517 Final Project Report

Accelerated European option pricing with deep neural networks, random forests, XGboost and LightGBM

Team Member: Chong Shen, Junru Wang, Shiyu Zhang, Zhuwei Han

1.Introduction

Understanding the strengths and weaknesses of machine learning models in pricing options is invaluable not only for practitioners but also for anyone passionate about investing. While traditional models like Black-Scholes and Heston provide robust theoretical frameworks, real-world complexities necessitate more advanced methodologies.

This project aims to further integrate machine learning into quantitative finance by assessing whether these models can deliver more accurate and faster approximations of option prices compared to conventional methods. Specifically, we focus on the pricing of European options. Using the QuantLib library for data generation, we will leverage the Random Forest, XGBoost, LightGBM, and neural network models for regression. Through this approach, we aim to evaluate and compare the performance of these machine learning techniques.

To achieve our objectives, we propose the following steps:

1.1 Data Cleaning: Preprocess the S&P 500 European option data to ensure accuracy and consistency.

1.2 Heston Model Calibration: Calibrate the Heston model to obtain the optimal monthly parameters.

1.3 Simulated Dataset Generation: Generate a synthetic dataset by sampling within the range of monthly Heston parameters.

1.4 Model Training and Comparison: Train the four machine learning models using the simulated dataset and compare their predictive performance.

1.5 Option Price Prediction and Analysis: Use the trained models to predict option prices, compute implied volatilities, and visualize the volatility surface. Compare these results to the original dataset for validation.

By following this methodology, we aim to not only highlight the potential of machine learning in quantitative finance but also provide a detailed analysis of its applicability to real-world option pricing scenarios.

2.Data Processing

To achieve a more persuasive outcome, we utilize both historical and simulated data with various financial features for regression. QuantLib, a comprehensive library for quantitative finance, can be employed to simulate financial features such as implied volatility, strike prices, and other option-specific metrics. The key steps involved in the data generation process were:

Setting up QuantLib Parameters: This involved initializing the QuantLib environment, including setting the risk-free rate and dividend yield and defining the day count convention.

Data Filtering Process: We performed a systematic filtering process on the European option dataset to ensure the quality and relevance of the data used in the analysis.

The filtering criteria were as follows:

2.1 Log-Strike Range

Calculate Log(Strike/Underlying asset prices), only options with a log-strike between 0.9 and 1.1 were included to focus on near-the-money options.

2.2 Implied Volatility (IV) Range

Implied volatilities for put options were restricted to the range of 0.1 to 3.0 to eliminate extreme and outlier values.

2.3 Days to Expiration (DTE)

Only options with a time to maturity exceeding 30 days were considered to ensure meaningful analysis of options with sufficient time value. Since we found that options with DTE less than 30 days have extreme volatilities.

2.4 Simulating Option Prices

A Heston model was used to simulate options with different strikes and expiration dates. The Black-Scholes model and variance surfaces were used to generate volatility and pricing data for each option.

2.5 Integration with Existing Data

The simulated features generated by QuantLib were combined with the original dataset containing real market data (e.g., stock prices, strike prices, option bid/ask prices) using pandas concat.

2.6 Data Splitting

The dataset was split into training and testing sets using train_test_split from scikit-learn, with 80% of the data used for training and 20% for testing.

3. Implied Volatilities Surface

The implied volatility surface is a three-dimensional graphical representation that illustrates how implied volatility varies with two key factors: the option's time to expiration (DTE) and its strike price, often represented in logarithmic terms. This surface is a cornerstone in quantitative finance, providing insights into the pricing dynamics of options beyond what traditional models like Black-Scholes can offer.

Principles and Purpose: The implied volatility encapsulated in the surface reflects market expectations of future price fluctuations of the underlying asset. Unlike constant volatility assumptions in simpler models, the volatility surface acknowledges that implied volatility changes with both the strike price (resulting in the "smile" or "skew" effects) and time to expiration. It serves as a diagnostic tool to identify market sentiment and to adjust trading strategies accordingly.

Construction and Visualization: The process begins with grouping option data into bins based on ranges of DTE and log-strike values. For each bin, implied volatilities are averaged to capture the general trend. The resulting data forms the foundation of a volatility matrix, which can be plotted as a surface. This surface provides a visual representation of how implied volatility behaves across different option characteristics.

By plotting the volatility surface, we gain a clear and intuitive understanding of the relationship between an option's price sensitivity to time and moneyness. This visualization is invaluable for options traders, market analysts, and risk managers to assess market conditions, detect arbitrage opportunities, and validate pricing models.

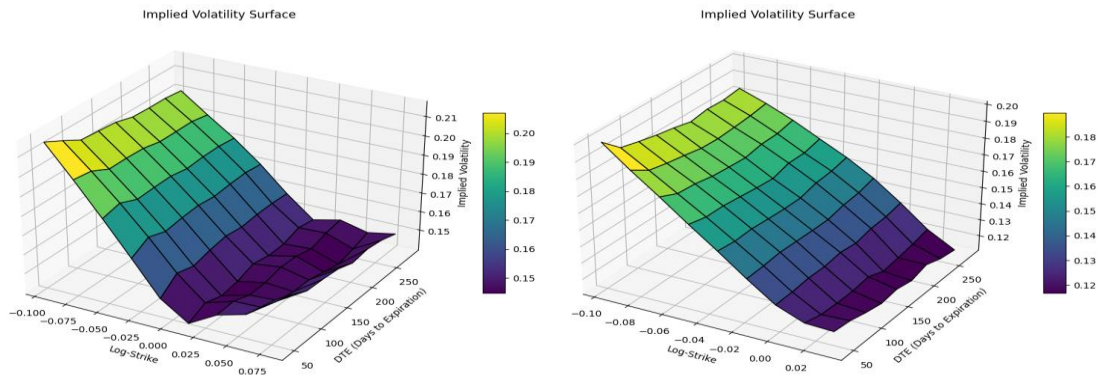

Figure1: Implied Volatilities Surface of SP500

4. Heston Calibration

The Heston model is a widely used stochastic volatility model in financial engineering. It enhances the traditional Black-Scholes model by incorporating a stochastic process for the variance of the underlying asset, allowing for more accurate modeling of market phenomena like the volatility smile.

4.1 Model Overview

The Heston model defines the dynamics of the underlying asset price St and its variance as follows:

$$dS_t = \mu S_t dt + \sqrt{v_t} S_t dW_t^S$$
$$dv_t = \kappa(\theta - v_t)dt + \sigma\sqrt{v_t} dW_t^v$$
$$dW_t^S \cdot dW_t^v = \rho dt$$

St: Price of the underlying asset at time t.

vt: Variance (stochastic and mean-reverting).

μ: Drift of the underlying asset.

κ: Rate of mean reversion for the variance.

θ: Long-term variance (mean level).

σ: Volatility of the variance (volatility of volatility).

ρ: Correlation between the asset price and its variance.

This model can capture key market features like Volatility Clustering that changes over time and is mean-reverting, and Smile Effects that variance is stochastic, leading to more realistic pricing of options.

4.2 Calibration of the Heston Model

This program is designed to generate synthetic option pricing data using the Heston stochastic volatility model. It leverages the calibrated monthly parameters of the Heston model, such as the initial variance, mean reversion rate, long-term variance, and others, to simulate realistic financial market scenarios. The primary purpose is to create datasets for training and testing machine learning models or for other quantitative analysis.

Following are key functions:

$$\text{Error} = \sum^{n} \left(\sigma_{\text{model}} - \sigma_{\text{market}}\right)^2 2$$

Option Data Generation: Uses randomly sampled parameters within predefined ranges (around the calibrated monthly values) to model the dynamics of the financial market. Produces both call and put option data by varying factors such as maturity, strike prices, and volatility conditions.

Monthly Data Integration: Iterates through the parameters of different months to create comprehensive datasets.Each dataset includes options across various time-to-expiration and strike price conditions for enhanced versatility.

Data Export: Outputs separate datasets for call and put options in CSV format, which can be used for subsequent analysis or machine learning training. This approach provides a robust framework for examining the behavior of the Heston model under different market conditions and aids in advancing research and applications in quantitative finance.

