Efficient Training of Neural Fractional-Order Differential Equation via Adjoint Backpropagation

Qiyu Kang[1], Xuhao Li[†2], Kai Zhao[3], Wenjun Cui[4], Yanan Zhao[3], Weihua Deng[5], Wee Peng Tay[3],

[1] University of Science and Technology of China
[2] Anhui University
[3] Nanyang Technological University
[4] Beijing Jiaotong University
[5] Lanzhou University

Abstract

Fractional-order differential equations (FDEs) enhance traditional differential equations by extending the order of differential operators from integers to real numbers, offering greater flexibility in modeling complex dynamic systems with nonlocal characteristics. Recent progress at the intersection of FDEs and deep learning has catalyzed a new wave of innovative models, demonstrating the potential to address challenges such as graph representation learning. However, training neural FDEs has primarily relied on direct differentiation through forward-pass operations in FDE numerical solvers, leading to increased memory usage and computational complexity, particularly in large-scale applications. To address these challenges, we propose a scalable adjoint backpropagation method for training neural FDEs by solving an augmented FDE backward in time, which substantially reduces memory requirements. This approach provides a practical neural FDE toolbox and holds considerable promise for diverse applications. We demonstrate the effectiveness of our method in several tasks, achieving performance comparable to baseline models while significantly reducing computational overhead.

Code — https://github.com/kangqiyu/torchfde

1 Introduction

Fractional calculus is a mathematical generalization of integer-order integration and differentiation, enabling the modeling of complex processes in physical systems beyond what traditional calculus can achieve. It finds applications across multiple disciplines, illustrating its versatility. For example, it characterizes viscoelastic materials (Coleman and Noll 1961), models population dynamics (Almeida, Bastos, and Monteiro 2016), enhances control systems (Podlubny 1994), improves signal processing (Machado, Kiryakova, and Mainardi 2011), supports financial modeling (Scalas, Gorenflo, and Mainardi 2000), and describes porous and fractal structures (Nigmatullin 1986; Mandelbrot and Mandelbrot 1982; Ionescu et al. 2017). Within these varied contexts, fractional-order differential equations (FDEs) serve as an enriched extension of traditional integer-order differential equations, providing a way to incorporate a continuum of past states into the present state. This generalization enables the capture of memory and non-locality effects inherent in various physical and engineering processes.

In the realm of deep learning, traditional neural Ordinary Differential Equations (ODEs) predominantly rely on integer-order differential equations, which can be conceptualized as continuous residual layers. (Weinan 2017; Chen et al. 2018; Kidger, Chen, and Lyons 2021). These models have been widely applied in areas such as content generation (Yang et al. 2023; Song et al. 2020), adversarial robustness (Kang et al. 2021; Yan et al. 2018), and physics modeling (Ji et al. 2021; Raissi, Perdikaris, and Karniadakis 2019; Lai et al. 2021). Despite their widespread success, integer-order ODEs struggle to effectively capture the complex memory-dependent characteristics of systems due to the inherent limitations of integer-order operators (Podlubny 1994). The integration of fractional calculus with deep learning has recently garnered interest from researchers in addressing the shortcomings of integer-order models. Innovations in this area include the application of fractional derivatives for optimizing parameters in neural networks (Liu et al. 2022), moving away from traditional gradient optimization methods like SGD or Adam (Kingma and Ba 2014). Furthermore, (Antil et al. 2020) has demonstrated that incorporating fractional calculus with its L1 approximation can enable networks to handle non-smooth data while addressing the vanishing gradient problem. Notably, in the field of physics-informed machine learning, the development of fractional physics-informed neural networks (fPINNs) (Pang, Lu, and Karniadakis 2019) highlights how these networks incorporate FDEs to embed physics principles, setting a new direction in the literature (Guo et al. 2022; Javadi et al. 2023; Wang, Zhang, and Jiang 2022). Additionally, the application of neural FDEs for updating graph hidden features has been shown to improve model performance, alleviate oversmoothing, and strengthen adversarial defense (Kang et al. 2024a,b; Zhao et al. 2024; Cui et al. 2025).

In contemporary research, training integer-order neural ODEs leverages reverse-mode differentiation by solving an augmented ODE backward in time, which is memory efficient (Chen et al. 2018). In contrast, training neural FDEs still relies on the basic automatic differentiation technique from leading platforms such as TensorFlow (Abadi et al. 2016)

[†]Corresponding author (lixh@ahu.edu.cn).

and PyTorch (Paszke et al. 2019). This approach involves backpropagation through forward-pass operations in FDE numerical solvers, requiring multiple iterations and are computationally intensive, which poses significant challenges for efficient training. To address this challenge, we introduce a scalable method that facilitates backpropagation by solving an augmented FDE backward in time. This method enables seamless end-to-end training of FDE-based modules within larger models and uses less memory. We validate our approach in practical tasks, demonstrating that it achieves performance on par with baseline models while significantly reducing computational memory demands during training.

Main contributions. Our key contributions are summarized as follows:

- We propose a novel and efficient method for training neural FDEs by solving an augmented FDE backward in time. This approach not only reduces computational training memory usage but also facilitates the efficient training of FDE components within larger models.
- We develop a practical neural FDE toolbox that has the potential for diverse applications. Our method has been successfully applied to practical graph representation learning tasks, achieving performance comparable to established baseline models while requiring less computational memory for training.

The rest of the paper is organized as follows: Section 2 offers a review of the literature related to the use of differential equations in machine learning. Section 3 briefly covers the mathematical foundations of fractional calculus to assist readers who are unfamiliar with it. Section 4 outlines our proposed neural FDE parameter backpropagation, detailing the rationale behind its design. Section 5 describes the experimental procedures and presents the results. We summarize and conclude the paper in Section 6.

2 Related Work

In this section, we provide a literature review of neural differential equations, fractional calculus, and their applications in machine learning.

2.1 Differential Equations and Machine Learning

The combination of differential equations with machine learning represents a significant leap forward in tackling complex problems across various domains (Raissi, Perdikaris, and Karniadakis 2019; Weinan 2017; Chen et al. 2018). This innovative approach fuses the accurate dynamic system modeling via differential equations with the high expressivity of neural networks. Among its many applications, particularly notable is its use in predictive modeling and simulation of physical systems. For instance, neural ODE (Chen et al. 2018) represents a significant advancement where the traditional layers of a neural network are replaced with continuous-depth models, enabling the network to learn complex dynamics with potentially fewer parameters and enhanced interpretability compared to standard deep learning models. Additionally, this integration may enhance neural network performance (Dupont, Doucet, and Teh 2019), stabilize gradients (Haber and Ruthotto 2017; Gravina, Bacciu, and Gallicchio 2022), and increase neural network robustness (Yan et al. 2018; Kang et al. 2021; Wang et al. 2023b). In computational fluid dynamics, machine learning models integrated with differential equations help in accelerating simulations without compromising on accuracy (Miyanawala and Jaiman 2017). Moreover, the integration of these disciplines also facilitates the development of data-driven discovery of differential equations. Techniques such as sparse identification of nonlinear dynamical systems (Brunton, Proctor, and Kutz 2016; Wang et al. 2023a) allow scientists to discover the underlying differential equations from experimental data, essentially learning the laws of physics governing a particular system. Overall, the synergy between differential equations and machine learning not only enhances computational efficiency and model accuracy but also opens up new research questions and methodologies. It promotes an active exchange of ideas and leads to innovations that could be beneficial across various domains.

