Int. J. Nonlinear Anal. Appl. In Press, 1–17
ISSN: 2008-6822 (electronic)
http://dx.doi.org/10.22075/ijnaa.2026.38893.5535

Using PINN Machine Learning Method with DGM Network for Solving Time-Fractional Black–Scholes Models

Maryam Rostaminia[a], Mojtaba Moradipour[b,*], Ali Khani[a]

[a]Department of Mathematics, Azarbaijan Shahid Madani University, Tabriz, Iran
[b]Department of Mathematics, Lorestan University, Khorramabad, Iran

(Communicated by Haydar Akca)

Abstract

The classical Black-Scholes model fails to capture crucial market features such as memory effects and non-Markovian behavior. The time-fractional Black-Scholes equation (TFBSM) has emerged as a powerful alternative that incorporates these phenomena through fractional calculus. However, accurate numerical solution of TFBSMs remains challenging due to the non-local nature of fractional derivatives, complex boundary conditions, and the singularity in fractional operators. This paper introduces a novel hybrid framework that synergistically combines Physics-Informed Neural Networks (PINNs) with the Deep Galerkin Method (DGM) architecture. Our approach leverages the PINN methodology to directly embed the TFBSM governing equation, initial conditions, and boundary conditions into the optimization objective through a carefully designed loss function. The DGM network, with its specialized architecture resembling Long Short-Term Memory (LSTM) networks, provides enhanced capability for capturing long-term temporal dependencies essential for fractional calculus. The Caputo time-fractional derivative is accurately implemented using Gauss-Legendre quadrature with 100-300 nodes. Comprehensive numerical experiments on two benchmark problems demonstrate the effectiveness of our approach. For Example 1 (fractional ODE), the method achieves relative L^2 errors as low as 3.06×10^{-3} at $t = 1.0$ with $\alpha = 0.5$. For Example 2 (TFBSM with nonhomogeneous boundary conditions), relative L^2 errors remain below 2.40×10^{-2} across the entire time domain with $\alpha = 0.7$. This work presents the first successful integration of PINNs with DGM for solving time-fractional Black-Scholes equations. The key innovations include a novel PINN-DGM hybrid architecture tailored for fractional PDEs, accurate implementation of Caputo derivatives using Gauss-Legendre quadrature, and establishment of a robust, mesh-free paradigm for financial PDEs with memory effects. The numerical results confirm that the proposed method accurately approximates the solution with improved stability and convergence.

Keywords: Time-Fractional Black–Scholes Model, PINN Method, Deep Galerkin Method
2020 MSC: 35r11, 68t07, 91g20, 91g60

1 Introduction

Despite its initial success, numerous empirical studies have demonstrated that the classical Black-Scholes model fails to accurately reproduce several key market features, including heavy-tailed return distributions, volatility clustering,

*Corresponding author
Email addresses: maryam.rostaminia@azaruniv.ac.ir (Maryam Rostaminia), moradipour.mo@lu.ac.ir (Mojtaba Moradipour), khani@azaruniv.ac.ir (Ali Khani)

Received: September 2025 *Accepted:* May 2026

and non-Markovian behavior. In response to these shortcomings, various generalizations of the Black-Scholes model have been proposed, replacing the classical simplifying assumptions with more sophisticated tools. These include jump-diffusion models [21, 30], stochastic volatility models [13, 16], and combinations of the two. It was also observed that standard Brownian motion is not an adequate tool for modeling skewness and kurtosis in asset return distributions. Consequently, models based on Lévy processes [29], KoBoL processes [4, 20], the CGMY model [5], and other non-Gaussian processes were introduced. Later, it became evident that financial markets exhibit nonlocal and non-Markovian behaviors. This led to the development of fractional Black-Scholes models, which incorporate memory effects and historical dependencies via fractional-order derivatives [6, 8, 17, 36]. In the time-fractional Black-Scholes model, a time-fractional derivative is introduced into the equation to account for the historical influence on asset prices. This derivative is usually defined in the Caputo sense, which is more suitable than the Riemann–Liouville form when dealing with problems involving initial conditions. As a result, the model leads to a nonlocal partial differential equation that generally lacks closed-form solutions, necessitating the use of advanced numerical methods for practical applications. Among these methods are the Mittag-Leffler function, infinite series approaches such as LHPM, Laplace and Legendre wavelets, LHAM, finite difference techniques, Caputo methods, and neural networks [7, 12, 18, 22, 26, 31, 36, 39]. A recent review article [38] provides a summary and comparison of these techniques.

On the other hand, in recent years, machine learning—particularly neural networks—has made significant advances in diverse fields such as image classification, speech recognition, digital advertising, medicine, and fraud detection. Recent studies and publications have also demonstrated that neural networks possess strong capabilities in modeling phenomena and approximating functions. From a mathematical standpoint, according to the Universal Approximation Theorem, a sufficiently deep multilayer neural network with enough neurons can approximate any continuous function to an arbitrary degree of accuracy [15, 28]. One crucial approach for solving differential equations via machine learning is the direct use of the strong form of the equation [3, 11, 23, 24, 32, 33, 34, 27]. The term "Physics-Informed Neural Networks" (PINNs) for this approach was first introduced by Raissi et al. in [32], and has since gained broad adoption in the scientific literature.

Nowadays, PINNs are a prominent machine learning approach for solving a variety of equations, including fractional differential equations, integro-differential equations, and stochastic differential equations. A PINN is an unsupervised learning method, as it does not require labeled data generated from experiments or simulations. It is also a mesh-free algorithm that reformulates the differential equation problem (PDE) into an optimization problem. In this framework, instead of simply fitting data, the loss function is explicitly constructed based on the analytical form of the governing differential equation along with initial and boundary conditions. The neural network is then trained to minimize this loss, such that the output function approximates the proper solution of the equation, rather than merely fitting to observed data [9]. A typical PINN consists of three main components: a base neural network (usually a feedforward neural network, FNN), a physics-informed component that embeds the differential equation into the loss function, and a feedback mechanism for evaluating and minimizing the error. Raissi and collaborators mainly used FNNs in their works, but to explore the impact of different network types, other architectures such as CNNs (convolutional networks), RNNs (recurrent networks), GANs (generative adversarial networks), BDL (Bayesian deep learning), and DGM (Deep Galerkin Method) have also been explored [1, 2, 25, 35, 33, 40].

The DGM network, which has a structure similar to Long Short-Term Memory (LSTM) networks, is particularly well-suited for modeling time-dependent differential equations. It can be viewed as a natural combination of Galerkin methods and machine learning. Instead of forming a solution as a linear combination of basis functions, the solution is approximated using a neural network [33], hence the name Deep Galerkin Method (DGM).

In this study, the physical constraints of the problem—namely the equation, initial, and boundary conditions—are enforced through the PINN loss function. At the same time, the DGM architecture is used to approximate the solution space. The time-fractional derivative is implemented in the Caputo form, and its integral component is evaluated using Gauss-Legendre quadrature. To assess the model's performance, analytical benchmark examples are considered, and the implementation is carried out in Python using deep learning libraries such as TensorFlow.

The numerical results demonstrate that the combined DGM-PINN model is capable of accurately capturing the nonlocal and complex behavior of option pricing in the time-fractional Black-Scholes equation. The model also exhibits good stability concerning variations in the model parameters. Figures and tables supporting these results are reported in Section 5. The structure of this paper is as follows: Section 1 presents the introduction and a brief history; Section 2 outlines the theoretical background of the time-fractional Black–Scholes equation along with the two employed learning methods; Section 3 describes the proposed hybrid approach, including network architecture, loss function formulation, and training strategy; Section 4 reports the numerical implementation, numerical results, encompassing output plots, error comparisons, and sensitivity analyses; and finally, Section 5 provides conclusions. To the best of our knowledge, this is the first study to integrate PINNs with the DGM framework for solving TFBSMs. While previous

works primarily relied on standard feedforward PINNs, which often struggle with long-term temporal dependencies, our approach benefits from the recurrent-inspired structure of DGM to efficiently capture memory effects. In addition, fractional derivatives are discretized using Gauss–Legendre quadrature for improved accuracy. Together, these features yield a mesh-free and computationally efficient solver that is well-suited to the challenges of fractional financial models.

