Mathematical Modeling and Algorithm Application
ISSN: 3006-0842 | Vol. 4, No. 1, 2025

Predicting Implied Volatility Via Improved Pinns–Informer to the BS Equation

Jixiao Liu, Yecheng Ma

School of Mathematics, Shanghai University of Finance and Economics, Shanghai, 200433, China

Abstract: This paper explores various methods for analyzing and predicting data through traditional statistical models, machine learning, and deep learning techniques. A detailed overview of SARIMA, a widely-used time series model, highlights its compo-nents-Autoregressive (AR), Moving Average (MA), Differencing, and Seasonal Differencing-that are crucial for handling time-based data dependencies. The paper also discusses machine learning approaches, such as Decision Trees and Random Forests, with an enhancement using Particle Swarm Optimization (PSO) for featureselection to improve performance. In the realm of deep learning, Convolutional Neural Networks (CNNs) are explored for their ability to handle complex data such as images, focusing on convolutional, pooling, and fully connected layers. Further, the paper presents a Physics-Informed Neural Network (PINN) model, which incorporates a dynamic loss function and attention mechanisms for improving model generalization. Finally, the hybrid model, combining deep learning with meta-learning, optimizes loss functions for faster training convergence and increased adaptability to market data. These approaches collectively aim to enhance prediction accuracy and computational efficiency across various domains.

Keywords: Implied Volatility; SARIMA; Machine Learning

1. Introduction

The globalization and informatization of financial markets have led to an increasing variety and quantity of financial derivatives, creating a growing need for precise pricing and efficient risk management solutions.[1–3] Traditional pricing methods like the Black-Scholes model face challenges such as high computational costs and limited accuracy with high-dimensional, nonlinear problems.[4] Researchers are exploring deep learning to enhance pricing efficiency and accuracy.[5,6]

Increased uncertainty and volatility in global markets, driven by crises and geopolitical risks, have highlighted the importance of deep learning in pricing and risk management of derivatives.[7,8] Deep learning models can learn patterns from extensive historical data, providing robust pricing strategies amid market uncertainties.[9,10]

Deep learning effectively addresses high-dimensional and nonlinear challenges.[3,5] While Monte Carlo methods are accurate, they are computationally demanding. In contrast, deep learning models utilize parallel computing to reduce computation time significantly.[6] Physics-Informed Neural Networks (PINNs) improve pricing accuracy by incorporating physical constraints, avoiding reliance on sample solutions.[8,9]

Generative models like GANs and VAEs further enhance model performance by generating realistic synthetic market data, aiding in training, particularly in complex scenarios.[10,11]

Real-world pricing models must consider specific market assumptions.[1,7] For instance, implied volatility predictions depend on conditions such as interest rates and asset price volatility.[8] Integrating deep learning with these assumptions can yield more stable predictions.[9]

Assuming geometric Brownian motion allows PINNs to solve the Black-Scholes equation in high-dimensional spaces, enhancing option pricing accuracy.4 Uncertainty Quantification (UQ) techniques can also assess model reliability.[3,6]

Deep learning has made significant strides in financial derivative pricing.[2,7,9] For example, DNNs optimize option pricing by capturing volatility patterns from historical data, while LSTM networks effectively predict future trends and volatility.[4,8] Major financial institutions are incorporating deep learning into trading platforms for real-time pricing and risk prediction.[6,9]

The Black-Scholes model, established in 1973, operates under strict assumptions that often misalign with actual market conditions, causing biases.[2] Traditional statistical models like ARIMA also struggle with nonlinear relationships, limiting their predictive capacity.[3,8]

Existing hybrid neural networks often rely on backpropagation, which suffers from slow learning, especially with time-series data.[10,11] Even LSTMs can face low training efficiency due to fixed parameters.[4]

This research conducts a comparative analysis of implied volatility using machine learning and deep learning methods.[8,9] We utilize high-frequency and low-frequency SSE 50ETF option data to ensure robust analysis.[10] A meta-learning approach reduces the need for extensive data, enhancing model efficiency. The Informer network architecture integrated into the PINN framework further improves performance.[4]

To optimize training convergence, we introduce an empirical loss function for adaptive adjustments during training, capturing dynamic changes in implied volatility more accurately.[3,5,6]

2. Data Source

The data in this article is based on the underlying asset of the SSE 50ETF options—low-frequency data (daily closing prices) and high-frequency data (5-minute trading data) of the SSE 50ETF. The SSE 50ETF options were introduced by the Shanghai Stock Exchange in 2015, and are European-style options with Huaxia SSE 50ETF (code 510050) as the underlying asset. Specific information is listed in Table 1:

Table 1. Detailed Information of SSE 50ETF Options

Contract Identifier	SSE 50 Exchange-Traded Fund (ETF)("50ETF") Open-ended Index Securities Investment Fund
Contract Type	Call Options and Put Options
Contract Unit	10,000 shares
Contract Expiration Month	Current month, next month, and the following two quarters
Strike Price	9 prices (1 at-the-money, 4 out-of-the-money, 4 in-the-money)
Exercise Style	European Style (Exercised at expiration)
Settlement Method	Physical delivery (unless otherwise specified by business rules)
Expiration Date	Fourth Friday of the expiration month (postponed if it falls on a public holiday)
Trading Hours	9:15-9:25, 9:30-11:30 (9:15-9:25 for call auction)) 13:00-15:00 (14:57-15:00 for call auction)
Order Types	Regular limit order, market-to-limit order, market fill or kill order,market withdrawal order, full amount immediate limit order, full amount immediate market order, and other types specified by business rules
Minimum Quote Unit	0.0001yuan
Application Unit	1 contract or its integer multiples

Choose SSE 50ETF options with expiration dates in March 2023, July 2023, and November 2023. Each month, select call and put options with strike prices of 2.4 yuan, 2.6 yuan, and 2.8 yuan, totaling 18 options. Detailed information is shown in Table 2.

