Journal of Theoretical and Applied Information Technology
31st December 2024. Vol.102. No. 24
© Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

LSTM ADAPTIVE HYPERPARAMETER TUNING FOR FINANCIAL TIME SERIES FORECASTING USING CUSTOM GRADIENT-BASED METHODS: A COMPARATIVE STUDY WITH BAYESIAN OPTIMIZATION

ADNANE EL OUARDI[1], BRAHIM ER-RAHA[1], KHALID TATANE[1]

[1] ESTIDMA research team, National School of Applied Sciences, Agadir, Morocco

E-mail: [1]adnane.elouardi.7@gmail.com, [1]b.raha@uiz.ac.ma, [1]k.tatane@uiz.ac.ma

ABSTRACT

This paper presents an adaptive hyperparameter tuning system for Long Short-Term Memory (LSTM) models, focusing on a gradient-based approach to achieve efficient and precise optimization. The proposed system is designed to dynamically adjust critical hyperparameters such as the learning rate, number of units, dropout rate, and batch size during the training process. By leveraging gradient information, the system iteratively refines the hyperparameter values, aiming to minimize the Mean Squared Error (MSE) and enhance the model's predictive accuracy. To evaluate the effectiveness of the gradient-based approach, the study includes Bayesian optimization as a benchmark, a method widely recognized for its probabilistic framework and established success in hyperparameter tuning. Comparative analysis is conducted in terms of loss metrics, including MSE and additional performance indicators, as well as execution time, highlighting the trade-offs between optimization efficiency and computational cost. The results demonstrate the gradient-based system's ability to adapt to high-dimensional and complex hyperparameter spaces with reduced computational overhead, while consistently achieving superior performance. The experiment is applied to the challenging task of forecasting the S&P 500 index, a real-world financial time series problem that demands robustness and precision.

Keywords: *Hyperparameter Optimization, Gradient-Based Tuning, LSTM Networks, Bayesian Optimization, Time Series Forecasting*

1. INTRODUCTION

Hyperparameter tuning is a critical step in the development of Long Short-Term Memory (LSTM) models for time series forecasting. Identifying optimal hyperparameters, such as the number of units, learning rate, dropout rate, and batch size, is essential to improve the model's performance and accuracy. Traditional methods for hyperparameter tuning, such as grid search and random search, are computationally expensive and often impractical for complex models with large parameter spaces [1] [2].

Recent advances have explored more efficient methods for hyperparameter optimization. Bayesian optimization has gained popularity due to its probabilistic approach to model performance estimation, allowing for more informed hyperparameter selection and reduced computational overhead. However, Bayesian optimization can struggle with the high-dimensional and non-convex nature of hyperparameter spaces in deep learning models [3] [4].

An alternative approach is gradient-based hyperparameter tuning, which leverages gradient information to iteratively refine hyperparameters. This method shows promise in navigating large parameter spaces more effectively and adapting hyperparameters dynamically during the training process. For instance, recent research has demonstrated the use of gradient-based methods for hyperparameter optimization in complex neural network architectures, showing significant improvements in convergence and performance [5] [6] [7].

Recent studies have further demonstrated the efficacy of hybrid optimization techniques, which blend the strengths of gradient-based methods and probabilistic approaches like Bayesian optimization. For instance, advancements in differentiable optimization frameworks have made gradient-based methods more scalable and suitable for complex hyperparameter spaces [8][9].

In this study, we propose an adaptive hyperparameter tuning system for LSTM models, focusing on a custom gradient-based method. This system dynamically adjusts hyperparameters based on performance feedback, aiming to minimize the Mean Squared Error (MSE) in forecasting the S&P 500 index close prices. To benchmark the effectiveness of our approach, we compare it against the performance of Bayesian optimization, a commonly used technique in hyperparameter tuning [2] [10].

