), denoted by $N(X,p)$, which encompasses all the network's weights and biases. Therefore, the DFMLNN formulation, inspired by [36], can be described as follows:

$$\hat{u}(X,p) = A(x,t) + x(1-x)tN(X,p) \tag{11}$$

where $X = (x,t)^T$, $N(X,p)$ is the output of DFMLNN, p is the parameters of DFMLNN, and

$$A(x,t) = (1-x)g_1(t) + xg_2(t) + g_0(x) - (1-x)g_0(0) - xg_0(1) \tag{12}$$

To begin, we utilize the Laplace transform technique to approximate the Caputo-type space-fractional derivative:

$$\begin{aligned}\mathcal{L}\left\{\frac{\partial^\alpha u(x,t)}{\partial x^\alpha}\right\} &= p^\alpha \overline{u}(p,t) - p^{\alpha-1}u(0,t) \\ &= s^\alpha\left[\overline{u}(p,t) - p^{-1}u(0,t)\right]\end{aligned} \tag{13}$$

where $\overline{u}(p,t)$ represents the Laplace transform of $u(x,t)$. Given that $0 < \alpha < 1$, we can linearize the term p^α as follows:

$$p^\alpha \approx \alpha p + (1-\alpha) \tag{14}$$

Next, we substitute this linearized term into (13), resulting in:

$$\begin{aligned}\mathcal{L}\left\{\frac{\partial^{\alpha}u(x,t)}{\partial x^{\alpha}}\right\} &\approx (\alpha p+(1-\alpha))\left[\overline{u}(p,t)-p^{-1}u(0,t)\right]\\ &=\alpha p\left[\overline{u}(p,t)-p^{-1}u(0,t)\right]+(1-\alpha)\left[\overline{u}(p,t)-p^{-1}u(0,t)\right]\end{aligned} \tag{15}$$

Thus, the inverse Laplace transform gives us:

$$\frac{\partial^{\alpha}u(x,t)}{\partial x^{\alpha}} \approx \alpha\frac{\partial u(x,t)}{\partial x}+(1-\alpha)[u(x,t)-u(0,t)] \tag{16}$$

So (9) can be rewritten as:

$$\begin{aligned}&\alpha\frac{\partial u(x,t)}{\partial x}+(1-\alpha)[u(x,t)-u(0,t)]=H\left(x,t,u_t,u_{xx}\right),\\ &u(x,0)=g_0(x),\quad u(0,t)=g_1(t),\quad u(1,t)=g_2(t)\end{aligned} \tag{17}$$

To optimize the DFMLNN, we define a cost function as follows:

$$C(p)=\frac{1}{2}\sum_{i=1}^{S}\left(D_x^{\alpha}\hat{u}(x,t)-H\left(x,t,\hat{u}_t,\hat{u}_{xx}\right)\right)^2 \tag{18}$$

here, S represents the total count of discretized sample points within the domain. By employing optimization algorithms to minimize this cost function, we can fine-tune the weights and biases of the DFMLNN to accurately approximate the solution of the specified FPDEs.

3.3 | Hybrid Model Training Approach

This subsection outlines the hybrid optimization strategy used in our work, which combines the strengths of the Adam method and the L-BFGS algorithm. This two-stage approach utilizes Adam's fast convergence for the initial training phase and L-BFGS's high precision for the fine-tuning phase, ensuring both efficiency and accuracy in training neural networks.

3.3.1 | Phase 1: Adam Optimizer

The Adam approach, proposed by [37], integrates the benefits of two other improvements to stochastic gradient descent: the adaptive gradient algorithm and root mean square propagation. Adam calculates distinct adaptive learning rates for various parameters by approximating the first and second moments of the gradients.

Initialization:

1. Parameter initialization: The neural network parameters p are randomly initialized to ensure a diverse starting point.

2. Adam hyperparameters:

- Learning rate (η): Controls the step size during the gradient descent process.
- First moment decay rate (β_1): Determines the pace at which the first moment estimations decay exponentially.
- Second moment decay rate (β_2): Specifies the rate at which the second moment estimations decay exponentially.

3. Moment vectors initialization: The initial values for the first and second moment vectors are set to zero: $m_0=0$ and $v_0=0$. This initialization ensures that the initial estimates are unbiased and provides a solid foundation for subsequent updates.

Initial training with Adam: The Adam algorithm begins the training process by effectively managing large-scale optimization through adjusting the learning rate for each parameter. The procedure is as follows:

First moment estimate update: Refine the biased first moment estimate to incorporate the new gradient:

$$m_t=\beta_1 m_{t-1}+\left(1-\beta_1\right)g_t. \tag{19}$$

Second moment estimate update: Refine the biased second moment estimate to incorporate the new gradient's square:

$$v_t=\beta_2 v_{t-1}+\left(1-\beta_2\right)g_t^2 \tag{20}$$

Bias-corrected first moment estimate: Adjust the first moment estimate to correct for bias introduced during initialization:

$$\hat{m}_t=\frac{m_t}{1-\beta_1} \tag{21}$$

Bias-corrected second moment estimate: Adjust the second moment estimate to correct for bias introduced during initialization:

$$\hat{v}_t=\frac{v_t}{1-\beta_2} \tag{22}$$

Parameter update: Implement the adjustment to the parameters, guiding them toward minimizing the loss function:

$$p_t=p_{t-1}-\eta\frac{\hat{m}_t}{\sqrt{\hat{v}_t}+\epsilon} \tag{23}$$

here, ϵ is a small constant (e.g., 10^{-8}) to prevent division by zero. This step applies the refined updates to the parameters, steering them towards minimizing the loss function.

After the Adam optimization phase is completed, the final parameters, denoted as p_{Adam}, are stored for the subsequent phase. These parameters act as the starting point for the L-BFGS optimization phase, ensuring a well-converged initial solution.

3.3.2 | Phase 2: L-BFGS Optimization

The L-BFGS method is a limited-memory quasi-Newton approach designed for unconstrained optimization, particularly suitable for problems involving numerous variables. It approximates the BFGS algorithm while using a restricted amount of memory [38].

Transition to L-BFGS: The parameters obtained from the Adam optimization phase, denoted as p_{Adam}, serve as the initial parameters for the L-BFGS optimizer:

ALGORITHM 2 | DFMLNN algorithm.

Input: Prepare the training data $\{(x_i, t_i)\}_{i=1}^{n}$, initial parameters p_0, learning rate η, Adam hyperparameters $(\beta_1, \beta_2, \epsilon)$, maximum iterations T_{Adam} and $T_{\text{L-BFGS}}$
Output: $\hat{u}(x,t)$
Define the Problem:
Given a FPDEs of the form (17)
Design the DFMLNN architecture by determining the number of layers and neurons in each layer.
Establish the cost function as in (18)
Phase 1: Adam Method
Initialize $m_0 = 0$ and $v_0 = 0$
for $t = 1$ to T_{Adam} **do**
    Compute the gradient: $g_t = \nabla_p C(p_{t-1})$
    Update parameters using (23)
end for
$p_{\text{Adam}} = p_{T_{\text{Adam}}}$
Phase 2: L-BFGS method [38]
Use the parameters from Adam as the starting point: $p_0^{\text{L-BFGS}} = p_{\text{Adam}}$, and integer $m > 0$.
for $k = 0$ to $T_{\text{L-BFGS}}$ **do**
    Initialize $H_0 = \gamma_k I$
    Determine search direction: $j_k = -H_k g_k$ from **Algorithm 1**
    Update parameters using (35), where α_k is chosen to satisfy the Wolfe conditions
    if $k > m$ **then**
        Discard the pair $\{s_{k-m}, y_{k-m}\}$ from storage
        break
    end if
    Compute and save: $s_k = p_{k+1} - p_k$, $y_k = g_{k+1} - g_k$
    $k = k + 1$
end for
Return $\hat{u}(x,t)$

$$p_0^{\text{L-BFGS}} = p_{\text{Adam}}$$

This transition capitalizes on the rapid convergence of Adam and the precision of L-BFGS, effectively merging the advantages of both optimization techniques.

