Research Paper
Computational Methods for Differential Equations
http://cmde.tabrizu.ac.ir
Vol. *, No. *, *, pp. 1-14
DOI:10.22034/cmde.2025.63059.2803

A novel neural network architecture for solving fractional differential equations

Hassan Dana Mazraeh[1], Ali Nosrati Firoozsalari[2], Alireza Afzal Aghaei[2], and Kourosh Parand[2,3,4,*]

[1]School of Mathematics and Computer Sciences, Damghan University, Damghan,P.O. Box 36715-364, Iran.

[2]Department of Computer and Data Sciences, Faculty of Mathematical Sciences, Shahid Beheshti University, Tehran, Iran.

[3]Department of Cognitive Modeling, Institute for Cognitive and Brain Sciences, Shahid Beheshti University, Tehran, Iran.

[4]International Business University, Toronto, Canada.

Abstract

The primary objective of this research is to develop a neural network-based method for solving fractional differential equations. The proposed design incorporates a Gaussian integration rule and an $L1$ discretization technique for solving fractional (integro-) differential equations. In each equation, a multi-layer neural network is employed to approximate the unknown function. To demonstrate the versatility of the method, three forms of fractional differential equations are examined: a fractional ordinary differential equation, a fractional integro-differential equation, and a fractional partial differential equation. The results indicate that the proposed architecture demonstrates good accuracy for these different types of equations.

Keywords. Artificial neural networks, Machine learning, Partial differential equations, Fractional calculus.

2010 Mathematics Subject Classification. 26A33,34K37,68T05.

1. Introduction

Fractional functional equations, characterized by the inclusion of one or more non-integer order derivative or integral operators alongside the classical ones, represent a growing class of mathematical problems with significant applications. Such equations are instrumental in numerous branches of science and engineering, including physics, chemistry, and cognitive sciences [25, 37]. For instance, they provide highly accurate models for complex phenomena such as viscoelastic materials [13], electrochemistry [22] and Resistor-Inductor-Capacitor (RLC) circuits [27].

While these equations offer superior modeling capabilities, their solution presents considerable challenges. The complexity increases further when integral operators are included. Analytical methods often prove insufficient for solving such complex problems. Consequently, researchers have turned to semi-analytical methods, such as the Adomian Decomposition Method (ADM) [11], the Variational Iteration Method (VIM) [21], and the Homotopy Perturbation Method (HPM) [12]. However, these techniques often suffer from drawbacks, a primary one being slow convergence. Numerical approaches like finite difference methods [17], meshless methods [18], and spectral methods [33] are also widely used. Yet, these methods have their own inherent limitations. For instance, finite difference methods can suffer from low accuracy, meshless methods may exhibit numerical instability, and spectral methods struggle with complex geometries.

In recent years, neural networks, particularly through the framework of Physics-Informed Neural Networks (PINNs) [26], have emerged as a powerful alternative for solving differential equations [7, 8]. In this paradigm, the governing physical laws, expressed as a functional equation, are directly encoded as a residual term within the neural network's loss function. The network is then trained to minimize this loss function, yielding an approximate solution to the problem. This framework is particularly potent when observational data is available, as neural networks can learn the underlying system dynamics directly from this data [31]. For a functional equation (FE), possibly with fractional or

Received: 20 August 2024 ; Accepted: 10 September 2025.
∗ Kourosh Parand Email: k_parand@sbu.ac.ir.

integer-order derivatives and integrals, subject to a set of initial and boundary conditions, and incorporating known data points, a PINN minimizes a composite loss function:

$$\text{Loss}_{\text{PINN}}(u(\mathbf{x})) = \text{Loss}_{FE}(u(\mathbf{x})) + \text{Loss}_{Boundaries}(u(\mathbf{x})) + \text{Loss}_{Initial}(u(\mathbf{x})) + \text{Loss}_{Data}(u(\mathbf{x})),$$

where $u(\mathbf{x})$ is the output of a multi-layer neural network (DNN) [1, 9, 23, 26]. PINNs offer several advantages over traditional methods: they can naturally incorporate data through Loss_{Data} [31], handle high-dimensional problems effectively [14], manage complex geometries without requiring mesh generation [36], solve nonlinear problems with ease, and leverage parallel computing on hardware like Graphical Processing Units (GPUs) to accelerate training [26].

A cornerstone of the PINN framework is automatic differentiation (AD), an algorithm that computes exact derivatives of the network's output with respect to its inputs by traversing its computational graph. However, AD is fundamentally incompatible with fractional derivatives and integrals, as these are non-local operators by definition. This presents a significant difficulty for applying PINNs to fractional functional equations (fPINNs).

Several studies have attempted to bridge this gap. Some approaches involve developing neural networks within symbolic computer algebra systems to leverage analytical methods for fractional operators. For example, Kheyrinataj and Nazemi [15] solved fractional optimal control problems using a fractional power neural network, Allahviranloo et al. [2] solved higher order fractional integro-differential equations of the Caputo type, and Chaharborj et al. [6] developed a semi-analytical Chebyshev neural network for fractional integro-differential equations. Numerical discretization methods are also studied. For example, Martire et al. [5] utilized Gaussian quadrature to approximate the Caputo fractional derivative, Saadat et al. [28] employed the trapezoidal rule to approximate the Riemann-Liouville fractional operator and Taheri et al. [34] developed operational matrices for approximating the Caputo derivative. Nevertheless, these methods often suffer from numerical errors and poor stability because fractional operators are usually defined as singular integrals.

To overcome these limitations, this paper introduces a PINN methodology for solving fractional differential equations, which may also include integral operators. Our proposed approach integrates an accurate finite difference scheme, namely the $L1$ discretization, to approximate fractional derivatives [20], and employs Gaussian quadrature for the efficient and precise computation of integral terms. This ensures that fractional derivatives are approximated accurately. The proposed method is then evaluated on fractional differential equations in one and two dimensions, and on a fractional integro-differential equation.

The remainder of this paper is organized as follows: Section 2 provides the necessary background on numerical integration and fractional calculus, section 3 details the proposed methodology, and section 4 presents the numerical results with their discussion. Finally, section 5 offers concluding remarks and outlines potential directions for future research.

2. Backgrounds

This section first describes the approximation of integer-order integral operators, followed by a discussion of fractional derivatives and integrals defined through singular kernels. It then presents a finite difference approach that replaces singular integration with a more precise discretization scheme.

2.1. Gauss-Legendre quadrature.

In standard DNN frameworks, PINNs can not compute integrals directly due to the nonlocal nature of integral operators. These operators must therefore be approximated by suitable numerical schemes. Depending on the smoothness of the integrand and the integration domain, various methods are available, including Newton-Cotes formulas, Gaussian quadrature, tanh-sinh quadrature, Monte Carlo approaches, and adaptive techniques [10]. For sufficiently smooth functions over a finite interval, the Gauss-Legendre quadrature is both accurate and straightforward to implement. Given a function $u(x)$ defined on $[a, b]$, the Gauss-Legendre rule takes the form:

$$\int_a^b u(x)\,dx = \int_{-1}^{1} u\left(\frac{(b-a)t + (b+a)}{2}\right)\frac{(b-a)}{2}\,dt \approx \frac{(b-a)}{2}\sum_{i=1}^{n} w_i\, u\left(\frac{(b-a)t_i + (b+a)}{2}\right), \tag{2.1}$$

where $\{t_i\}_{i=1}^{n}$ are the roots of the Legendre polynomial of degree n mapped to $[-1, 1]$, and $w_i = 2\{(1-t_i^2)[P_n'(t_i)]^2\}^{-1}$ are the associated weights. This quadrature rule integrates exactly all polynomials of degree up to $2n-1$.