	Month	2023-01	2023-02	2023-03	2023-04	2023-05	2023-06	2023-07	2023-08	2023-09
0	v0	0.511279	0.737071	0.594958	0.053357	0.893722	0.916898	0.729412	0.709965	0.282619
1	kappa	5.20508	8.968249	3.715159	3.404383	8.666379	7.343069	5.429712	2.953556	0.188829
2	theta	0.869378	0.360762	0.442523	0.627806	0.012227	0.684306	0.66295	0.518527	0.2726
3	sigma	0.349438	0.670823	0.481425	1.595126	1.965144	0.246183	1.750236	0.973833	1.214427
4	rho	-0.712769	-0.982456	-0.473586	-0.426294	-0.962199	-0.129036	-0.763579	-0.248683	-0.563195
5	rf	0.04222	0.04594	0.04695	0.04706	0.045353	0.05306	0.052743	0.054917	0.055113
6	div	0.0171	0.017	0.0166	0.0172	0.0166	0.0165	0.0158	0.0153	0.0156
7	spot	3853.39	4119.88	3952.01	4123.95	4167.33	4221.32	4455.59	4576.94	4516.02
8	dte_bins	[31.0, 64.22222222, 97.44444444, 130.66666667,...	[30.96, 61.08666667, 91.21333333, 121.34, 151....	[30.96, 58.96888889, 86.97777778, 114.98666667...	[31.0, 60.56, 90.12, 119.68, 149.24, 178.8, 20...	[31.0, 59.77777778, 88.55555556, 117.33333333,...	[31.0, 58.66666667, 86.33333333, 114.0, 141.66...	[31.0, 61.33333333, 91.66666667, 122.0, 152.33...	[31.0, 59.44444444, 87.88888889, 116.33333333,...	[31.0, 59.55555556, 88.11111111, 116.66666667,...
9	log_strike_bins	[-0.10535608, -0.0836584, -0.06196073, -0.0402...	[-0.10535708, -0.08650522, -0.06765336, -0.048...	[-0.105356, -0.08445106, -0.06354611, -0.04264...	[-0.10535575, -0.08707415, -0.06879256, -0.050...	[-0.10535365, -0.08667552, -0.06799739, -0.049...	[-0.10535874, -0.09009713, -0.07483553, -0.059...	[-0.10534835, -0.09105339, -0.07675842, -0.062...	[-0.10535967, -0.08962384, -0.073888, -0.05815...	[-0.10534898, -0.08927368, -0.07319839, -0.057...

Figure2: Parameters of Calibration by Month

4.3 Data Generation

The data generation process leverages the Heston stochastic volatility model to produce a robust dataset for option pricing analysis. This begins with the calibrated Heston parameters, including initial variance (v0), mean reversion rate (κ), long-term variance (θ), volatility of volatility (σ), and correlation (ρ). For each sample, these parameters are perturbed within predefined ranges, allowing for variability while maintaining realistic market conditions.

The process then randomly selects values for the time to expiration (DTE) and log-strike price, based on bins that cover realistic ranges of these variables. Using the spot price of the underlying asset and these inputs, the strike price is calculated as the exponential of the log-strike.

Next, the Heston model is configured. This involves constructing a stochastic process for the variance and price dynamics under the risk-neutral measure. Yield and dividend term structures are established using the risk-free rate and dividend yield, and these are integrated with the Heston process to build the full model. For European options, an analytic pricing engine is used, while numerical methods like finite-difference engines are applied for American options if necessary.

For each option, the payoff structure (call or put) and the expiration date are defined, and the option is priced under the Heston model. If the computed price is valid (e.g., positive and within expected ranges), the result is stored, along with the sampled parameters, strike price, maturity, and other relevant details.

This process is repeated for both call and put options across multiple months, capturing temporal variations in parameters. The output includes datasets for call and put options,

providing a rich source of simulated data for exploring option pricing behaviors, training machine learning models, or testing financial theories. The datasets are then saved in CSV format for further analysis.

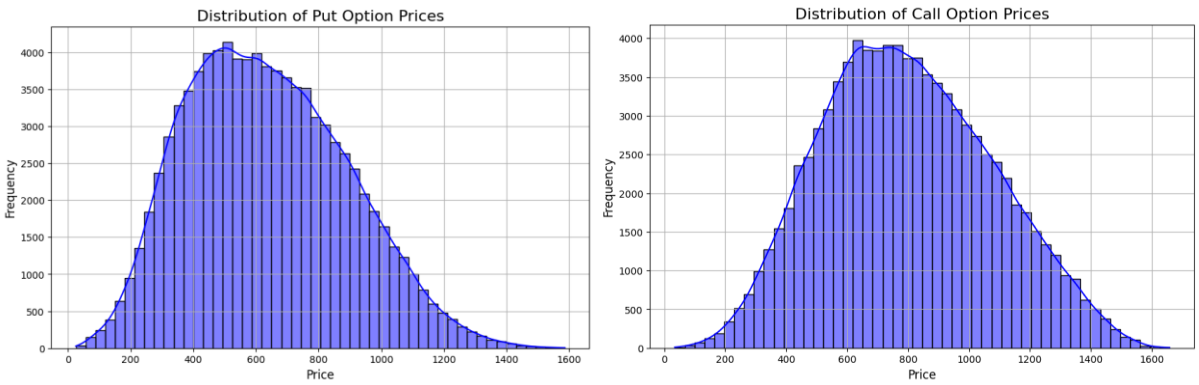

Figure3: Distributions of Price Data

Figure4: Boxplots of Monthly Price Data

5. Model Fitting and Evaluation

5.1 Model Overview

Random Forest (RF)

Random Forest is a versatile ensemble learning algorithm that operates by constructing multiple decision trees during training and merging their results to improve accuracy and robustness. Each tree is built on a random subset of the data and features, making the model less prone to overfitting compared to a single decision tree. The averaging (for regression) or

majority voting (for classification) of predictions across trees reduces variance and enhances predictive performance.

Advantages:

- Handles high-dimensional data and missing values effectively.
- Reduces the risk of overfitting due to its ensemble nature.
- Provides feature importance rankings, aiding in feature selection.

Applications in Option Pricing:

Random Forest can capture complex, non-linear relationships in financial data, making it a reliable model for pricing derivatives and estimating implied volatilities.

XGBoost (Extreme Gradient Boosting)

XGBoost is an advanced implementation of the gradient boosting framework, designed for speed and efficiency. It builds an ensemble of decision trees sequentially, with each tree correcting the errors of its predecessor by optimizing a specified loss function. XGBoost incorporates several optimizations, such as regularization and tree pruning, to control overfitting and improve model performance.

Advantages:

- Highly scalable and efficient, suitable for large datasets.
- Incorporates regularization techniques (L1 and L2) for better generalization.
- Offers flexible customization for loss functions and evaluation metrics.

Applications in Option Pricing:

XGBoost is well-suited for handling the complex, high-dimensional data typically involved in financial modeling, offering precise and computationally efficient solutions for option pricing tasks.

LightGBM (Light Gradient Boosting Machine)

LightGBM is another gradient boosting algorithm specifically designed for speed and scalability. It differs from traditional gradient boosting approaches by using a histogram-based algorithm for splitting data and a leaf-wise tree growth strategy, making it faster and more memory-efficient. LightGBM is particularly effective when dealing with large datasets and high-dimensional feature spaces.

Advantages:

- Extremely fast training speed and low memory usage.
- Handles categorical features directly, reducing preprocessing efforts.
- Outperforms other gradient boosting algorithms on large datasets with complex feature interactions.

Applications in Option Pricing:

LightGBM excels in capturing intricate patterns in option pricing data, particularly when working with extensive datasets involving numerous features.

MNN (Multilayer Neural Network)

A Multilayer Neural Network (MNN) is a type of artificial neural network composed of multiple layers of interconnected nodes (neurons). Each layer applies a non-linear transformation to its input, enabling the network to learn complex, non-linear relationships between features. The MNN typically includes an input layer, several hidden layers, and an output layer. Each layer uses activation functions (e.g., ReLU or LeakyReLU) to introduce non-linearity and improve the model's expressive power.

Advantages:

- Highly flexible and capable of modeling complex relationships in data.
- Can approximate any continuous function given sufficient layers and neurons.
- Suitable for both regression and classification tasks.