Fine-tuning with L-BFGS: The L-BFGS optimizer fine-tunes the parameters by utilizing a second-order approximation of the inverse Hessian matrix, which captures the curvature of the loss function, enabling more accurate parameter updates. The detailed procedure is as follows:

Initialization:

- The initial inverse Hessian approximation H_0 is computed using the scaling factor γ_k:

$$H_0 = \gamma_k I, \tag{24}$$

where

$$\gamma_k = \frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}, \tag{25}$$

TABLE 1 | Comparison of exact and DFMLNN solutions for Example 4.1 with varying values of α.

		DFMLNN		
(x,t)	Exact	$\alpha = 1$	$\alpha = 0.99$	$\alpha = 0.85$
(0,0)	0	0	0	0
(0.1,0.1)	0.020000	0.020004	0.019958	0.019469
(0.2,0.2)	0.080000	0.0800009	0.079899	0.078999
(0.3,0.3)	0.180000	0.179993	0.179910	0.179396
(0.4,0.4)	0.320000	0.319991	0.320011	0.320729
(0.5,0.5)	0.500000	0.499996	0.500166	0.502731
(0.6,0.6)	0.720000	0.720001	0.720324	0.725040
(0.7,0.7)	0.979999	0.980001	0.980448	0.987100
(0.8,0.8)	1.280000	1.280001	1.280502	1.287974
(0.9,0.9)	1.619999	1.620001	1.620407	1.626233
(1,1)	2.000000	2.000000	2.000000	2.000000

TABLE 2 | Absolute errors for $\alpha = 1$ in Example 4.1.

(x,t)	AEs
(0,0)	0
(0.1,0.1)	4.85E-06
(0.2,0.2)	9.83E-07
(0.3,0.3)	6.95E-06
(0.4,0.4)	8.22E-06
(0.5,0.5)	3.39E-06
(0.6,0.6)	1.01E-06
(0.7,0.7)	1.66E-06
(0.8,0.8)	9.53E-07
(0.9,0.9)	1.66E-06
(1,1)	0

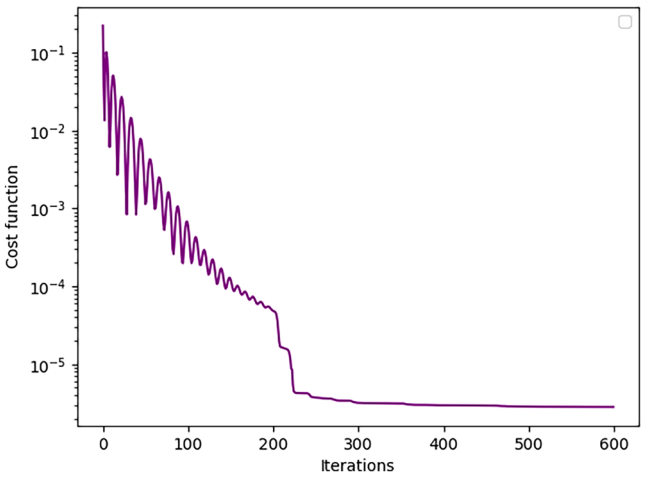
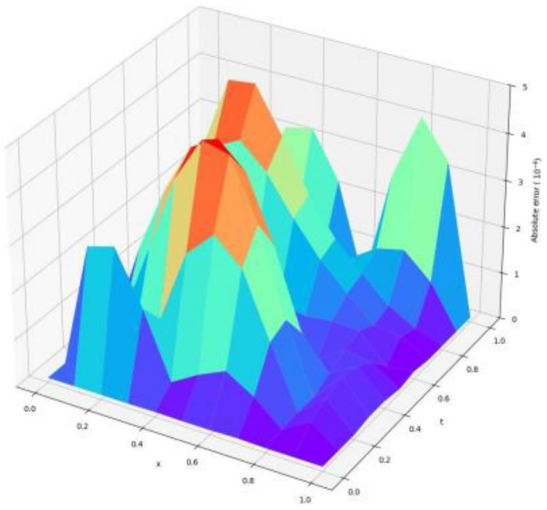
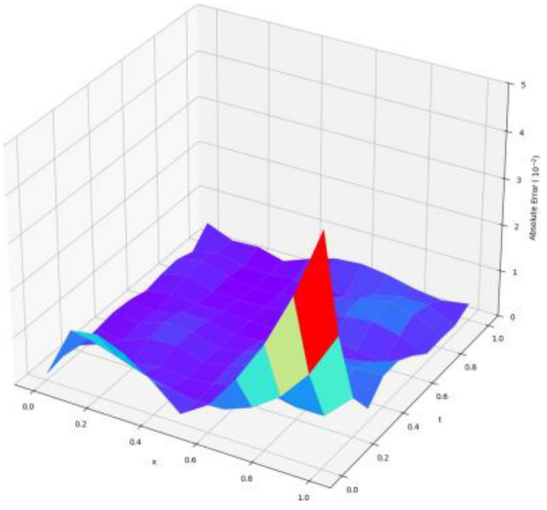


FIGURE 2 | Convergence of the cost function at $\alpha = 0.99$ for Example 4.1.

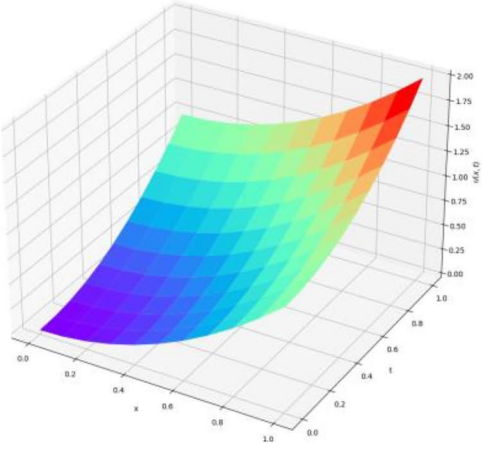


(a) Absolute error at $\alpha = 1$.

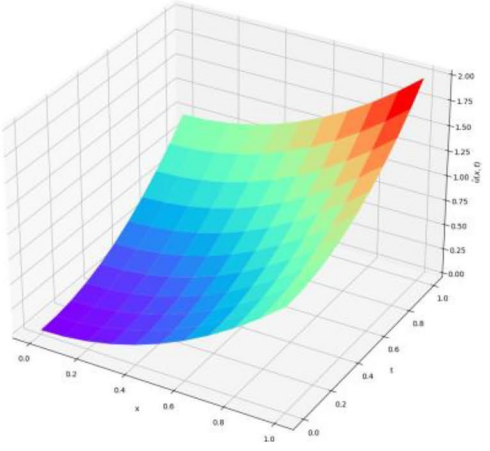


(b) Absolute error of cost function at $\alpha = 0.99$.

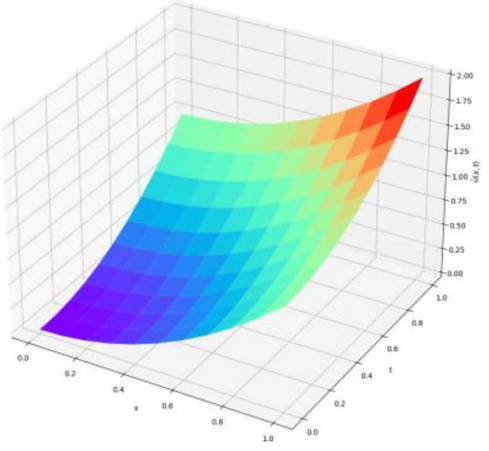
FIGURE 3 | Absolute error at $\alpha = 1$, and absolute error of cost function at $\alpha = 0.99$ for Example 4.1.