2.2. **Fractional Derivatives.** The definition of a non-integer order operator, whether a derivative or an integral, in fractional calculus is not unique. Each physical phenomenon may require a particular type of fractional operator suited to its specific properties. Some definitions, such as the Grünwald-Letnikov, rely on discrete summations; others, including the Riemann-Liouville and Caputo, involve singular integrals, while operators like the Caputo-Fabrizio use non-singular integrals [19]. Among these, the Caputo derivative is the most widely employed, as it recovers the integer-order derivative when its fractional order approaches an integer. This operator is defined in terms of the Riemann-Liouville fractional integral.

Definition 2.1 (Riemann-Liouville fractional integral)**.** Let $u \in C[a,b]$ and $\alpha \in \mathbb{R}^+$. The Riemann-Liouville fractional integral of order α of u on $[a,x]$ is defined by

$$ {}_a\mathcal{I}_x^{\alpha} u(x) = \frac{1}{\Gamma(\alpha)} \int_a^x (x-s)^{\alpha-1} u(s)\, ds, \tag{2.2}$$

where $\Gamma(\alpha)$ denotes the gamma function given by

$$ \Gamma(\alpha) = \int_0^{\infty} t^{\alpha-1} e^{-t}\, dt, \quad \alpha > 0. $$

Definition 2.2 (Riemann-Liouville fractional derivative)**.** The Riemann-Liouville fractional derivative of order $\alpha > 0$ is given by

$$ {}_a\mathcal{D}_x^{\alpha} u(x) = \frac{1}{\Gamma(n-\alpha)} \frac{d^n}{dx^n} \int_a^x (x-s)^{n-\alpha-1} u(s)\, ds, \tag{2.3}$$

where $n = \lceil \alpha \rceil$ (the smallest integer greater than or equal to α). In this formulation, the integral is evaluated first, followed by the integer-order derivative.

Definition 2.3 (Caputo fractional derivative)**.** In contrast, the Caputo fractional derivative of order $\alpha > 0$ is defined by [4, 32]:

$$ {}_a^C\mathscr{D}_x^{\alpha} u(x) = \frac{1}{\Gamma(n-\alpha)} \int_a^x (x-s)^{n-\alpha-1} u^{(n)}(s)\, ds, \quad n-1 < \alpha \le n, \tag{2.4}$$

where the integer-order derivative of u is computed first, followed by the fractional integral. Compatibility with classical integer-order derivatives is maintained in the limit $\alpha \to n$ under this definition.

Property 2.4. The Caputo fractional derivative satisfies the following fundamental properties for suitable functions u and v:

- **Linearity.** For scalar constants a, b,
  $ {}_a^C\mathscr{D}_x^{\alpha}[a\,u(x) + b\,v(x)] = a\, {}_a^C\mathscr{D}_x^{\alpha} u(x) + b\, {}_a^C\mathscr{D}_x^{\alpha} v(x).$
- **Partial derivatives.** For $u(\mathbf{x})$ with $\mathbf{x} = (x_1, \ldots, x_d)$, the Caputo fractional partial derivative with respect to x_i is defined by:
  $$ {}_a^C\mathscr{D}_{x_i}^{\alpha} u(\mathbf{x}) = \frac{1}{\Gamma(n-\alpha)} \int_a^{x_i} (x_i - s)^{n-\alpha-1} \frac{\partial^n u}{\partial x_i^n}(\mathbf{x}_{(x_i \to s)})\, ds. $$
- **Composition with integer derivatives.** For integer n and $\alpha > 0$,
  $$ {}_a^C\mathscr{D}_x^{n+\alpha} u(x) = \frac{d^n}{dx^n}\big({}_a^C\mathscr{D}_x^{\alpha} u(x)\big), $$
  under suitable smoothness conditions.
- **Action on monomials.** For monomials $u(x) = x^m$ with $m \in \mathbb{N}_0$,
  $$ {}_0^C\mathscr{D}_x^{\alpha} u(x) = \begin{cases} 0, & m < \lceil \alpha \rceil, \\ \dfrac{\Gamma(m+1)}{\Gamma(m+1-\alpha)} x^{m-\alpha}, & m \ge \lceil \alpha \rceil. \end{cases} $$

To address the computational challenge of incorporating the Caputo fractional operator into PINNs, where automatic differentiation is not applicable, either Gaussian quadrature or robust numerical discretization methods can be applied [3, 34]. However, Gaussian quadrature typically yields low accuracy because the Caputo derivative involves a weakly singular kernel. The following section introduces the $L1$ discretization method, which produces higher approximation accuracy for weakly singular fractional operators.

2.3. **$L1$-discretization.** The $L1$ scheme is a widely adopted and well-analyzed method for approximating the Caputo fractional derivative, particularly for orders $0 < \alpha < 1$. The derivation begins by discretizing the time domain into uniform steps $t_j = j\Delta t$, and by evaluating the Caputo derivative at $t = t_{n+1}$. For $0 < \alpha < 1$, the Caputo definition is:

$$ {}_0^C\mathscr{D}_t^\alpha u(t_{n+1}) = \frac{1}{\Gamma(1-\alpha)} \int_0^{t_{n+1}} (t_{n+1}-s)^{-\alpha} \frac{du}{ds}(s)\, ds. \tag{2.5} $$

The $L1$ scheme approximates this integral by partitioning it into a sum over the discrete intervals $[t_j, t_{j+1}]$. On each interval, the integer-order derivative $u'(s)$ is approximated by a first-order backward difference, $u'(s) \approx \frac{u(t_{j+1})-u(t_j)}{\Delta t} = \frac{u^{j+1}-u^j}{\Delta t}$. Substituting this piecewise constant approximation into the integral gives:

$$ \begin{aligned} {}_0^C\mathscr{D}_t^\alpha u(t_{n+1}) &\approx \frac{1}{\Gamma(1-\alpha)} \sum_{j=0}^{n} \int_{t_j}^{t_{j+1}} (t_{n+1}-s)^{-\alpha} \frac{u^{j+1}-u^j}{\Delta t}\, ds \\ &= \frac{1}{\Gamma(1-\alpha)\Delta t} \sum_{j=0}^{n} (u^{j+1}-u^j) \left[-\frac{(t_{n+1}-s)^{1-\alpha}}{1-\alpha} \right]_{s=t_j}^{t_{j+1}} \\ &= \frac{(\Delta t)^{-\alpha}}{\Gamma(2-\alpha)} \sum_{j=0}^{n} (u^{j+1}-u^j) \left((n+1-j)^{1-\alpha} - (n-j)^{1-\alpha} \right). \end{aligned} $$

In the last step, we used the identity $\Gamma(z+1) = z\Gamma(z)$, such that $\Gamma(2-\alpha) = (1-\alpha)\Gamma(1-\alpha)$.

Defining the coefficients $b_j = (j+1)^{1-\alpha} - j^{1-\alpha}$ for $j \geq 0$ and rearranging the summation produces the standard $L1$ formula. The approximation at time t_{n+1} is given by:

$$ {}_0^C\mathscr{D}_t^\alpha u(t_{n+1}) \approx \frac{(\Delta t)^{-\alpha}}{\Gamma(2-\alpha)} \left[b_0 u^{n+1} - b_n u^0 - \sum_{j=1}^{n} (b_{n-j} - b_{n-j+1}) u^j \right]. \tag{2.6} $$

This formulation is numerically stable, with a local truncation error $R_{\Delta t}^{n+1}$ that is well characterized [3]. For a sufficiently smooth function $u(t)$, the $L1$ scheme is known to have an order of accuracy of $O(\Delta t^{2-\alpha})$. Hence, the approximation improves as the time step Δt decreases.