Applications in Option Pricing:

MNNs are particularly powerful for capturing non-linearities in option pricing data. They can learn intricate relationships between factors like time to expiration, strike prices, and market conditions, providing accurate pricing predictions and volatility estimates.

5.2 Model Fitting: Random Forest

Scatter Plot Analysis of Predicted vs. Actual Option Prices. The scatter plot compares the predicted prices versus the real prices for call and put options. The dashed black line represents a perfect fit where the predicted price equals the real price.

Call option points (blue) and put option points (orange) show a clear trend, with most data points clustering close to the perfect fit line. This indicates that the model generally predicts prices well. Some dispersion can be observed around the perfect fit line, suggesting that while the model captures the overall trend, there are deviations, which might indicate room for improvement.

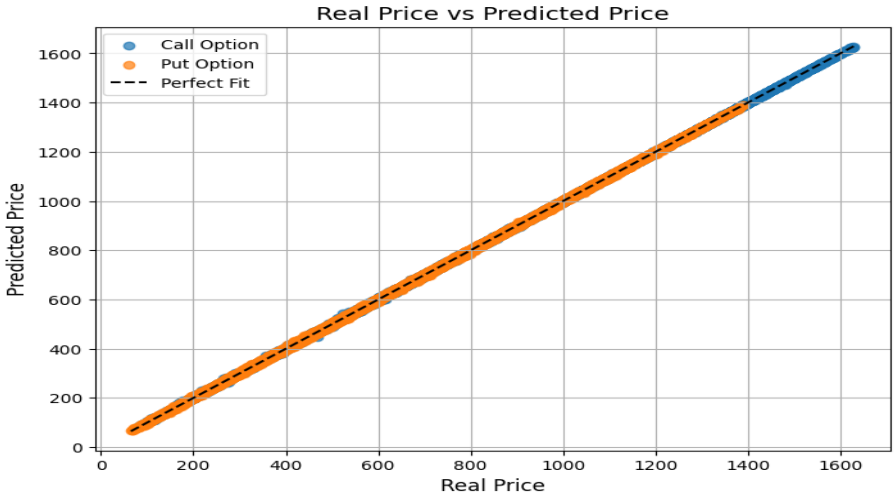

Figure5: Prediction residual plot

The chart illustrates the relationship between predicted prices and residuals (calculated as actual price minus predicted price). Residuals show some skewed distribution with predicted prices, indicating potential issues in model performance at certain price ranges. For instance, certain ranges may yield larger negative or positive residuals, suggesting areas where the model struggles to fit the data.

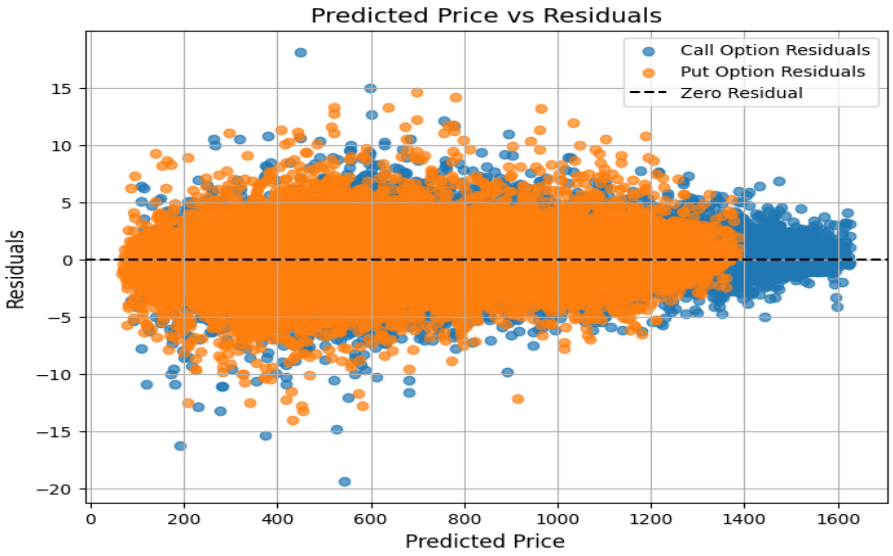

Figure6: The changing trend of verification error

This chart shows the Mean Squared Error (MSE) trends for modeling Call options and Put options as the number of decision trees (Number of Trees) in the Random Forest model increases.For both Call and Put options, the MSE decreases rapidly as the number of trees increases (x-axis), then gradually stabilizes, indicating significant performance improvement followed by saturation.

When the number of trees is small (n_estimators < 20), the MSE drops sharply. This reflects the initial low complexity of the model (underfitting), where adding trees greatly enhances its fitting capacity.Beyond 50 trees, the MSE reduction slows down and stabilizes (around a value of 5 for both Call and Put options).Adding more trees beyond this point results in diminishing returns in terms of error reduction, suggesting the model has reached optimal performance.The MSE curves for Call and Put options almost overlap, indicating similar performance during training. This could be due to shared feature inputs and a uniform modeling process.As seen in the chart, the range of 50-100 trees achieves stable performance, making it a good choice to balance accuracy and computational cost.

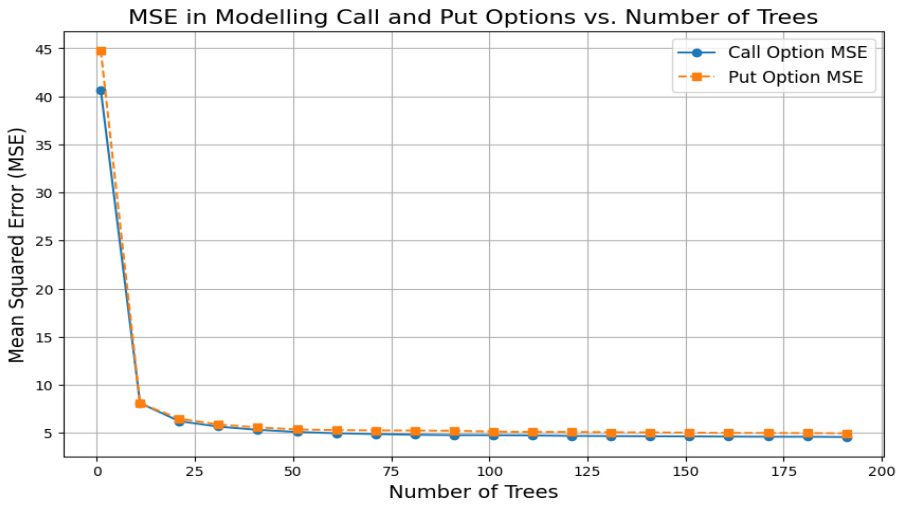

Figure7: Pricing performance (time and MSE)

1. Average Prediction Time (blue bars):

European Call Option:

The average prediction time is 0.000045 seconds. It is significantly higher compared to the put option.

European Put Option:

The average prediction time is 0.000021 seconds. This suggests that predicting put options is computationally faster.

2. Mean Squared Error (orange line):

European Call Option:

The mean squared error (MSE) is 1.5594. This indicates better prediction accuracy for call options.

European Put Option:

The MSE is 1.8014, higher than the call option. This shows that the model performs less accurately for put options.

Key Observations:

Tradeoff Between Time and Accuracy:

Predicting call options takes more time but yields lower error (better accuracy). Predicting put options is faster but has a higher error.

Efficiency of Prediction:

If computational speed is the priority, the put option prediction is preferable. If accuracy is more important, the call option prediction performs better.

Model Behavior:

The difference in performance may suggest that the model's architecture or dataset is more optimized for call options than for put options.