Table 2. Select Detailed Information for 50ETF Options

Code	Expiration Date
10004877.SH	2023-03-22
10004881.SH	2023-03-22
10004885.SH	2023-03-22
10005399.SH	2023-07-26
10005403.SH	2023-07-26
10005407.SH	2023-07-26
10006025.SH	2023-11-22
10006029.SH	2023-11-22
10006033.SH	2023-11-22
10004886.SH	2023-03-22
10004890.SH	2023-03-22
10004894.SH	2023-03-22
10005408.SH	2023-07-26
10005412.SH	2023-07-26
10005416.SH	2023-07-26
10006034.SH	2023-11-22
10006038.SH	2023-11-22
10006042.SH	2023-11-22

3. Hybrid Model

3.1. Constructing the Loss Function for the Meta-Learning Enhanced PINN Model

The traditional PINN model is formulated as follows:

$$\frac{\partial u}{\partial t} + M[u;\lambda] = 0, x \in \Omega, \quad t \in [0,T] \tag{1}$$

where u is the quantity to be solved. In our case, u(t ,x) = C(τ , S), where τ is the time to maturity and S is the stock price. According to the Black-Scholes model, we have:

$$M[u;\lambda] = M[C;r,\sigma] = -rC + rS\frac{\partial C}{\partial S} + \frac{1}{2}\sigma^2 S^2 \frac{\partial^2 C}{\partial S^2} \tag{2}$$

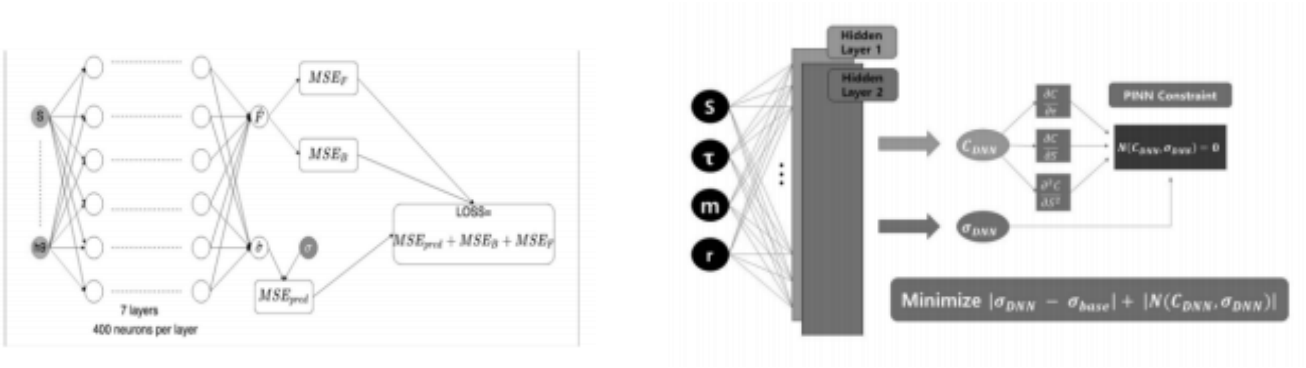
However, this setting considers volatility as a constant. Therefore, to incorporate more realistic time-varying volatility in our PINN model, we also set σ as a target function to be solved. Moreover, to leverage neural networks for finding data-driven solutions, we construct the target solutions C and σ as functions of all observable variables (V, T, m, r), thereby expanding the spatial variables to include mandr. Let x = (C, m, r) ∈ R3, and define

$$N(C,\sigma) := \frac{\partial C}{\partial \tau} + M[C,\sigma] \tag{3}$$

The loss function is then given by:

$$L_{PINN} = ||\sigma_{DNN} - \sigma_{base}||_1 + ||N(C_{DNN}, \sigma_{DNN})||_1 \tag{4}$$

where σbase is the baseline volatility used for interpolation. The deep neural network structures of CDNN and σDNN consist of single hidden layer neural networks with 1000 nodes and Softplus activation functions. Figure 1 depicts two sets of neural networks:



(a) Neural Network 1 **(b)** Neural Network 2

Figure 1. Two sets of neural network models

There are two sets of inputs: the first set includes option information, such as V, T, m, r, and the second set includes past information used to predict future volatility. For example, in the figure, we represent the last input as h9, indicating that there are 10 past information inputs, ranging from 1 day to 10 days delay. Each intermediate layer is transformed by a ReLU function. After multiple layers of processing, our model outputs a two-dimensional vector.

3.2. Constructing the Dual Attention Mechanism

To address a new problem (not only the implied volatility changes but also the high-dimensional PDE problem), we adopt PINN combined with the attention mechanism and incorporate meta-learning. To more accurately identify and utilize the interactions among input features while capturing the feature relationships inherent in longtime series, we integrate encoding and decoding layers into the outer framework of PINN. This combination helps the model more flexibly capture crucial input features when handling high-dimensional PDEs. Specifically, the self-attention mechanism employed is capable of deeply exploring internal interactions among input features, rather than being limited to their relationship with the overall task. Additionally, the standard PINN primarily relies on a predefined loss function for training. However, in some computational regions, the approximation difficulty may exceed that in other regions. Thus, we propose a dynamic loss function that allows the network to adaptively allocate weights to various regions, with particular focus on those hard-to-approximate parts. Unlike traditional fixed weights, these weights are dynamic and are adjusted based on real-time feedback during the training process, enabling the network to self-learn and determine which regions require more attention.