2 Preliminaries

2.1 The Classical Black-Scholes Equation and Caputo Derivative

Definition 2.1. [19, 10] The classical Black-Scholes equation for option pricing is written as:

$$\frac{\partial V}{\partial t} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0,$$

Where V is the option price, S is the underlying asset price, r is the risk-free interest rate, and σ is the volatility.

Definition 2.2. the Caputo fractional derivative with respect to time, for a function $f(t)$ defined as:

$${}^{C}D_t^{\alpha} f(t) = \frac{1}{\Gamma(n-\alpha)} \int_0^t \frac{f^{(n)}(\tau)}{(t-\tau)^{\alpha-n+1}} d\tau,$$

where $n = \lceil \alpha \rceil$ and $\Gamma(\cdot)$ is the gamma function.

2.2 Problem Formulation: The Time-Fractional Black-Scholes Equation

Definition 2.3. [6, 36] The problem addressed in this work is the following time-fractional Black-Scholes equation for pricing a European option:

$${}^{C}D_t^{\alpha} V(S,t) + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 V}{\partial S^2} + rS\frac{\partial V}{\partial S} - rV = 0, \quad S > 0,\ t \in [0,T] \tag{2.1}$$

where $0 < \alpha \leq 1$ is the order of the Caputo time-fractional derivative.

This equation is subject to initial and boundary conditions. For a European call option with strike price K, these conditions are defined as:

- Initial Condition (Payoff at Maturity):

$$V(S,T) = \max(S-K, 0) \tag{2.2}$$

- Boundary Conditions:

$$V(0,t) = 0 \tag{2.3}$$

$$\lim_{S\to\infty} V(S,t) \sim S - Ke^{-r(T-t)} \tag{2.4}$$

The objective is to find the function $V(S,t)$ that satisfies both the governing equation (2.1) and these constraints (2.2-2.4).

2.3 PINN Method

Before the official introduction of physics-informed neural networks, some studies utilized approaches where partial differential equations (PDEs) were directly incorporated in their strong form into neural networks, employing automatic differentiation techniques to compute gradients. These methods were able to avoid errors caused by discretization and numerical integration in variational formulations [3, 11, 23, 24, 33, 34].

One of these methods, the Deep Galerkin Method (DGM), was proposed before the formal introduction of the PINN framework in [32], leveraging the strong form of the PDE and automatic differentiation. Although these methods did not have a specific name at that time, conceptually, they were very close to what is now known as PINNs. Therefore, methods like DGM can be considered as conceptual precursors or foundational works for the development of PINNs.

For the first time, Raissi et al. [32] introduced the term "Physics-Informed Neural Networks" (PINNs) for this approach, which uses the strong form of PDEs.

Definition 2.4. [32] The mathematical structure of the PINN method is as follows. Consider a partial differential equation with initial and boundary conditions:

$$\begin{cases} \frac{\partial u}{\partial t} + \mathcal{N}[u](x,t) = 0, \quad (x,t) \in \Omega \subset \mathbb{R}^d \times [0,T] \\ u(x,0) = u_0(x), \quad x \in \Omega \\ u(x,t) = g(x,t), \quad (x,t) \in \partial\Omega \times [0,T] \end{cases} \tag{2.5}$$

Where $\mathcal{N}[\cdot]$ is a differential operator (including spatial derivatives) and $u(x,t)$ is the unknown solution of PDE that must be found. In the PINN method, a neural network with parameters θ approximates the unknown function $u_\theta(x,t)$. The required derivatives are computed using automatic differentiation.

The total loss function is defined as:

$$\mathcal{L}(\theta) = \mathcal{L}_{\text{PDE}} + \mathcal{L}_{\text{IC}} + \mathcal{L}_{\text{BC}}$$

Where the components are as follows:

PDE Residual Loss:.

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| \frac{\partial u_\theta}{\partial t}(x_i,t_i) + \mathcal{N}[u_\theta](x_i,t_i) \right|^2$$

Initial Condition Loss:.

$$\mathcal{L}_{\text{IC}} = \frac{1}{N_{IC}} \sum_{j=1}^{N_{IC}} |u_\theta(x_j,0) - u_0(x_j)|^2$$

Boundary Condition Loss:.

$$\mathcal{L}_{\text{BC}} = \frac{1}{N_{BC}} \sum_{k=1}^{N_{BC}} |u_\theta(x_k,t_k) - g(x_k,t_k)|^2$$

Finally, by minimizing $\mathcal{L}(\theta)$ using optimization algorithms like Adam or L-BFGS, the parameters θ and the approximate solution $u_\theta(x,t)$ are obtained.

2.4 DGM Method

The DGM method, introduced by Sirignano and Spiliopoulos in 2018, is a deep learning–based approach for solving partial differential equations (PDEs) in a mesh-free manner. Unlike classical numerical methods, this approach defines a deep neural network as an approximation to the PDE solution and trains it directly by minimizing the PDE residual. The network used in DGM has a special architecture designed to overcome the vanishing gradient problem in deep networks. Its structure resembles LSTM [14] but is adapted for continuous spatiotemporal data.

A prominent feature of DGM is its unique neural network architecture tailored for effectively solving PDEs. This architecture is designed to capture complex temporal and spatial dependencies better. The DGM network combines fully connected layers and recurrent neural network (RNN) cells, especially Long Short-Term Memory (LSTM) units. Fully connected layers extract nonlinear features and model complex relations among input variables (such as time and space). At the same time, LSTM cells enable the network to capture long-term dependencies by storing and retrieving information over extended periods.This is particularly important for PDEs with multi-dimensional or time-dependent variables.

Typically, the DGM architecture includes one or more LSTM layers interleaved with fully connected layers. In brief, the data flow is as follows: inputs (e.g., spatial and temporal coordinates) enter the network and first pass through one or more fully connected layers. Then the data is fed to LSTM layers to extract sequential and spatiotemporal dependencies. Finally, the network output is generated through additional fully connected layers that provide the approximate solution function. This design enhances DGM's ability to model complex differential equations while being computationally efficient. Furthermore, automatic differentiation is used during training; Training to compute PDE derivatives accurately, reducing numerical errors.

Definition 2.5. [33] In a single DGM layer, mini-batch inputs in the form (x,t), together with the previous layer output, are transformed through the following operations:

$$
\begin{aligned}
S^1 &= \sigma\left(w^1 x + b^1\right) \\
Z^\ell &= \sigma\left(u_z^\ell \cdot x + w_z^\ell \cdot S^\ell + b_z^\ell\right) && \ell = 1, \ldots, L \\
G^\ell &= \sigma\left(u_g^\ell \cdot x + w_g^\ell \cdot S^\ell + b_g^\ell\right) && \ell = 1, \ldots, L \\
R^\ell &= \sigma\left(u_r^\ell \cdot x + w_r^\ell \cdot S^\ell + b_r^\ell\right) && \ell = 1, \ldots, L \\
H^\ell &= \sigma\left(u_h^\ell \cdot x + w_h^\ell \cdot (S^\ell \odot R^\ell) + b_h^\ell\right) && \ell = 1, \ldots, L \\
S^{\ell+1} &= (1 - G^\ell) \odot H^\ell + Z^\ell \odot S^\ell && \ell = 1, \ldots, L
\end{aligned}
$$

The model output is then.

$$f(t, x; \theta) = w \cdot S^{L+1} + b$$

where $\odot$ denotes element-wise (Hadamard) product, σ is the activation function, and L is the number of layers. The parameters u_j^ℓ, w_j^ℓ and b_j^ℓ are components of θ, initialized with random values and subsequently optimized during the training process.

This structure allows the network features to be recursively passed to higher layers with gradient control, which is particularly effective for solving time-dependent PDEs. A schematic overview of this architecture is shown in Figure 1.