To achieve this, the study will:

- ✓ Design and implement a personalized gradient-based hyperparameter tuning mechanism that leverages gradient information to refine hyperparameters iteratively.
- ✓ Integrate a feedback loop to dynamically adjust hyperparameters during the training process, ensuring continuous improvement in model performance.
- ✓ Benchmark the gradient-based method against Bayesian optimization to evaluate its effectiveness and efficiency. Bayesian optimization will serve as a comparative baseline due to its established use in hyperparameter tuning [2] [3].
- ✓ Assess the performance of the proposed system by applying it to the task of forecasting the S&P 500 index.

The anticipated outcomes include a comprehensive evaluation of the gradient-based hyperparameter tuning system, demonstrating its potential to enhance the predictive accuracy of LSTM models. This study will contribute to the existing body of knowledge by providing insights into the practical application of gradient-based methods for hyperparameter optimization in complex machine learning models [6] [5] [11].

2. LITERATURE REVIEW

In this section, we review key methods for hyperparameter optimization, including Bayesian optimization, gradient-based approaches, and the use of LSTM networks for time series forecasting, providing a foundation for the study.

2.1 Bayesian Optimization in Hyperparameter Tuning

Bayesian optimization has been widely recognized for its efficacy in hyperparameter tuning, leveraging probabilistic models to make informed decisions on sampling hyperparameters [2] [8]. This approach has been successfully applied in various machine learning tasks, including neural network training, due to its ability to handle the high-dimensional and non-convex nature of hyperparameter spaces [4] [3]. Despite its strengths, Bayesian optimization can be computationally intensive, especially for models with numerous hyperparameters, as the complexity of the probabilistic models scales with the number of dimensions and iterations. Some recent approaches attempt to mitigate this issue by using surrogate models or approximations to reduce computational costs while maintaining optimization efficiency [12] [13].

2.2 Gradient-Based Hyperparameter Optimization

Gradient-based methods offer an alternative by utilizing gradient information to iteratively adjust hyperparameters, aiming for more efficient convergence. Scientists introduced reversible learning, enabling the computation of exact gradients of cross-validation performance with respect to hyperparameters, significantly improving the optimization of complex neural network architectures [5]. Recent advancements include bilevel optimization frameworks, which enhance the efficiency and scalability of gradient-based tuning by structuring the optimization problem as a nested set of tasks, often referred to as upper and lower levels. This approach has been demonstrated to outperform traditional gradient-based methods in several deep learning applications [6] [11] [7]. Recent approaches have explored meta-gradient methods to accelerate the tuning of deep learning models, offering improved adaptability to dynamic data distributions [12][13]. These methods complement traditional bilevel frameworks by reducing computational overhead in high-dimensional parameter spaces. Additionally, automatic differentiation techniques and differentiable optimization frameworks further facilitate the practical application of these methods to a broader range of machine learning problems [14] [15].

2.3 LSTM for Time Series Forecasting

Long Short-Term Memory (LSTM) networks, a type of recurrent neural network (RNN), are particularly suited for time series forecasting due

to their capability to capture long-term dependencies. LSTM models have been extensively applied in financial market predictions, demonstrating substantial improvements in accuracy over traditional methods, such as linear regression or ARIMA models [16] [17]. The effectiveness of LSTM networks in modeling sequential data, such as stock prices and economic indicators, makes them a prime candidate for exploring advanced hyperparameter tuning techniques. Moreover, LSTMs have been shown to handle the non-linear and non-stationary nature of financial time series effectively, further supporting their widespread use in forecasting tasks [18].
Figure 1 represents the internal workings of an LSTM cell, illustrating the flow of information through various gates and operations, including the forget gate, input gate, and output gate, which collectively determine the cell's ability to retain or discard information over time [19].