3.3. Attention Mechanism for Inner Layer - Loss Function (Targeting CDNN)

• **A.** Task Definition: In the context of meta-learning, we need multiple tasks. In the context of PDEs, each task may correspond to different boundary conditions, initial conditions, or other specific PDE parameters.

• **B.** Task Similarity Measure:

$$\mathbf{C}_{\mu i,\mu j} = \frac{\mathbf{w}_{\mu i}\mathbf{w}_{\mu j}}{\|\mathbf{w}_{\mu i}\|^2 \times \|\mathbf{w}_{\mu j}\|^2}$$

and Euclidean Distance:

$$E_{\mu i,\mu j} = \| \mathrm{w}_{\mu i} - \mathrm{w}_{\mu j} \|_2$$

to encourage the model to find similar solutions for different tasks.

• **C.** In previous work, the PINN model consisted of two deep neural networks: one for predicting option prices (denoted as CDNN(V, T, m, r)), and another for predicting volatility (denoted as σDNN(V, T, m, r)). The inputs are the underlying asset price V, time to maturity T, value m, and risk-free interest rate r. Here, we improve by using only a single layer network, retaining the layer for predicting volatility.

• **D.** Define Residual Loss, Boundary Loss, and Initial Condition Loss:

$$L_r(w,\lambda_r) = \frac{1}{2}\sum_{i=1}^{N_r} m(\lambda_{ir}) \left|\nabla V,T\left[\mathrm{CDNN}(V_{ir},T_{ir},m,r;w)\right] - f(V_{ir},T_{ir})\right|^2 \tag{5}$$

$$L_b(w,\lambda_b) = \frac{1}{2}\sum_{i=1}^{N_b} m(\lambda_{ib}) \left|\mathrm{BV},T\left[\mathrm{CDNN}(V_{ir},T_{ir},m,r;w)\right] - g(V_{ib},T_{ib})\right|^2 \tag{6}$$

$$L_0(w,\lambda_0) = \frac{1}{2}\sum_{i=1}^{N_0} m(\lambda_{i0}) \left|\mathrm{CDNN}(V_{i0},T_0,m,r;w) - h(V_{i0})\right|^2 \tag{7}$$

where f is the right-hand side of the PDE, representing the internal driving force or source of the system. g is the boundary condition, describing the system behavior at the boundary. His the initial condition, giving the state of the system at the initial moment.

Based on these, we can construct the loss function for the meta-learning enhanced PINN model as follows:

$$L_{\text{meta}}(w,\lambda_r,\lambda_b,\lambda_0) = L_{\text{PINN}}(w) + \sum_{i=1}^{N}[L_r(w,\lambda_{ri}) + L_b(w,\lambda_{bi}) + L_0(w,\lambda_{0i})] + \sum_{i=1}^{N}\sum_{j=i+1}^{N}[1 - C_{\tau_i,\tau_j} + E_{\tau_i,\tau_j}] \tag{8}$$

Here, LPINN(w) is the original loss of the PINN model. Lr, Lb, and L0 are the residual, boundary, and initial condition losses for the i-th task, respectively. Cτi,τj and Eτi,τj are the cosine similarity and Euclidean distance between the weights of tasks τi and τj, respectively. λ is an adaptive parameter used to balance task loss and weight similarity among tasks.

- **E.** Final Problem Formulation: Once the loss function is constructed, another crucial aspect is to update and initialize the weights. Given that high-dimensional PDEs typically require extensive data and computational resources for training and convergence, meta-learning can help the model quickly adapt to new PDE tasks with few iterations, avoiding the need for training from scratch. This greatly enhances the generalization and adaptability of the model. We adopt the Model-Agnostic Meta-Learning (MAML) approach for this purpose. The specific construction is as follows:

– **Initialize Weights:** For each task τi, we use the updated weights w′τi to compute the loss on the validation set and update the initialization weights.

a. Compute Validation Loss:

$$L_{\tau_i}^{val}(w'_{\tau_i}) = \sum_{(x,y)\in D_{\tau_i}^{val}} \mathcal{L}(u(x;w'_{\tau_i}),y) \tag{9}$$

b. Update Initialization Weights: We compute the gradient of the average validation loss across all tasks with respect to the initial weights and update the initial weights:

$$w = w - \beta \nabla_w \left(\frac{1}{N}\sum_{i=1}^{N} L_{\tau_i}^{val}(w'_{\tau_i}) \right) \tag{10}$$

where β is the learning rate for the outer loop. This process ensures that the network weights are initialized to a value that can easily adapt to different boundary conditions, initial conditions, etc., allowing the network to quickly learn various tasks.

–Update Weights: Assuming we have N tasks, each denoted as τi, i = 1, 2, . . . , N, and each task has its own training dataset Dτitrain and validation dataset Dτival . For each task τi, we perform the following steps:

a. Compute Task Loss: For each training sample of task τi, we compute the loss:

$$L_{\tau_i}^{train}(w) = \sum_{(x,y)\in D_{\tau_i}^{train}} (u(x;w),y) \tag{11}$$

where u(x; w) is the network output parameterized by weights w for input x, and L isthe pointwise loss function.

b. Gradient Descent: For each task, we update the weights via gradient descent:

$$w'_{\tau_i} = w - \alpha \nabla_w L_{\tau_i}^{train}(w) \tag{12}$$

where α is the learning rate.

3.4. Outer Layer Attention Mechanism

In deep learning, the self-attention mechanism, particularly the version in the Transformer architecture, has become the mainstream method for various tasks such as machine translation, text generation, and time series prediction. To better capture temporal features, this paper innovatively incorporates the Informer network architecture into the outer layer of the original PINN network, as illustrated in the Figure 2.