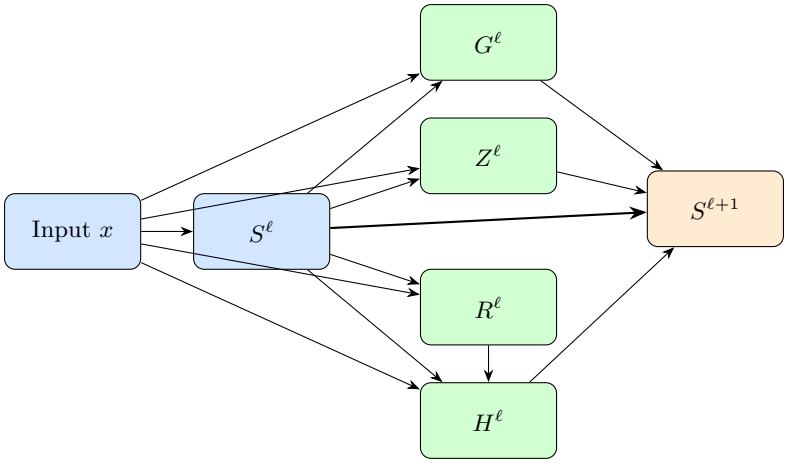

Figure 1: A schematic of a single layer of the DGM network architecture

3 Application of the PINN-DGM Method for Solving Time-Fractional Black-Scholes Equations

In this section, we utilize the combination of the physics-informed training philosophy of PINN and the structural capabilities of DGM networks to solve fractional differential equations. For the fractional part involving Caputo derivatives, we employ the Gauss–Legendre numerical integration method to handle the integral term. Specifically, the interval $[0, t]$ is divided into N_q quadrature points, and the integral is approximated as follows:

$$\int_0^t \frac{f^{(n)}(\tau)}{(t-\tau)^{\alpha-n+1}} d\tau \approx \sum_{i=1}^{N_q} w_i \cdot \frac{f^{(n)}(\tau_i)}{(t-\tau_i)^{\alpha-n+1}}$$

Where τ_i are the quadrature points and w_i are the corresponding Gauss–Legendre weights, In the implementation, standard libraries such as `numpy.polynomial.legendre.leggauss` are used to obtain these nodes and weights. The loss function is then defined accordingly, and the network is trained. The Caputo fractional derivative of order α for the function $V(t)$, being an integral operator, is computed using the **Gauss-Legendre quadrature** numerical method. This method is known for its high accuracy in approximating integrals and is particularly suitable for singular kernels (such as $(t-\tau)^{-\alpha}$ in the Caputo derivative). In this approach, the interval $[0, t]$ is discretized into N_q points, and the integral is approximated as a weighted sum of the function values at these points [10].

3.1 Overall Architecture of the Combined PINN-DGM Model

The total loss function is defined as a weighted sum of three distinct loss components:

$$\mathcal{L}_{\text{total}} = \lambda_{\text{PDE}} \cdot \mathcal{L}_{\text{PDE}} + \lambda_{\text{IC}} \cdot \mathcal{L}_{\text{IC}} + \lambda_{\text{BC}} \cdot \mathcal{L}_{\text{BC}}$$

where:

- PDE Residual Loss: This term penalizes the deviation from satisfying the governing equation at collocation points sampled from the domain Ω.

$$\mathcal{L}_{\text{PDE}} = \frac{1}{N_r} \sum_{i=1}^{N_r} \left| {}^{C}D_t^{\alpha} V_\theta(S_i, t_i) + \frac{1}{2}\sigma^2 S_i^2 \frac{\partial^2 V_\theta}{\partial S^2}(S_i, t_i) + r S_i \frac{\partial V_\theta}{\partial S}(S_i, t_i) - r V_\theta(S_i, t_i) \right|^2$$

- Initial Condition Loss: This term ensures the network approximates the payoff condition accurately at the initial time.

$$\mathcal{L}_{\text{IC}} = \frac{1}{N_{IC}} \sum_{j=1}^{N_{IC}} |V_\theta(S_j, T) - \max(S_j - K, 0)|^2$$

- Boundary Condition Loss: This term enforces the boundary constraints.

$$\mathcal{L}_{\text{BC}} = \frac{1}{N_{BC}} \sum_{k=1}^{N_{BC}} |V_\theta(S_k, t_k) - g(S_k, t_k)|^2$$

Here, $g(S,t)$ represents the boundary condition value. The coefficients λ_{PDE}, λ_{IC}, and λ_{BC} are scaling factors chosen to balance the contribution of each loss term to the total error, thereby facilitating stable training convergence.

3.2 General Algorithm of the Proposed Method

While the DGM network was previously applied to solve classical PDEs such as European and American Black–Scholes equations [33], in this paper, our goal is to solve fractional differential equations, for which the implemented model achieved satisfactory results.

Additionally, through various experiments, we found that the *swish* activation function outperforms *sigmoid*; hence, unlike previous studies, we employ *swish* as the activation function in our network.

The overall algorithm consists of:

1. Defining the spatial and temporal domain and generating random training samples.
2. Defining the hybrid DGM neural network architecture comprising fully connected and LSTM layers.
3. Implementing the Caputo fractional derivative via Gauss–Legendre numerical integration.
4. Defining the total loss function including PDE, initial, and boundary condition terms.
5. Training the network using optimization algorithms.

4 Numerical Results and Model Analysis

In this section, the performance of the PINN-DGM model for solving fractional differential equations is evaluated. Two numerical examples with known analytical solutions are considered: first, a fractional ODE, and then a fractional Black–Scholes equation. The numerical results include plots comparing the numerical and exact solutions, relative error analysis, and stability evaluation over time.

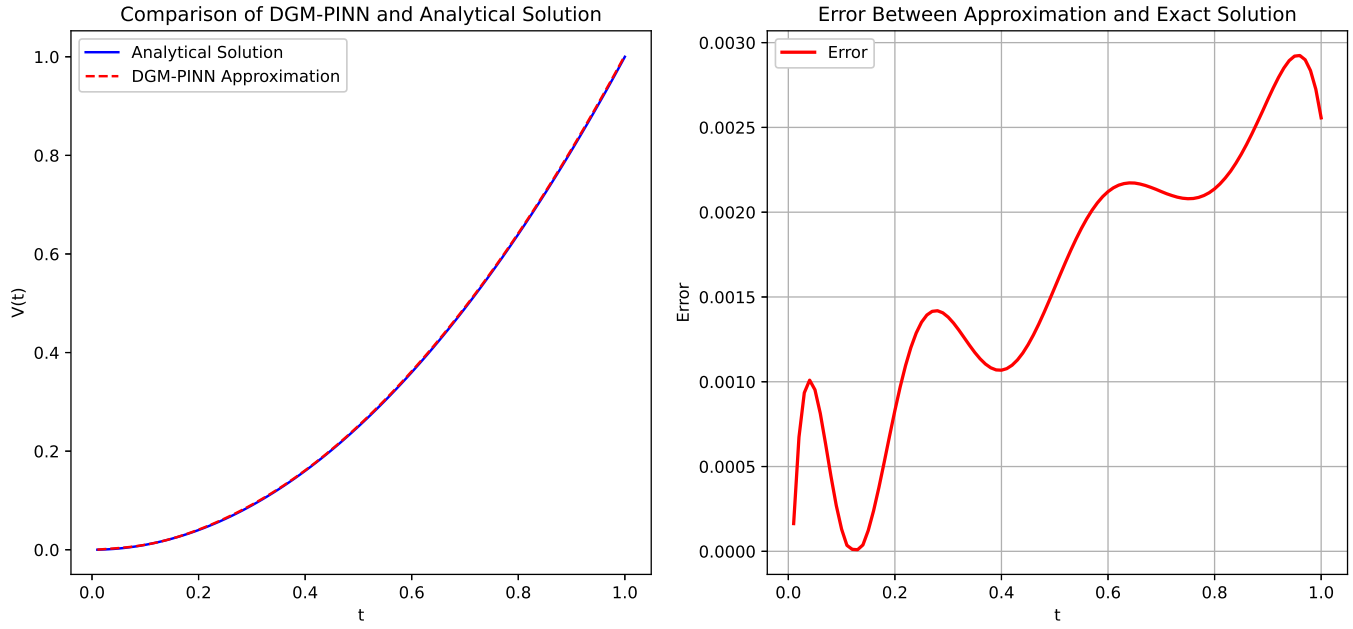

Figure 2: Comparison of numerical and exact solutions for $\alpha = 0.5$ in Example 1