The coefficients b_j possess key properties that ensure the stability of the scheme. For $0 < \alpha < 1$, it can be shown that $b_j > 0$ for all $j \geq 0$. Furthermore, the sequence is monotonically decreasing, i.e., $b_{j+1} < b_j$, and converges to zero as $j \to \infty$. The first coefficient is always $b_0 = (1)^{1-\alpha} - 0 = 1$. The weights applied to the historical values of u decay over time, ensuring consistency with the fading memory characteristic of systems modeled by fractional calculus.

3. Methodology

The previous section provided the necessary approximation strategies for evaluating integrals and the fractional Caputo derivative. In this section, we employ these approximations to construct the proposed method. We handle the integrals in two distinct categories. The first category arises from the definition of the Caputo fractional derivative, which involves a weakly singular kernel of the form $(x-s)^{-\alpha}$; for these, we adopt the $L1$ discretization scheme, which is particularly effective for addressing such singularities. Although the theoretical formulation involves singular kernels, their discrete implementation within DNNs composes them with the smooth neural network output, ensuring that the integrals remain well-behaved. This composition ensures that the resulting integrals remain well-behaved in practice, which results in numerical stability. The second category includes integrals with sufficiently smooth kernels, where

Gauss-Legendre quadrature provides an accurate and efficient numerical approximation due to its high precision for smooth integrands.

To approximate the solution of a functional equation, we employ a standard feedforward neural network with $\tanh(\cdot)$ activation functions, whose output is denoted by $\hat{u}(\cdot)$ and defined as

$$\hat{u}(\mathbf{x}) = \sum_{j=1}^{N_L} w_j^{(L)} \tanh\left(\sum_{i=1}^{N_{L-1}} w_{ji}^{(L-1)} \tanh\left(\cdots \tanh\left(\sum_{k=1}^{N_1} w_{ik}^{(1)} x_k + b_i^{(1)} \right) \cdots \right) + b_j^{(L-1)} \right) + b^{(L)}. \tag{3.1}$$

Here, L denotes the number of layers and N_ℓ the number of neurons in the ℓ-th layer. The trainable weights w_{ij} and biases b_i are updated using the Adam optimizer:

$$\begin{aligned} m^{(m)} &= \beta_1 m^{(m-1)} + (1-\beta_1)\, \widehat{\nabla}_\theta \operatorname{Loss}(\theta^{(m)}), \\ v^{(m)} &= \beta_2 v^{(m-1)} + (1-\beta_2)\, \left(\widehat{\nabla}_\theta \operatorname{Loss}(\theta^{(m)}) \right)^2, \\ \hat{m}^{(m)} &= \frac{m^{(m)}}{1-\beta_1^m}, \quad \hat{v}^{(m)} = \frac{v^{(m)}}{1-\beta_2^m}, \\ \theta^{(m+1)} &= \theta^{(m)} - \eta \frac{\hat{m}^{(m)}}{\sqrt{\hat{v}^{(m)}} + \epsilon}, \end{aligned} \tag{3.2}$$

where θ collectively represents all weights and biases, η is the learning rate, β_1, β_2 are the exponential decay rates, and ϵ is a small positive constant. In this setting, derivatives required by the governing equation or with respect to network parameters are computed by automatic differentiation in PyTorch. The fractional derivatives are discretized using the $L1$ scheme, while integral terms are approximated by Gauss-Legendre quadrature; both depend only on the network output $\hat{u}(\cdot)$. Figure 1 illustrates the diagram of the proposed method.

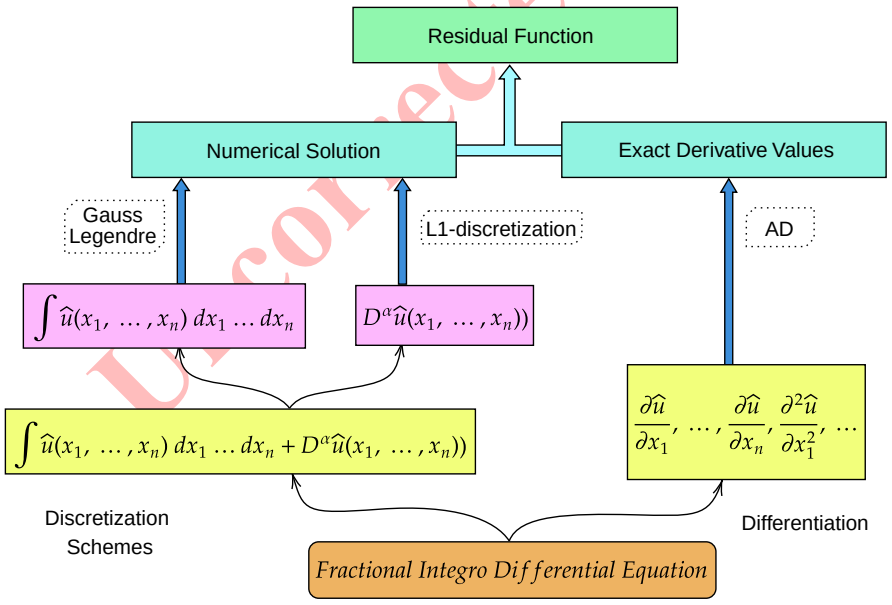

FIGURE 1. A diagram illustrating the proposed method

We now explicitly define the general form of the fractional differential equations under consideration and the corresponding loss functions. We consider three types of fractional differential equations to demonstrate the versatility of our approach:

(1) **Fractional Ordinary Differential Equations (FODEs)**: We address FODEs of the form:

$${}_0^C\mathscr{D}_x^\alpha u(x) + f(x, u(x)) = 0, \quad x \in [a, b], \quad \alpha \in (0, 1),$$

where ${}_0^C\mathscr{D}_x^\alpha$ denotes the Caputo fractional derivative of order α, $u(x)$ is the unknown function, and $f(x, u(x))$ is a possibly nonlinear function. The equation is subject to initial conditions, e.g., $u(a) = \kappa$. The loss function associated with FODEs is as follows:

$$\text{Loss} = \lambda_1 \text{Loss}_{Residual} + \lambda_2 \text{Loss}_{Initial}, \tag{3.3}$$

where:

$$\text{Loss}_{Residual} = \left({}_0^C\mathscr{D}_x^\alpha u(x_i) + f(x_i, u(x_i))\right)^2,$$

$$\text{Loss}_{Initial} = \left(\hat{u}(a) - \kappa\right)^2, \quad \kappa \in \mathbb{R}$$

where $\hat{u}(x)$ is the output of the network approximating $u(x)$, λ_1 and λ_2 are hyperparameter in $\mathbb{R}$.

(2) Fractional Partial Differential Equations (FPDEs): We consider FPDEs of the form:

$${}_0^C\mathscr{D}_t^\alpha u(x, t) + \mathcal{L}u(x, t) = g(x, t), \quad (x, t) \in \Omega \times [0, T], \quad \alpha \in (0, 1),$$

where ${}_0^C\mathscr{D}_t^\alpha$ is the Caputo fractional time derivative, $\mathcal{L}$ is a spatial differential operator (e.g., $\partial^2/\partial x^2$), and $g(x, t)$ is a source term. Boundary conditions ($\phi_1(x)$ and $\phi_2(x)$) and initial condition ($\xi(t)$) are specified on the domain $\Omega \times \{0\}$.