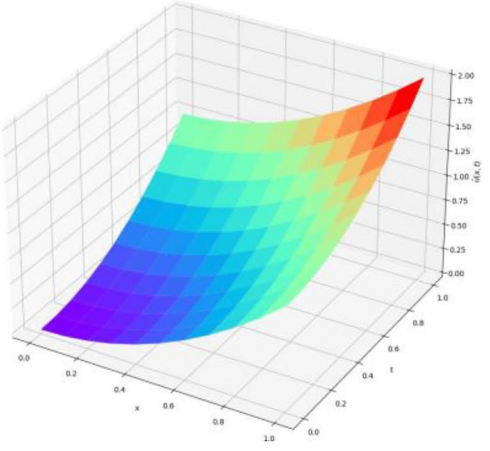
(a) Exact solution.



(b) DFMLNN solution at $\alpha = 1$.



(c) DFMLNN solution at $\alpha = 0.99$.



(d) DFMLNN solution at $\alpha = 0.85$.

FIGURE 4 | Visualization of the exact and numerical DFMLNN solutions at varying α values for Example 4.1.

TABLE 3 | Comparison of exact and DFMLNN solutions for Example 4.2 with varying values of α.

(x,t)	Exact	DFMLNN		
		$\alpha = 1$	$\alpha = 0.99$	$\alpha = 0.7$
(0,0)	0	0	0	0
(0.1,0.1)	0.011000	0.011009	0.011228	0.014798
(0.2,0.2)	0.048000	0.047997	0.047825	0.057486
(0.3,0.3)	0.116999	0.116983	0.116101	0.135133
(0.4,0.4)	0.224000	0.223989	0.222560	0.251674
(0.5,0.5)	0.375000	0.375004	0.373133	0.411685
(0.6,0.6)	0.576000	0.576013	0.573801	0.623170
(0.7,0.7)	0.833000	0.833008	0.830932	0.905996
(0.8,0.8)	1.152000	1.152000	1.150640	1.270791
(0.9,0.9)	1.538999	1.53899	1.538313	1.597327
(1,1)	2.000000	2.000000	2.000000	2.000000

here, s_{k-1} is the difference in parameters and y_{k-1} is the difference in gradients between two successive iterations.

Gradient computation: Evaluate the gradient g_k of the cost function with respect to the current parameters p_k:

$$g_k = \nabla_p C(p_k). \tag{26}$$

Search direction: Identify the direction j_k for adjusting the parameters. This direction is determined by using the L-BFGS two-loop recursion:

$$j_k = -H_k \nabla_p C(p_k). \tag{27}$$

The L-BFGS two-loop recursion method is as follows:

Line search: Perform a line search to determine the step length α_k that satisfies the Wolfe conditions:

$$C(p_k + \alpha_k j_k) \le C(p_k) + c_1 \alpha_k \nabla_p C^T(p_k) j_k \tag{28}$$

$$\nabla_p C(p_k + \alpha_k j_k)^T j_k \ge c_2 \nabla_p C^T(p_k) j_k \tag{29}$$

with $0 < c_1 < c_2 < 1$

Evaluating differences: Evaluate the changes in parameters and gradients:

$$s_k = p_{k+1} - p_k \tag{30}$$

$$y_k = g_{k+1} - g_k \tag{31}$$

Hessian update factor: Determine the factor ρ_k to be used in updating the Hessian approximation:

$$\rho_k = \frac{1}{y_k^T s_k} \tag{32}$$

TABLE 4 | Absolute errors for $\alpha = 1$ in Example 4.2.

(x,t)	AEs
(0,0)	0
(0.1,0.1)	9.00E-06
(0.2,0.2)	2.99E-06
(0.3,0.3)	1.63E-05
(0.4,0.4)	1.08E-05
(0.5,0.5)	4.70E-06
(0.6,0.6)	1.30E-06
(0.7,0.7)	8.58E-06
(0.8,0.8)	1.19E-07
(0.9,0.9)	0
(1,1)	0

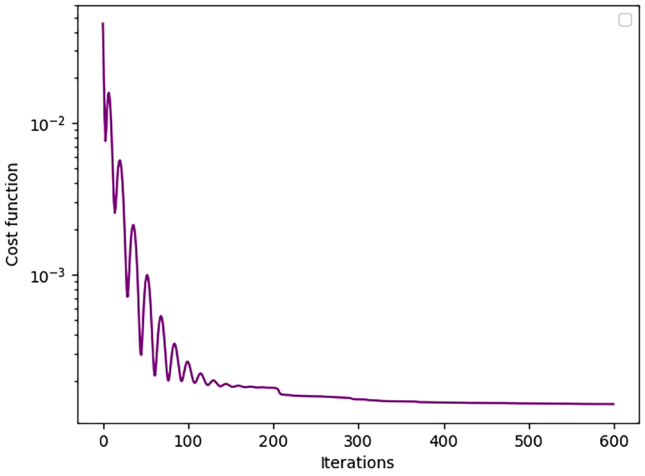


FIGURE 5 | Convergence of the cost function at $\alpha = 0.99$ for Example 4.2.

Intermediate matrix V_k: Formulate the intermediate matrix V_k necessary for the Hessian update:

$$V_k = I - \rho_k y_k s_k^T \tag{33}$$

Inverse Hessian approximation update:

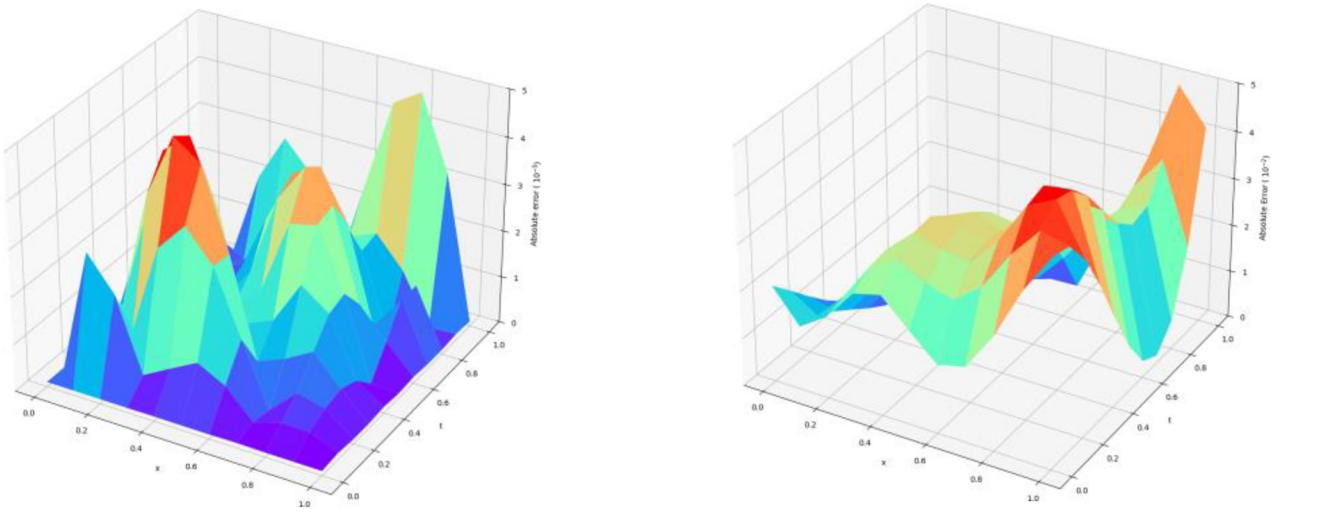
$$H_{k+1} = V_k^T H_k V_k + \rho_k s_k s_k^T \tag{34}$$

An implicit modified version of H_k is stored by keeping a specified number m of vector pairs $\{s_i, y_i\}, i = k-m, \cdots, k-1$, where $m \le 100$ [39].