2.2 Fractional Calculus and Its Applications

The field of fractional calculus, extending the traditional definitions of calculus to non-integer orders, has evolved significantly, offering new perspectives and tools for solving various scientific and engineering problems. This approach is pivotal for processes exhibiting anomalous or non-local properties that classical integer-order methods fail to capture. In engineering, fractional calculus enhances system control, achieving greater stability and performance with fractional-order controllers (Podlubny 1994). It also aids in modeling electrical circuits and materials more accurately (Kaczorek and Rogowski 2015). In physics, it provides a framework for describing anomalous diffusion in media like geological formations, capturing dynamics unexplained by classical theories (Diaz-Diaz and Estrada 2022; Sornette 2006). In finance, fractional derivatives model the heavy tails and memory effects of financial time series, leading to more precise risk management tools (Scalas, Gorenflo, and Mainardi 2000). Meanwhile, in medicine and biology, it models phenomena such as blood flow in aneurysms and epidemic dynamics, offering more accurate descriptions than traditional models (Krapf 2015; Chen et al. 2021; Yu, Perdikaris, and Karniadakis 2016). Recent studies combine FDEs with shadow neural networks, primarily applied in computational neuroscience (Anastasio 1994) and models like Hopfield networks (Kaslik and Sivasundaram 2012). These studies primarily engage with numerical simulations and delve into the bifurcation and stability dynamics within such networks. In recent advancements within deep learning research, (Liu et al. 2022) have pioneered the application of fractional derivatives for optimizing neural network parameters, offering an alternative to conventional methods such as SGD and Adam (Kingma and Ba 2014). Building on the theoretical underpinnings of fractional calculus, (Antil et al. 2020) utilize an L1 approximation of fractional derivatives to enhance the architecture of densely connected neural networks, addressing challenges associated with the vanishing gradient problem. Additionally, studies on FDE-based Graph Neural Networks (GNNs) (Kang et al. 2024a) have employed fractional diffusion and oscillator mechanisms to enhance graph representation learning,

achieving superior performance over traditional integer-order models. This approach demonstrates that neural FDEs with fractional derivatives can effectively model the updating and propagation of hidden features over a graph, offering both theoretical and practical advantages.

3 Preliminaries

Fractional calculus excels in modeling systems characterized by non-local interactions, where the system's future state is determined by its extensive historical context. Here, we provide a succinct introduction to the fundamental concepts of fractional calculus. Throughout this paper, we adhere to standard assumptions that ensure the well-posedness of our formulations, e.g., the validity of integrals and the existence and uniqueness of solutions to differential equations, as detailed in foundational texts (Diethelm and Ford 2009, 2002).

3.1 Fractional Calculus

Traditional Calculus: In traditional calculus, the first-order derivative of a scalar function $f(t)$ quantifies the local rate of change, defined as:

$$\frac{\mathrm{d}f(t)}{\mathrm{d}t} \equiv \dot{f}(t) \coloneqq \lim_{\Delta t \to 0} \frac{f(t+\Delta t) - f(t)}{\Delta t}. \tag{1}$$

We denote by J the operator that maps a function f, assumed to be (Riemann) integrable over the compact interval $[0, T]$, to its primitive centered at 0, i.e.,

$$Jf(x) \coloneqq \int_0^x f(t)\,\mathrm{d}t \quad \text{for } 0 \le x \le T. \tag{2}$$

For any integer $n \in \mathbb{N}^+$, the notation J^n represents the n-fold iteration of J, defined such that $J^1 := J$ and $J^n := JJ^{n-1}$ for $n \ge 2$. Equivalently, by using integration by parts, we have (Diethelm 2010)[Lemma 1.1.]:

$$J^n f(x) = \frac{1}{(n-1)!} \int_0^x (x-t)^{n-1} f(t)\,\mathrm{d}t \text{ with } n \in \mathbb{N}^+. \tag{3}$$

Fractional Integrals: The concept of a fractional integral generalizes the classical integral operator. Two commonly used definitions are the left- and right-sided Riemann-Liouville fractional integrals (Tarasov 2011)[page 4], denoted by J^β_{left} and J^β_{right} respectively, with $\beta \in \mathbb{R}^+$. These operators are defined as:

$$\begin{aligned} J^\beta_{\text{left}} z(t) &\coloneqq \frac{1}{\Gamma(\beta)} \int_0^t (t-u)^{\beta-1} z(u)\,\mathrm{d}u, \\ J^\beta_{\text{right}} z(t) &\coloneqq \frac{1}{\Gamma(\beta)} \int_t^T (u-t)^{\beta-1} z(u)\,\mathrm{d}u, \end{aligned} \tag{4}$$

where $\Gamma(\beta)$ is the gamma function, which extends the factorial function to complex and real number arguments. Unlike the integer-order n in traditional integrals (3), the order β in (4) can take any positive real value.

Fractional Derivatives: The fractional derivative extends the concept of differentiation to non-integer orders. The left- and right-sided Riemann-Liouville fractional derivatives ${}_{\text{left}}D^\beta_{\text{RL}}$ and ${}_{\text{right}}D^\beta_{\text{RL}}$ are formally defined as (Tarasov 2011)[page 385]:

$$\begin{aligned} {}_{\text{left}}D^\beta_{\text{RL}} z(t) &\coloneqq \frac{\mathrm{d}^m}{\mathrm{d}t^m} J^{m-\beta}_{\text{left}} z(t) \\ &= \frac{1}{\Gamma(m-\beta)} \frac{\mathrm{d}^m}{\mathrm{d}t^m} \int_0^t \frac{z(\tau)d\tau}{(t-\tau)^{\beta-m+1}} \\ {}_{\text{right}}D^\beta_{\text{RL}} z(t) &\coloneqq (-1)^m \frac{\mathrm{d}^m}{\mathrm{d}t^m} J^{m-\beta}_{\text{right}} z(t) \\ &\quad \frac{(-1)^m}{\Gamma(m-\beta)} \frac{\mathrm{d}^m}{\mathrm{d}t^m} \int_t^T \frac{z(\tau)d\tau}{(\tau-t)^{\beta-m+1}}, \end{aligned} \tag{5}$$

where m is an integer such that $m-1 < \beta \le m$. Similarly, the expressions ${}_{\text{left}}D^\beta_C$ and ${}_{\text{right}}D^\beta_C$ represent the left- and right-sided Caputo fractional derivatives, respectively, as detailed in (Tarasov 2011)[page 386]. They are defined as follows:

$$\begin{aligned} {}_{\text{left}}D^\beta_C z(t) &\coloneqq J^{m-\beta}_{\text{left}} \frac{\mathrm{d}^m}{\mathrm{d}t^m} z(t), \\ &= \frac{1}{\Gamma(m-\beta)} \int_0^t \frac{\frac{\mathrm{d}^m}{\mathrm{d}\tau^m} z(\tau)\,\mathrm{d}\tau}{(t-\tau)^{\beta-m+1}}, \\ {}_{\text{right}}D^\beta_C z(t) &\coloneqq (-1)^m J^{m-\beta}_{\text{right}} \frac{\mathrm{d}^m}{\mathrm{d}t^m} z(t) \\ &= \frac{(-1)^m}{\Gamma(m-\beta)} \int_t^T \frac{\frac{\mathrm{d}^m}{\mathrm{d}\tau^m} z(\tau)\,\mathrm{d}\tau}{(\tau-t)^{\beta-m+1}}. \end{aligned} \tag{6}$$

From the definitions, it becomes clear that fractional derivatives integrate the historical states of the function through the integral term, emphasizing their non-local, memory-dependent characteristics. Unlike integer-order derivatives that solely represent the local rate of change of the function at a specific point, fractional derivatives encapsulate a broader spectrum of the function's past behavior, providing a richer analysis tool in dynamic systems where history plays a crucial role. As the fractional order β approaches an integer, these fractional operators naturally converge to their classical counterparts (Diethelm 2010), ensuring a smooth transition from fractional to traditional calculus. For vector-valued functions, fractional derivatives and integrals are defined component-wise across each dimension, similar to the treatment in integer-order calculus.