The loss function associated with FPDEs is as follows:

$$\text{Loss} = \lambda_1 \text{Loss}_{Residual} + \lambda_2 \text{Loss}_{Initial} + \lambda_3 \text{Loss}_{Boundary}, \tag{3.4}$$

where:

$$\text{Loss}_{Residual} = \left({}_0^C\mathscr{D}_x^\alpha \hat{u}(x_i, t_j) + \mathcal{L}\hat{u}(x_i, t_j) - g(x_i, t_j)\right)^2,$$

$$\text{Loss}_{Initial} = \left(\hat{u}(a, t_j) - \xi(t)\right)^2,$$

$$\text{Loss}_{Boundary} = \left(\hat{u}(x_i, a) - \phi_1(x_i)\right)^2 + \left(\hat{u}(x_i, b) - \phi_2(x_i)\right)^2,$$

where $\xi(t), \phi_1(x), \phi_2(x)$ are known functions, $\hat{u}(x, t)$ is the output of the network approximating $u(x, t)$, λ_1, λ_2, and λ_3 are hyperparameters in $\mathbb{R}$.

(3) Fractional Integro-Differential Equations (FIDEs): We solve FIDEs of the form:

$${}_0^C\mathscr{D}_x^\alpha u(x) + \int_a^x K(x, s)u(s)ds = h(x), \quad u(a) = \kappa, \quad x \in [a, b], \quad \alpha \in (0, 1),$$

where $K(x, s)$ is a smooth kernel of the integral term, and $h(x)$ is a given function, with appropriate initial or boundary conditions.

The loss function associated with FIDEs is as follows:

$$\text{Loss} = \lambda_1 \text{Loss}_{Residual} + \lambda_2 \text{Loss}_{Initial}, \tag{3.5}$$

where:

$$\text{Loss}_{Residual} = \left({}_0^C\mathscr{D}_x^\alpha \hat{u}(x_i) - h(x_i) + \int_a^{x_i} K(x, s)\hat{u}(s)ds\right)^2,$$

$$\text{Loss}_{Initial} = \left(\hat{u}(a) - \kappa\right)^2, \quad \kappa \in \mathbb{R},$$

where $\hat{u}(x)$ is the output of the network approximating $u(x)$, λ_1 and λ_2 are hyperparameters in $\mathbb{R}$.

In either case, we consider the following assumptions:

- The solution $\hat{u}(x, t)$ is assumed to be sufficiently smooth in space and time to allow differentiation and numerical approximation using neural networks.
- The fractional derivative is taken in the Caputo sense, and the order $\alpha \in (0, 1]$ is fixed for each problem instance.
- Initial and boundary conditions (as applicable) are assumed to be known and well-defined for each problem.

- The neural network approximator $\hat{u}(x)$ or $\hat{u}(x,t)$ is trained by minimizing a composite loss function that incorporates the residual of the differential equation, the boundary conditions, and the initial conditions.

In each iteration of the proposed method, a random point x_i is generated in $\Omega = [a, b]$ and the value $\hat{u}(x_i)$ is calculated using the neural network. The random value is then chosen as the maximum value in both the integration scheme and to estimate the fractional derivative. The equation is considered part of the loss function, and the parameters are learned using the squared error of the values in each equation and the initial and boundary conditions. The shared parameters will be learned by minimizing the loss function using the Adam optimizer. Figure 2 illustrates the main flow diagram of the proposed framework. The methodology is applied to specific examples in section 4, including one FODE, one FPDE, and one FIDE, to validate the approach.

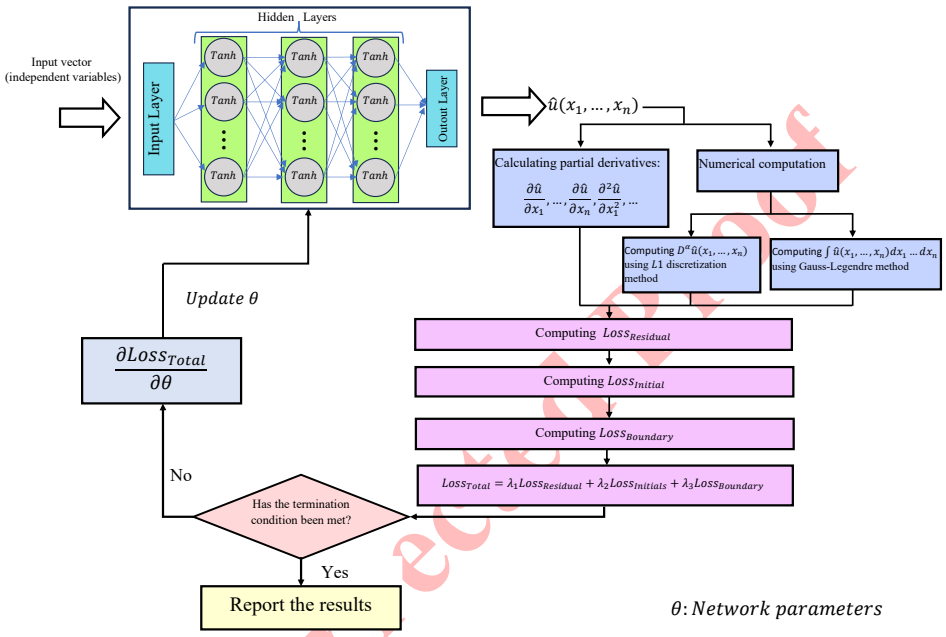

FIGURE 2. The main flow diagram of the proposed framework.

4. Numerical Results

In this section, some examples are investigated and solved using the given methods to demonstrate the efficacy of the proposed method. To demonstrate the generality of the proposed method, we have selected some different examples from different classes of fractional order differential equations, including a fractional order differential equation, a fractional order integro-differential equation, and a fractional order partial differential equation. Before proposing the numerical results, we discuss the training configurations.

4.1. **Network and Training Configuration.** In all experiments, we used a fully connected feedforward neural network consisting of five layers with 32 neurons in each hidden layer. The activation function used for all layers was the hyperbolic tangent function $\tanh(\cdot)$. The networks were implemented in PyTorch. For training, the Adam optimizer was employed with learning rates chosen empirically for each example to ensure fast convergence:

- Example 1: learning rate = 0.01
- Example 2: learning rate = 0.005
- Example 3: learning rate = 0.001

The batch size was set equal to the number of training points (full-batch gradient descent), and the number of training epochs was 1000 unless otherwise noted. The number of quadrature points in the Gauss-Legendre method and the number of discretization points for $L1$ are carefully chosen to achieve a balance between computational cost and numerical accuracy. Our experiments indicate that beyond a certain threshold, increasing the number of points does not produce significant gains, which supports the efficiency and robustness of the proposed approach. All experiments were conducted using a standard desktop with an NVIDIA 1080 GPU, and training times were on the order of minutes for each example. Specifically, training for Example 1 took approximately 2 minutes, Example 2 took 3 minutes, and Example 3 took 5 minutes. Once trained, the proposed network can provide solutions almost instantaneously for new input points, typically within milliseconds. Table 1 presents a summary of neural network architecture and training hyperparameters for each example. Furthermore, all hyperparameters λ_i are set to 1.

TABLE 1. Summary of neural network architecture and training hyperparameters for each example.