Parameter update: Apply the determined step length to adjust the parameters:

$$p_{k+1} = p_k + \alpha_k j_k \tag{35}$$

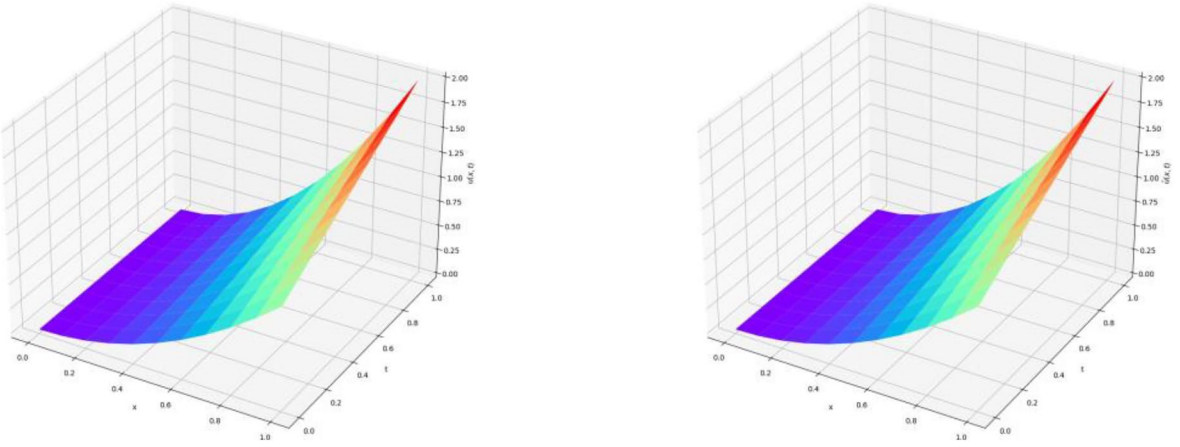
The L-BFGS optimizer enhances the precision and accuracy of the model by refining the parameters through the implementation of



(a) Absolute error at $\alpha = 1$.

(b) Absolute error of cost function at $\alpha = 0.99$.

FIGURE 6 | Absolute error at $\alpha = 1$, and absolute error of cost function at $\alpha = 0.99$ for Example 4.2.



(a) Exact solution.

(b) DFMLNN solution at $\alpha = 1$.

(c) DFMLNN solution at $\alpha = 0.99$.

(d) DFMLNN solution at $\alpha = 0.7$.

FIGURE 7 | Visualization of the exact solution and the numerical solution derived using DFMLNN at varying α values for Example 4.2.

TABLE 5 | Comparison of exact and DFMLNN solutions for Example 4.3 with varying values of α.

(x,t)	Exact	DFMLNN		
		$\alpha=1$	$\alpha=0.99$	$\alpha=0.6$
(0,0)	0	0	0	0
(0.1,0.1)	1.015222	1.015213	1.016055	1.028981
(0.2,0.2)	1.062258	1.06222	1.064675	1.104907
(0.3,0.3)	1.144346	1.144295	1.148317	1.222277
(0.4,0.4)	1.266516	1.266450	1.271759	1.380334
(0.5,0.5)	1.435901	1.435830	1.442061	1.580305
(0.6,0.6)	1.662081	1.662024	1.668695	1.825769
(0.7,0.7)	1.671957	1.671935	1.963893	2.123118
(0.8,0.8)	2.337887	2.337888	2.343205	2.481775
(0.9,0.9)	2.822881	2.822892	2.826109	2.914302
(1,1)	3.436563	3.436563	3.436563	3.436563

this comprehensive approach. This method combines Adam's rapid convergence with the careful adjustment of parameters in L-BFGS, resulting in an ideal trade-off between training speed and accuracy.

The following algorithm outlines the DFMLNN training process with hybrid optimization for solving FPDE:

4 | Numerical Evaluation

This section presents the numerical results derived from the proposed DFMLNN model (Algorithm 2) combined with the hybrid optimization approach. We assess the performance of our method by solving various FPDEs. The DFMLNN model's effectiveness and accuracy are illustrated through multiple examples. For each example, we offer detailed comparisons between the numerical and exact solutions, emphasizing the precision and robustness of our approach.

Example 4.1. Consider the following FPDES [40]:

$$\begin{cases} \dfrac{\partial^2 u(x,t)}{\partial x^2} - \dfrac{\partial^\alpha u(x,t)}{\partial x^\alpha} - \dfrac{\partial u(x,t)}{\partial t} = f(x,t),\ 0\le x\le 1, 0\le t\le 1, 0<\alpha\le 1, \\ u(x,t)=x^2, \\ u(0,t)=t^2,\ u(1,t)=1+t^2 \end{cases}$$

where $f(x,t)=2-\frac{\Gamma(3)}{\Gamma(3-\alpha)}x^{2-\alpha}-2t$ and for $\alpha=1$, $u(x,t)=x^2+t^2$. According to (10), the specified DFMLNN is: $\hat{u}(X,p)=(1-x)t^2+x\left(t^2+1\right)+x^2-x+x(1-x)tN(X,p)$.

The network is trained over 600 iterations, beginning with 200 iterations using the Adam optimizer, followed by up to 400 iterations with the L-BFGS optimizer. A discretized mesh grid with 11×11 points in both time and space is employed to thoroughly explore the solution domain. The parameters β_1 and β_2 are set to 0.95 and 0.999, respectively, and the learning rate η is set to 0.01.

TABLE 6 | Absolute errors for $\alpha=1$ in Example 4.3.

(x,t)	AEs
(0,0)	0
(0.1,0.1)	9.05E-06
(0.2,0.2)	2.93E-05
(0.3,0.3)	5.05E-05
(0.4,0.4)	6.62E-05
(0.5,0.5)	7.09E-05
(0.6,0.6)	5.68E-05
(0.7,0.7)	2.76E-05
(0.8,0.8)	1.66E-07
(0.9,0.9)	1.12E-05
(1,1)	0

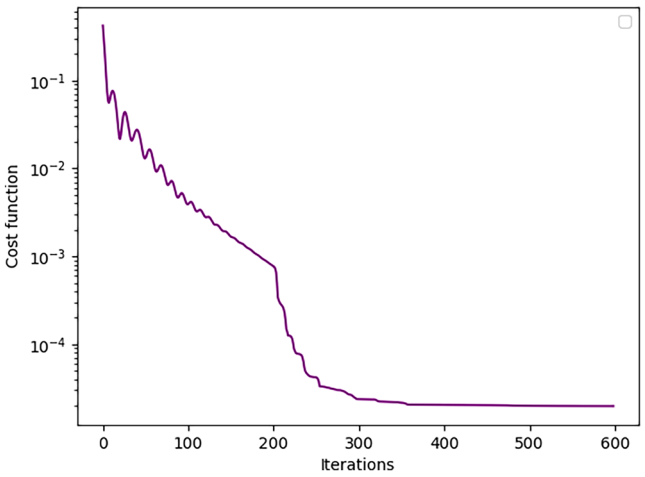
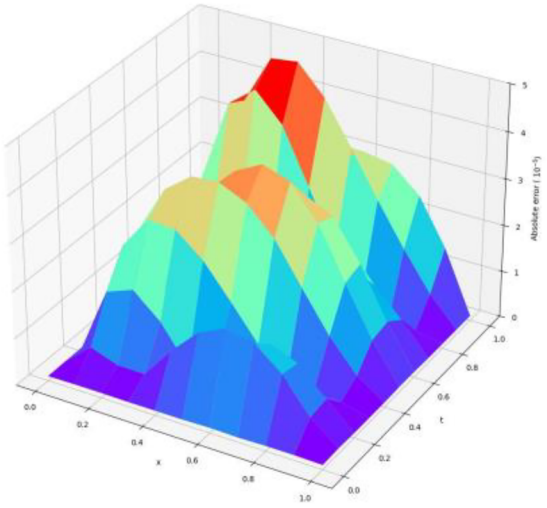
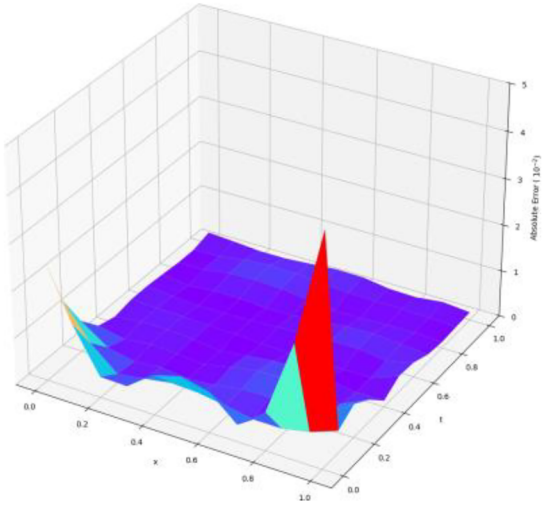


FIGURE 8 | Convergence of the cost function at $\alpha=0.99$ for Example 4.3.