3.2 First-Order Neural ODEs

In a neural ODE layer, the transformation from the initial feature $\mathbf{z}(0) = \mathbf{z}_0 \in \mathbb{R}^k$ to the output $\mathbf{z}(T) \in \mathbb{R}^k$ is governed by the differential equation:

$$\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}t} = f(t, \mathbf{z}(t); \boldsymbol{\theta}) \tag{7}$$

where the function f, mapping from $[0, \infty) \times \mathbb{R}^k$ to $\mathbb{R}^k$, encapsulates the layer's trainable dynamics for updating hidden features, parameterized by $\boldsymbol{\theta}$. The trajectory $\mathbf{z}(t)$ represents the continuous evolution of the system's hidden state. A notable technical challenge in training neural ODEs involves performing backpropagation. Direct differentiation using automatic differentiation techniques (Paszke et al. 2017) is

feasible but can incur significant memory costs and introduce numerical inaccuracies. To address this issue, (Chen et al. 2018) introduced the adjoint sensitivity method, originally proposed by Pontryagin (Pontryagin et al. 1962). This method efficiently computes the gradients of parameters $\boldsymbol{\theta}$ by constructing an augmented ODE that operates backward in time.

4 Neural FDE and Adjoint Backpropagation

In this section, we introduce the neural FDE framework, which utilizes a neural network to parameterize the fractional derivative of the hidden feature state. This approach integrates a continuum of past states into the current state, allowing for rich and flexible modeling of hidden features. We then propose an effective strategy for training the neural FDE by utilizing an augmented FDE in the reverse direction. Additionally, we describe the method for efficiently solving this augmented FDE.

4.1 Neural FDE

In our study, we adopt the Caputo fractional derivative in a manner akin to the approach described by (Kang et al. 2024a). The dynamics of the hidden units are modeled by the following neural FDE:

$$ {}_{\text{left}}D_C^{\beta}\mathbf{z}(t) = f(t, \mathbf{z}(t); \boldsymbol{\theta}), \quad 0 < \beta \leq 1. \tag{8} $$

In this formulation, f, parameterized by $\boldsymbol{\theta}$, is a function that maps $[0, \infty) \times \mathbb{R}^k$ to $\mathbb{R}^k$ and represents the trainable fractional derivatives of the hidden state. The system state $\mathbf{z}(t)$, starting from the initial condition $\mathbf{z}(0) = \mathbf{z}_0$, evolves up to a predetermined terminal time T. This terminal state $\mathbf{z}(T)$ is then used for downstream tasks such as classification or regression. For simplicity, we consider only $0 < \beta \leq 1$ without loss of generality, as higher orders can be converted to this range (Diethelm 2010), akin to how higher integer-order ODEs can be transformed into first-order systems with augmented states.

The computation of $\mathbf{z}(T)$ is achieved using a forward FDE solver. Classic solvers such as the fractional explicit Adams–Bashforth–Moulton (Diethelm, Ford, and Freed 2004) and the implicit L1 solver (Gao and Sun 2011; Sun and Wu 2006) can be employed. These methods demonstrate how time can serve as a continuous analog to discrete layer indices in traditional neural networks, similar to integer-order ODEs (Chen et al. 2018). To illustrate, we introduce the following iterative method from (Diethelm, Ford, and Freed 2004) to solve (8).

Predictor: Let h be a small positive discretization parameter. Consider a uniform grid spanning $[0, T]$ defined by $\{t_k = kh : k = 0, 1, \ldots, N\}$, where $h = \frac{T}{N}$. Let $\mathbf{z}_k$ be the numerical approximation of $\mathbf{z}(t_k)$. The basic predictor approximation is given by:

$$ \mathbf{z}_k^{\mathrm{P}} = \mathbf{z}_0 + \frac{1}{\Gamma(\beta)} \sum_{j=0}^{k-1} \mu_{j,k} f(t_j, \mathbf{z}_j; \boldsymbol{\theta}), \tag{9} $$

where $\mu_{j,k} = \frac{h^\beta}{\beta}\left((k-j)^\beta - (k-1-j)^\beta\right)$, and k denotes the discrete time index (iteration).

Corrector: The Predictor only provides a rough approximation of the true solution. To refine this approximation, the corrector formula from (Diethelm, Ford, and Freed 2004), a fractional variant of the one-step Adams-Moulton method, can be implemented using $\mathbf{z}_k^{\mathrm{P}}$ as follows:

$$ \begin{aligned} \mathbf{z}_k = \mathbf{z}_0 &+ \frac{1}{\Gamma(\beta)} \sum_{j=0}^{k-1} \eta_{j,k} f(t_j, \mathbf{z}_j; \boldsymbol{\theta}) \\ &+ \frac{1}{\Gamma(\beta)} \eta_{k,k} f(t_k, \mathbf{z}_k^{\mathrm{P}}; \boldsymbol{\theta}). \end{aligned} \tag{10} $$

The coefficients $\eta_{j,k}$ (Diethelm, Ford, and Freed 2004) are defined as follows: $\eta_{0,k}(\beta) = \frac{h^\beta}{\beta(\beta+1)}\left((k-1)^{\beta+1} - (k-1-\beta)k^\beta\right)$; for $1 \leq j \leq k-1$, $\eta_{j,k}(\beta) = \frac{h^\beta}{\beta(\beta+1)}\left((k-j+1)^{\beta+1} + (k-1-j)^{\beta+1} - 2(k-j)^{\beta+1}\right)$; and $\eta_{k,k}(\beta) = \frac{h^\beta}{\beta(\beta+1)}$.

4.2 Adjoint Parameter Backpropagation

From the fractional explicit Adams–Bashforth–Moulton formulations (9) and (10), it is evident that solving the neural FDE during the forward pass involves numerous iterations. The works (Kang et al. 2024a,b) train neural FDEs using the basic automatic differentiation techniques from PyTorch (Paszke et al. 2019). This method is computationally demanding, involving backpropagation through the numerous forward-pass iterations in FDE solvers. To address this challenge, we derive an adjoint method to compute gradients of the parameters $\boldsymbol{\theta}$. Echoing the methodology used in integer-order neural ODEs (Chen et al. 2018), this technique entails solving a secondary, augmented FDE in reverse time. While the complete derivation is detailed in the Appendix due to space constraints, we outline the critical steps here.