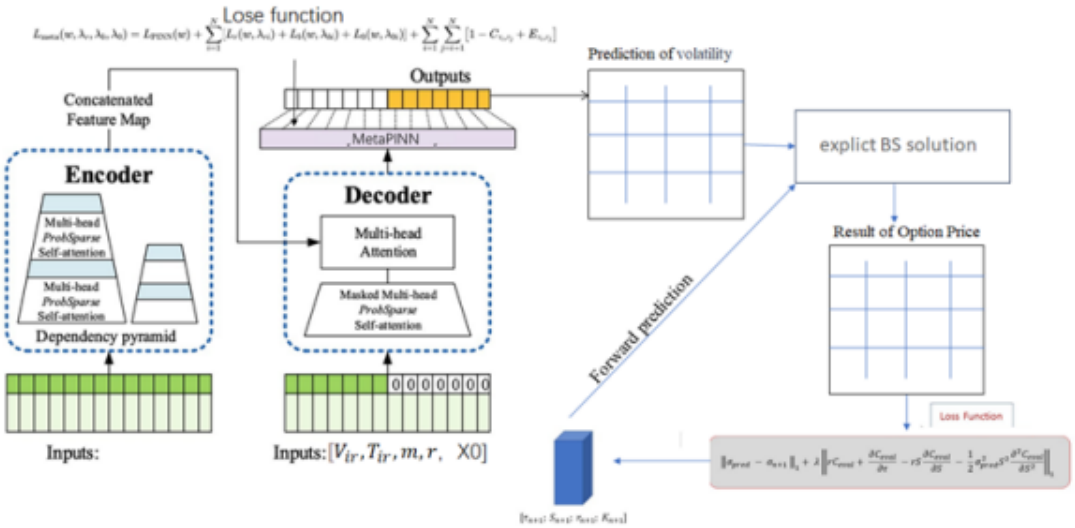


Figure 2. Main Model Framework

In our approach, we use the implied volatility predictions obtained from the improved Informer model at each maturity and strike price to determine the call option prices. This method is based on the fact that professionals reverse-engineer the Black-Scholes formula to determine implied volatility based on option pricing details. Thus, the standard Black-Scholes model can be employed to obtain Ceval:

$$C_{\text{eval}(i,j)} = S_{n+1(i,j)} \Phi\left(d_{1(i,j)}\right) - X_{n+1(i,j)} e^{-r_{n+1(i,j)} \tau_{n+1(i,j)}} \Phi\left(d_{2(i,j)}\right) \tag{13}$$

where:

$$d_{1_{(i,j)}} = \frac{\ln\left(S_{n+1_{(i,j)}} / X_{n+1_{(i,j)}}\right) + \left(r_{n+1_{(i,j)}} + \frac{1}{2}\sigma^2_{\text{pred}_{(i,j)}}\right)\tau_{n+1_{(i,j)}}}{\sigma_{\text{pred}_{(i,j)}}\sqrt{\tau_{n+1_{(i,j)}}}} \tag{14}$$

$$d_{2_{(i,j)}} = d_{1_{(i,j)}} - \sigma_{\text{pred}_{(i,j)}}\sqrt{\tau_{n+1_{(i,j)}}} \tag{15}$$

The results are fed into the PINN model, and the weights are continuously updated to minimize the loss function. Since the PINN model attempts to approximate a general volatility function from daily data points, its training is more in-depth compared to other models. The batch size is chosen to be larger because the sampled inputs are not tensors representing the daily volatility surface, but vectors corresponding to each point on the surface. Thus, a larger batch size is chosen.

The following is a theoretical analysis of the loss function improvement:

3.5. Theoretical Analysis of Improved Loss Functions: Part One

To further optimize the model and accelerate training convergence, we introduce an empirical loss function based on the distribution of implied volatility in options. This loss function, combined with the cross-entropy of a uniform distribution, is added to the original loss function as an additional term.

Specifically, the empirical loss function can be defined as follows:

$$L_{\text{exp}} = -\sum_{i=1}^{N} p_i \log q_i \tag{16}$$

where pi represents the true distribution of implied volatility, and qi is the predicted distribution of implied volatility by the model.

To incorporate the empirical loss function into the original loss function, we construct the total loss function Ltotal, which includes both the original loss function and the empirical loss function:

$$L_{\text{total}} = L_{\text{meta}}(w, \lambda_r, \lambda_b, \lambda_0) + \gamma L_{\text{exp}} \tag{17}$$

where γ is a weighting coefficient used to balance the importance of the original loss function and the empirical loss function.

By introducing the empirical loss function, the model not only achieves optimization in fitting the data-driven target functions C and σ, but also improves adaptability to market data and training convergence speed by considering the true distribution of implied volatility.

This enhancement enables the loss function to adaptively adjust during training, thereby capturing the dynamic changes in implied volatility more accurately, significantly improving the model's predictive performance and stability.

The improved loss function can be expressed as follows:

$$L_{\text{total}} = L_{\text{PINN}}(w) + \sum_{i=1}^{N}\left[L_r(w,\lambda_{ri}) + L_b(w,\lambda_{bi}) + L_0(w,\lambda_{0i})\right] + \sum_{i=1}^{N}\sum_{j=i+1}^{N}\left[1 - C_{\tau_i,\tau_j} + E_{\tau_i,\tau_j}\right] + \gamma\left[-\sum_{i=1}^{N} p_i \log q_i\right] \tag{18}$$

where LPINN(w) is the original PINN loss function, Lr, Lb, and L0 are the residual loss, boundary loss, and initial condition loss, respectively, Cτi,τj and Eτi,τj represent the cosine similarity and Euclidean distance between the weights of tasks τi and τj ,

respectively, and γ is the balancing coefficient. The term $-\Sigma^{N}_{i=1} p_i \log$ qi is the empirical loss term.