Table 1 compares the exact and DFMLNN solutions for various values of α. As α approaches 1, the DFMLNN solution closely aligns with the exact solution, indicating high accuracy. Table 2 presents the absolute errors (AEs) for $\alpha=1$. The extremely small error values demonstrate the DFMLNN model's precision in predicting the solutions, with negligible discrepancies. Figure 2 illustrates the convergence behavior of the cost function for $\alpha=0.99$. The cost function decreases rapidly during the initial training phase, indicating effective early-stage learning. After approximately 300 iterations, the rate of decrease slows, and the cost function gradually stabilizes, confirming successful convergence of the training process. Figure 3 displays the absolute error at $\alpha=1$ and the absolute error cost function at $\alpha=0.99$. Figure 3a shows that the absolute error is predominantly low throughout most of the domain, with a maximum value of approximately 10^{-6}. Figure 3b illustrates that the overall absolute error cost remains relatively low across the domain, peaking at around 10^{-2}. Figure 4 provides a visualization of the exact solution and the DFMLNN solutions for different α values. The close agreement among all sets of solutions underscores the model's effectiveness and its strong generalization capability.

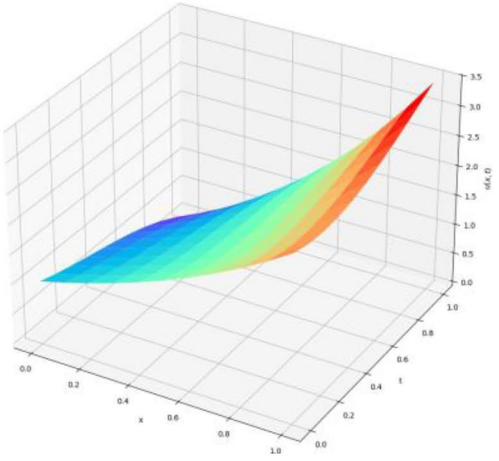


(a) Absolute error at $\alpha = 1$.

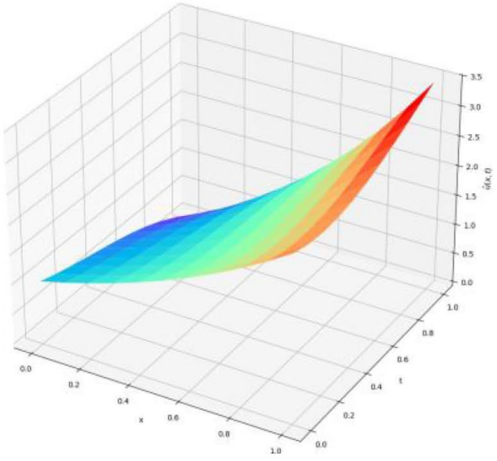


(b) Absolute error of cost function at $\alpha = 0.99$.

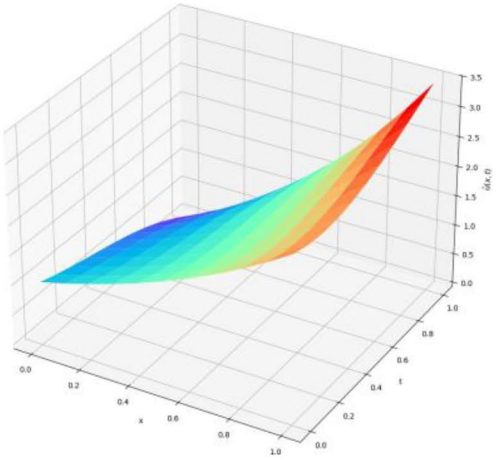
FIGURE 9 | Absolute error at $\alpha = 1$, and absolute error of cost function at $\alpha = 0.99$ for Example 4.3.



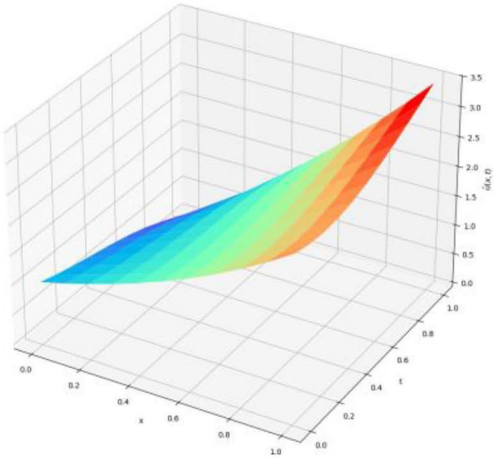
(a) Exact solution.



(b) DFMLNN solution at $\alpha = 1$.



(c) DFMLNN solution at $\alpha = 0.99$.



(d) DFMLNN solution at $\alpha = 0.6$.

FIGURE 10 | Visualization of the exact solution and the numerical solution derived using DFMLNN at varying α values for Example 4.3.

TABLE 7 | Comparison of exact and DFMLNN solutions for Example 4.4 with varying values of α.

(x,t)	Exact	DFMLNN		
		$\alpha=1$	$\alpha=0.99$	$\alpha=0.5$
(0,0)	0	0	0	0
(0.1,0.1)	0.200000	0.2000004	1.016055	1.028981
(0.2,0.2)	0.400000	0.4000043	1.064675	1.104907
(0.3,0.3)	0.600000	0.6000082	1.148317	1.222277
(0.4,0.4)	0.800000	0.8000087	1.271759	1.380334
(0.5,0.5)	1.000000	1.0000056	1.442061	1.580305
(0.6,0.6)	1.200000	1.2000010	1.668695	1.825769
(0.7,0.7)	1.399999	1.3999978	1.963893	2.123118
(0.8,0.8)	1.600000	1.5999975	2.343205	2.481775
(0.9,0.9)	1.799999	1.7999997	2.826109	2.914302
(1,1)	2.000000	2.000000	2.000000	2.000000

Example 4.2. Let consider the following space FPDE of order α $(0<\alpha\leq 1)$ as [41]:

$$\begin{cases} \dfrac{\partial^{\alpha}u(x,t)}{\partial x^{\alpha}}+\dfrac{\partial u(x,t)}{\partial t}=f(x,t), 0<x\leq 1, 0<t\leq 1, \\ u(x,0))=x^2, \\ u(0,t)=0, u(1,t)=(t+1) \end{cases}$$

where the exact solution for $\alpha=1$ is $u(x,t)=x^2(1+t)$ and $f(x,t)=\frac{\Gamma(3)}{\Gamma(3-\alpha)}(1+t)x^{2-\alpha}+x^2$. According to (10), the assigned DFMLNN is $\widehat{u}(X,p)=(1+t)x+x^2-x+x(1-x)tN(X,p)$.