Consider a scalar-valued loss function $L(\cdot)$ that depends on the terminal state $\mathbf{z}(T)$. Our primary goal is to minimize $L(\mathbf{z}(T))$ with respect to $\boldsymbol{\theta}$. We aim to compute the gradient $\frac{\mathrm{d}L(\mathbf{z}(T))}{\mathrm{d}\boldsymbol{\theta}}$ for gradient descent. The strategy is to find a Lagrangian function $\lambda(t)$ to circumvent the direct computation of challenging derivatives such as $\frac{\mathrm{d}\mathbf{z}(T)}{\mathrm{d}\boldsymbol{\theta}}$ or $\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}\boldsymbol{\theta}}$. To this end, we consider the following optimization problem:

$$ \begin{aligned} \min_{\boldsymbol{\theta}} \quad & L\left(\mathbf{z}\left(T\right)\right) \\ \text{subject to} \quad & F\left({}_{\text{left}}D_C^{\beta}\mathbf{z}(t), \mathbf{z}(t), \boldsymbol{\theta}, t\right) \\ & \qquad := {}_{\text{left}}D_C^{\beta}\mathbf{z}(t) - f(t, \mathbf{z}(t); \boldsymbol{\theta}) = 0 \\ & \text{for all } t \in [0, T], \\ & \mathbf{z}\left(0\right) = \mathbf{z}_0. \end{aligned} \tag{11} $$

We define the augmented objective function ψ as

$$ \psi = L\left(\mathbf{z}\left(T\right)\right) - \int_0^T \lambda(t) F\left({}_{\text{left}}D_C^{\beta}\mathbf{z}(t), \mathbf{z}(t), \boldsymbol{\theta}, t\right) \mathrm{d}t. \tag{12} $$

Since $F\left({}_{\text{left}}D_C^{\beta}\mathbf{z}(t), \mathbf{z}(t), \boldsymbol{\theta}, t\right) = 0$ for all $t \in [0, T]$, the derivative $\frac{\mathrm{d}L(\mathbf{z}(T))}{\mathrm{d}\boldsymbol{\theta}}$ is the same as $\frac{\mathrm{d}\psi}{\mathrm{d}\boldsymbol{\theta}}$. For the second term

on the right-hand side of (12), we can get that

$$\int_0^T \lambda(t)F\,\mathrm{d}t = \lambda(T)J_{\text{left}}^{1-\beta}\mathbf{z}(T) - \mathbf{z}(0)J_{\text{right}}^{1-\beta}\lambda(0) + \int_0^T {}_{\text{right}}D_C^{\beta}\lambda(t)\mathbf{z}(t)\,\mathrm{d}t - \int_0^T \lambda f(t,\mathbf{z}(t);\boldsymbol{\theta})\,\mathrm{d}t.$$

The detailed intermediate steps to achieve this are provided in the Appendix due to space constraints. Take the derivative with respect to $\boldsymbol{\theta}$, we have

$$\frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left[\int_0^T \lambda F\,\mathrm{d}t\right] = \lambda\left(T\right)\frac{\mathrm{d}J_{\text{left}}^{1-\beta}\mathbf{z}(T)}{\mathrm{d}\boldsymbol{\theta}} + \int_0^T \left({}_{\text{right}}D_C^{\beta}\lambda(t) - \lambda(t)\frac{\partial f}{\partial \mathbf{z}}\right)\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\boldsymbol{\theta}}\,\mathrm{d}t - \int_0^T \lambda\frac{\partial f}{\partial \boldsymbol{\theta}}\,\mathrm{d}t \tag{13}$$

where we have used $\frac{\mathrm{d}\mathbf{z}(0)}{\mathrm{d}\boldsymbol{\theta}} = 0$ since $\mathbf{z}_0$ is the initial input hidden feature and it is not dependent on $\boldsymbol{\theta}$. It follows that

$$\begin{aligned}\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}} = \frac{\mathrm{d}\psi}{\mathrm{d}\boldsymbol{\theta}} &= \frac{\mathrm{d}L}{\mathrm{d}J_{\text{left}}^{1-\beta}\mathbf{z}(T)}\frac{\mathrm{d}J_{\text{left}}^{1-\beta}\mathbf{z}(T)}{\mathrm{d}\boldsymbol{\theta}} - \frac{\mathrm{d}}{\mathrm{d}\boldsymbol{\theta}}\left[\int_0^T \lambda F\,\mathrm{d}t\right] \\ &= \left(\frac{\mathrm{d}L}{\mathrm{d}J_{\text{left}}^{1-\beta}\mathbf{z}(T)} - \lambda\left(T\right)\right)\frac{\mathrm{d}J_{\text{left}}^{1-\beta}\mathbf{z}(T)}{\mathrm{d}\boldsymbol{\theta}} \\ &\quad - \int_0^T \left({}_{\text{right}}D_C^{\beta}\lambda(t) - \lambda(t)\frac{\partial f}{\partial \mathbf{z}}\right)\frac{\mathrm{d}\mathbf{z}}{\mathrm{d}\boldsymbol{\theta}}\,\mathrm{d}t + \int_0^T \lambda\frac{\partial f}{\partial \boldsymbol{\theta}}\,\mathrm{d}t\end{aligned} \tag{14}$$

To avoid the direct computation of challenging derivatives such as $\frac{\mathrm{d}\mathbf{z}(T)}{\mathrm{d}\boldsymbol{\theta}}$ or $\frac{\mathrm{d}\mathbf{z}(t)}{\mathrm{d}\boldsymbol{\theta}}$, we let $\lambda(t)$ satisfy the following FDE:

$$\begin{aligned}{}_{\text{right}}D_C^{\beta}\lambda(t) &= \lambda(t)\frac{\partial f}{\partial \mathbf{z}} \\ \text{with } \lambda\left(T\right) &= \frac{\mathrm{d}L}{\mathrm{d}J_{\text{left}}^{1-\beta}\mathbf{z}(T)}\end{aligned} \tag{15}$$

Consequently, as the first two terms in (14) vanish, we obtain

$$\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}} = -\int_T^0 \lambda(t)\frac{\partial f}{\partial \boldsymbol{\theta}}\,\mathrm{d}t. \tag{16}$$

To facilitate computation, we approximate the constraint on the last time point as $\frac{\mathrm{d}L}{\mathrm{d}J_{\text{left}}^{1-\beta}\mathbf{z}(T)} \approx \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(T)}$ in (15). This approximation represents the gradient of the loss with respect to the final state of the system. Efficient evaluation of the vector-Jacobian products, $\lambda(t)\frac{\partial f}{\partial \mathbf{z}}$ and $\lambda(t)\frac{\partial f}{\partial \boldsymbol{\theta}}$, specified in (15) and (16), is achieved using automatic differentiation, offering a computational cost on par with that of evaluating f directly.

In the reverse model, the systems described in (15) and (16) are computed simultaneously. Echoing the methodologies employed in (9) and (10), we utilize standard quadrature techniques to solve these equations. Our numerical iterations confirm that the coefficients are consistent with those reported in (9). Additionally, the trajectory $\mathbf{z}(t)$ generated during the forward pass can be efficiently reused. The complete numerical scheme is detailed in Section 4.3. Algorithm 1 outlines the procedure for constructing the required dynamics and applying the solver described in Section 4.3 to simultaneously compute all gradients.

Algorithm 1: Reverse-mode Differentiation for a Neural FDE

1: **Input:** Dynamics parameters $\boldsymbol{\theta}$, initial time 0, final time T, trajectory $\mathbf{z}(t)$, loss gradient $\mathrm{d}L/\,\mathrm{d}\mathbf{z}(T)$
2: **Objective:** Compute the gradients $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}}$ using reverse-time integration.
3: **Initialize:** Set $\lambda(T) = \frac{\mathrm{d}L}{\mathrm{d}\mathbf{z}(T)}$ and $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}}(T) = \mathbf{0}_{|\boldsymbol{\theta}|}$.
4: Solve the reverse-time FDE for $\lambda(t)$:

$${}_{\text{right}}D_C^{\beta}\lambda(t) = \lambda(t)\frac{\partial f}{\partial \mathbf{z}}, \quad \text{(Adjoint equation)}$$

5: Simultaneously compute the gradient with respect to parameters $\boldsymbol{\theta}$:

$$\frac{\mathrm{d}L/\,\mathrm{d}\boldsymbol{\theta}}{\mathrm{d}t} = \lambda(t)\frac{\partial f}{\partial \boldsymbol{\theta}}, \quad \text{(Parameter sensitivity)}$$

6: **Output:** Return the computed gradient $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}}$ upon completion of the integration from T to 0.