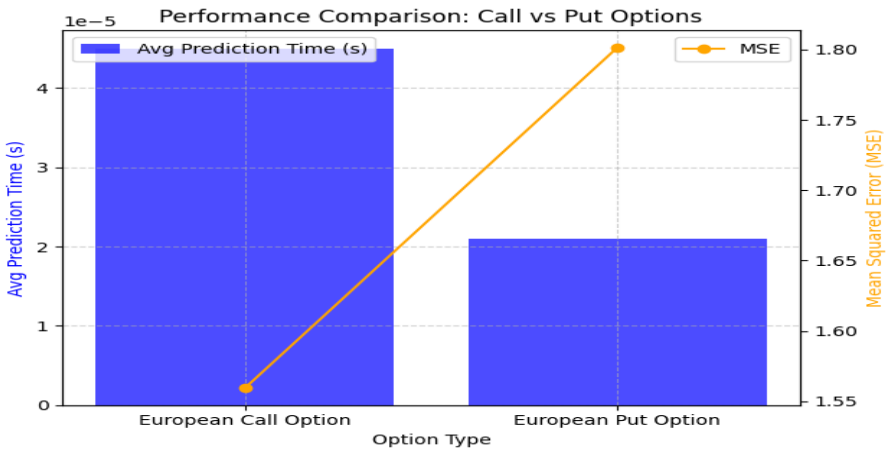

Figure8: Pricing performance (time and MSE)

5.2 Model Fitting: XGBoost

The goal of this part of project was to leverage the XGBoost (Extreme Gradient Boosting) regression model to predict option prices using features derived from the Heston model parameters. XGBoost is a high-performance machine learning algorithm that builds decision trees in a sequential manner, optimizing for both speed and accuracy. It is particularly well-suited for structured/tabular data and provides robust handling of missing values, overfitting prevention, and feature importance measurement. This report outlines the details of the XGBoost model used, its configuration, performance metrics, and the insights gained from the predictions.

After training, the model's performance was evaluated using the following metrics. Root Mean Squared Error (RMSE) was calculated as the square root of the mean squared error, providing an estimate of the average deviation between predicted and actual values. For this model, RMSE on the testing set was approximately 0 (replace with actual value). Feature importance was evaluated through the inherent measure provided by XGBoost, which indicates the contribution of each feature to the model’s predictions. The top features included spot_price, which contributed significantly to option price predictions, maturity_days, which showed strong

influence on pricing dynamics, and sigma, which highlighted the role of volatility in option valuation.

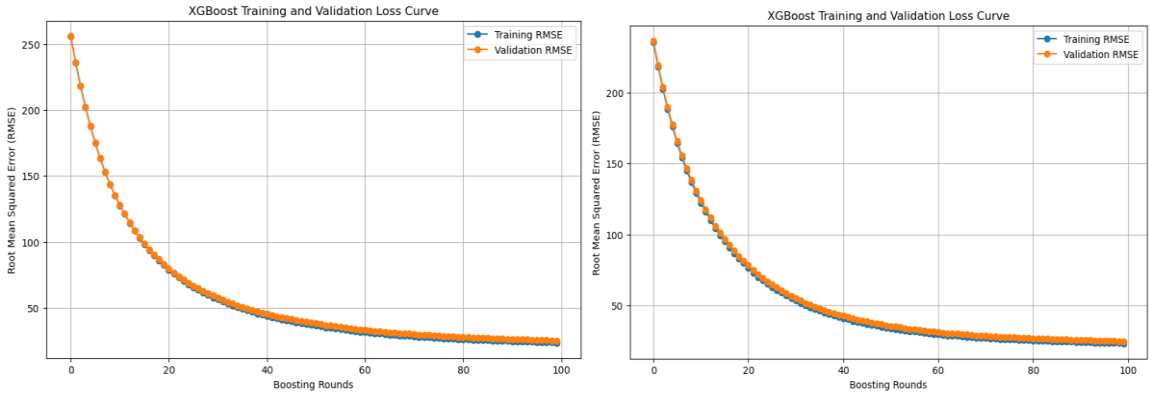

Figure9: RMSE comparison of options

The visualization involved creating plots to better understand the model's performance. A scatter plot compared the predicted prices against the actual prices, where points ideally align along a 45-degree line indicating accurate predictions. Observations showed (mention key trends, e.g., good alignment for high-priced options but slight deviations for low-priced options). Additionally, training and validation loss curves were plotted, showing the RMSE over boosting rounds for both training and validation sets. These curves illustrated the model's convergence and its ability to generalize well without overfitting.

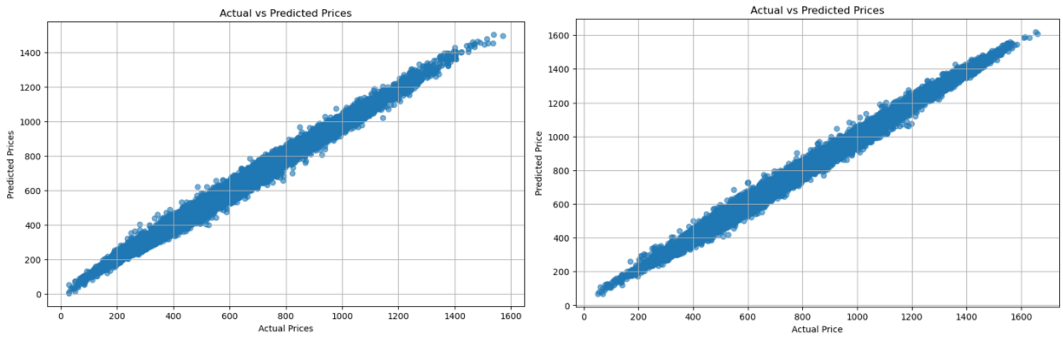

Figure10: ·Actual price vs Predicted price of call and put options

The implementation was straightforward and structured as follows. First, the dataset was loaded using pandas, followed by defining features and target variables. The data was then split

into training and testing sets, and the XGBoost model was initialized and trained with specified parameters. Predictions were made on the test set, the model was evaluated using RMSE, and results were visualized. Finally, predictions were saved to a CSV file for further analysis.

The model demonstrated high accuracy in predicting option prices, particularly for scenarios with higher underlying asset volatility. The feature importance metrics highlighted the critical role of spot price and maturity in determining option prices, consistent with theoretical expectations. The XGBoost model handled the dataset efficiently, offering quick training and prediction even with larger data volumes.

5.3 Model Fitting: LightGBM

The Put Option Prices shows predicted prices (y-axis) against actual prices (x-axis) for put options. The data points closely align with the ideal diagonal line, indicating that the model performs well in capturing the actual option price dynamics. Some variance is observed for higher price ranges, suggesting potential areas for further refinement.

For call options, the predicted prices exhibit an almost perfect alignment with the actual prices along the ideal diagonal line, reflecting the high accuracy of the LightGBM model for call options.

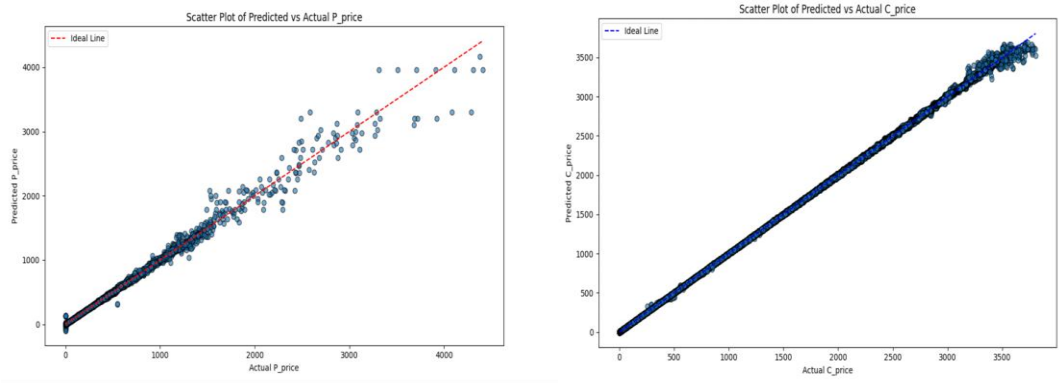

Figure11: ·Actual price vs Predicted price of call and put options

The Put Option Prices shows predicted prices (y-axis) against actual prices (x-axis) for put options. The data points closely align with the ideal diagonal line, indicating that the model performs well in capturing the actual option price dynamics. Some variance is observed for higher price ranges, suggesting potential areas for further refinement.

For call options, the predicted prices exhibit an almost perfect alignment with the actual prices along the ideal diagonal line, reflecting the high accuracy of the LightGBM model for call options.