With this improvement, the model can learn the dynamic changes in implied volatility more efficiently, enhancing training convergence speed and prediction accuracy.

3.6. Theoretical Analysis of Improved Loss Functions: Part Two

- Mean Squared Error (MSE) Loss Function: We introduce the mean squared error loss term to measure the difference between the predicted implied volatility σpred and the true volatility σtrue. The MSE loss is defined as follows:

$$L_{\text{MSE}} = \frac{1}{N}\sum_{i=1}^{N}\left(\sigma_{\text{pred}_i} - \sigma_{\text{true}_i}\right)^2 \tag{19}$$

- Logarithmic Error Loss Function: To better handle the distribution characteristics of implied volatility, we introduce a logarithmic error loss term. The logarithmic error loss is defined as follows:

$$L_{\log} = \frac{1}{N}\sum_{i=1}^{N}\left(\log\left(\sigma_{\text{pred}_i}+\varepsilon\right)-\log(\sigma_{\text{true}_i}+\varepsilon)\right)^2 \tag{20}$$

where ε is a small constant to avoid logarithm of zero.

• K-Nearest Neighbors (KNN) Based Loss Function: We introduce a KNN loss term to measure the consistency of predictions for neighboring points in high-dimensional space. Specifically, for each predicted point, we calculate the K nearest neighbors in high-dimensional space and compute the difference in predicted values among these points. The KNN loss is defined as follows:

$$L_{\text{KNN}} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{K}\sum_{j\in\mathcal{N}_i}(\sigma_{\text{pred}_i}-\sigma_{\text{pred}_j})^2 \tag{21}$$

where Ni denotes the set of K nearest neighbors of the i-th point.

• Knowledge-aware Neural Network (KAN) Loss Function: We introduce a KAN loss term to incorporate external knowledge to improve model prediction accuracy. Specifically, we use a prior distribution σprior constructed from domain knowledge and measure the difference between the model's predictions and the prior distribution. The KAN loss is defined as follows:

$$L_{\text{KAN}} = \frac{1}{N}\sum_{i=1}^{N}(\sigma_{\text{pred}_i}-\sigma_{\text{prior}_i})^2 \tag{22}$$

By incorporating these improved loss functions into the total loss function, we construct a new total loss function Ltotal that includes the original loss function, empirical loss function, mean squared error loss, logarithmic error loss, KNN loss, and KAN loss:

$$L_{\text{total}} = L_{\text{meta}}(w,\lambda_r,\lambda_b,\lambda_0)+\gamma_1 L_{\exp}+\gamma_2 L_{\text{MSE}}+\gamma_3 L_{\log}+\gamma_4 L_{\text{KNN}}+\gamma_5 L_{\text{KAN}} \tag{23}$$

where γ1 , γ2 , γ3 , γ4, and γ5 are weighting coefficients used to balance the importance of each loss term. The improved loss function can be expressed as follows:

$$\begin{aligned}L_{\text{total}} =& L_{\text{PINN}}(w)+\sum_{i=1}^{N}\left[L_r(w,\lambda_{ri})+L_b(w,\lambda_{bi})+L_0(w,\lambda_{0i})\right]\\&+\sum_{i=1}^{N}\sum_{j=i+1}^{N}\left[1-C_{\tau_i,\tau_j}+E_{\tau_i,\tau_j}\right]+\gamma_1\left[-\sum_{i=1}^{N}p_i\log q_i\right]\\&+\gamma_2\left[\frac{1}{N}\sum_{i=1}^{N}(\sigma_{\text{pred}_i}-\sigma_{\text{true}_i})^2\right]+\gamma_3\left[\frac{1}{N}\sum_{i=1}^{N}\left(\log\left(\sigma_{\text{pred}_i}+\varepsilon\right)-\log(\sigma_{\text{true}_i}+\varepsilon)\right)^2\right]\\&+\gamma_4\left[\frac{1}{N}\sum_{i=1}^{N}\frac{1}{K}\sum_{j\in\mathcal{N}_i}(\sigma_{\text{pred}_i}-\sigma_{\text{pred}_j})^2\right]+\gamma_5\left[\frac{1}{N}\sum_{i=1}^{N}(\sigma_{\text{pred}_i}-\sigma_{\text{prior}_i})^2\right]\end{aligned}$$

By introducing these improvements, the model achieves better performance in fitting the data-driven target functions C and σ , while further improving adaptability to market data and training convergence speed by considering the true distribution of implied volatility. The incorporation of mean squared error loss, logarithmic error loss, KNN loss, and KAN loss allows the model to capture the dynamic changes in implied volatility more accurately, significantly enhancing predictive performance and stability.

3.7. Statistical Analysis of the Improved Loss Functions

In this section, we perform a statistical analysis of the proposed loss functions to understand their performance characteristics. The overall aim is to establish their ability to approximate the true distribution of implied volatility and improve model convergence.

• **A.** Empirical Loss Function The empirical loss function Lexp is defined as:

$$L_{\exp} = -\sum p_i \log q_i, \tag{24}$$

where pi and qi represent the true and predicted distributions of implied volatility, respectively. To analyze its statistical properties, we consider the expected value:

$$\mathbb{E}[L_{\exp}] = -\sum_{i=1}^{N}\mathbb{E}[p_i]\log\mathbb{E}[q_i]. \tag{25}$$

Using Jensen's inequality, since the logarithm is a concave function, we derive:

$$\mathbb{E}[L_{\exp}] \geq -\mathbb{E}[p]\log\mathbb{E}[q], \tag{26}$$

indicating that minimizing the empirical loss drives q towards p.