Example	Hidden Layers	Neurons	Activation	Optimizer	Learning Rate	Epochs	Batch Size
1	3	32	Tanh	Adam	0.01	1000	Full batch
2	3	32	Tanh	Adam	0.005	1000	Full batch
3	3	32	Tanh	Adam	0.001	2000	Full batch

Example 4.1. First, we consider a fractional ordinary differential equation as follows [24]:

$$ {}_0^C\mathscr{D}_x^{\alpha} u(x) + u^2(x) = x + \left(\frac{x^{\alpha+1}}{\Gamma(\alpha+2)}\right)^2, \quad 0 < \alpha \leq 1, \quad 0 \leq x \leq 1. \tag{4.1} $$

The initial condition for this equation is $u(0) = 0$ and the exact solution is $u(x) = \frac{1}{\Gamma(v+2)} x^{\alpha+1}$. The loss function will consist of the initial condition and the equation itself:

$$ \text{Loss} = \text{Loss}_{Residual} + \text{Loss}_{Initial}, \tag{4.2} $$

where:

$$ \text{Loss}_{Residual} = \left({}_0^C\mathscr{D}_x^{\alpha} \hat{u}(x_i) + \hat{u}^2(x_i) - x_i - \left(\frac{x_i^{\alpha+1}}{\Gamma(\alpha+2)}\right)^2 \right)^2, $$

$$ \text{Loss}_{Initial} = \left(\hat{u}(0)\right)^2. $$

The results are presented in Table 2, where the absolute error is calculated for different values for α with $n = 1000$ discretization points and 1000 epochs. Figure 3 depicts the exact and predicted solution to the presented problem, along with the residual. According to our experiments, the best results are achieved when α is 0.5 for different values of x. Since we already have an initial condition, it is expected to have better results initially, and we can observe it in the results presented in Table 2. The accuracy did not improve after increasing epochs further than 1000 epochs. Based on our experiments, we converged the fastest using the learning rate of 0.01.

Example 4.2. Now we consider a fractional integro-differential equation [35]:

$$ \begin{cases} {}_0^C\mathscr{D}_x^{0.5} u(x) = u(x) + \frac{8}{\sin(0.5)} x^{1.5} - x^2 - \frac{1}{3}x^3 + \int_0^x u(t)\, dt, & 0 < x < 1, \\ u(0) = 0, \end{cases} \tag{4.3} $$

The exact solution to this equation is $u(x) = x^2$. The loss function of PINN will consist of the initial condition and the equation itself:

$$ \text{Loss} = \text{Loss}_{Residual} + \text{Loss}_{Initial}, $$

where

$$ \left({}_0^C\mathscr{D}_x^{0.5} \hat{u}(x_i) - \hat{u}(x_i) - \frac{8}{\sin(0.5)} x_i^{1.5} + x_i^2 + \frac{1}{3}x_i^3 - \int_0^{x_i} \hat{u}(t)\, dt, \right)^2, \tag{4.4} $$

TABLE 2. Absolute error for different values of α and 1000 epochs for Example 4.1.

x \ α	0.1	0.3	0.5	0.7	0.9
0	3.23×10^{-5}	7.14×10^{-6}	6.25×10^{-6}	3.43×10^{-5}	6.31×10^{-5}
0.1	1.45×10^{-5}	7.75×10^{-5}	1.86×10^{-5}	4.12×10^{-5}	1.11×10^{-4}
0.2	2.51×10^{-4}	5.37×10^{-5}	1.42×10^{-5}	3.04×10^{-5}	7.69×10^{-5}
0.3	8.31×10^{-5}	6.13×10^{-5}	4.44×10^{-5}	1.49×10^{-5}	1.42×10^{-4}
0.4	1.25×10^{-4}	2.33×10^{-5}	1.92×10^{-5}	1.21×10^{-5}	2.15×10^{-4}
0.5	1.56×10^{-5}	2.59×10^{-5}	8.61×10^{-6}	1.89×10^{-6}	2.37×10^{-4}
0.6	9.12×10^{-5}	2.40×10^{-6}	2.46×10^{-5}	2.39×10^{-5}	2.12×10^{-4}
0.7	4.85×10^{-5}	3.96×10^{-5}	2.54×10^{-5}	2.42×10^{-5}	1.83×10^{-4}
0.8	2.86×10^{-5}	2.48×10^{-5}	9.58×10^{-6}	1.28×10^{-5}	1.91×10^{-4}
0.9	3.38×10^{-5}	1.59×10^{-5}	2.11×10^{-5}	1.31×10^{-5}	2.31×10^{-4}
1	1.18×10^{-4}	3.73×10^{-5}	2.95×10^{-5}	1.11×10^{-5}	2.11×10^{-4}

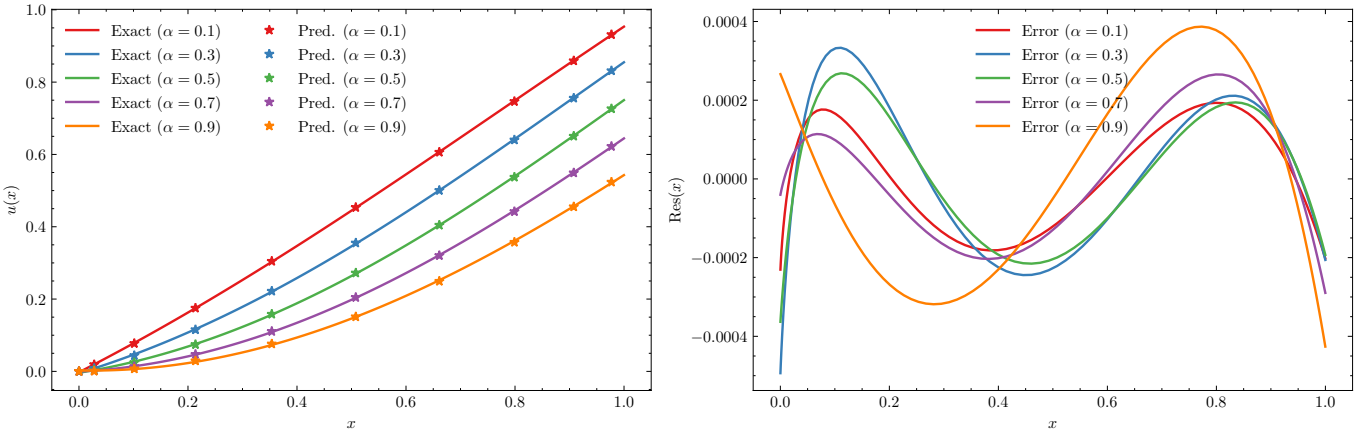

FIGURE 3. Predicted solutions and their residual using various values of α for Example 4.1.

and

$$\text{Loss}_{Initial} = (\hat{u}(0))^2, \tag{4.5}$$

The results are presented in Table 3 with varying numbers of discretization points. Figure 4 depicts the predicted and exact solutions, and the residual graph for 1000 epochs. This equation consists of a fractional part and an integral term. The fractional component is calculated with various numbers of discretization points as shown in Table 3; however, we have utilized 400 discretization points in the Gauss-Legendre method to estimate the integral part. The rationale behind choosing 400 discretization points is that if we were to use any further discretization points, the accuracy would not change considerably; on the other hand, the time complexity would increase dramatically. According to our experiments, a reasonable trade-off between accuracy and time complexity is reached at 400 discretization points. As it is evident from Table 3, in general, the accuracy of the model decreases as x approaches 1. This is due to the fact that we have an initial condition for this problem at $x = 0$. In general, augmenting the quantity of discretization points leads to enhanced accuracy in the model. Nevertheless, due to the resultant escalation in time complexity, we have selected to set the top limit at 1000 points for our experimental purposes. The conducted tests have shown that the observed trade-off between the execution time and accuracy of the model is favorable. Based on our experiments, we converged the fastest using the learning rate of 0.005. Additionally, in this example we use a polynomial layer as part of our neural network to estimate the answer. The first layer has 6 outputs from x up to x^6 and these polynomial

features are fed into another layer with a linear transformation, then through a tanh(·) activation function which can help the model learn to predict the answer.