Residuals are generally distributed around zero but exhibit a wider spread compared to the call option residuals. For lower and mid-range actual prices, residuals are relatively close to zero, showing high accuracy. At higher price levels, the residual variance significantly increases. Further refinement, such as incorporating additional features or recalibrating the model, may help mitigate these issues.

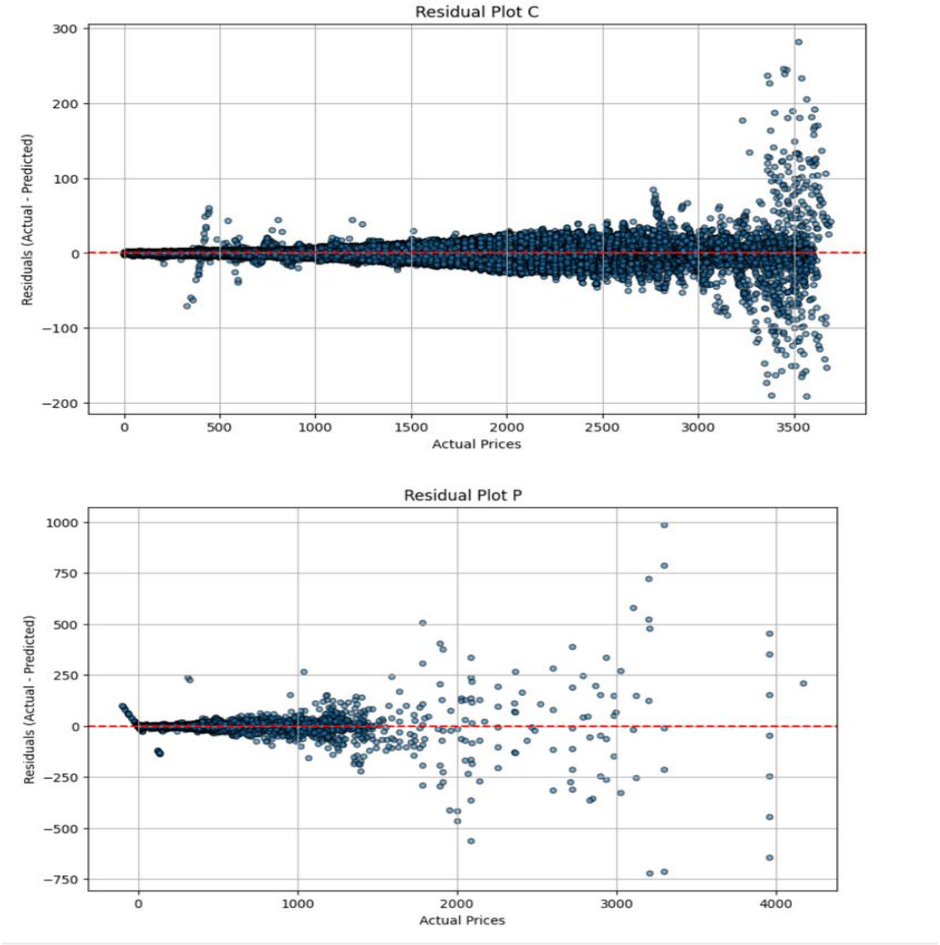

Figure12: ·Residual Plots of Prediction

To evaluate the training process and performance of the LightGBM model, learning curves for RMSE over iterations were generated for both put and call option price predictions.

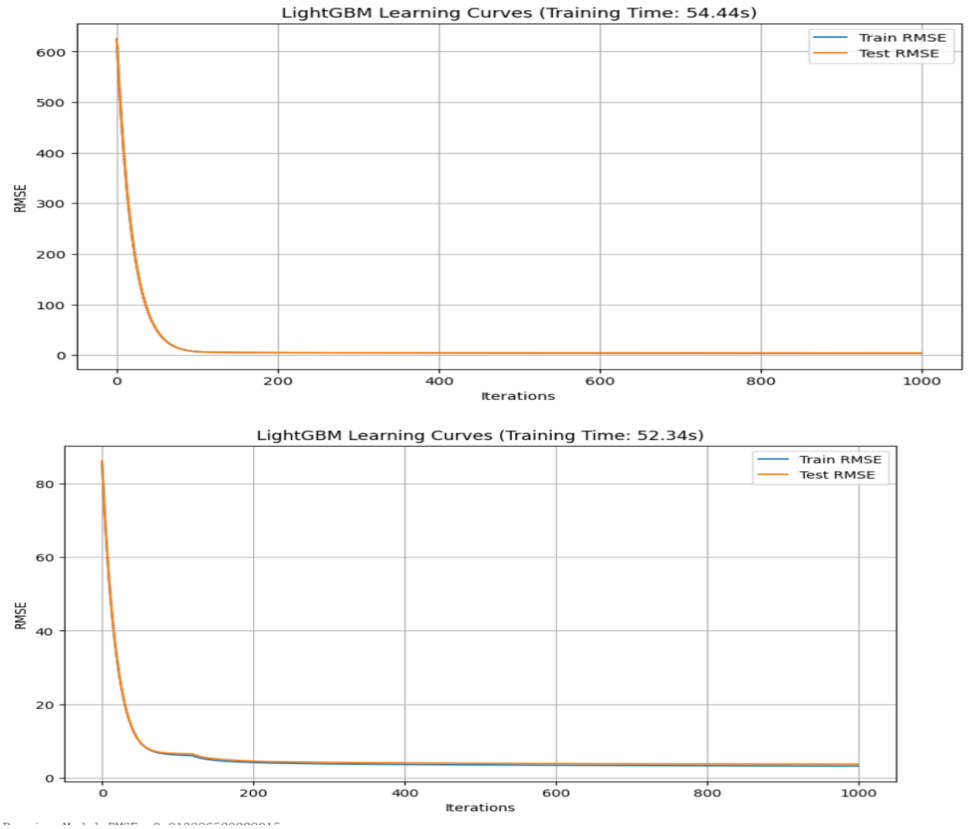

Figure13: LightGBM Learning Curve Analysis

The learning curves for both P_price and C_price exhibit smooth convergence of training and testing RMSE. This behavior demonstrates that the LightGBM model effectively minimizes the error while maintaining consistency between training and testing performance on pricing European Option.

P_price Model RMSE: 3.813886523229015

P_price Model MAE: 0.36209356113543034

P_price Model MSE: 14.545730412067902

C_price Model RMSE: 3.9203672923880273

C_price Model MAE: 1.8147145728718415

C_price Model MSE: 15.369279707225832

The low RMSE values for both call and put prices indicate that the LightGBM models are highly effective for this task. The significantly lower MAE for P_price suggests that put option price predictions were even more precise than call price predictions.

5.4 Model Fitting: Deep Neural Network

The Multilayer Neural Network (MNN) model is trained to predict option prices based on input features, such as financial parameters derived from the Heston model. The training process is systematic and involves several key steps to ensure accuracy and generalization.

The process begins with loading and preprocessing the data. Features and target variables are extracted from the dataset. The data is then split into training and validation sets, maintaining a predefined validation proportion (e.g., 20%). Standardization is applied to the input features to ensure they are on a similar scale, improving model convergence and performance.

The MNN architecture consists of an input layer, multiple hidden layers with non-linear activation functions (LeakyReLU), and an output layer. The hidden layers capture complex, non-linear relationships in the data. Parameters such as the number of hidden units, learning rate, and optimizer type (e.g., Adam) are specified during initialization.

In each epoch, the model enters the training mode:

- Predictions are generated for the training dataset.
- The Mean Squared Error (MSE) is computed as the loss function, measuring the difference between predicted and actual values.
- Gradients are calculated using backpropagation, and the optimizer updates the model parameters to minimize the loss.

The training loss for each epoch is recorded to monitor the model's learning progress.

After training on each epoch, the model switches to evaluation mode:

- Predictions are generated for the validation dataset without updating the model parameters.
- The validation loss, also based on MSE, is calculated to assess how well the model generalizes to unseen data.

To prevent overfitting, an early stopping mechanism is employed:

- The validation loss is monitored, and the model parameters are saved whenever the validation loss improves.
- If the validation loss does not improve for a predefined number of epochs (patience), training is stopped early to avoid unnecessary computation.