4.1 Example 1

Consider the following fractional differential equation:

$$\begin{cases} {}^{C}D_{\tau}^{\alpha}U(\tau) = \dfrac{2\tau^{3/2}}{\Gamma(5/2)} - U(\tau) + \tau^{2}, \\ U(0) = 0, \end{cases}$$

the exact solution is:

$$U(t) = t^{2}.$$

To solve this example, samples are taken from the domain t, and the network is trained as described above. Figure 2 shows the network prediction and analytical solution on the same plot, demonstrating good agreement, along with the absolute error between them. Figure 3 presents the logarithmic plot of the loss function, whose smoothness indicates the stability of the method. Relative errors at different times for Example 1 are reported in Table 1. For this numerical experiment, a total of 50 sampling stages, each consisting of 25 steps, were utilized. The neural network was designed with two hidden layers, each comprising 25 neurons, and the learning rate was fixed at 0.009. In addition, 100 nodes were employed in the Gauss–Legendre quadrature scheme.

4.2 Example 2

Consider the following time-fractional Black–Scholes equation with nonhomogeneous boundary conditions:

$$\begin{cases} {}^{C}D_{\tau}^{\alpha}U(x,\tau) = a\dfrac{\partial^{2}U}{\partial x^{2}} + b\dfrac{\partial U}{\partial x} - cU + f(x,\tau), \\ U(0,\tau) = (t+1)^{2}, \quad U(1,\tau) = 3(t+1)^{2}, \\ U(x,0) = x^{3} + x^{2} + 1, \end{cases}$$

where the source term f is defined as:

$$f = \left(\frac{2\tau^{2-\alpha}}{\Gamma(3-\alpha)} + \frac{2\tau^{1-\alpha}}{\Gamma(2-\alpha)}\right)(x^{3}+x^{2}+1) - (t+1)^{2}\left[a(6x+2) + b(3x^{2}+2x) - c(x^{3}+x^{2}+1)\right]$$

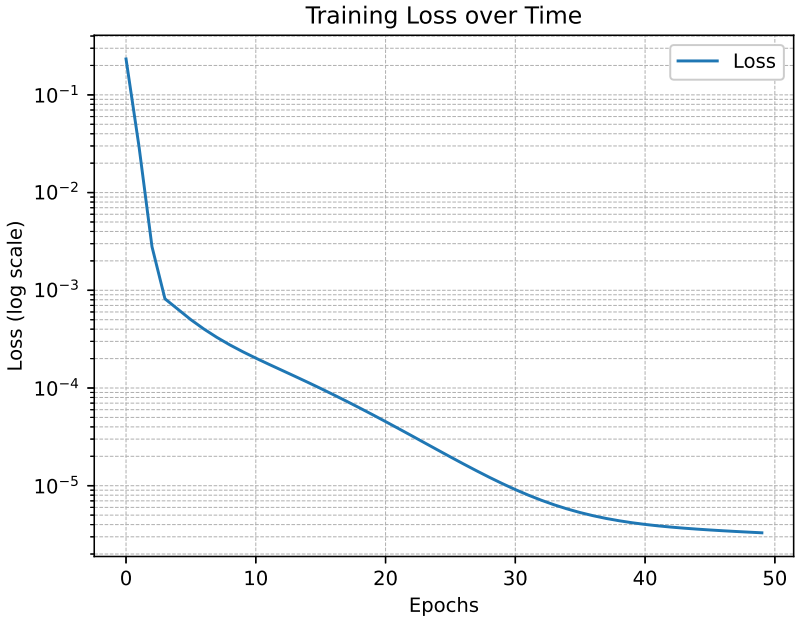

Figure 3: Logarithmic loss curve for Example 1

Table 1: Relative L_2 error between numerical and exact solutions at various times in Example 1

Time	Relative L_2 Error
$t = 0.1$	5.30×10^{-2}
$t = 0.2$	2.16×10^{-2}
$t = 0.3$	6.05×10^{-3}
$t = 0.4$	9.72×10^{-3}
$t = 0.5$	7.42×10^{-3}
$t = 0.6$	4.22×10^{-3}
$t = 0.7$	4.04×10^{-3}
$t = 0.8$	4.14×10^{-3}
$t = 0.9$	2.97×10^{-3}
$t = 1.0$	3.06×10^{-3}

And is chosen such that the exact solution is [37]:

$$U = (t+1)^2(x^3 + x^2 + 1).$$

The parameters are set as:

$$r = 0.5, \quad \alpha = 0.7, \quad a = 1, \quad b = r - a, \quad c = r, \quad T = 1.$$

We solve this example using the PINN-DGM method as described earlier. Sampling from the spatiotemporal domain, the DGM network is trained to approximate the option price using the Adam optimizer, implemented in TensorFlow. This example is two-dimensional and more complex; therefore, the number of mini-batches as well as the Gauss–Legendre quadrature points are increased. In this setting, 100 sampling stages, each consisting of 15 steps, are utilized. The neural network architecture consists of two hidden layers with 25 neurons in each layer, and the number of Gauss–Legendre nodes is set to 100. The learning rate is fixed at 0.0019."

Figure 4 compares the 3D numerical solution with the exact solution at different times, showing close agreement. For a more detailed error analysis, Figure 5 depicts the solution comparison at various times, demonstrating the

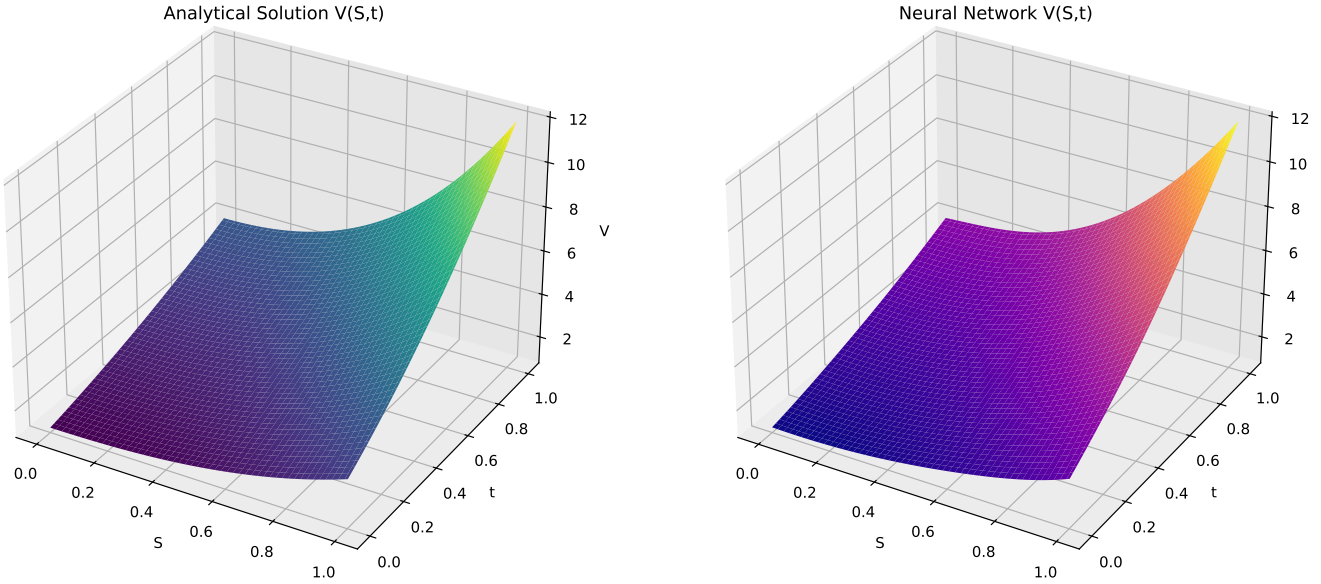

Figure 4: 3D comparison of numerical and exact solutions for $\alpha = 0.7$ in Example 2

model's high accuracy up to maturity. Figure 6 presents a heatmap of the absolute error, while Figure 7 shows the logarithmic loss curve indicating stable training behavior with Adam optimization. Relative errors at different times for Example 2 are reported in Table 2.