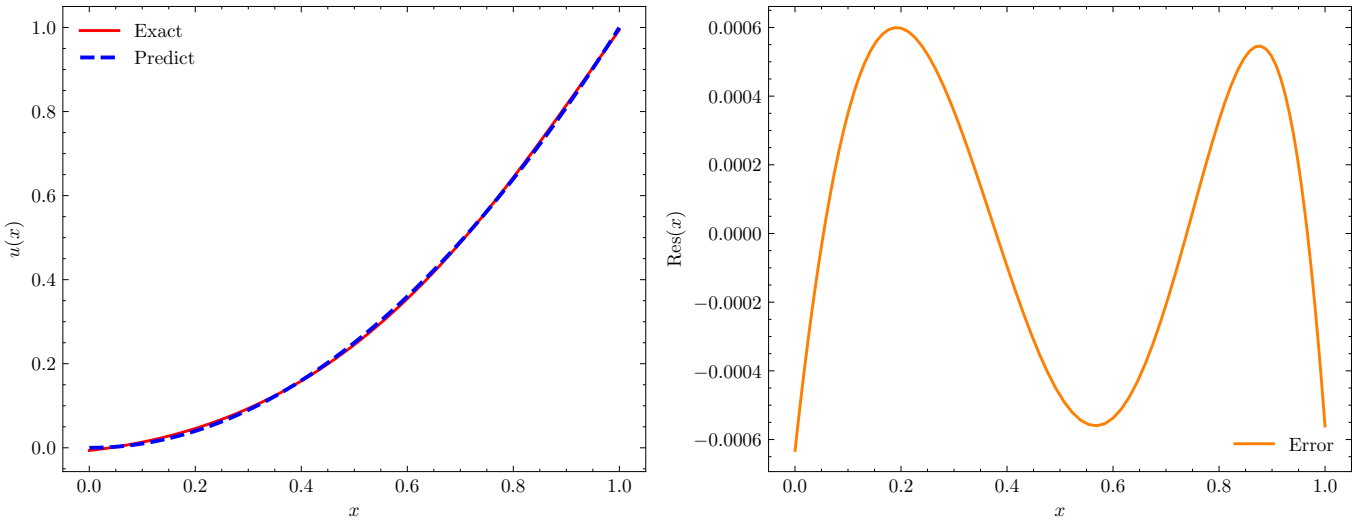

FIGURE 4. Predicted solution and its error for Example 4.2.

TABLE 3. Absolute error for Example 4.2 with different discretization points and 1000 epochs.

x \ n	100	250	500	750	1000
0	2.22×10^{-3}	1.19×10^{-3}	1.30×10^{-3}	5.94×10^{-4}	2.26×10^{-3}
0.1	1.01×10^{-2}	6.55×10^{-3}	1.64×10^{-3}	2.36×10^{-3}	1.44×10^{-3}
0.2	1.02×10^{-2}	7.02×10^{-3}	2.62×10^{-3}	2.47×10^{-3}	1.42×10^{-3}
0.3	6.48×10^{-3}	4.97×10^{-3}	2.40×10^{-3}	1.46×10^{-3}	2.94×10^{-4}
0.4	2.45×10^{-3}	2.46×10^{-3}	2.17×10^{-3}	6.45×10^{-4}	1.75×10^{-3}
0.5	3.29×10^{-4}	8.79×10^{-4}	3.02×10^{-3}	6.67×10^{-4}	1.58×10^{-3}
0.6	2.24×10^{-4}	5.26×10^{-4}	4.87×10^{-3}	1.41×10^{-3}	1.72×10^{-4}
0.7	4.38×10^{-4}	6.07×10^{-4}	5.92×10^{-3}	2.22×10^{-3}	1.87×10^{-3}
0.8	5.05×10^{-4}	1.80×10^{-4}	4.47×10^{-3}	2.57×10^{-3}	1.76×10^{-3}
0.9	2.73×10^{-3}	8.18×10^{-4}	1.89×10^{-3}	2.72×10^{-3}	1.09×10^{-3}
1	9.65×10^{-3}	8.60×10^{-3}	7.86×10^{-4}	1.90×10^{-3}	2.16×10^{-3}

Example 4.3. Finally, we consider an initial-boundary problem of fractional partial differential equation [29]:

$$\frac{\partial^\alpha u(x,t)}{\partial t^\alpha} + x\frac{\partial u(x,t)}{\partial x} + \frac{\partial^2 u(x,t)}{\partial x^2} = 2t^\alpha + 2x^2 + 2, \quad 0 < x < 1, 0 < t < 1, \tag{4.6}$$

where α is between 0 and 1, the initial condition is $u(x,0) = x^2$ and the boundary conditions are: $u(0,t) = 2\frac{\Gamma(\alpha+1)}{\Gamma(2\alpha+1)}t^{2\alpha}$, and $u(1,t) = 1 + 2\frac{\Gamma(\alpha+1)}{\Gamma(2\alpha+1)}t^{2\alpha}$.

The loss function will consist of the initial condition, and the equation itself is as follows:

$$\text{Loss} = \text{Loss}_{Residual} + \text{Loss}_{Initial} + \text{Loss}_{Boundary},$$

$$\text{Loss}_{Residual} = \left(\frac{\partial^\alpha \hat{u}(x,t)}{\partial t^\alpha} + x\frac{\partial \hat{u}(x,t)}{\partial x} + \frac{\partial^2 \hat{u}(x,t)}{\partial x^2} - 2t^\alpha - 2x^2 - 2\right)^2,$$

$$\text{Loss}_{Initial} = \left(\hat{u}(x_i,0) - x_i^2\right)^2,$$

$$\text{Loss}_{Boundary} = \left(\hat{u}(0,t_j) = 2\frac{\Gamma(\alpha+1)}{\Gamma(2\alpha+1)}t_j^{2\alpha}\right)^2 + \left(u(1,t_j) = 1 + 2\frac{\Gamma(\alpha+1)}{\Gamma(2\alpha+1)}t_j^{2\alpha}\right)^2.$$

TABLE 4. Comparison of mean absolute error between Wavelet Method and our method for α=0.5 and t=0.5 in Example 4.3.

x	Wavelet Method		[30]	Presented Method
	$m = 32$	$m = 64$		
0.1	6.09×10^{-3}	1.21×10^{-3}		8.43×10^{-3}
0.2	4.84×10^{-3}	1.25×10^{-3}		8.45×10^{-3}
0.3	2.75×10^{-2}	1.86×10^{-3}		7.64×10^{-3}
0.4	1.93×10^{-2}	7.41×10^{-3}		6.19×10^{-3}
0.5	1.00×10^{-6}	1.00×10^{-6}		4.45×10^{-3}
0.6	4.35×10^{-2}	7.46×10^{-3}		2.73×10^{-3}
0.7	1.73×10^{-2}	1.72×10^{-3}		1.23×10^{-3}
0.8	7.75×10^{-2}	4.99×10^{-3}		8.98×10^{-5}
0.9	4.44×10^{-2}	1.67×10^{-2}		5.91×10^{-4}