4.3 Solving the Reverse-Mode FDE

To solve the equations described in (15) and (16), we first convert (15) into the corresponding Volterra integral equation:

$$\lambda(t) = \lambda(T) + \frac{1}{\Gamma(\beta)}\int_t^T (s-t)^{\beta-1}\lambda(s)\frac{\partial f}{\partial \mathbf{z}}\,\mathrm{d}s. \tag{17}$$

Consider a small positive discretization parameter h, and a uniform grid over the interval $[0,T]$, defined by $\{t_k = kh \mid k = 0,1,\ldots,N\}$, where $h = \frac{T}{N}$. Let λ_k denote the numerical approximation of $\lambda(t_k)$. Using the product rectangle rule and initializing with $\lambda_N = \lambda(T)$, we derive the following basic predictor to iteratively compute λ_k:

$$\lambda_{N-k-1} = \lambda(T) + \frac{1}{\Gamma(\beta)}\sum_{j=N-k}^{N} b_{j,k+1}\lambda_j\frac{\partial f}{\partial \mathbf{z}_j}, \tag{18}$$

where the coefficients $b_{j,k+1}$ are given by:

$$b_{j,k+1} = \frac{h^{\beta}}{\beta}\left((j-(N-k-1))^{\beta} - (j-(N-k))^{\beta}\right).$$

Note that the full discretized trajectory $\{\mathbf{z}_j\}$, obtained from the forward pass in (9) and (10), can be reused during the backward computation.

For the integration in (16), we employ a basic Euler scheme using the same uniform time grid. Let us denote the numerical approximation of $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}}(t_k)$ as $g_k^{\boldsymbol{\theta}}$, initialized as $g_N^{\boldsymbol{\theta}} = \mathbf{0}_{|\boldsymbol{\theta}|}$, where $\mathbf{0}_{|\boldsymbol{\theta}|}$ denotes a vector of zeros with the same dimensionality as $\boldsymbol{\theta}$. We then compute:

$$g_{N-k-1}^{\boldsymbol{\theta}} = g_{N-k}^{\boldsymbol{\theta}} + \lambda_{N-k}\frac{\partial f}{\partial \boldsymbol{\theta}}, \tag{19}$$

Finally, we obtain $\frac{\mathrm{d}L}{\mathrm{d}\boldsymbol{\theta}} \approx g_0^{\boldsymbol{\theta}}$ and will be used as the gradient for backpropagation. While advanced corrector formulas could potentially offer more accurate integration to compute the gradient, this work only considers the basic predictor to solve the reverse-mode FDE (15) and (16). The adjoint

Method	Cora	Citeseer	Pubmed	CoauthorCS	Computer	Photo	CoauthorPhy	Ogbn-arxiv
GCN (Kipf and Welling 2017)	81.5±1.3	71.9±1.9	77.8±2.9	91.1±0.5	82.6±2.4	91.2±1.2	92.8±1.0	72.2±0.3
GAT (Veličković et al. 2018)	81.8±1.3	71.4±1.9	78.7±2.3	90.5±0.6	78.0±19.0	85.7±20.3	92.5±0.9	**73.7±0.1**
HGCN (Chami et al. 2019)	78.7±1.0	65.8±2.0	76.4±0.8	90.6±0.3	80.6±1.8	88.2±1.4	90.8±1.5	OOM
CGNN (Xhonneux et al. 2020)	81.4±1.6	66.9±1.8	66.6±4.4	92.3±0.2	80.3±2.0	91.4±1.5	91.5±1.8	58.7±2.5
GDE (Poli et al. 2019)	78.7±2.2	71.8±1.1	73.9±3.7	91.6±0.1	82.9±0.6	92.4±2.0	91.3±1.1	56.7±10.9
GRAND-l	83.6±1.0	73.4±0.5	78.8±1.7	92.9±0.4	83.7±1.2	92.3±0.9	93.5±0.9	71.9±0.2
GRAND-nl	82.3±1.6	70.9±1.0	77.5±1.8	92.4±0.3	82.4±2.1	92.4±0.8	91.4±1.3	71.2±0.2
F-GRAND-l	84.8±1.1	74.0±1.5	79.4±1.5	93.0±0.3	84.4±1.5	92.8±0.6	**94.5±0.4**	72.6±0.1
F-GRAND-nl	83.2±1.1	74.7±1.9	79.2±0.7	92.9±0.4	84.1±0.9	93.1±0.9	93.9±0.5	71.4±0.3
Adj-F-GRAND-l	**85.0±1.0**	**75.0±1.3**	**79.7±1.6**	**93.1±0.3**	86.9±1.4	**93.3±0.5**	94.0±0.5	72.5±0.3
Adj-F-GRAND-nl	82.6±1.3	74.6±1.9	78.5±1.5	92.8±0.3	**87.5±0.8**	92.5±0.8	93.8±0.6	72.6±0.3

Table 1: Node classification results(%) for random train-val-test splits. The best and the second-best results for each criterion are highlighted in bold and underlined, respectively.

backpropagation method is illustrated in Fig. 1, offering a visual depiction of the gradient computation process within the reverse-mode framework.

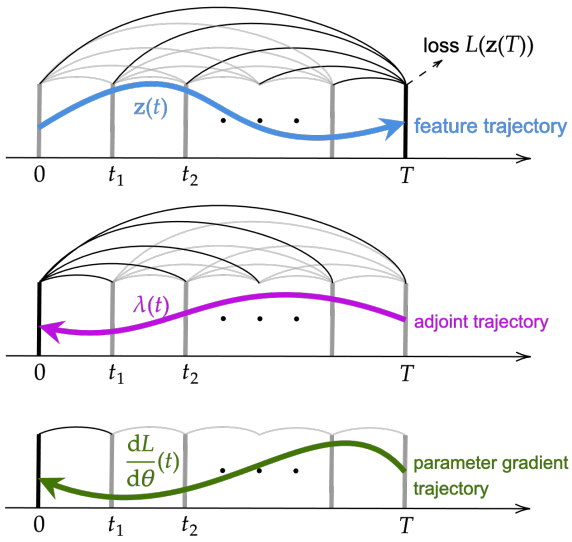

Figure 1: The visualization of the adjoint backpropagation method for training neural FDEs by solving an augmented FDE backward in time.

4.4 Model Complexity

During the forward pass, as described in (9), the process necessitates repeated computation of $f(t_k, \mathbf{z}_k; \boldsymbol{\theta})$ at each iteration. The total computational time complexity over the entire process can be expressed as $\sum_{k=0}^{N}(C + O(k))$, where $O(k)$ represents the computational overhead of summing and weighting the k terms at each step. Here, $N = \frac{T}{h}$ denotes the number of discretization (iteration) steps necessary for the integration process, and C indicates the computational complexity of the function f. This leads to a total cost of $O(NC + N^2)$. If there is a fast algorithm for the convolution computations, we typically require $O(N\log(N))$ for the convolution (Mathieu, Henaff, and LeCun 2013), resulting in $O(NC + N\log(N))$. The computational memory complexity is represented by $O(NM)$, with M denoting the memory requirement for the hidden state $\mathbf{z}_k$. During the backward phase, as indicated by computations in (18) and (19), a sequence of vector-Jacobian products is required. According to (Griewank 2003), computing these products demands approximately at most 2 to 3 times the computational time complexity compared to evaluating the original function f. The time complexity is again $O(NC + N^2)$. And the computational memory complexity remains at $O(NM)$. Overall, solving the reverse-mode FDE is nearly equivalent in complexity to the forward pass.