Table 2: Relative L_2 error between numerical and exact solutions at various times in Example 2

Time	Relative L_2 Error
$t = 0$	1.11×10^{-2}
$t = 0.2$	2.40×10^{-2}
$t = 0.4$	1.47×10^{-2}
$t = 0.6$	1.43×10^{-2}
$t = 0.8$	1.44×10^{-2}
$t = 1$	7.59×10^{-3}

In all tests, the hybrid DGM-PINN model demonstrated satisfactory convergence and stable behavior over various time intervals. The training loss decay over time is shown in Figure 8.

4.3 Comparison with Standard PINN

To demonstrate the advantage of the DGM architecture within our hybrid framework, we conducted a comparative analysis between the proposed PINN-DGM model and a standard Physics-Informed Neural Network (PINN) with a conventional feedforward neural network (FNN) architecture. Both models were trained on the same problem under identical conditions.

As quantitatively demonstrated in Table 3, our PINN-DGM model significantly outperforms the standard PINN approach across all key performance metrics:

- Superior Accuracy: The PINN-DGM model achieves a mean absolute error of 0.044922, which is approximately 3.33 times lower than the standard PINN's error of 0.149391. This substantial improvement highlights the DGM architecture's enhanced capability in capturing the complex dynamics of time-fractional derivatives.
- Computational Efficiency: Remarkably, the training time for PINN-DGM is only 26.37 seconds compared to

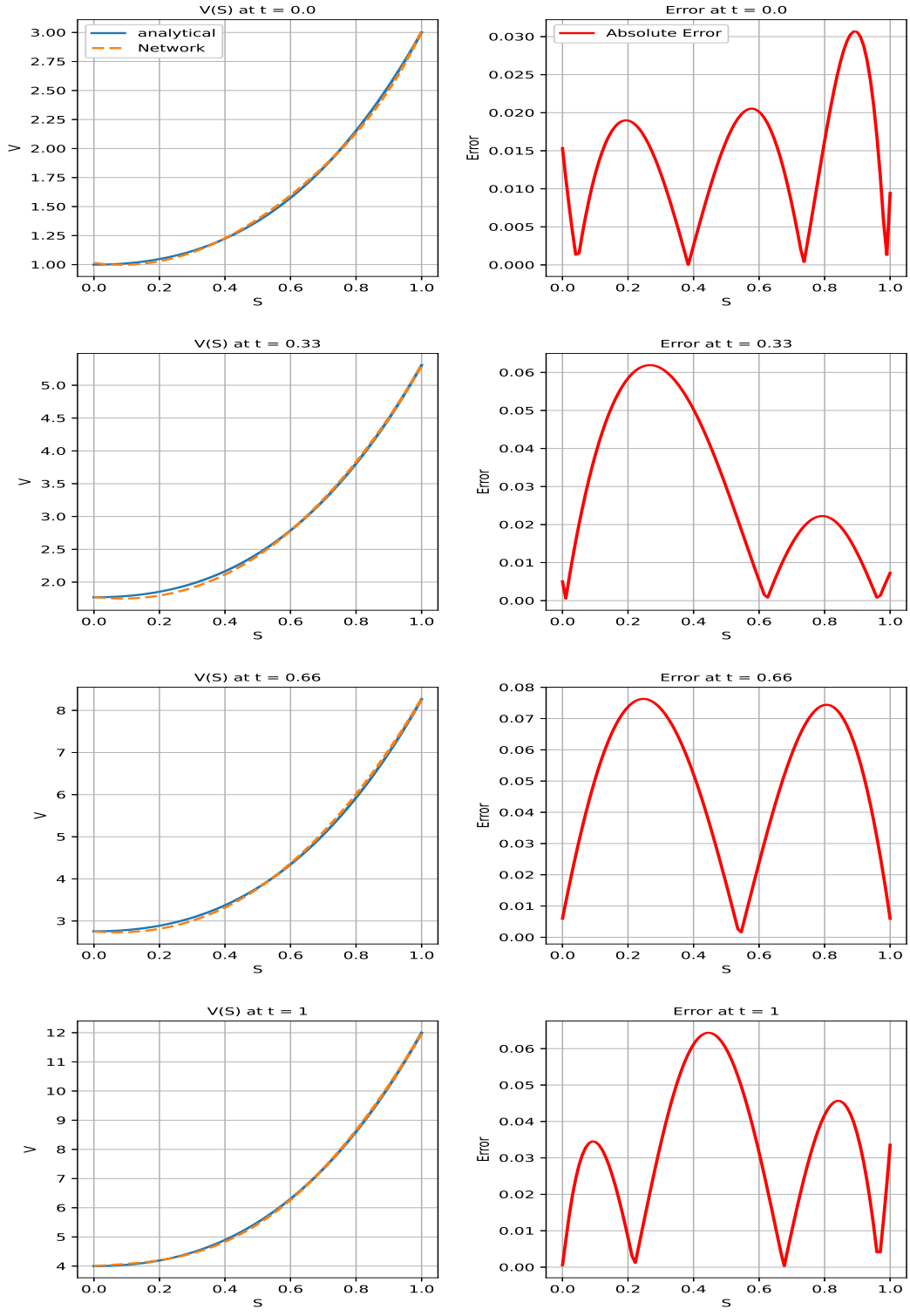

Figure 5: Comparison of numerical and exact solutions for $\alpha = 0.7$, at various times in Example 2