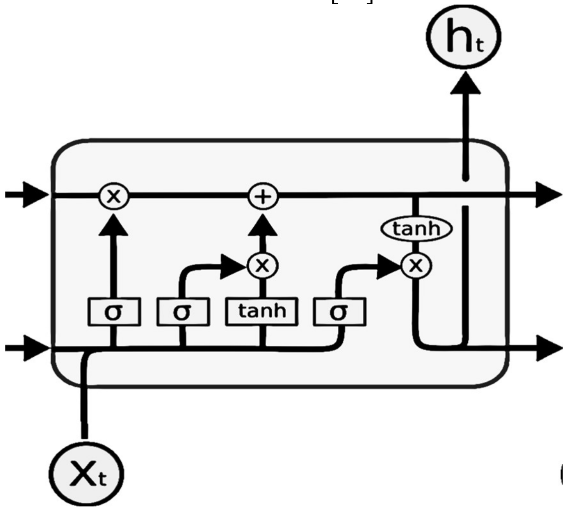

Figure 1: Components of an LSTM

Here's a breakdown of its components and how the LSTM functions:

- Input (X_t): The current input at time step t, denoted as X_t, enters the LSTM cell alongside the previous hidden state and cell state.
- Forget Gate (σ): The forget gate determines which parts of the previous cell state (C_{t-1}) should be kept or forgotten. It uses a sigmoid function (σ), outputting a value between 0 and 1. If it outputs 1, all information is kept, and if it outputs 0, all information is forgotten. Mathematically, the output is:

$$f_t = \sigma(W_f \cdot [h_{t-1}, X_t] + b_f) \quad (1)$$

- Input Gate (σ): The input gate controls how much of the new information (proposed by the tanh activation) should be added to the cell state. It consists of a sigmoid function (σ) to decide which values to update, and a tanh activation to generate candidate values to update the state:

$$i_t = \sigma(W_i \cdot [h_{t-1}, X_t] + b_i) \quad (2)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, X_t] + b_C) \quad (3)$$

- Cell State Update: The cell state is updated by combining the previous state (C_{t-1}) and the new candidate state ($\tilde{C}_t$) controlled by the forget gate and input gate:

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (4)$$

This ensures that the LSTM retains long-term information and incorporates new relevant data.

- Output Gate (σ): The output gate controls the final output of the LSTM cell. It uses a sigmoid function to determine how much of the cell state should be passed through the hidden state. The cell state is passed through a tanh activation to normalize it between -1 and 1 before multiplying by the output gate's value:

$$o_t = \sigma(W_o \cdot [h_{t-1}, X_t] + b_o) \quad (5)$$

$$h_t = o_t * \tanh C_t \quad (6)$$

- Hidden State (h_t): The hidden state is passed to the next time step and serves as the output of the current LSTM unit.

Functionality:
Long-Term Memory: The LSTM's main feature is its ability to preserve information over long sequences through the cell state, which is updated selectively by the forget gate and input gate.
Selective Updates: By controlling the flow of information via gates, the LSTM decides what to remember, forget, and output, making it highly effective for tasks where long-term dependencies are important (e.g., time series, language modeling).
Non-Linearity: The use of sigmoid (σ) and tanh activations introduces non-linearity, helping the network capture more complex patterns in sequential data.
Overall, this diagram shows how LSTMs manage both short-term and long-term information, ensuring that relevant data is retained across different time

steps while discarding irrelevant or outdated information.

2.4 Comparison of Hyperparameter Optimization Methods

Numerous studies have compared various hyperparameter optimization strategies, including grid search, random search, Bayesian optimization, and gradient-based methods. Previous research highlighted the inefficiencies of grid search and random search in high-dimensional spaces. These comparisons underscore the need for adaptive and efficient hyperparameter tuning approaches.

3. METHODOLOGY

This section outlines the methodology for developing and evaluating our adaptive hyperparameter tuning system for LSTM models. Key steps include data preprocessing, dynamic hyperparameter optimization using a generator, and iterative refinement based on model performance. The LSTM model's architecture, training process, and techniques like early stopping are also detailed to ensure robust and efficient time series forecasting.