Table 3 contrasts the exact and DFMLNN solutions for various α values. As α nears 1, the DFMLNN solution aligns closely with the exact solution, demonstrating high precision. Table 4 details the AEs for $\alpha=1$. The minimal error values confirm that the DFMLNN model accurately predicts the solutions with negligible discrepancies. Figure 5 depicts the convergence trend of the cost function for $\alpha=0.99$. During the early training phase, the cost function drops sharply, reflecting efficient initial learning. After about 200 iterations, the decline slows down, and the cost function progressively levels off, indicating that the training has successfully converged. Figure 6 presents the absolute error for $\alpha=1$ and the absolute cost function for $\alpha=0.99$. Figure 6a demonstrates that the absolute error remains consistently low across the majority of the domain, with a peak value near 10^{-5}. In Figure 6b, the absolute cost function is shown to stay relatively low throughout the domain, reaching a maximum of approximately 10^{-2}. Figure 7 illustrates the exact solution alongside the DFMLNN solutions for various α values. The strong alignment between these results highlights the model's accuracy and robust generalization performance.

Example 4.3. Let consider the following FPDEs as [42]:

TABLE 8 | Absolute errors for $\alpha=1$ in Example 4.4.

(x,t)	AEs
(0,0)	0
(0.1,0.1)	4.91E-07
(0.2,0.2)	4.32E-06
(0.3,0.3)	8.22E-06
(0.4,0.4)	8.76E-06
(0.5,0.5)	5.60E-06
(0.6,0.6)	9.53E-07
(0.7,0.7)	2.14E-06
(0.8,0.8)	2.50E-06
(0.9,0.9)	2.38E-07
(1,1)	0

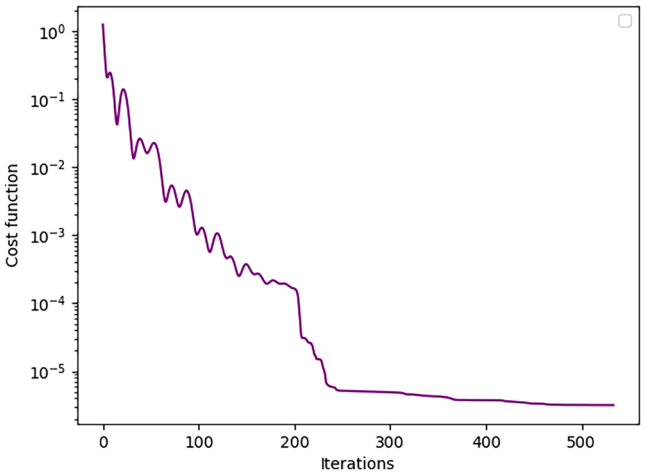
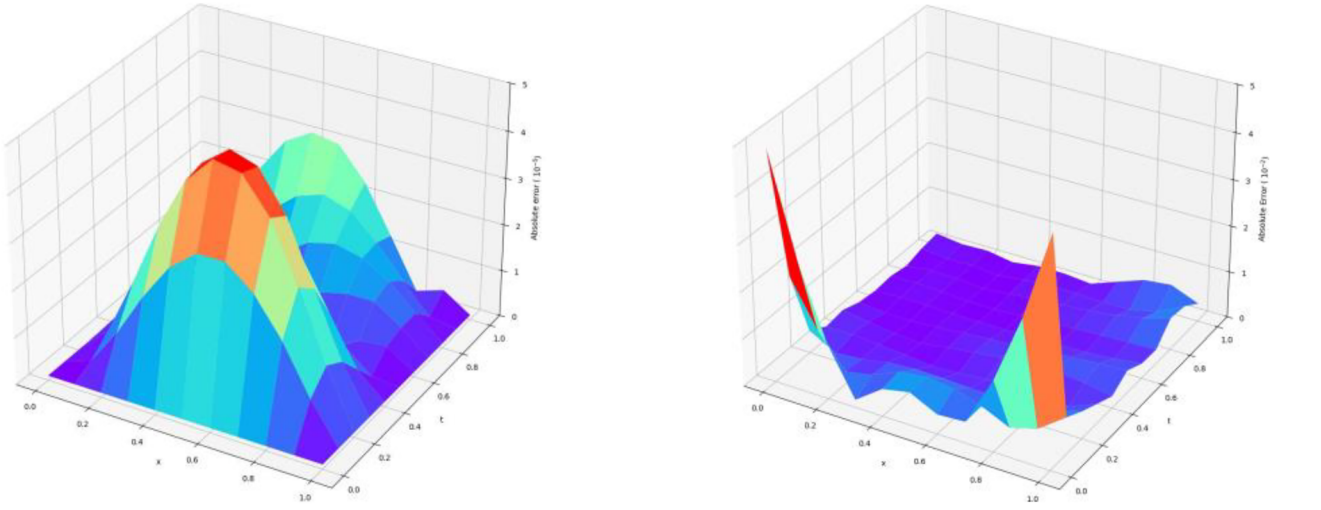


FIGURE 11 | Convergence of the cost function at $\alpha=0.99$ for Example 4.4.

$$\begin{cases} \dfrac{\partial^{\alpha}u(x,t)}{\partial x^{\alpha}}+\dfrac{\partial u(x,t)}{\partial t}-x\dfrac{\partial^2 u(x,t)}{\partial x^2}=f(x,t), 0<x\leq 1, 0<t\leq 1, \\ u(x,0))=e^{x}, \\ u(0,t)=\left(1+t^2\right)(1-t), u(1,t)=\left(1+t^2\right)(e-t) \end{cases}$$

where $f(x,t)$ is chosen such that the exact solution to the problem is $\alpha=1, u(x,t)=\left(1+t^2\right)(e^{x}-t)$. According to (10), the assigned DFMLNN is $\widehat{u}(X,p)=(1-x)\left(1+t^2\right)(1-t)+x\left(1+t^2\right)(e-t)+e^{x}-(1-x)-xe+x(1-x)tN(X,p)$.

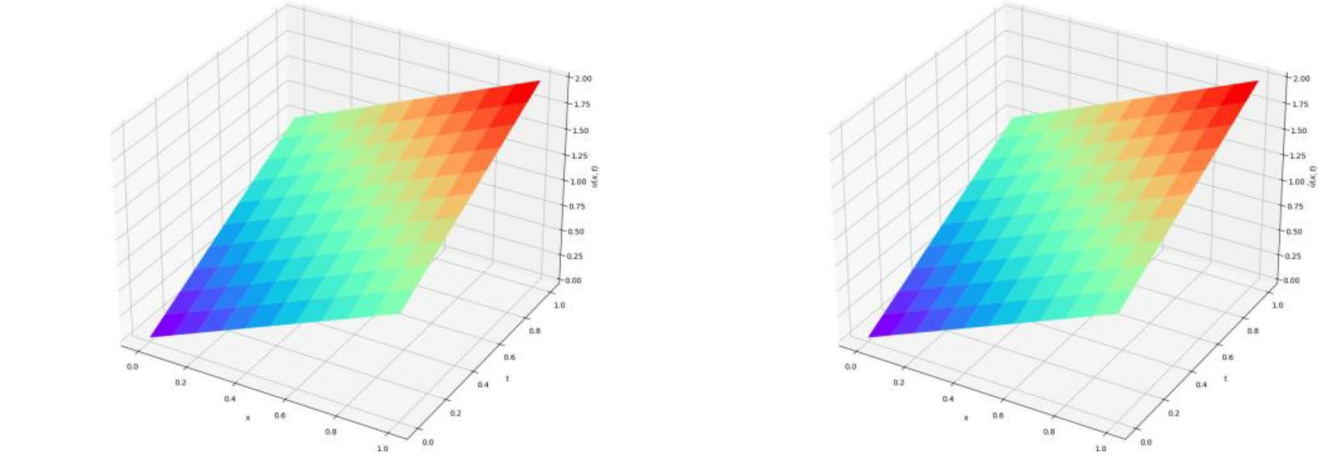
Table 5 compares the exact solutions with those produced by the DFMLNN model across different α values. As α approaches 1, the DFMLNN results exhibit a close match with the exact solutions, indicating high precision. Table 6 reports the AEs for $\alpha=1$, where the notably small error values confirm the DFMLNN model's ability to accurately approximate the solutions with minimal deviation. Figure 8 shows the convergence behavior of the cost



(a) Absolute error at $\alpha = 1$.