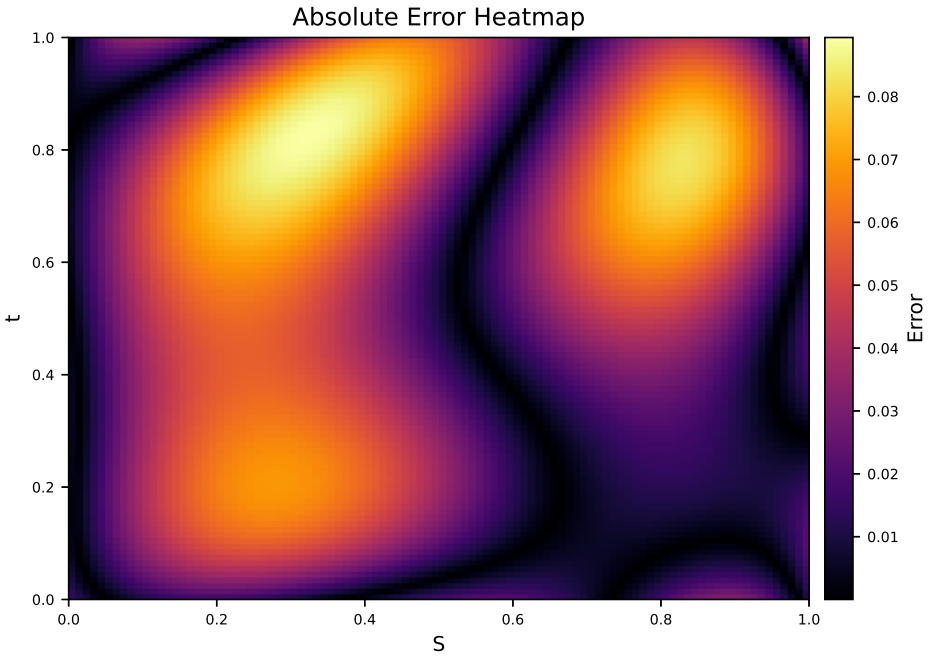

Figure 6: Heatmap of absolute error for Example 2

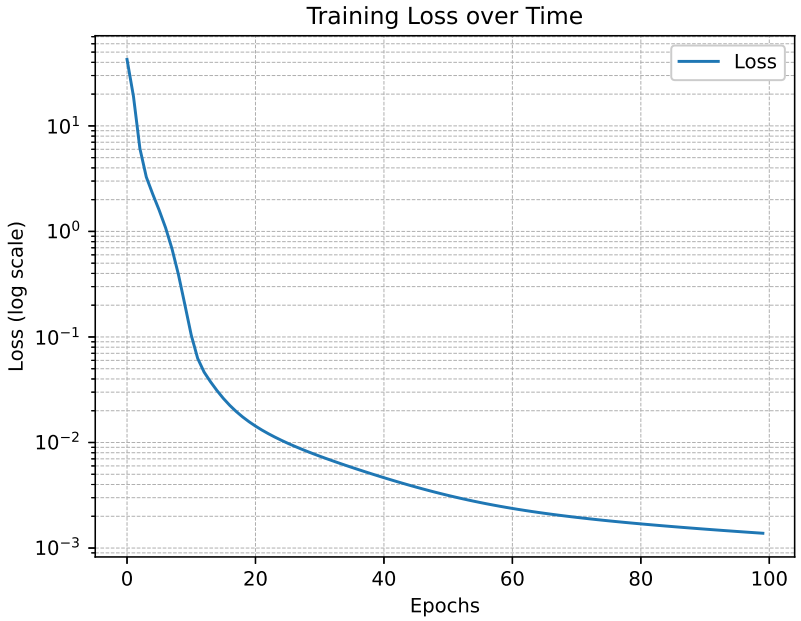

Figure 7: Logarithmic loss curve, indicating stable learning using Adam optimizer in Example 2

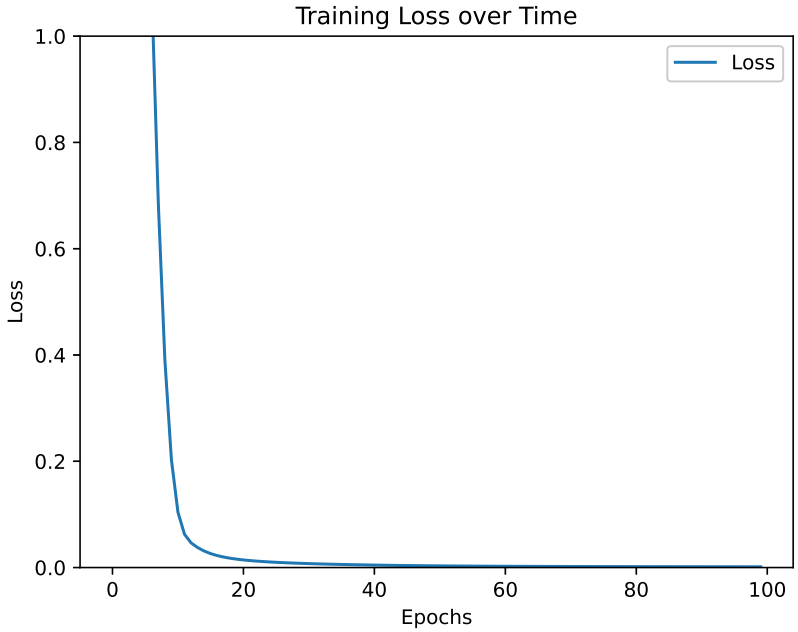

Figure 8: Training loss decay during model training for Example 2

Table 3: Comparative performance analysis between standard PINN and PINN-DGM

Metric	PINN-DGM (Ours)	Standard PINN	Improvement	Ratio
Mean Absolute Error	0.044922	0.149391	0.104469	3.33×
Training Time (seconds)	26.37	73.05	46.68	2.77×
Training Efficiency (Error/Time)	0.001704	0.002045	-	1.20×

73.05 seconds for standard PINN, representing a 2.77-fold speedup. This efficiency gain is particularly notable given that DGM architectures are typically more complex than standard FNNs.

- Overall Performance: When considering the trade-off between accuracy and computational cost, PINN-DGM demonstrates better training efficiency (Error/Time ratio of 0.001704 vs. 0.002045), confirming its practical advantage for real-world applications.

The visual evidence supporting this quantitative analysis is presented in Figure 9, which clearly show that PINN-DGM provides more accurate solutions with significantly reduced error distribution across the computational domain.

4.4 Sensitivity Analysis

To investigate the impact of the network's architectural parameters on model performance, a sensitivity analysis was conducted on Example 2. The influence of three key parameters was examined: (1) the number of hidden layers (network depth), (2) the number of neurons per layer (network width), and (3) the learning rate. The final relative L^2 error at $t = 1$ served as the evaluation metric.

- Number of Layers: Increasing the network depth from 1 to 2 layers resulted in reduced error; however, further increasing to 3 layers did not yield significant improvement and increased training time. Consequently, an optimal depth of 2 layers was selected.
- Number of Neurons: Varying the number of neurons per layer between 20 and 50 did not significantly impact the final accuracy, so we have used 25 neurons for each layer.

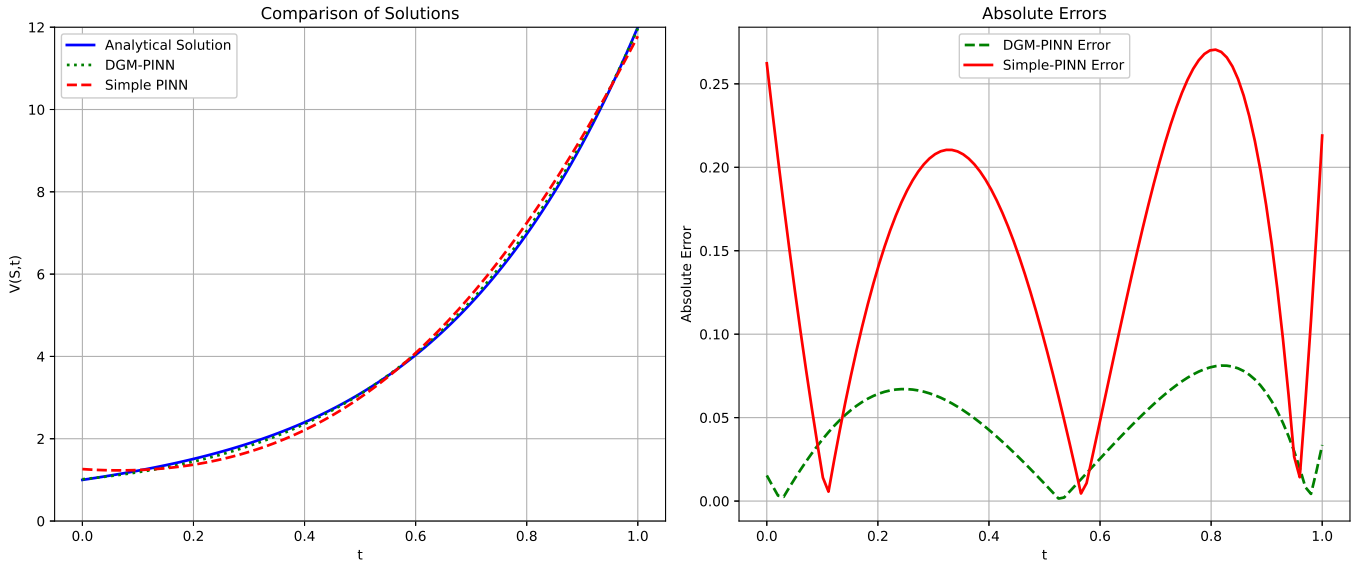

Figure 9: Comparison between DGM-PINN and simple PINN methods for Example 2

- Learning Rate: A very small learning rate (below 10^{-4}) led to exceedingly slow convergence. Conversely, a very large learning rate (above 10^{-2}) caused instability during training. The optimal rate was found to be within the range $[0.001, 0.002]$.