Direct differentiation through the forward pass (9) requires storing both intermediate states and their gradients with respect to all model parameters. This is crucial for calculating gradients of the loss function using backpropagation from the output back to the inputs, by applying the chain rule throughout the intermediate computations. The memory requirement increases to $O(N(M+P))$, where P is the parameter dimensionality necessary at each step, thereby exceeding $O(NM)$, particularly for large-dimensional $\boldsymbol{\theta}$. In contrast, the adjoint method is more memory-efficient.

5 Experiments

To evaluate the efficiency of our neural FDE solvers, experiments were conducted on three tasks: biological system FDE discovery, image classification, and graph node classification. The experiments in this section are designed to achieve two main objectives: 1) To verify that our adjoint FDE training accurately computes gradients and supports backpropagation. This is demonstrated in the small-scale problem described in Section 5.1, where the estimated parameters are shown to converge to the ground-truth values following the adjoint gradients. 2) To show that our adjoint FDE training is memory-efficient for large-scale problems. Experiments in Sections 5.2 and 5.3, particularly with the large-scale Ogbn-Products dataset, support this claim. It is important to note that our primary goal is to showcase the efficiency of the proposed adjoint backpropagation method for training neural FDEs by solving the augmented FDE backward in time, rather than achieving state-of-the-art results.

Model	MLP	Node2vec	GCN	GraphSAGE	GRAND-l	F-GRAND-l	F-GRAND-nl	Adj-F-GRAND-l	Adj-F-GRAND-nl
Accuracy	61.06±0.08	72.49±0.10	75.64±0.21	78.29±0.16	75.56±0.67	77.25±0.62	77.01±0.22	**78.36±0.32**	<u>78.33±0.20</u>

Table 2: Node classification accuracy(%) on Ogbn-products dataset. The best and the second-best results for each criterion are highlighted in bold and underlined, respectively.

Model	Test Error	Train. GPU Mem (MB)	Training Time (s)	Inf. GPU Mem (MB)	Inference Time (s)
Direct	0.39%	3612	1.46	1628	0.54
Adjoint	0.36%	2432	1.41	1628	0.54

Table 3: Comparative performance of direct differentiation vs. adjoint backpropagation on the MNIST dataset with $T = 20$.

Our empirical tests demonstrate that this approximation not only reduces the computational memory required for training but also maintains reasonable performance across various experimental setups. The experiments were conducted on a workstation running Ubuntu 20.04.1, equipped with an AMD Ryzen Threadripper PRO 3975WX with 32 cores and an NVIDIA RTX A5000 GPU with 24GB of memory.

5.1 Fractional Lotka-Volterra Model

We consider a nonlinear fractional Lotka-Volterra system, comprising two differential equations that describe the dynamics of biological systems where two species interact, one as a predator and the other as a prey:

$$
\begin{aligned}
{}_{\text{left}}D_C^{\beta} x &= x(a - cy) \\
{}_{\text{left}}D_C^{\beta} y &= -y(b - dx)
\end{aligned}
$$

where x and y represent prey and predator populations, respectively, and a, b, c, d are constants indicating interaction dynamics. We set the ground truth parameters as $[a, b, c, d] = [1.0, 0.5, 1.0, 0.3]$. Using synthetic data generated with these parameters and initial conditions randomly selected from $[0.5, 5]$, we train a model to estimate the parameters. The model uses the Adam optimizer (Kingma and Ba 2014) with a learning rate of 0.01. After 30 epochs, the estimated parameters, $[0.99, 0.48, 1.05, 0.33]$, closely match the true values, demonstrating the efficiency of our adjoint backpropagation. We do not validate the gradient in high-dimensional cases because these scenarios have many local minima, and there is no guarantee of achieving the global minimum.

5.2 Image Classification

This experiment evaluates the performance of neural FDEs on the MNIST dataset (LeCun et al. 1998), with a focus on comparing adjoint backpropagation to direct differentiation through forward-pass operations. The aim is to assess these models in terms of both accuracy and computational cost. Following the model architecture from (Chen et al. 2018), the input is downsampled twice. The model's hidden features are then updated following a neural FDE. Here, $f(t, \mathbf{z}(t); \boldsymbol{\theta})$ is configured as a convolution module. The step size h is set to 0.1, and the fractional order β is set to 0.5. This configuration implies a continuous analog to discrete layer indices in traditional neural networks, corresponding to approximately $N = T/h$ layers. Subsequently, a fully connected layer is applied to the extracted features for classification. In both the training and testing phases, the batch size is set to 128. In the first setting, gradients are backpropagated directly through forward-pass operations (9), while in the second setting, we solve the proposed reverse-mode FDE in Section 4.3 to obtain the gradients.

T	1	5	10	20	30	40	50	60	70	100	200
$\mathrm{M}_{\mathrm{dir}}$	1.8	2.2	2.7	3.6	4.6	5.5	6.5	7.4	8.4	11.2	20.7
$\mathrm{M}_{\mathrm{adj}}$	1.8	1.9	2.1	2.4	2.5	2.9	3.2	3.5	3.9	5.0	8.4
$\frac{\mathrm{M}_{\mathrm{adj}}}{\mathrm{M}_{\mathrm{dir}}}$	1.00	0.88	0.79	0.67	0.55	0.51	0.49	0.48	0.46	0.44	0.40

Table 4: GPU memory usage (GB) during training on the MNIST dataset across different integral times T. Notations $\mathrm{M}_{\mathrm{dir}}$ and $\mathrm{M}_{\mathrm{adj}}$ represent the memory usage using direct differentiation and adjoint backpropagation. The ration $\frac{\mathrm{M}_{\mathrm{adj}}}{\mathrm{M}_{\mathrm{dir}}}$ quantifies the relative memory consumption between the two methods.

The model's test accuracy, along with training and testing memory and time, are presented in Table 3 when $T = 20$. Both training methods achieve an accuracy of over 99.5%. Comparing the training memory, we find that training neural FDEs by solving the augmented FDE backward reduces memory usage by nearly 33% in this setting. Furthermore, we set integral times T ranging from 1 to 200 and record the GPU memory usage in Table 4. We observe that with sufficiently large T, adjoint backpropagation consumes only 40% of the training memory compared to direct differentiation.

Model	Adj-F-GRAND-l	F-GRAND-l	Adj-F-GRAND-nl	F-GRAND-nl
Inference Time (s)	0.102	0.102	0.185	0.185
Inf. GPU Mem. (MB)	3982	3982	4314	4314
Training Time (s)	0.319	0.352	0.785	0.806
Train. GPU Mem. (MB)	5570	8527	9086	18180

Table 5: Computation cost of models on the Ogbn-arxiv dataset: integral time $T = 10$ and step size of 1.

5.3 Node Classification on Graph Dataset

In this section, we validate the efficiency of our neural FDE solvers through experiments conducted across a range of graph node classification tasks, as outlined in (Kang et al. 2024a). These experiments utilize the neural FDE model F-GRAND, detailed in (Kang et al. 2024a)[Sec. 3.1.1]. It includes two variants, F-GRAND-nl and F-GRAND-l, denoting nonlinear and linear graph feature dynamics f, respectively. Utilizing our neural FDE toolbox, we aim to demonstrate that our solver can match the performance of traditional solvers (Kang et al. 2024a) that rely on standard automatic differentiation techniques from PyTorch without using adjoint backpropagation techniques. Models trained using our adjoint backpropagation techniques are prefixed with Adjoint- or Adj-. Moreover, we show that our neural FDE toolbox significantly reduces computational memory demand during training.