• **B.** Mean Squared Error (MSE) Loss Function The MSE loss function is defined as:

$$L_{\text{MSE}} = \frac{1}{N}\sum(\sigma_{\text{pred}_i}-\sigma_{\text{true}_i})^2. \tag{27}$$

The expected MSE can be expressed as:

$$\mathbb{E}[L_{\text{MSE}}] = \text{Bias}^2+\text{Var}+\sigma^2, \tag{28}$$

where Bias2 reflects the error due to model approximation, Var measures the variability of the predictions, and σ2 is the

irreducible error.

- **C.** Logarithmic Error Loss Function For the logarithmic error loss:

$$L_{\log} = \frac{1}{N}\sum_{i=1}^{N}\left(\log\left(\sigma_{\text{pred}_i}+\varepsilon\right)-\log(\sigma_{\text{true}_i}+\varepsilon)\right)^2, \tag{29}$$

we analyze its properties through Taylor expansion around the true value:

$$\log\left(\sigma_{\text{pred}_i}+\varepsilon\right) \approx \log(\sigma_{\text{true}_i}+\varepsilon)+\frac{1}{\sigma_{\text{true}_i}+\varepsilon}(\sigma_{\text{pred}_i}-\sigma_{\text{true}_i}). \tag{30}$$

Thus, the expected logarithmic loss captures both bias and variance effectively.

- **D.** K-Nearest Neighbors (KNN) Based Loss Function The KNN loss function is defined as:

$$L_{\text{KNN}} = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{K}\sum_{j\in\mathcal{N}_i}(\sigma_{\text{pred}_i}-\sigma_{\text{pred}_j})^2. \tag{31}$$

This loss term fosters local consistency in predictions. Its expected value can be analyzed as:

$$\mathbb{E}[L_{\text{KNN}}] = \frac{1}{N}\sum_{i=1}^{N}\text{Var}(\sigma_{\text{pred}}|\mathcal{N}_i), \tag{32}$$

indicating that minimizing this loss promotes smoothness in high-dimensional spaces.

- **E.** Knowledge-aware Neural Network (KAN) Loss Function The KAN loss function is defined as:

$$\text{LKAN} = \frac{1}{N}\sum_{i=1}^{N}(\sigma_{\text{pred}_i}-\sigma_{\text{prior}_i})^2. \tag{33}$$

This term integrates prior knowledge into the model, and its expected value can similarly be analyzed through the lens of bias-variance decomposition:

$$\text{E}[L\text{KAN}] = \text{Bias2} + \text{Var}. \tag{34}$$

The complete loss function, incorporating all components, can be expressed as:

$$\text{Ltotal} = \text{Lmeta} + \sum_{k=1}^{5}\gamma\text{kLk}, \tag{35}$$

where Lk represents each of the individual loss components. By balancing the coefficients γk, we can optimize the model's convergence and stability. Through this statistical analysis, we establish that the improved loss functions are designed to effectively capture the dynamics of implied volatility, thus enhancing both the convergence speed and predictive performance of the model. Each loss term contributes uniquely to minimizing bias and variance, ensuring robust performance across varying market conditions.

3.8. Theoretical Complexity Analysis

In this section, we analyze the computational complexity of the proposed meta-learning enhanced PINN model. Let N be the number of training samples, D be the number of dimensions of the input space, and L be the number of layers in the neural network.

The forward pass of the neural network requires O(N·L·D) operations, as each input passes through L layers, each requiring a computation of O(D). Thus, the total complexity for one forward pass is:

$$O\text{forward} = O(N\cdot L\cdot D). \tag{36}$$

For the backward pass, which involves calculating gradients, the complexity remains O(N ·L·D) due to the backpropagation algorithm. Therefore, the total complexity for one iteration of training is:

$$O\text{iteration} = O(N\cdot L\cdot D). \tag{37}$$

If we denote T as the total number of training iterations, the overall complexity for training the model can be expressed as:

$$O\text{total} = O(T\cdot N\cdot L\cdot D). \tag{38}$$

This analysis indicates that the computational complexity of our model scales linearly with the number of training samples and the depth of the network, making it suitable for practical applications with large datasets.

3.9 Convergence Analysis

In this section, we rigorously analyze the convergence properties of our meta-learning enhanced PINN model. Let L(w) denote the loss function associated with the model, where w represents the model parameters. The goal is to minimize L(w) over the parameter space W.

Assuming that the loss function is μ-strongly convex, we have:

$$L(w) \geq L(w^*) + \langle \nabla L(w^*), w-w^*\rangle + \frac{\mu}{2}\|w-w^*\|^2, \tag{39}$$

where w* is the optimal parameter that minimizes L(w), and μ > 0 is the strong convexity constant.