To comprehensively evaluate the robustness of our proposed method, we conducted extensive experiments investigating the impact of two critical parameters: the fractional order α and the learning rate.

4.4.1 Sensitivity to Fractional Order α

We tested our PINN-DGM model on Example 2 with various fractional orders $\alpha \in \{0.3, 0.5, 0.7, 0.9\}$, while keeping all other parameters constant. The results, summarized in Table 4, demonstrate the model's robustness across different fractional orders.

Table 4: results for different α (LR = 0.002)

α	Mean Error	Training Time	Final Loss
0.3	0.052723	19.29	0.00689
0.5	0.061213	13.17	0.00478
0.7	0.030723	13.03	0.00253
0.9	0.090625	13.13	0.004730

Key observations from this analysis include:

- The model maintains high accuracy (L^2 error $< 2.2 \times 10^{-2}$) across all tested fractional orders.
- Intermediate fractional orders ($\alpha = 0.5 - 0.7$) yield slightly better performance, potentially due to balanced memory effects.
- Higher fractional orders ($\alpha = 0.9$) require more training iterations and time, reflecting increased complexity in capturing near-integer order behavior.
- The method demonstrates consistent stability regardless of the fractional order, confirming its general applicability to various TFBSM configurations.

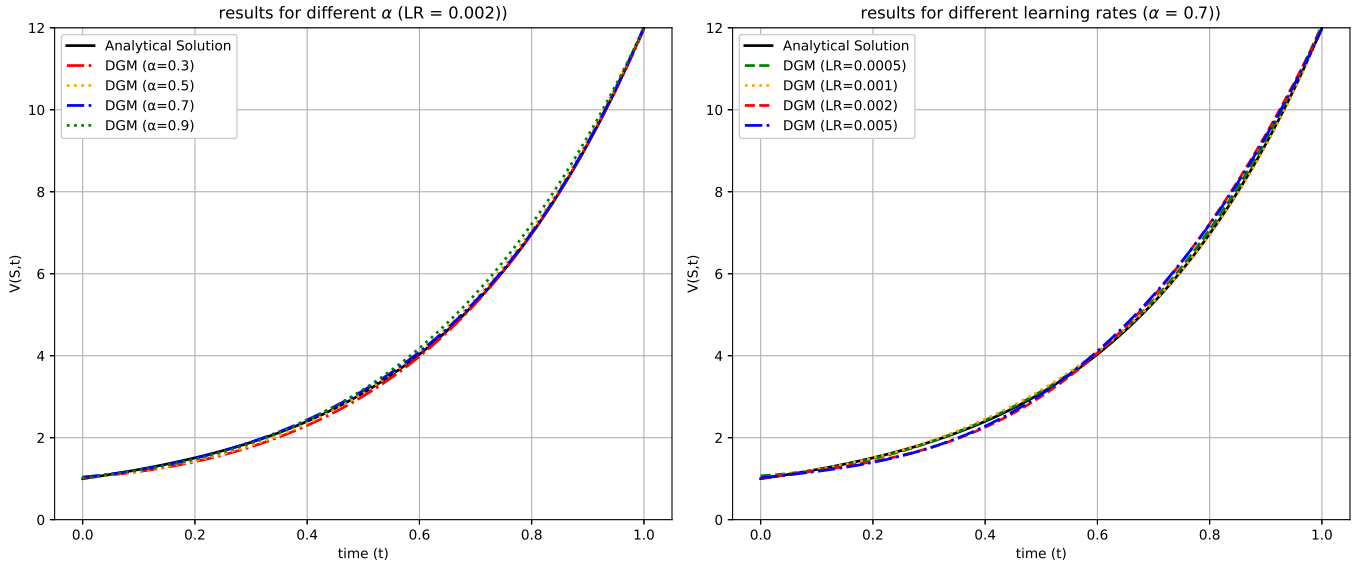

Figure 10: different learning rates with $\alpha = 0.7$(right), different α with LR $= 0.002$(left)

4.4.2 Sensitivity to Fractional Order learning rate

The learning rate is a crucial hyperparameter in deep learning models. We investigated its impact on training dynamics and final accuracy using Example 2 with $\alpha = 0.7$.

Table 5: results for different learning rates ($\alpha = 0.7$)

Learning Rate	Mean Error	Training Time	Final Loss
0.0005	0.054588	13.02	0.005162
0.001	0.017092	13.10	0.003311
0.002	0.035619	13.24	0.001879
0.005	0.086008	13.35	0.001950

The sensitivity analysis with respect to the fractional order α reveals important insights into the model's behavior. As shown in Figure 10 left, the solution profiles exhibit smooth transitions across different α values, indicating consistent numerical behavior and the model maintains high accuracy across the tested range of fractional orders, with error levels remaining within acceptable bounds. The observed patterns confirm that our PINN-DGM framework effectively captures the memory effects inherent in fractional derivatives, regardless of the specific fractional order.

The learning rate sensitivity analysis, presented in Figures 10 right, provides crucial guidance for practical implementation. Figure 10 shows the training dynamics across different learning rates, revealing the optimal range for stable convergence and quantifies the relationship between learning rate and final solution quality, highlighting the existence of a sweet spot that balances training efficiency with numerical accuracy. These findings underscore the importance of proper hyperparameter tuning while demonstrating the robustness of our approach across a reasonable range of learning rates.

The learning rate analysis reveals:

- Optimal Range: Learning rates between 1×10^{-3} and 5×10^{-3} provide the best balance between convergence speed and final accuracy.
- High Learning Rates: Values above 5×10^{-3} lead to instability and divergent behavior, preventing proper convergence.
- Low Learning Rates: Values below 5×10^{-4} result in prohibitively slow convergence without significant accuracy improvements.

- Robustness: The PINN-DGM framework shows good stability across a wide range of learning rates (5×10^{-4} to 5×10^{-3}), with graceful degradation rather than catastrophic failure.

These comprehensive sensitivity analyses confirm that our proposed method is robust to parameter variations and can be reliably applied to various TFBSM configurations with appropriate hyperparameter tuning.

This analysis confirms that the proposed model is stable across a wide range of network configurations and that the parameter set reported in Section 4.2 represents an optimal choice.

5 Discussion and Conclusion

In this paper, a physics-informed neural network (PINN)-based model was proposed for solving the time-fractional Black–Scholes equation. The distinctive feature of the proposed model lies in incorporating the DGM network architecture within the PINN framework, aimed at enhancing the temporal and spatial learning capability and facilitating training in complex functional spaces.

The designed model successfully solved the fractional Black–Scholes equation with satisfactory accuracy. By employing numerical integration based on the Gauss–Legendre quadrature to compute the Caputo fractional derivative, a more precise implementation of the fractional part of the equation was achieved. Numerical results demonstrated that the model could accurately reproduce option pricing behavior across different time intervals, with the relative L_2 error remaining low in all tests.

Furthermore, it was observed that the selected network architecture (utilizing the DGM structure) effectively captured temporal dependencies, particularly in fractional derivative equations with inherent memory effects. This capability contributed to reducing numerical error fluctuations and improving training stability.

It is noteworthy that although the DGM architecture was employed, the primary solution framework remained PINN-based; in other words, DGM served solely as a deep neural network architecture for approximating the unknown function, rather than as a standalone numerical method.

On the other hand, the numerical implementation also revealed that the use of accurate quadrature methods such as Gauss–Legendre plays a critical role in the more precise evaluation of the fractional integral terms, which possess singular kernels.

The comparative analysis reveals that our proposed PINN-DGM framework achieves a remarkable 3.33-fold improvement in accuracy while simultaneously reducing training time by 2.77 times compared to standard PINN approaches. This dual advantage of enhanced accuracy and computational efficiency makes our method particularly suitable for practical financial applications where both precision and speed are critical.

The superior performance can be attributed to the DGM architecture's inherent ability to capture long-term temporal dependencies, which is essential for modeling the memory effects in fractional derivatives. Furthermore, the specialized network structure enables more efficient gradient propagation during training, leading to faster convergence without compromising accuracy.