We follow the experimental setup from GRAND (Chamberlain et al. 2021), conducting experiments on homophilic datasets. We adopt the same dataset splitting method as in (Chamberlain et al. 2021), using the Largest Connected Component (LCC) and performing random splits. For the Ogbn-products dataset, we employ a mini-batch training approach as outlined in the paper (Zeng et al. 2020). For detailed information on the dataset and implementation specifics, please refer to the Appendix.

From Table 1, we observe that our adjoint backpropagation delivers comparable performance across all datasets on node classification tasks. This demonstrates the effectiveness of our proposed gradient computation using the proposed reverse-mode FDE. Furthermore, Table 2 includes the large-scale Ogbn-Products dataset, with 2449029 nodes and 61859140 edges. The memory efficiency of adj-F-GRAND enables the use of larger batch sizes, which contributes to improved classification outcomes. In our experiments, the batch size for adj-F-GRAND is set to 20,000 compared to 10,000 for F-GRAND when executed on the same GPU. Setting the F-GRAND batch size to 20000, however, leads to Out-Of-Memory (OOM) errors.

We also investigate the computational memory costs associated with these training methods with the other settings all the same. From Table 5 and Table 6, it is evident that our adjoint solvers significantly reduce computational memory costs during the training phase for the same model. Especially for the GRAND-nl model, which recomputes the attention score at each integration step, our adjoint solvers require only half the memory compared to traditional solvers. This highlights the remarkable efficiency of our adjoint method.

Model	Adj-F-GRAND-l	F-GRAND-l	Adj-F-GRAND-nl	F-GRAND-nl
Inference Time (s)	34.39	34.39	36.11	36.11
Inf. GPU Mem. (MB)	2678	2678	4468	4468
Training Time (s)	34.04	35.15	43.24	44.41
Train. GPU Mem. (MB)	6602	7950	11238	17210

Table 6: Computation cost of models on the Ogbn-products dataset: integral time $T = 10$ and step size of 1.

6 Conclusion

In this paper, we propose an efficient neural FDE training strategy by solving an augmented FDE backward in time, which substantially reduces memory requirements. Our approach provides a practical neural FDE toolbox and holds considerable promise for diverse applications. We demonstrate the effectiveness of our solver in image classification, biological system FDE discovery, and graph node classification. Our training using adjoint backpropagation can perform comparably to baseline models while significantly reducing computational overhead. The new neural FDE training technique will benefit the community by enabling more efficient use of computational resources and has the potential to scale to large FDE systems.

Acknowledgments

This research is supported by the National Research Foundation, Singapore and Infocomm Media Development Authority under its Future Communications Research and Development Programme. It is also supported by the National Natural Science Foundation of China under Grant Nos. 12301491, 12225107 and 12071195, the Major Science and Technology Projects in Gansu Province-Leading Talents in Science and Technology under Grant No. 23ZDKA0005, the Innovative Groups of Basic Research in Gansu Province under Grant No. 22JR5RA391, and Lanzhou Talent Work Special Fund. To improve the readability, parts of this paper have been grammatically revised using ChatGPT (OpenAI 2022).

References

Abadi, M.; Barham, P.; Chen, J.; Chen, Z.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Irving, G.; Isard, M.; et al. 2016. TensorFlow: a system for Large-Scale machine learning. In *12th USENIX symposium on operating systems design and implementation (OSDI 16)*, 265–283.

Almeida, R.; Bastos, N. R.; and Monteiro, M. T. T. 2016. Modeling some real phenomena by fractional differential equations. *Mathematical Methods in the Applied Sciences*, 39(16): 4846–4855.

Anastasio, T. J. 1994. The fractional-order dynamics of brainstem vestibulo-oculomotor neurons. *Biological cybernetics*, 72(1): 69–79.

Antil, H.; Khatri, R.; Löhner, R.; and Verma, D. 2020. Fractional deep neural network via constrained optimization. *Mach. Learn.: Sci. Technol.*, 2(1): 015003.

Brunton, S. L.; Proctor, J. L.; and Kutz, J. N. 2016. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the national academy of sciences*, 113(15): 3932–3937.

Chamberlain, B. P.; Rowbottom, J.; Goronova, M.; Webb, S.; Rossi, E.; and Bronstein, M. M. 2021. GRAND: Graph Neural Diffusion. In *Proc. Int. Conf. Mach. Learn.*

Chami, I.; Ying, Z.; Ré, C.; and Leskovec, J. 2019. Hyperbolic graph convolutional neural networks. In *Advances Neural Inf. Process. Syst.*

Chen, R. T.; Rubanova, Y.; Bettencourt, J.; and Duvenaud, D. 2018. Neural ordinary differential equations. In *Advances Neural Inf. Process. Syst.*

Chen, Y.; Liu, F.; Yu, Q.; and Li, T. 2021. Review of fractional epidemic models. *Applied mathematical modelling*, 97: 281–307.

Coleman, B. D.; and Noll, W. 1961. Foundations of linear viscoelasticity. *Rev. Modern Phys.*, 33(2): 239.

Cui, W.; Kang, Q.; Li, X.; Zhao, K.; Tay, W. P.; Deng, W.; and Li, Y. 2025. Neural Variable-Order Fractional Differential Equation Networks. In *Proc. AAAI Conference on Artificial Intelligence*. Philadelphia, USA.

Diaz-Diaz, F.; and Estrada, E. 2022. Time and space generalized diffusion equation on graph/networks. *Chaos, Solitons and Fractals*, 156: 111791.

Diethelm, K. 2010. *The analysis of fractional differential equations: an application-oriented exposition using differential operators of Caputo type*, volume 2004. Lect. Notes Math.

Diethelm, K.; and Ford, N. J. 2002. Analysis of fractional differential equations. *J. Math. Anal. Appl.*, 265(2): 229–248.

Diethelm, K.; and Ford, N. J. 2009. Numerical analysis for distributed-order differential equations. *J. Comput. Appl. Math.*, 225(1): 96–104.

Diethelm, K.; Ford, N. J.; and Freed, A. D. 2004. Detailed error analysis for a fractional Adams method. *Numer. Algorithms*, 36: 31–52.

Dupont, E.; Doucet, A.; and Teh, Y. W. 2019. Augmented neural odes. In *Advances Neural Inf. Process. Syst.*, 1–11.

Gao, G.-h.; and Sun, Z.-z. 2011. A compact finite difference scheme for the fractional sub-diffusion equations. *Journal of Computational Physics*, 230(3): 586–595.

Gravina, A.; Bacciu, D.; and Gallicchio, C. 2022. Anti-symmetric dgn: A stable architecture for deep graph networks. In *Proc. Int. Conf. Learn. Representations*.

Griewank, A. 2003. A mathematical view of automatic differentiation. *Acta Numerica*, 12: 321–398.

Guo, L.; Wu, H.; Yu, X.; and Zhou, T. 2022. Monte Carlo fPINNs: Deep learning method for forward and inverse problems involving high dimensional fractional partial differential equations. *Comput. Methods Appl. Mechanics Eng.*, 400: 115523.

Haber, E.; and Ruthotto, L. 2017. Stable architectures for deep neural networks. *Inverse Problems*, 34(1): 1–23.

Ionescu, C.; Lopes, A.; Copot, D.; Machado, J. T.; and Bates, J. H. 2017. The role of fractional calculus in modeling biological phenomena: A review. *Commun. Nonlinear Sci. Numer. Simul.*, 51: 141–159.

Javadi, R.; Mesgarani, H.; Nikan, O.; and Avazzadeh, Z. 2023. Solving Fractional Order Differential Equations by Using Fractional Radial Basis Function Neural Network. *Symmetry*, 15(6): 1275.