- **A.** Gradient Descent Updates

We apply gradient descent updates as follows:

$$\text{w(t +1)} = \text{w(t)} — \eta\nabla\text{L(w(t))} \tag{40}$$

where η is the learning rate. By substituting into the loss function, we obtain:

$$\text{L(w(t +1))} \leq \text{L(w(t))} - \eta\|\nabla\text{L(w(t))}\|2 + \frac{L_L}{2}\ \eta\ 2 \tag{41}$$

where LL is the Lipschitz constant of the gradient. By using the Cauchy-Schwarz inequality, we can further bound the loss as:

$$L(w^{(t+1)}) - L(w^*) \leq (1 - \eta\mu)(L(w^{(t)}) - L(w^*)) + \frac{L_L \eta^2}{2}. \tag{42}$$

• **B.** Convergence to Local Minimum

If we choose $\eta < \frac{2}{\mu + L_L}$, we can ensure that the term $1 - \eta\mu$ remains positive. Therefore, the convergence behavior can be analyzed using the following inequality:

$$L(w^{(t+1)}) - L(w^*) \leq (1 - \eta\mu)^t (L(w^{(0)}) - L(w^*)) + \frac{L_L \eta^2}{2} \sum_{k=0}^{t-1} (1 - \eta\mu)^k. \tag{43}$$

Using the formula for the sum of a geometric series, we find:

$$L(w^{(t+1)}) - L(w^*) \leq (1 - \eta\mu)^t (L(w^{(0)}) - L(w^*)) + \frac{L_L \eta^2}{2} \cdot \frac{1}{\eta\mu} (1 - (1 - \eta\mu)^t). \tag{44}$$

Taking the limit as $t \to \infty$:

$$L(w^{(t)}) - L(w*) \to 0, \tag{45}$$

which demonstrates that the proposed model converges to the optimal solution.

• C. Rate of Convergence

To analyze the rate of convergence, we can express it in terms of the number of iterations required to reach a desired precision ε:

$$(1 - \eta\mu)^t \leq \frac{\varepsilon}{L(w^{(0)}) - L(w^*)}. \tag{46}$$

Taking logarithms, we find:

$$t \geq \frac{\log\left(\frac{L(w^{(0)}) - L(w^*)}{\varepsilon}\right)}{-\log(1 - \eta\mu)}. \tag{47}$$

Thus, the convergence rate is governed by the learning rate η and the strong convexity constant μ, highlighting the importance of these hyperparameters in practical applications.

4. Results

Experimental Approach: A certain proportion of the initial options data is used as the training set, while the latter portion is used as the validation set. The implied volatility predicted from the model is then used to reverse-calculate the option prices, which are compared with the actual option prices. The root mean square error (RMSE) is employed as the evaluation metric. The performance of the proposed model is compared with traditional statistical methods(ARIMA),machine learning methods(PSO-RF), and deep learning methods(CNN) to demonstrate its superiority.

5. Summary of Experimental Section

In this section, we analyze and summarize the experimental results of time series models, machine learning models, and deep learning models in predicting the implied volatility of 50ETF options. By testing call and put options with different expiration dates and strike prices, we have drawn the following conclusions.

• Accuracy: The deep learning model outperforms time series and machine learning models in terms of R2 and MAE values, indicating its highest accuracy in predicting implied volatility. The machine learning model performs better than the time series model, especially showing more stable performance on the test set.

• Stability: The deep learning model exhibits consistent performance across different expiration dates and strike prices, indicating good stability. The performance of the time series model fluctuates significantly, particularly with large variations in MAE values on the test set.

• Generalization Ability: The deep learning model demonstrates strong generalization ability, with minimal errors on the test set. The machine learning model follows, with some test sets showing higher MAE values. The time series model has the weakest generalization ability, particularly when dealing with highly volatile data.

• Training Speed and Computational Complexity: The time series model has the fastest training speed but the lowest accuracy and stability. The machine learning model has moderate training speed and balances accuracy and stability to a certain extent. The deep learning model has the slowest training speed and highest computational complexity, but its significant accuracy and stability make it highly valuable in practical applications.

Based on the above experimental results, we can conclude that the deep learning model performs the best in predicting implied volatility, followed by the machine learning model, with the time series model performing relatively poorly. Deep neural networks can better capture complex patterns and features in the data through multi-layer nonlinear transformations, thus having significant advantages in prediction accuracy and stability. This also proves the important application value and broad prospects of deep learning models in handling high-dimensional nonlinear problems. Further research can explore how to optimize the training speed and computational complexity of deep learning models for better application in actual financial markets. Detailed information is shown in Figure 3.