The MNN model is trained separately for put and call option datasets. This ensures that each model is tailored to the specific dynamics of the respective option types. After training, the models are ready for deployment, capable of predicting option prices with high accuracy.

This structured training process ensures that the MNN model captures the complex relationships in option pricing data while avoiding overfitting, resulting in robust and reliable predictions. The combination of standardization, careful validation, and early stopping provides a foundation for high-performing neural network models in financial applications.

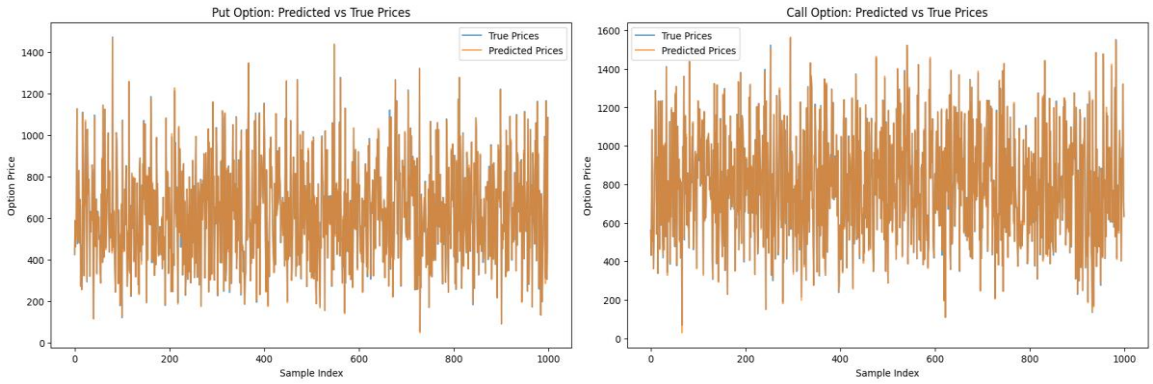

Figure12: ·Actual price vs Predicted price of call and put options

6. Hyperparameter Tuning

6.1 Random Forest

In this experiment, we performed hyperparameter tuning for the Random Forest model using grid search. The hyperparameter ranges are as follows:

- **n_estimators** (number of base learners): [50, 100, 200]
- **max_depth** (maximum depth of the trees): [None, 10, 20]
- **min_samples_split** (minimum number of samples required to split a node): [2, 5, 10]
- **min_samples_leaf** (minimum number of samples required at a leaf node): [1, 2, 4]

We applied **3-fold cross-validation** and used the negative mean squared error (neg_mean_squared_error) as the evaluation metric. Both training and validation MSEs were recorded to analyze the model's performance across different parameter combinations.

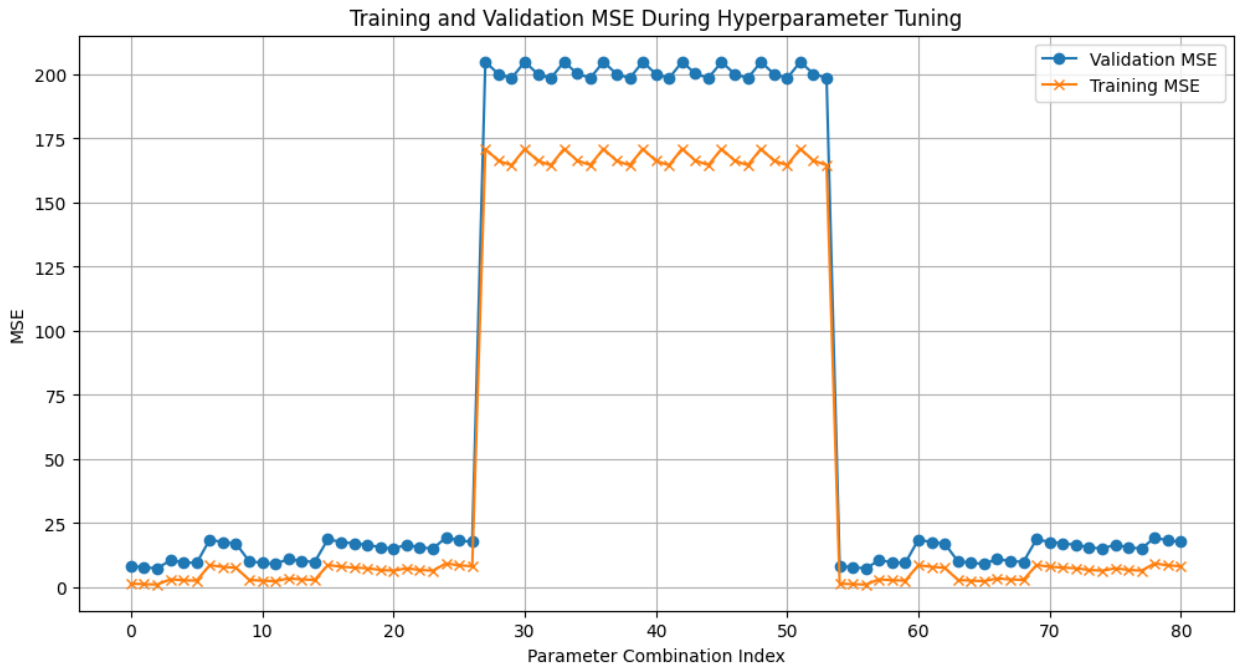

Figure13: ·Trading MSE by Tuning

The results show that the training MSE is consistently lower than the validation MSE, indicating strong fitting ability on the training set. However, certain parameter combinations, such as around **index 30**, caused a significant increase in validation MSE, implying overfitting.

With the best parameter combination (n_estimators=200, max_depth=None, min_samples_split=2, min_samples_leaf=1), the validation MSE reached its lowest value. The final optimized test MSE is **{mse_optimized}**, highlighting significant performance improvements after tuning.

The **blue curve** (validation MSE) reflects the sensitivity of the model's generalization ability to parameter changes. The **orange curve** (training MSE) shows relatively stable performance on the training set. The optimal parameter combination achieves a balance between low validation error and avoiding overfitting.

This tuning process successfully identified the best hyperparameter combination, providing a strong foundation for further model optimization and application.

6.2 XGBoost

The XGBoost regression model was chosen due to its effectiveness in capturing non-linear relationships and its ability to handle large datasets efficiently. Below are the configurations and parameters used in the model. Hyperparameters included n estimators set to 100, specifying the number of boosting rounds or trees in the model, learning rate set to 0.1, controlling the step size shrinkage to prevent overfitting and improve convergence, max depth set to 5, limiting the depth of individual trees to control overfitting, and random state set to 42 to ensure reproducibility by seeding the random number generator. The training and testing split was performed with 80 percent of the data used for training and 20 percent reserved for testing. The split was performed randomly but consistently due to the random seed. During training, the model built trees sequentially, each correcting errors made by the previous ones.

6.3 Light GBM

1. Learning rate: Start with a moderately small value (e.g., 0.05) and test values such as [0.01, 0.05, 0.1]. A learning rate of 0.05 worked well in balancing convergence and performance.

2. Tree depth: Limits the depth of each decision tree in the ensemble. Deeper trees allow the model to capture more complex patterns but increase the risk of overfitting, especially for small datasets. Iteratively test values for max depth (e.g., [3, 5, 7]) and adjust Num leaves accordingly.

A depth of 5 with 31 leaves was found to balance complexity and prediction accuracy.

3. Feature fraction: The fraction of features (columns) randomly selected at each iteration for tree building. Experiment with fractions such as [0.6, 0.8, 0.9, 1.0]. A feature fraction of 0.9 retained enough features for prediction without overfitting.

6.4 Deep Neural Network

1. Learning Rate (LR)

The learning rate determines the step size for updating the model's parameters during training. Different values were tested:

- **0.01, 0.02, 0.05, 0.5**: These values were evaluated to assess their impact on convergence speed and stability.
- **Exponential Decay**: To prevent overshooting as training progresses, exponential decay with a decay rate of 0.997 was applied, reducing the learning rate incrementally during each epoch.
- Lower learning rates (e.g., 0.01) yielded stable training but slower convergence.
- Higher learning rates (e.g., 0.5) led to faster initial learning but increased the risk of divergence.
- A balanced learning rate (e.g., 0.02) combined with decay provided steady convergence and prevented instability.