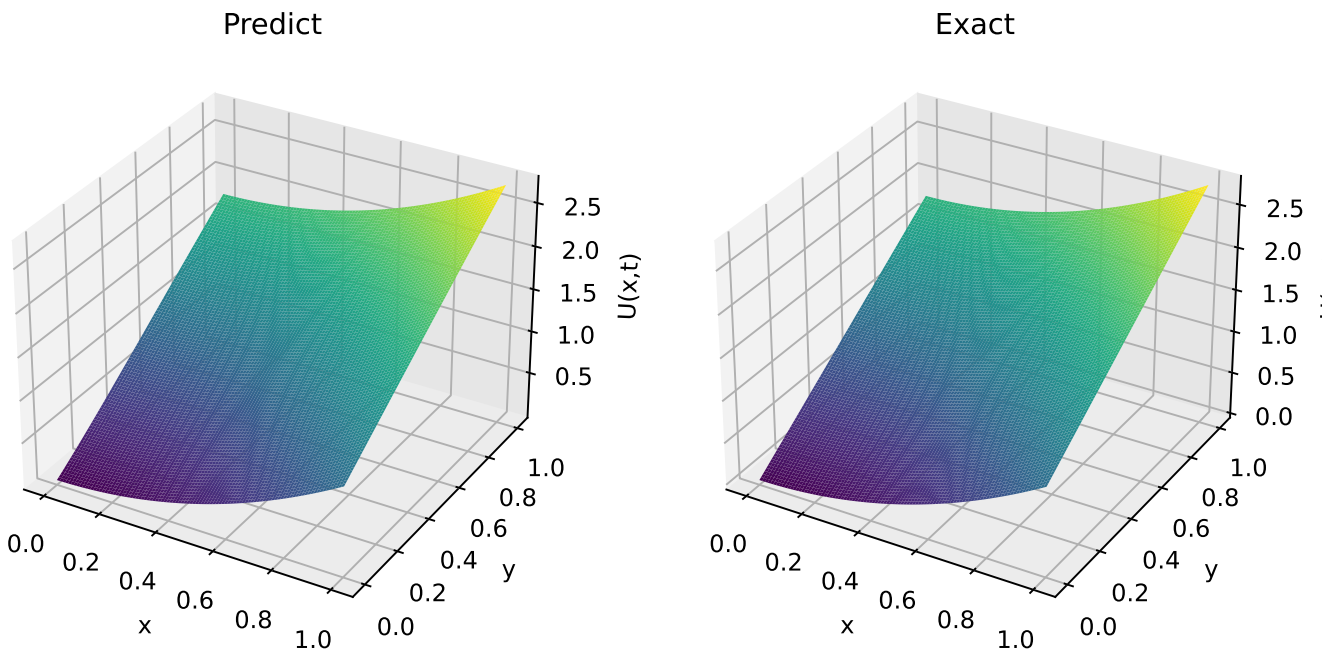

FIGURE 5. Predicted solution and the exact solution for Example 4.3 with 2000 epochs.

The results presented in Table 5 and Figure 5 show the approximate solution obtained by the current method using $n = 100$ discretization points along with the exact solution. The comparison between the obtained solutions

TABLE 5. Mean absolute error for Example 4.3 with different values for x and t and 1000 epochs.

x \ t	0.1	0.3	0.5	0.7	0.9
0	8.07×10^{-3}	1.65×10^{-2}	7.71×10^{-3}	3.81×10^{-3}	1.13×10^{-2}
0.1	6.34×10^{-3}	1.59×10^{-2}	8.43×10^{-3}	2.49×10^{-3}	1.07×10^{-2}
0.2	4.67×10^{-3}	1.44×10^{-2}	8.45×10^{-3}	1.12×10^{-3}	8.95×10^{-3}
0.3	2.91×10^{-3}	1.21×10^{-2}	7.64×10^{-3}	7.14×10^{-5}	6.59×10^{-3}
0.4	1.11×10^{-3}	9.19×10^{-3}	6.19×10^{-3}	6.34×10^{-4}	3.93×10^{-3}
0.5	6.15×10^{-4}	6.10×10^{-3}	4.45×10^{-3}	1.14×10^{-3}	1.09×10^{-3}
0.6	2.09×10^{-3}	3.18×10^{-3}	2.73×10^{-3}	1.58×10^{-3}	1.85×10^{-3}
0.7	3.15×10^{-3}	6.91×10^{-4}	1.23×10^{-3}	2.07×10^{-3}	4.88×10^{-3}
0.8	3.67×10^{-3}	1.19×10^{-3}	8.98×10^{-5}	2.70×10^{-3}	8.01×10^{-3}
0.9	3.51×10^{-3}	2.36×10^{-3}	5.91×10^{-4}	3.60×10^{-3}	1.14×10^{-2}
1	2.56×10^{-3}	2.73×10^{-3}	6.55×10^{-4}	4.98×10^{-3}	1.52×10^{-2}

and the Wavelet method is also provided in Table 4. In comparison to the wavelet method, our presented method is more reliable and consistent, as we can observe that the accuracy of the Wavelet method fluctuates highly at different internal points; however, the accuracy of our method changes smoothly. Furthermore, since we have a boundary condition at $x = 1$, we expect to have better accuracy around this point, and the results are in total agreement with this fact. Increasing the number of discretization points often results in improved accuracy in the model. However, as a consequence of the increased complexity of computation, we have made the decision to establish a maximum restriction of 1000 points for our experimental objectives. The experiments undertaken have shown that there exists a good trade-off between the execution time and accuracy of the model. The results of our trials indicate that the most rapid convergence was achieved while using a learning rate of 0.001.

The obtained results demonstrate that the current method can reliably evaluate the solution to the presented problem.

5. CONCLUSION

Solving the fractional differential equations using neural networks is a challenging task, especially considering that many fractional definitions have a singular kernel and cannot be directly computed. In this paper, we propose a framework to solve the fractional differential equations using $L1$ discretization and the Gauss-Legendre discretization to discretize the integral component. The method can be used in several different fields, including biological systems, medical imaging, stock prices, and control systems [16]. To demonstrate the effectiveness of the model, we considered several fractional equations, including an ODE, a PDE, and an integro-differential equation. Solving fractional equations using neural networks is a relatively new research area, and while previous works have contributed to this field, a solid framework for obtaining effective solutions to these equations remains lacking. By utilizing the aforementioned methodologies, we have developed a new and reliable framework to calculate the solutions to these equations. While earlier studies primarily addressed fractional differential equations without explicitly discretizing integral terms, our approach incorporates both Gauss–Legendre quadrature for nonlocal integral evaluation and the $L1$ scheme for Caputo-type fractional derivatives. Future extensions may involve exploring alternative integration rules, such as Gauss–Lobatto quadrature or adaptive schemes, as well as employing higher-order fractional discretizations like the $L1-2$ method. Our findings demonstrate that different discretization methods can be efficiently incorporated into a neural network; nevertheless, the effectiveness and usability of the method can be affected by parameters such as the depth of the neural network, activation functions, and the overall structure of the network. Using specialized quadrature schemes designed for weakly singular kernels (such as Gauss-Jacobi quadrature or adaptive quadrature) could be more accurate in theory, and we plan to explore these alternatives in future works. However, for the scope of the current paper, our numerical evidence justifies the use of Gauss-Legendre quadrature in this PINN-based architecture. Furthermore, the effectiveness of different discretization methods in solving various fractional definitions is yet

another subject that can be investigated in future work; nonetheless, the choice of the specific discretization method may depend on the specific characteristics of the equation and the structure of the neural network, notably the order of the fractional equation, the convergence order of the method, and the computational complexity of the method, which could in some cases severely hinder the speed and performance of our model. Finally, our findings suggest that the proposed method and, in general, discretization methods are very valuable and could serve as a strong foundation for further research in this area.

Acknowledgment

We would like to thank Amir Hossein Hadian Rasanan for introducing us to the concepts of PINNs and fPINNs, and also Dr. Fatemeh Baharifard for her useful insights. Their introduction inspired us to further investigate these areas.

Author Contribution

Hassan Dana Mazraeh: Writing - Original Draft, Writing - Review & Editing, Software, Methodology;
Ali Nosrati Firoozsalari: Writing - Original Draft, Software, Methodology;
Alireza Afzal Aghaei: Writing - Review & Editing, Methodology, Software;
Kourosh Parand: Supervision, Resources, Validation.