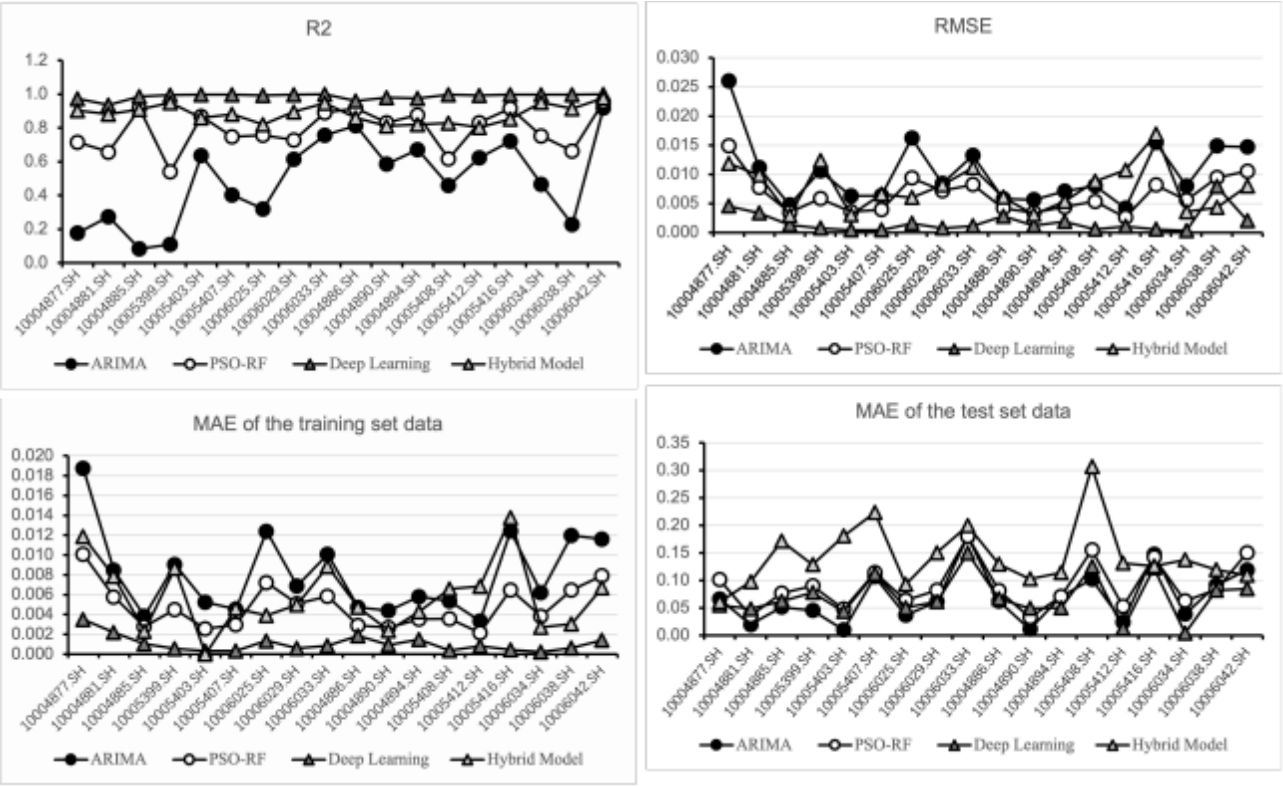


Figure 3. Fit Metrics

6. Future Prospects

Despite the significant advancements in the application of deep learning to the pricing of financial derivatives, there remain numerous areas worthy of exploration. First and foremost, enhancing the generalization ability of deep learning models in high-dimensional, complex pricing problems is an important challenge. Secondly, further integration of physical information with deep learning methods to develop more efficient and accurate solving algorithms will be beneficial in addressing complex issues in actual financial markets. Lastly, exploring the application of deep learning in real-time pricing, particularly in fields requiring rapid responses such as high-frequency trading and risk management, holds great significance.

Moreover, the promotion of interdisciplinary research will further foster innovation in the field of financial pricing through deep learning. For example, combining quantum computing with deep learning techniques can solve complex pricing problems on a larger scale and in higher dimensions. The parallel processing capabilities of quantum computing, coupled with the pattern recognition abilities of deep learning, will bring new breakthroughs in pricing. On the other hand, developing more efficient data generation and augmentation techniques will provide richer and more diverse training data for deep learning models, thereby enhancing the models' generalization ability and robustness.

In summary, the application of deep learning in the pricing of financial derivatives demonstrates immense potential, offering new ideas and methods for research and practice in related fields. With the continuous development and innovation of deep learning technology, it is anticipated that deep learning will achieve more breakthroughs and progress in the field of financial pricing.

References

[1] Wu, K. & Xiu, D. Data-driven deep learning of partial differential equations in modal space. J. Comput. Phys. 408, 109307, DOI: https://doi.org/10.1016/j.jcp.2020.109307 (2020).

[2] Khan, M. M.-U.-R., Arefin, M. R. & Tanimoto, J. Investigating the trade-off between self-quarantine and forced quarantine provisions to control an epidemic: An evolutionary approach. Appl. Math. Comput. 432, 127365, DOI: https://doi.org/10.1016/j.amc.2022.127365 (2022).

[3] Long, Z., Lu, Y. & Dong, B. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. J. Comput. Phys. 399, 108925, DOI:https://doi.org/10.1016/j.jcp.2019.108925 (2019).

[4] Li, P., Niu, Z. & Zhong, W. Temperature modeling and weather derivative pricing based on forward-looking information - a case study of zhengzhou city. Chin. J. Eng. Math. 39, 357–378 (2022).

[5] Meng, X., Li, Z., Zhang, D. & Karniadakis, G. E. Ppinn: Parareal physics-informed neural network for time-dependent pdes. Comput. Methods Appl. Mech. Eng. 370, 113250, DOI:https://doi.org/10.1016/j.cma.2020.113250 (2020).

[6] Wang, X., Li, J. & Li, J. A deep learning based numerical pde method for option pricing. Comput. economics 62, 149–164, DOI: https://doi.org/10.1007/s10614-022-10220-8 (2023).

[7] Raissi, M. & Karniadakis, G. E. Hidden physics models: Machine learning of nonlinear partial differential equations. J. Comput. Phys. 357, 125–141, DOI: https://doi.org/10.1016/j.jcp.2017.11.078 (2018).

[8] Zhu, Y., Zabaras, N., Koutsourelakis, P.-S. & Perdikaris, P. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. J. Comput. Phys. 394, 56–81, DOI: https://doi.org/10.1016/j.jcp.2019.05.024 (2019).

[9] Rodriguez-Torrado, R. et al. Physics-informed attention-based neural network for hyperbolic partial differential equations: application to the buckley–leverett problem. Sci. reports 12, 7557, DOI: https://doi.org/10.1038/s41598-022-11782-1 (2022).

[10] Sun, L., Gao, H., Pan, S. & Wang, J.-X. Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data. Comput. Methods Appl. Mech. Eng. 361, 112732, DOI:https://doi.org/10.1016/j.cma.2020.112732 (2020).

[11] Glau, K. & Wunderlich, L. The deep parametric pde method and applications to option pricing. Appl. Math. Comput. 432, 127355, DOI: https://doi.org/10.1016/j.amc.2022.127355 (2022).