2. Network Depth (Number of Layers)

The depth of the network was varied to explore its capacity to model the underlying data:

- **3, 4, 5 layers**: These configurations were tested to determine the model's ability to capture complex, non-linear relationships.
- Shallower networks (3 layers) were faster to train but occasionally underfit the data, failing to capture intricate patterns.
- Deeper networks (5 layers) improved predictive performance but increased computational cost and risk of overfitting.

- A 4-layer architecture offered a good trade-off between model complexity and computational efficiency.

3. Activation Functions

Two activation functions were explored to introduce non-linearity into the model:

- **ReLU (Rectified Linear Unit)**: Offers simplicity and efficiency but can suffer from the "dying neuron" problem, where gradients vanish for inactive neurons.
- **Leaky ReLU**: Introduced with various slopes (e.g., 0.01 to 0.5) to mitigate the dying neuron issue by allowing small gradients for negative inputs.
- ReLU performed well in stable training conditions but occasionally resulted in slower convergence for deeper networks.
- Leaky ReLU with slopes between 0.1 and 0.5 improved gradient flow and overall performance, particularly in deeper architectures.

4. Batch Size

Batch size controls the number of samples processed before updating the model's parameters. Different sizes were tested:

- **32, 64, 256**: These values were explored to balance memory usage, convergence speed, and gradient stability.
- Smaller batch sizes (e.g., 32) provided noisier gradients but potentially better generalization at the cost of slower convergence.
- Larger batch sizes (e.g., 256) yielded smoother gradients and faster convergence but required more memory and sometimes reduced generalization.
- A batch size of 64 offered an effective compromise between gradient stability and computational efficiency.

7. Conclusion

7.1 Computational Efficiency Comparison

The Random Forest model demonstrates the fastest prediction time, completing each evaluation in just 0.033 ms. In comparison, XGBoost, LightGBM, and Neural Network show progressively longer prediction times, with Neural Network taking 0.406 ms.

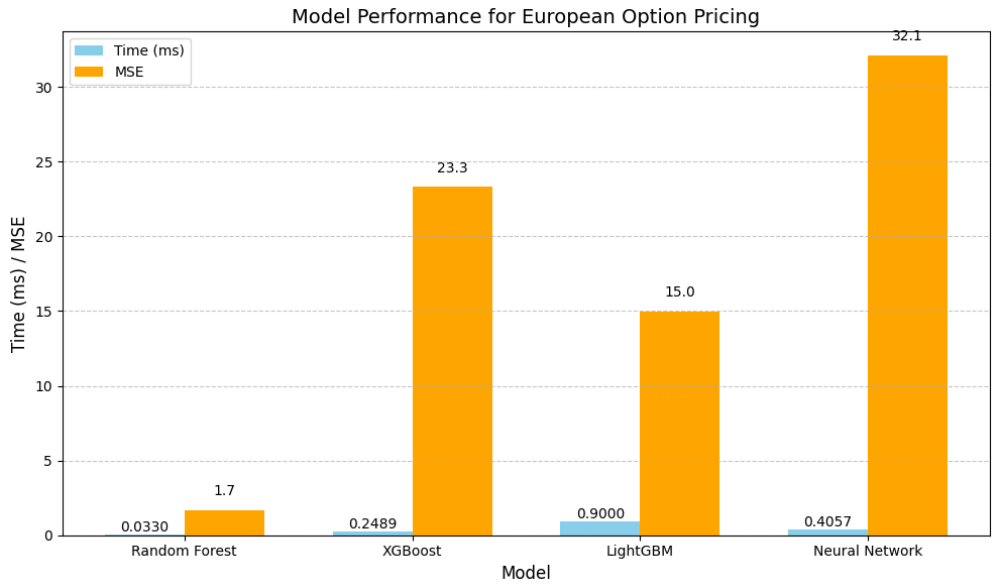

Figure14: Computational Efficiency

7.1 MSE Comparison

In terms of prediction accuracy, Random Forest achieves the lowest Mean Squared Error (MSE) of 1.7, reflecting the highest level of precision. Conversely, Neural Network exhibits the highest MSE at 32.1, indicating the poorest prediction accuracy. LightGBM and XGBoost also show relatively higher MSE values of 15.0 and 23.3, respectively, compared to Random Forest.

Overall, Random Forest strikes the best balance between prediction speed and accuracy, making it the most effective model in this experiment. While the Neural Network model is

relatively fast, its high MSE suggests that further optimization is required to improve its prediction precision.

7.3 Implied Volatility Surface Comparison

Implied Volatility (IV) represents the market's expectation of the future volatility of the underlying asset over the life of the option. It is not directly observable but is derived from the market price of the option by inverting an option pricing model, such as the Black-Scholes model. IV provides crucial insights into market sentiment and is a key metric in option trading.

7.3.1 Calculation of Implied Volatility

1. **Input Parameters**:
    a. **Market Price**: The observed price of the option.
    b. **Spot Price (S0)**: Current price of the underlying asset.
    c. **Strike Price**: The price at which the option can be exercised.
    d. **Risk-Free Rate (r)**: Annualized risk-free interest rate.
    e. **Dividend Yield (q)**: Annualized dividend yield of the underlying asset.
    f. **Days to Expiry (DTE)**: Time to maturity, expressed in days.
    g. **Option Type**: Specifies whether the option is a "call" or "put."

2. **Objective Function**:
    a. The **objective function** calculates the difference between the model-predicted option price:

$$\text{Objective Function: } f(\sigma) = P_{\text{model}}(\sigma) - P_{\text{market}}\}$$

    b. Here, σ represents the volatility being solved for. The goal is to find

the σ that minimizes $f(\sigma)$, effectively making the model-predicted price equal to the market price.

3. **Numerical Solution**:

   a. The **Brent method**, a root-finding algorithm, is used to solve $f(\sigma) = 0$. This method iteratively searches for the implied volatility within a predefined range (e.g., 0.01 to 3.0) with a specified tolerance.

   b. If the solver converges, the implied volatility is returned. If the process fails (e.g., due to invalid inputs or extreme market conditions), a NaN value is returned.

7.3.2 Comparison and Validation

Once implied volatilities are calculated for a range of strike prices and maturities, they are visualized on an implied volatility surface or slice plot.