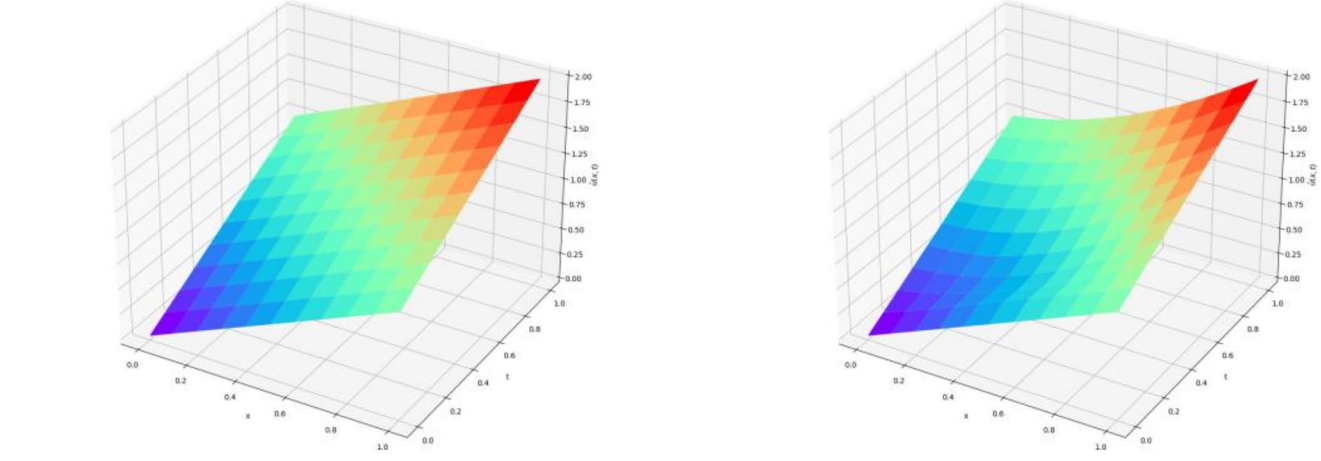
(b) Absolute error of cost function at $\alpha = 0.99$.

FIGURE 12 | Absolute error at $\alpha = 1$, and absolute error of cost function at $\alpha = 0.99$ for Example 4.4.



(a) Exact solution.

(b) DFMLNN solution at $\alpha = 1$.



(c) DFMLNN solution at $\alpha = 0.99$.

(d) DFMLNN solution at $\alpha = 0.5$.

FIGURE 13 | Visualization of the exact solution and the numerical solution derived using DFMLNN at varying α values for Example 4.4.

function for $\alpha = 0.99$. The cost function decreases rapidly during the initial stages of training, reflecting effective early learning. After approximately 300 iterations, the rate of decrease slows, and the function gradually stabilizes, signifying that the model has successfully converged. Figure 9 displays both the absolute error for $\alpha = 1$ and the absolute error cost function for $\alpha = 0.99$. Specifically, Figure 9a illustrates that the absolute error remains low across most of the domain, peaking at around 10^{-5}. Figure 9b reveals that the cost function also stays relatively low throughout the domain, with a maximum value close to 10^{-2}. Finally, Figure 10 presents a comparison between the exact solution and the DFMLNN solutions for several α values. The close correspondence among these solutions emphasizes the model's effectiveness and its strong capacity for generalization.

Example 4.4. Let consider the following FPDE [43]:

$$\begin{cases} \dfrac{\partial^{\alpha} u(x,t)}{\partial x^{\alpha}} = \dfrac{\partial u(x,t)}{\partial t} - \dfrac{\partial^2 u(x,t)}{\partial x^2}, 0 < x \le 1, 0 < t \le 1, \\ u(x,0)) = x, \\ u(0,t) = t, u(1,t) = x \end{cases}$$

where the exact solution for $\alpha = 1$ is $u(x,t) = x + t$. According to (10), the assigned DFMLNN is $\hat{u}(X,p) = (1-x)t + x(1+t) + x(1-x)tN(X,p)$.

Table 7 presents a comparison between the exact solutions and those generated by the DFMLNN model for various values of α. As α approaches 1, the DFMLNN solutions closely align with the exact ones, demonstrating the model's high accuracy and reliability. Table 8 provides the AEs for $\alpha = 1$, where the consistently small values further validate the DFMLNN model's capability to produce precise approximations with minimal deviation. The convergence pattern of the cost function for $\alpha = 0.99$ is illustrated in Figure 11. A rapid decrease is observed during the initial training phase, reflecting effective early learning. After approximately 250 iterations, the rate of decline slows, and the cost function gradually stabilizes, indicating successful convergence of the training process. Figure 12 highlights both the absolute error at $\alpha = 1$ and the cost function behavior at $\alpha = 0.99$. In Figure 12a, the absolute error remains consistently low across the domain, with a maximum of approximately 10^{-5}. Figure 12b shows that the cost function also maintains low values throughout, peaking near 10^{-2}. Lastly, Figure 13 visualizes the exact solution alongside the DFMLNN-generated results for different α values. The strong agreement between them highlights the model's robustness and excellent generalization performance.

5 | Conclusion

This study presents a novel hybrid deep learning framework for solving FPDEs, utilizing a DFMLNN. The proposed approach introduces a two-phase optimization strategy that combines the Adam algorithm for rapid initial convergence with the L-BFGS algorithm for precise fine-tuning. This integration effectively leverages the complementary strengths of both optimizers to achieve a robust balance between efficiency and accuracy. A further innovation lies in incorporating the Laplace transform method to approximate Caputo space-fractional derivatives, which significantly enhances the framework's performance compared to traditional methods. Extensive numerical experiments validate the model's accuracy and reliability, demonstrating excellent agreement with exact solutions and minimal error margins. These results underscore the effectiveness of the DFMLNN model and hybrid training strategy in solving complex FPDEs. This approach opens promising directions for future research in scientific computing and the broader application of deep learning to fractional DEs.

Acknowledgments

The authors are very thankful to Malaysia Ministry of Education for awarding the Fundamental Research Grant Scheme (Ref. No. FRGS/1/2022/STG06/UPM/02/2) for supporting this work.

Data Availability Statement

The data that support the findings of this study are available from the corresponding author upon reasonable request.

References

1. V. Daftardar-Gejji, *Fractional Calculus* (Alpha Science International Limited, 2013).

2. I. Podlubny, *Fractional Differential Equations: An Introduction to Fractional Derivatives, Fractional Differential Equations, to Methods of Their Solution and Some of Their Applications* (Elsevier, 1998).

3. H. G. Sun, Y. Zhang, D. Baleanu, W. Chen, and Y. Q. Chen, "A New Collection of Real World Applications of Fractional Calculus in Science and Engineering," *Communications in Nonlinear Science and Numerical Simulation* 64 (2018): 213–231.

4. V. E. Tarasov, "No Nonlocality. No Fractional Derivative," *Communications in Nonlinear Science and Numerical Simulation* 62 (2018): 157–163.

5. O. P. Agrawal, J. Sabatier, and J. A. T. Machado, *Advances in Fractional Calculus: Theoretical Developments and Applications in Physics and Engineering* (Springer, 2007).

6. O. Abdulaziz, I. Hashim, and E. S. Ismail, "Approximate Analytical Solution to Fractional Modified Kdv Equations," *Mathematical and Computer Modelling* 49, no. 1–2 (2009): 136–145.

7. Z. Odibat and S. Momani, "A Generalized Differential Transform Method for Linear Partial Differential Equations of Fractional Order," *Applied Mathematics Letters* 21, no. 2 (2008): 194–199.