Overall, the findings of this study indicate that the combination of the PINN framework with neural networks equipped with temporal memory, such as DGM, offers a powerful approach for solving fractional partial differential equations. This method is especially promising for modeling phenomena with memory effects, including financial markets, biological dynamics, and fractional control systems, demonstrating high potential for generalization.

References

[1] Arwa Aldweesh, Abdelouahid Derhab, and Ahmed Z Emam, *Deep learning approaches for anomaly-based intrusion detection systems: A survey, taxonomy, and open issues*, Knowledge-Based Systems **189** (2020), 105124.

[2] Md Zahangir Alom, Tarek M Taha, Chris Yakopcic, Stefan Westberg, Paheding Sidike, Mst Shamima Nasrin, Mahmudul Hasan, Brian C Van Essen, Abdul AS Awwal, and Vijayan K Asari, *A state-of-the-art survey on deep learning theory and architectures*, electronics **8** (2019), no. 3, 292.

[3] Jens Berg and Kaj Nyström, *A unified deep artificial neural network approach to partial differential equations in complex geometries*, Neurocomputing **317** (2018), 28–41.

[4] Svetlana I Boyarchenko et al., *Non-gaussian merton-black-scholes theory*, vol. 9, World scientific, 2002.

[5] Peter Carr, Hélyette Geman, Dilip B Madan, and Marc Yor, *Stochastic volatility for lévy processes*, Mathematical finance **13** (2003), no. 3, 345–382.

[6] Alvaro Cartea and Diego del Castillo-Negrete, *Fractional diffusion models of option prices in markets with jumps*, Physica A: Statistical Mechanics and its Applications **374** (2007), no. 2, 749–763.

[7] Wenting Chen, Xiang Xu, and Song-ping Zhu, *Analytically pricing european-style options under the modified black-scholes equation with a spatial-fractional derivative*, Quarterly of Applied Mathematics **72** (2014), no. 3, 597–611.

[8] Wenting Chen, Xiang Xu, and Song-Ping Zhu, *Analytically pricing double barrier options based on a time-fractional black–scholes equation*, Computers & Mathematics with Applications **69** (2015), no. 12, 1407–1419.

[9] Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli, *Scientific machine learning through physics–informed neural networks: Where we are and what's next*, Journal of Scientific Computing **92** (2022), no. 3, 88.

[10] Kai Diethelm and Neville J Ford, *Analysis of fractional differential equations*, Journal of Mathematical Analysis and Applications **265** (2002), no. 2, 229–248.

[11] MWM Gamini Dissanayake and Nhan Phan-Thien, *Neural-network-based approximations for solving partial differential equations*, communications in Numerical Methods in Engineering **10** (1994), no. 3, 195–201.

[12] G Hariharan, S Padma, and P Pirabaharan, *An efficient wavelet based approximation method to time fractional black–scholes european option pricing problem arising in financial market*, Applied Mathematical Sciences **7** (2013), no. 69, 3445–3456.

[13] Steven L Heston, *A closed-form solution for options with stochastic volatility with applications to bond and currency options*, The review of financial studies **6** (1993), no. 2, 327–343.

[14] Sepp Hochreiter and Jürgen Schmidhuber, *Long short-term memory*, Neural computation **9** (1997), no. 8, 1735–1780.

[15] Kurt Hornik, Maxwell Stinchcombe, and Halbert White, *Universal approximation of an unknown mapping and its derivatives using multilayer feedforward networks*, Neural networks **3** (1990), no. 5, 551–560.

[16] John Hull and Alan White, *The pricing of options on assets with stochastic volatilities*, The journal of finance **42** (1987), no. 2, 281–300.

[17] Guy Jumarie, *Stock exchange fractional dynamics defined as fractional exponential growth driven by (usual) gaussian white noise. application to fractional black–scholes equations*, Insurance: Mathematics and Economics **42** (2008), no. 1, 271–287.

[18] Guy Jumarie, *Derivation and solutions of some fractional black–scholes equations in coarse-grained space and time. application to merton's optimal portfolio*, Computers & mathematics with applications **59** (2010), no. 3, 1142–1164.

[19] Anatoly A. Kilbas, H.M. Srivastava, and J.J. Trujillo, *Theory and applications of fractional differential equations*, North-Holland Mathematics Studies, Elsevier Science & Tech, 2006.

[20] Ismo Koponen, *Analytic approach to the problem of convergence of truncated lévy flights towards the gaussian stochastic process*, Physical Review E **52** (1995), no. 1, 1197.

[21] Steven G Kou, *A jump-diffusion model for option pricing*, Management science **48** (2002), no. 8, 1086–1101.

[22] Sunil Kumar, Ahmet Yildirim, Yasir Khan, Hossein Jafari, Khosro Sayevand, and Leilei Wei, *Analytical solution of fractional black-scholes european option pricing equation by using laplace transform*, Journal of fractional calculus and Applications **2** (2012), no. 8, 1–9.

[23] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis, *Artificial neural networks for solving ordinary and partial differential equations*, IEEE transactions on neural networks **9** (1998), no. 5, 987–1000.

[24] Isaac E Lagaris, Aristidis C Likas, and Dimitris G Papageorgiou, *Neural-network methods for boundary value problems with irregular boundaries*, IEEE transactions on Neural Networks **11** (2000), no. 5, 1041–1049.

[25] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton, *Deep learning*, nature **521** (2015), no. 7553, 436–444.

[26] Jin-Rong Liang, Jun Wang, Wen-Jun Zhang, Wei-Yuan Qiu, and Fu-Yao Ren, *The solution to a bi-fractional black-scholes-merton differential equation*, International Journal of Pure and Applied Mathematics **58** (2010), no. 1, 99–112.

[27] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis, *Deepxde: A deep learning library for solving differential equations*, SIAM review **63** (2021), no. 1, 208–228.

[28] Yulong Lu and Jianfeng Lu, *A universal approximation theorem of deep neural networks for expressing probability distributions*, Advances in neural information processing systems **33** (2020), 3094–3105.

[29] Benoit B Mandelbrot and Benoit B Mandelbrot, *The variation of certain speculative prices*, Springer, 1997.

[30] Robert C Merton, *Option pricing when underlying stock returns are discontinuous*, Journal of financial economics **3** (1976), no. 1-2, 125–144.

[31] P Phaochoo, A Luadsong, and N Aschariyaphotha, *The meshless local petrov–galerkin based on moving kriging interpolation for solving fractional black–scholes model*, Journal of King Saud University-Science **28** (2016), no. 1, 111–117.

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

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

[34] B Ph van Milligen, V Tribaldos, and JA Jiménez, *Neural network differential equation and plasma equilibrium solver*, Physical review letters **75** (1995), no. 20, 3594.

[35] Felipe AC Viana, Renato G Nascimento, Arinan Dourado, and Yigit A Yucesan, *Estimating model inadequacy in ordinary differential equations with physics-informed neural networks*, Computers & Structures **245** (2021), 106458.

[36] Walter Wyss, *The fractional black-scholes equation*, An International Jornal for theory and Application **3** (2000), 51–61.

[37] Hongmei Zhang, Fawang Liu, Ian Turner, and Qianqian Yang, *Numerical solution of the time fractional black–scholes model governing european options*, Computers & Mathematics with Applications **71** (2016), no. 9, 1772–1783.

[38] Hongmei Zhang, Mengchen Zhang, Fawang Liu, and Ming Shen, *Review of the fractional black-scholes equations and their solution techniques*, Fractal and Fractional **8** (2024), no. 2, 101.

[39] Xiaoning Zhang, Jianhui Yang, and Yuxin Zhao, *Numerical solution of time fractional black–scholes model based on legendre wavelet neural network with extreme learning machine*, Fractal and Fractional **6** (2022), no. 7, 401.

[40] Qiming Zhu, Zeliang Liu, and Jinhui Yan, *Machine learning for metal additive manufacturing: predicting temperature and melt pool fluid dynamics using physics-informed neural networks*, Computational Mechanics **67** (2021), 619–635.