References

[1] A. A. Aghaei, K. Parand, A. Nikkhah, and S. Jaberi, *Solving Falkner-Skan type equations via Legendre and Chebyshev Neural Blocks*, arXiv preprint arXiv:2308.03337, (2023).
[2] T. Allahviranloo, A. Jafarian, R. Saneifard, N. Ghalami, S. Measoomy Nia, F. Kiani, U. Fernandez-Gamiz, and S. Noeiaghdam, *An application of artificial neural networks for solving fractional higher-order linear integro-differential equations*, Boundary Value Problems, *2023.1*(74) (2023).
[3] J. Cao, C. Li, and Y. Chen, *High-order approximation to Caputo derivatives and Caputo-type advection-diffusion equations (II)*, Fractional calculus and Applied analysis, *18*(3) (2015), 735-761.
[4] M. Caputo, *Linear Models of Dissipation whose Q is almost Frequency Independent—II*, Geophysical Journal International, *13*(5) (1967), 529-539.
[5] M. Cenci, M. Alessandra Congedo, A. Luciano Martire, and B. Rogo, *Fractional Volterra Integral Equations: A Neural Network Approach*,Roma TrE-Press, *1*(1) (2022).
[6] S. S. Chaharborj, S. S. Chaharborj, and Y. Mahmoudi, *Study of fractional order integro-differential equations by using Chebyshev neural network*, J. Math. Stat, *13*(1) (2017), 1-13.
[7] H. Dana Mazraeh and K. Parand, *Approximate symbolic solutions to differential equations using a novel combination of Monte Carlo tree search and physics-informed neural networks approach*, Engineering with Computers, (2025).
[8] H. Dana Mazraeh and K. Parand, *An innovative combination of deep Q-networks and context-free grammars for symbolic solutions to differential equations*, Engineering Applications of Artificial Intelligence, *142* (2025), 109733.
[9] H. Dana Mazraeh and K. Parand, *A three-stage framework combining neural networks and Monte Carlo tree search for approximating analytical solutions to the Thomas–Fermi equation*, Journal of Computational Science, *87* (2025), 102582.
[10] Ph. J. Davis and Ph. Rabinowitz, *Methods of numerical integration*, Courier Corporation, 2007.
[11] J. Sh. Duan, R. Rach, D. Baleanu, and A. M. Wazwaz, *A review of the Adomian decomposition method and its applications to fractional differential equations*, Communications in Fractional Calculus, *3*(2) (2012), 73-99.
[12] Kh. A. Gepreel, *The homotopy perturbation method applied to the nonlinear fractional Kolmogorov–Petrovskii–Piskunov equations*, Applied Mathematics Letters, *24*(8) (2011), 1428-1434.
[13] N. Heymans and J. -C. Bauwens, *Fractal rheological models and fractional differential equations for viscoelastic behavior*, Rheologica acta, *33* (1994), 210-219.
[14] Z. Hu, Kh. Shukla, G. E. Karniadakis, and K. Kawaguchi, *Tackling the curse of dimensionality with physics-informed neural networks*, Neural Networks, *176* (2024), 106369.

[15] F. Kheyrinataj and A. Nazemi, *Fractional power series neural network for solving delay fractional optimal control problems*, Connection Science, *32*(1) (2020), 53-80.

[16] A. A. Kilbas, H. M. Srivastava, and J. J. Trujillo, *Theory and Applications of Fractional Differential Equations*, North-Holland Mathematics Studies, 2006.

[17] Ch. Li and F. Zeng, *Finite difference methods for fractional differential equations*, International Journal of Bifurcation and Chaos, *22*(4) (2012), 1230014.

[18] Q. Liu, Y. T. Gu, P. Zhuang, F. Liu, and Y. F. Nie, *An implicit RBF meshless approach for time fractional diffusion equations*, Computational Mechanics, *48* (2011), 1-12.

[19] K. Miller and B. Ross, *An Introduction to the Fractional Calculus and Fractional Differential Equations*, Wiley, 1993.

[20] A. Nosrati Firoozsalari, A. Afzal Aghaei, and K. Parand, A machine learning framework for efficiently solving Fokker–Planck equations, Computational and Applied Mathematics, *43*(6) (2024).

[21] Z. M. Odibat and S. Momani, *Application of variational iteration method to nonlinear differential equations of fractional order*, International Journal of Nonlinear Sciences and Numerical Simulation, *7*(1) (2006), 27-34.

[22] K. B. Oldham, *Fractional differential equations in electrochemistry*, Advances in Engineering software, *41*(1) (2010), 9-12.

[23] K. Parand, A. A. Aghaei, S. Kiani, T. Ilkhas Zadeh, and Z. Khosravi, *A neural network approach for solving nonlinear differential equations of Lane–Emden type*, Engineering with Computers, *40*(1) (2023), 953–969.

[24] P. Rahimkhani, Y. Ordokhani, and E. Babolian, *Fractional-order Bernoulli wavelets and their applications, Applied Mathematical Modelling*, *40*(17) (2016), 8087-8107.

[25] M. Rahimy, *Applications of Fractional Differential Equations*, Applied Mathematical Sciences, *4*(50) (2010), 2453-2461.

[26] M. Raissi, P. Perdikaris, and G. E. Karniadakis, *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations*, Journal of Computational Physics, *378* (2019), 686-707.

[27] A. Rohul, S. Kamal, A. Muhammad, I. Khan, and F. Ullah, *An efficient algorithm for numerical solution of fractional integro-differential equations via Haar wavelet*, Journal of Computational and Applied Mathematics, *381* (2021), 113028.

[28] M. Saadat, M. Deepak, and S. Jamali, *UniFIDES: Universal fractional integro-differential equations solver*, APL Machine Learning, *3*(1) (2025), 16116.

[29] A. Saadatmandi, M. Dehghan, and M. Azizi, *The Sinc–Legendre collocation method for a class of fractional convection–diffusion equations with variable coefficients*, Communications in Nonlinear Science and Numerical Simulation, *17*(11) (2012), 4125-4136.

[30] R. Schumer, M.M. Meerschaert, and B. Baeumer, *Fractional advection-dispersion equations for modeling transport at the Earth surface*, Journal of Geophysical Research: Earth Surface, *114*(F4) (2009).

[31] P. Shaeri, S. AlKhaled, and A. Middel, *A multimodal physics-informed neural network approach for mean radiant temperature modeling*, arXiv:2503.08482, (2025).

[32] T. Taheri, A. A. Aghaei, and K. Parand, *Bridging machine learning and weighted residual methods for delay differential equations of fractional order*, Applied Soft Computing, *149* (2023), 110936.

[33] T. Taheri, A. Afzal Aghaei, and K. Parand, *A new kernel-based approach for solving general fractional (integro)-differential-algebraic equations*, Engineering with Computers, *41* (2025), 845-864.

[34] T. Taheri, A. Afzal Aghaei, and K. Parand, *Accelerating fractional PINNs using operational matrices of derivative*, arXiv:2401.14081, (2024).

[35] A. Toma and O. Postavaru, *A numerical method to solve fractional Fredholm-Volterra integro-differential equations*, Alexandria Engineering Journal, *68* (2023), 469-478.

[36] M. Wang, H. Li, H. Zhang, X. Wu, and N. Li, *PINN-MG: A physics-informed neural network for mesh generation*, arXiv:2503.00814, (2025).

[37] M. Zaslavsky, *Some applications of fractional equations*, Communications in Nonlinear Science and Numerical Simulation, *8*(3) (2023), 273-281.