Ji, W.; Qiu, W.; Shi, Z.; Pan, S.; and Deng, S. 2021. Stiff-pinn: Physics-informed neural network for stiff chemical kinetics. *The Journal of Physical Chemistry A*, 125(36): 8098–8106.

Kaczorek, T.; and Rogowski, K. 2015. *Fractional linear systems and electrical circuits*. Springer.

Kang, Q.; Song, Y.; Ding, Q.; and Tay, W. P. 2021. Stable neural ode with Lyapunov-stable equilibrium points for defending against adversarial attacks. *Advances in Neural Information Processing Systems*, 34: 14925–14937.

Kang, Q.; Zhao, K.; Ding, Q.; Ji, F.; Li, X.; Liang, W.; Song, Y.; and Tay, W. P. 2024a. Unleashing the Potential of Fractional Calculus in Graph Neural Networks with FROND. In *Proc. International Conference on Learning Representations*. Vienna, Austria.

Kang, Q.; Zhao, K.; Song, Y.; Xie, Y.; Zhao, Y.; Wang, S.; She, R.; and Tay, W. P. 2024b. Coupling Graph Neural Networks with Fractional Order Continuous Dynamics: A Robustness Study. In *Proc. AAAI Conference on Artificial Intelligence*. Vancouver, Canada.

Kaslik, E.; and Sivasundaram, S. 2012. Nonlinear dynamics and chaos in fractional-order neural networks. *Neural networks*, 32: 245–256.

Kidger, P.; Chen, R. T. Q.; and Lyons, T. J. 2021. "Hey, that's not an ODE": Faster ODE Adjoints via Seminorms. *International Conference on Machine Learning*.

Kingma, D. P.; and Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kipf, T. N.; and Welling, M. 2017. Semi-Supervised Classification with Graph Convolutional Networks. In *Proc. Int. Conf. Learn. Representations*.

Krapf, D. 2015. Mechanisms underlying anomalous diffusion in the plasma membrane. *Current Topics Membranes*, 75: 167–207.

Lai, Z.; Mylonas, C.; Nagarajaiah, S.; and Chatzi, E. 2021. Structural identification with physics-informed neural ordinary differential equations. *Journal of Sound and Vibration*, 508: 116196.

LeCun, Y.; Bottou, L.; Bengio, Y.; and Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278–2324.

Liu, Z.; Wang, Y.; Luo, Y.; and Luo, C. 2022. A Regularized Graph Neural Network Based on Approximate Fractional Order Gradients. *Mathematics*, 10(8): 1320.

Machado, J. T.; Kiryakova, V.; and Mainardi, F. 2011. Recent history of fractional calculus. *Communications in nonlinear science and numerical simulation*, 16(3): 1140–1153.

Mandelbrot, B. B.; and Mandelbrot, B. B. 1982. *The fractal geometry of nature*, volume 1. WH freeman New York.

Mathieu, M.; Henaff, M.; and LeCun, Y. 2013. Fast training of convolutional networks through ffts. *arXiv preprint arXiv:1312.5851*.

Miyanawala, T. P.; and Jaiman, R. K. 2017. An efficient deep learning technique for the Navier-Stokes equations: Application to unsteady wake flow dynamics. *arXiv preprint arXiv:1710.09099*.

Nigmatullin, R. 1986. The realization of the generalized transfer equation in a medium with fractal geometry. *Physica status solidi (b)*, 133(1): 425–430.

OpenAI. 2022. ChatGPT-4. Available at: https://www.openai.com (Accessed: 10 April 2024).

Pang, G.; Lu, L.; and Karniadakis, G. E. 2019. fPINNs: Fractional physics-informed neural networks. *SIAM J. Sci. Comput.*, 41(4): A2603–A2626.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in pytorch. In *Advances Neural Inf. Process. Syst.*

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

Podlubny, I. 1994. Fractional-order systems and fractional-order controllers. *Institute of Experimental Physics, Slovak Academy of Sciences, Kosice*, 12(3): 1–18.

Poli, M.; Massaroli, S.; Park, J.; Yamashita, A.; Asama, H.; and Park, J. 2019. Graph neural ordinary differential equations. *arXiv preprint arXiv:1911.07532*.

Pontryagin, L. S.; Mishchenko, E.; Boltyanskii, V.; and Gamkrelidzem, R. 1962. *The Mathematical theory of optimal processes*. Routledge.

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

Scalas, E.; Gorenflo, R.; and Mainardi, F. 2000. Fractional calculus and continuous-time finance. *Physica A: Statistical Mechanics and its Applications*, 284(1-4): 376–384.

Song, Y.; Sohl-Dickstein, J.; Kingma, D. P.; Kumar, A.; Ermon, S.; and Poole, B. 2020. Score-based generative modeling through stochastic differential equations. *arXiv preprint arXiv:2011.13456*.

Sornette, D. 2006. *Critical phenomena in natural sciences: chaos, fractals, selforganization and disorder: concepts and tools*. Springer Science & Business Media.

Sun, Z.-z.; and Wu, X. 2006. A fully discrete difference scheme for a diffusion-wave system. *Applied Numerical Mathematics*, 56(2): 193–209.

Tarasov, V. E. 2011. *Fractional dynamics: applications of fractional calculus to dynamics of particles, fields and media*. Springer Science & Business Media.

Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Liò, P.; and Bengio, Y. 2018. Graph Attention Networks. In *Proc. Int. Conf. Learn. Representations*, 1–12.

Wang, H.; Fu, T.; Du, Y.; Gao, W.; Huang, K.; Liu, Z.; Chandak, P.; Liu, S.; Van Katwyk, P.; Deac, A.; et al. 2023a. Scientific discovery in the age of artificial intelligence. *Nature*, 620(7972): 47–60.

Wang, S.; Kang, Q.; She, R.; Tay, W. P.; Hartmannsgruber, A.; and Navarro, D. N. 2023b. RobustLoc: Robust Camera Pose Regression in Challenging Driving Environments. In *Proc. AAAI Conference on Artificial Intelligence*.

Wang, S.; Zhang, H.; and Jiang, X. 2022. Fractional physics-informed neural networks for time-fractional phase field models. *Nonlinear Dyn.*, 110(3): 2715–2739.

Weinan, E. 2017. A proposal on machine learning via dynamical systems. *Commun. Math. Statist.*, 1(5): 1–11.

Xhonneux; Louis-Pascal; Qu, M.; and Tang, J. 2020. Continuous graph neural networks. In *Proc. Int. Conf. Mach. Learn.*, 10432–10441.

Yan, H.; Du, J.; Tan, V. Y.; and Feng, J. 2018. On robustness of neural ordinary differential equations. In *Advances Neural Inf. Process. Syst.*, 1–13.

Yang, L.; Zhang, Z.; Song, Y.; Hong, S.; Xu, R.; Zhao, Y.; Zhang, W.; Cui, B.; and Yang, M.-H. 2023. Diffusion models: A comprehensive survey of methods and applications. *ACM Computing Surveys*, 56(4): 1–39.

Yu, Y.; Perdikaris, P.; and Karniadakis, G. E. 2016. Fractional modeling of viscoelasticity in 3D cerebral arteries and aneurysms. *Journal of computational physics*, 323: 219–242.

Zeng, H.; Zhou, H.; Srivastava, A.; Kannan, R.; and Prasanna, V. 2020. GraphSAINT: Graph Sampling Based Inductive Learning Method. arXiv:1907.04931.

Zhao, K.; Kang, Q.; Ji, F.; Li, X.; Ding, Q.; Zhao, Y.; Liang, W.; and Tay, W. P. 2024. Distributed-Order Fractional Graph Operating Network. In *Advances in Neural Information Processing Systems*. Vancouver, Canada.