3.1 Data

For this research, the primary focus is on the historical Close Price data of the S&P 500 index, sourced from Yahoo Finance via the *yfinance* library. The Close Price is a critical metric in financial time series forecasting, representing the final trading price of the index at the end of each trading day. This price is widely regarded as the most significant daily price, reflecting the market's consensus on the value of the index after all trading activities have concluded. Unlike other daily price metrics (such as open, high, or low prices), the Close Price is often used by investors, analysts, and traders to assess market trends and make informed decisions. It serves as a key indicator in various technical analysis tools and is commonly used in the calculation of moving averages, oscillators, and other financial indicators that guide trading strategies and portfolio management decisions.

The data collection process captures the maximum available historical Close Price data for the S&P 500 index, spanning from the earliest records to the most recent date. This extensive dataset provides a rich historical context, covering a wide range of market conditions, economic cycles, and significant financial events. By utilizing the full extent of the available data, the research ensures that the forecasting models can learn from a diverse set of market behaviors, including long-term trends, short-term fluctuations, and rare market events that include critical periods of financial history, such as:

- The Great Depression (1929-1939): A major economic downturn that had a profound impact on global financial markets.
- Post-War Economic Expansion (1945-1960s): A period marked by sustained economic growth and market development.
- The Oil Crisis (1973-1974): A period of economic and market instability triggered by geopolitical events and oil embargoes.
- The Dot-com Bubble (2000-2002): A period of speculative investment in internet-related companies, followed by a significant market correction.
- The Global Financial Crisis (2007-2009): A severe financial crisis that led to dramatic declines in global stock markets.
- The Long Bull Market (2009-2020): An extended period of market growth and recovery following the financial crisis.
- COVID-19 Pandemic (2020): A global health crisis that caused unprecedented market volatility and a rapid economic downturn, followed by recovery.
- Recent Market Trends (2021-Present): Including post-pandemic recovery, inflation concerns, interest rate changes, and geopolitical events.

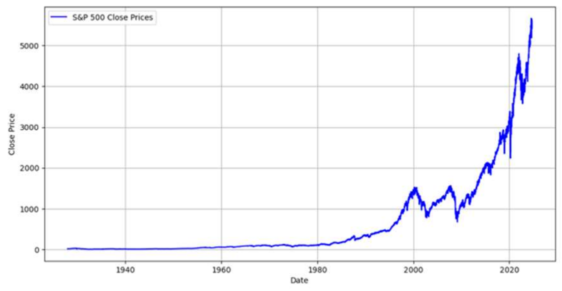

Figure 2: S&P 500 Historical Close Prices

3.1.1 Data Normalization

Normalization is the process of scaling the data to a specific range, typically between 0 and 1. This step is particularly important in machine learning models that are sensitive to the scale of the input data, such as neural networks [20]. It ensures that all input features contribute equally to the model, preventing any single feature from disproportionately influencing the model's predictions. In time series forecasting, normalization helps stabilize the learning process by reducing the impact of varying scales in the data, which in turn accelerates the training process and improves convergence [21].

Normalization should be applied to the entire dataset before splitting into training and testing sets to avoid data leakage and overfitting. This practice ensures that the model does not have access to information from the test set during training, which could bias the evaluation results and lead to overly optimistic performance estimates [22]. Once the data is normalized, it is ready for input into the machine learning models, helping maintain consistency and integrity across the training and evaluation phases.

MinMax Scaling: MinMax scaling transforms the data by scaling each value in the time series to a specified range, typically [0, 1]. The formula for MinMax scaling is as follows:

$$Xscaled = \frac{X - Xmin}{Xmax - Xmin}$$

Where:

- X is the original data value.
- Xmin is the minimum value in the dataset.
- Xmax is the maximum value in the dataset.
- Xscaled is the normalized data value.