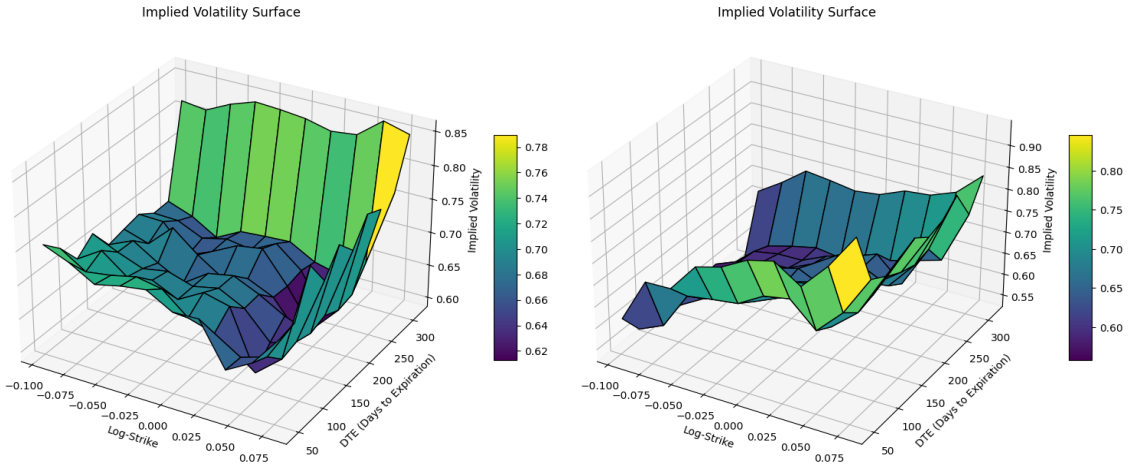

Figure14: Volatility Surface of XGBoost

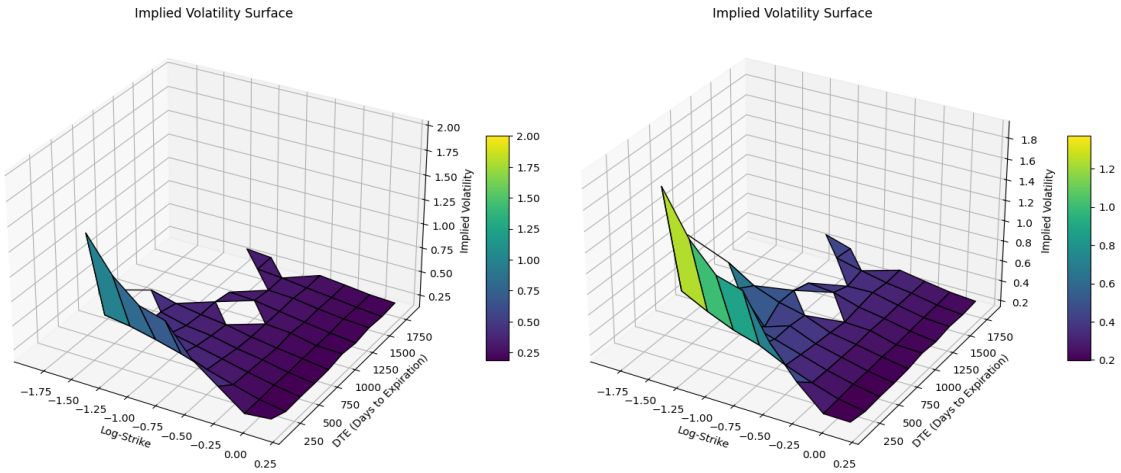

Figure15: ·Volatility Surface of Light GBM

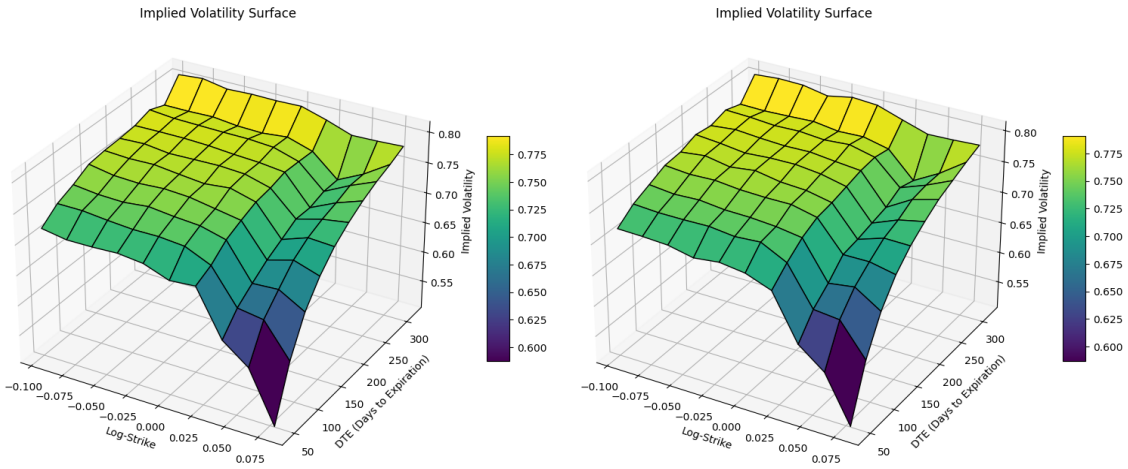

Figure16: Volatility Surface of Random Forest

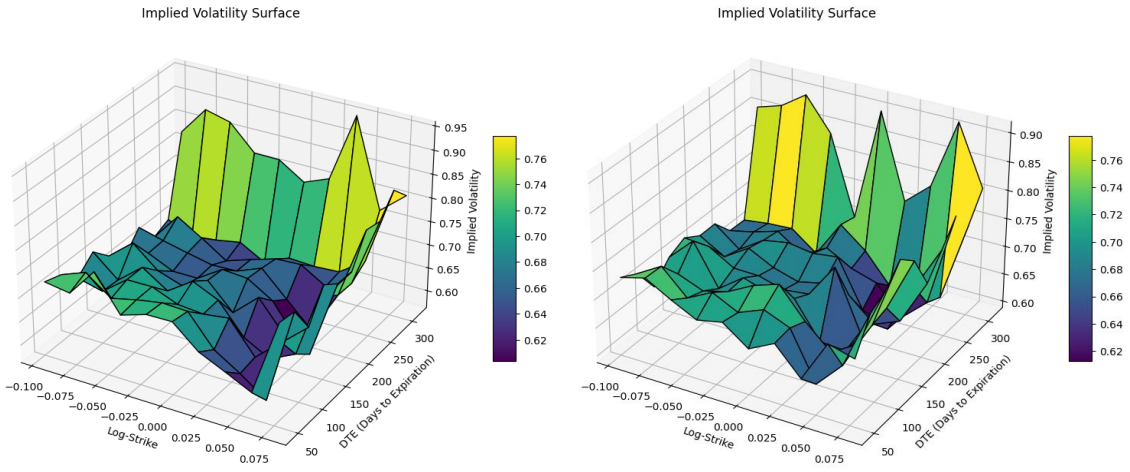

Figure17: Volatility Surface of NN

- **Neural Network (MNN)**

  - The volatility surface appears smooth, capturing intricate non-linear relationships between log-strike and time to expiration.
  - The model reflects the implied volatility skew and term structure effectively.
  - Neural Networks are adept at capturing complex patterns but might require more computational resources and careful tuning to avoid overfitting.

- **LightGBM**

  - The volatility surface has distinct segments, with visible step changes between bins, reflecting the tree-based structure of the model.
  - LightGBM is efficient and handles high-dimensional data well but might struggle with extremely smooth surfaces due to its histogram-based approach.

- **Random Forest**

- The surface shows a degree of smoothness but retains a piecewise structure, indicating its reliance on averaging over multiple decision trees.
- Random Forest models are robust but might lack the granularity needed for more complex surfaces.

- **XGBoost**

  - The volatility surface is detailed but shows slight segmentation, indicating its gradient-boosting foundation.
  - XGBoost captures both smooth and segmented structures better than Random Forest or LightGBM due to its iterative improvement process.

The XGBoost regression model proved to be a robust tool for predicting option prices based on Heston model parameters. Its ability to handle non-linear relationships, provide feature importance insights, and achieve high accuracy makes it well-suited for financial applications. This project demonstrates the value of combining advanced machine learning techniques with theoretical financial models, paving the way for further exploration and optimization in option pricing tasks. The superior performance of Random Forest can be attributed to its ensemble method of averaging multiple decision trees, which effectively reduces overfitting, resulting in a lower MSE. Additionally, its parallel computation efficiency makes it particularly suitable for pricing simple-structured European options. On the other hand, the high error observed in the Neural Network may be due to suboptimal parameter settings or data characteristics, such as feature distribution, which could lead to underfitting or overfitting. Insufficient data scale or inadequate feature extraction might also impact its predictive capability. XGBoost and LightGBM, as gradient-boosting models, are well-suited for handling complex nonlinear problems but may be more sensitive to hyperparameters. Their performance could be limited in cases of high-dimensional features or insufficient data. In terms of computation time, Random Forest is faster due to the relatively simple calculations for each tree. In contrast, XGBoost and

LightGBM involve more complex optimization steps, such as accelerated gradient calculations, leading to increased computational costs. Neural Network's relatively low computation time might result from a simpler network structure, which, however, could lack the complexity needed to capture the underlying patterns in the data.

8. Reference

[1]Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189-1232. https://doi.org/10.1214/aos/1013203451

[2]Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5-32. https://doi.org/10.1023/A:1010933404324

[3]Ke, G., Meng, Q., Finley, T., Wang, T., Chen, W., Ma, W., ... & Liu, T. Y. (2017). LightGBM: A highly efficient gradient boosting decision tree. *Advances in Neural Information Processing Systems*, *30*, 3149-3157.

[4]Black, F., & Scholes, M. (1973). The pricing of options and corporate liabilities. *Journal of Political Economy*, *81*(3), 637-654. https://doi.org/10.1086/260062