8. S. Momani, Z. Odibat, and V. S. Erturk, "Generalized Differential Transform Method for Solving a Space-and Time-Fractional Diffusion-Wave Equation," *Physics Letters A* 370, no. 5–6 (2007): 379–387.

9. S. Z. Rida, A. M. A. El-Sayed, and A. A. M. Arafa, "On the Solutions of Time-Fractional Reaction–Diffusion Equations," *Communications in Nonlinear Science and Numerical Simulation* 15, no. 12 (2010): 3847–3854.

10. K. Moaddy, S. Momani, and I. Hashim, "The Non-Standard Finite Difference Scheme for Linear Fractional Pdes in Fluid Mechanics," *Computers & Mathematics With Applications* 61, no. 4 (2011): 1209–1216.

11. O. Abu Arqub, "Numerical Solutions for the Robin Time-Fractional Partial Differential Equations of Heat and Fluid Flows Based on the Reproducing Kernel Algorithm," *International Journal of Numerical Methods for Heat & Fluid Flow* 28, no. 4 (2018): 828–856.

12. A. Omar Abu, "Numerical Simulation of Time-Fractional Partial Differential Equations Arising in Fluid Flows via Reproducing Kernel Method," *International Journal of Numerical Methods for Heat & Fluid Flow* 30, no. 11 (2020): 4711–4733.

13. A. Omar Abu and N. Shawagfeh, "Application of Reproducing Kernel Algorithm for Solving Dirichlet Time-Fractional Diffusion-Gordon Types Equations in Porous Media," *Journal of Porous Media* 22, no. 4 (2019): 411–434.

14. A. Omar Abu, "Computational Algorithm for Solving Singular Fredholm Time-Fractional Partial Integrodifferential Equations With Error Estimates," *Journal of Applied Mathematics and Computing* 59, no. 1 (2019): 227–243.

15. D. Li and D. Yu, "Deep Learning–Methods and Applications," *Foundations and Trends in Signal Processing* 7, no. 3–4 (2014): 197–387.

16. I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning* (MIT Press, 2016).

17. Y. LeCun, Y. Bengio, and G. Hinton, "Deep Learning," *Nature* 521, no. 7553 (2015): 436–444.

18. S. Li, W. Song, L. Fang, Y. Chen, P. Ghamisi, and J. A. Benediktsson, "Deep Learning for Hyperspectral Image Classification: An Overview," *IEEE Transactions on Geoscience and Remote Sensing* 57, no. 9 (2019): 6690–6709.

19. Y. Goldberg, "A Primer on Neural Network Models for Natural Language Processing," *Journal of Artificial Intelligence Research* 57 (2016): 345–420.

20. G. Helbing and M. Ritter, "Deep Learning for Fault Detection in Wind Turbines," *Renewable and Sustainable Energy Reviews* 98 (2018): 189–198.

21. K. Hornik, M. Stinchcombe, and H. White, "Universal Approximation of an Unknown Mapping and Its Derivatives Using Multilayer Feedforward Networks," *Neural Networks* 3, no. 5 (1990): 551–560.

22. Y. Lu and J. Lu, "A Universal Approximation Theorem of Deep Neural Networks for Expressing Probability Distributions," *Advances in Neural Information Processing Systems* 33 (2020): 3094–3105.

23. A. A. S. Almarashi, "Approximation Solution of Fractional Partial Differential Equations by Neural Networks," *Advances in Numerical Analysis* 2012, no. 1 (2012): 912810.

24. H. Qu, X. Liu, and Z. She, "Neural Network Method for Fractional-Order Partial Differential Equations," *Neurocomputing* 414 (2020): 225–237.

25. H. Qu, Z. She, and X. Liu, "Neural Network Method for Solving Fractional Diffusion Equations," *Applied Mathematics and Computation* 391 (2021): 125635.

26. H.-D. Qu, X. Liu, X. Lu, M. u. Rahman, and Z.-H. She, "Neural Network Method for Solving Nonlinear Fractional Advection-Diffusion Equation With Spatiotemporal Variable-Order," *Chaos, Solitons & Fractals* 156 (2022): 111856.

27. J.-L. Wei, G.-C. Wu, B.-Q. Liu, and Z. Zhao, "New Semi-Analytical Solutions of the Time-Fractional Fokker–Planck Equation by the Neural Network Method," *Optik* 259 (2022): 168896.

28. C. Biswas, A. Singh, M. Chopra, and S. Das, "Study of Fractional-Order Reaction-Advection-Diffusion Equation Using Neural Network Method," *Mathematics and Computers in Simulation* 208 (2023): 15–27.

29. W. Liu, Y. Liu, and H. Li, "Time Difference Physics-Informed Neural Network for Fractional Water Wave Models," *Results in Applied Mathematics* 17 (2023): 100347.

30. A. Ali, N. Senu, N. Wahi, N. Almakayeel, and A. Ahmadian, "An Adaptive Algorithm for Numerically Solving Fractional Partial Differential Equations Using Hermite Wavelet Artificial Neural Networks," *Communications in Nonlinear Science and Numerical Simulation* 137 (2024): 108121.

31. A. Ali, N. Senu, A. Ahmadian, and N. Wahi, "A Deep Learning Framework for Solving Fractional Partial Differential Equations," *Physica Scripta* 100, no. 4 (2025): 046012.

32. I. Podlubny, "Matrix Approach to Discrete Fractional Calculus," *Fractional Calculus and Applied Analysis* 3, no. 4 (2000): 359–386.

33. S. Shen, F. Liu, J. Chen, I. Turner, and V. Anh, "Numerical Techniques for the Variable Order Time Fractional Diffusion Equation," *Applied Mathematics and Computation* 218, no. 22 (2012): 10861–10870.

34. H. M. Fahad, M. U. Rehman, and A. Fernandez, "On Laplace Transforms With Respect to Functions and Their Applications to Fractional Differential Equations," *Mathematical Methods in the Applied Sciences* 46, no. 7 (2023): 8304–8323.

35. I. Podlubny, "Fractional Differential Equations," in *Mathematics in Science and Engineering* (Academic Press, 1999).

36. I. E. Lagaris, A. Likas, and D. I. Fotiadis, "Artificial Neural Networks for Solving Ordinary and Partial Differential Equations," *IEEE Transactions on Neural Networks* 9, no. 5 (1998): 987–1000.

37. D. P. Kingma and J. Ba, "Adam: A Method for Stochastic Optimization, arXiv," 2014, https://doi.org/10.48550/arXiv.1412.6980.

38. S. Wright and J. Nocedal, *Numerical Optimization* (Springer Science, 2006).

39. J. Rafati and R. F. Marcia, "Deep Reinforcement Learning via l-bfgs Optimization, arXiv," 2018, https://doi.org/10.48550/arXiv.1811.02693.

40. H. Jafari and H. Tajadodi, "Numerical Solutions of the Fractional Advection-Dispersion Equation," *Progress in Fractional Differentiation & Applications* 1, no. 1 (2015): 37–45.

41. K. K. Ali, M. A. Abd, E. Salam, and M. S. Mohamed, "Chebyshev Fifth-Kind Series Approximation for Generalized Space Fractional Partial Differential Equations," *AIMS Mathematics* 7, no. 5 (2022): 7759–7780.

42. F. Zhou and X. Xu, "The Third Kind Chebyshev Wavelets Collocation Method for Solving the Time-Fractional Convection Diffusion Equations With Variable Coefficients," *Applied Mathematics and Computation* 280 (2016): 11–29.

43. M. A. Firoozjaee, H. Jafari, A. Lia, and D. Baleanu, "Numerical Approach of Fokker–Planck Equation With Caputo–Fabrizio Fractional Derivative Using Ritz Approximation," *Journal of Computational and Applied Mathematics* 339 (2018): 367–373.