MinMax is particularly useful when the data has known bounds, as it preserves the relationships between data points while transforming the values into a standardized range. This makes it easier for the model to learn from the data, as all input features are on a comparable scale.

Inverse Transformation: After the model makes predictions, it is often necessary to convert the normalized predictions back to the original scale. This is done using the inverse transformation of the MinMax scaler which ensures that the predicted Close Prices are presented in their original scale, making them interpretable and actionable for financial analysis.

$$Xoriginal = Xscaled \times (Xmax - Xmin) + Xmin$$

This step is crucial for evaluating the model's performance in real-world terms, such as calculating absolute errors or comparing predicted prices to actual market prices.

3.1.2 Sequence Creation

Once the data is normalized, the next step is to prepare it for input into the forecasting models. This involves structuring the time series data in a way that is compatible with the model's requirements.

In time series forecasting, especially when using models like LSTM networks, it is necessary to create sequences of data points that the model can use to learn temporal patterns. This process involves segmenting the normalized time series into overlapping sequences, where each sequence contains a fixed number of past observations (often referred to as the "look-back period") that the model will use to predict the next value.

For example, if the look-back period is set to 30 days, the model will use the Close Prices from the previous 30 days to predict the Close Price on the 31st day. This process is repeated for the entire dataset, creating a large number of training examples that capture the sequential dependencies in the data.

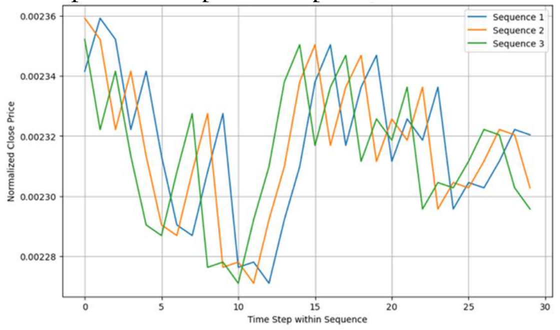

Figure 3: First 3 Sequences of S&P 500 Close Prices

The plot shows three sequences of normalized close prices, each with a sequence length of 30 days. These sequences represent chunks of the time series data that the model will use as inputs to predict the next day's close price. We observe the following:

- ✓ Overlapping Sequences: The sequences overlap because they are created by sliding a fixed-length window (30 days) across the time series. This approach ensures that the model is exposed to as much data as possible, even though the sequences overlap.
- ✓ Trends and Patterns: The sequences capture both upward and downward trends in the S&P 500 close prices. This is important because it allows the model to learn from various patterns in the data (e.g., rising and falling prices over time).
- ✓ Normalization: The close prices are normalized between 0 and 1, which helps the model train more efficiently by removing the influence of absolute price scales. Normalization also ensures that the model treats each sequence comparably, regardless of the actual price values.
- ✓ Variability: Although the three sequences start at different points in the data, they both capture a smooth, continuous trend of price movement. This variability in sequences is essential for training models that generalize well across different patterns.

In a practical sense, each sequence would be followed by a target value (the price on day 31) that the model will try to predict. This approach is typical in time series forecasting to learn from historical data and predict future values.

3.1.3 Data Splitting

The final step in data processing is to split the data into training and testing sets. The training set is used to train the model, while the testing set is reserved for evaluating the model's performance. It is important to ensure that the test set contains data that the model has not seen during training, as this provides an unbiased assessment of the model's ability to generalize to new, unseen data [23] [22]. This practice helps prevent data leakage, which can occur if information from the test set is inadvertently used during training, leading to artificially high-performance scores that do not reflect the model's true predictive capabilities [24]. Properly splitting the data ensures a fair and realistic evaluation, allowing for a more accurate understanding of how well the model performs on real-world data.

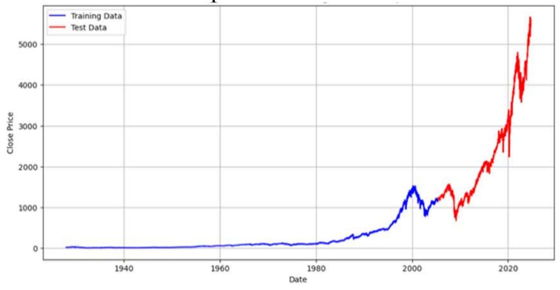

Figure 4: Training and Test Data Split

Typically, the data is split chronologically, with the earlier portion of the time series used for training (80%) and the later portion reserved for testing (20%). This approach mimics real-world forecasting scenarios, where future prices are predicted based on past data.

3.2 Hyperparameter Space

The hyperparameters space in the context of the provided model is a crucial component of the model design, determining the configuration of the LSTM network and influencing its performance in predicting the Close Price. Hyperparameters differ from model parameters in that they are not learned during the training process; instead, they must be set before training begins. The effectiveness of the model heavily depends on the careful selection and optimization of these hyperparameters.

In this system, the hyperparameters are dynamically generated and optimized by the Hyperparameter Generator. The generator explores a defined space of possible hyperparameter values, adjusting them iteratively based on the model's performance. Here, we break down the specific hyperparameters involved, their ranges, and their significance in the model's performance.

Table 1: Hyperparameter Space

Hyperparameter	Range/Values
Number of Layers	1 to 10 (Integers)
Number of Units	1 to 10 (Integers)
Learning Rate	10^{-5} to 0.1 (Logarithmic Scale)
Batch Size	16 to 2048 (Powers of 2)
Dropout Rate	0.1 to 0.9 (Increments of 0.1)
Sequence Length	1 to 250

Table 2: Description and Significance of Hyperparameters

Hyper-parameter	Description	Significance
Number of Layers	Number of LSTM layers in the model. Each LSTM layer adds depth to the network, enabling it to capture more complex patterns in the time series data.	**Fewer Layers:** - Less complexity - Underfitting.
		More Layers: - More complex - Risk of overfitting - More computational resources - Longer training time.
Number of Units Per Layer	specifies the number of units (neurons) in each LSTM layer. Each unit processes the input data and passes the information through the network.	**Fewer Units:** The model may struggle to learn the underlying patterns in the data, potentially leading to underfitting.
		More Units: Increase the model's capacity to learn and capture more details from the input sequences. increases the risk of overfitting and the computational complexity of the model.
Learning Rate	The learning rate controls the size of the steps the optimizer takes when updating the model's weights during training. It is a critical hyperparameter that influences the speed and stability of the training process.	**Low Learning Rate:** Makes the training process more stable but slower. It may help in fine-tuning the model, but if too low, it can result in the model getting stuck in local minima.
		High Learning Rate: Can speed up training but at the risk of overshooting the optimal solution, leading to divergence or suboptimal performance.

Batch Size	The batch size defines the number of samples processed before the model's internal parameters are updated. It affects the model's training stability and the convergence speed.	**Small Batch Size:** Introduces more noise into the gradient estimation, which can help the model escape local minima but may also result in slower convergence.
		Large Batch Size: Provides more accurate gradient estimates, leading to smoother convergence. However, requires more memory and computational resources and may result in the model getting stuck in local minima.
Dropout Rate	Dropout is a regularization technique used to prevent overfitting in neural networks. During training, dropout randomly sets a fraction of input units to zero at each update, which helps the model generalize better by preventing it from relying too heavily on any one feature.	**Low Dropout Rate:** More units are active during training, which can lead to better performance on the training data but may increase the risk of overfitting.
		High Dropout Rate: Forces the model to learn more robust features by relying on a smaller set of units at each update, reducing overfitting but potentially making the training process less efficient.
Sequence Length	The sequence length determines the number of past time steps (Close Prices) used as input to the model to predict the next time step. This hyperparameter is crucial for capturing temporal dependencies in the data.	**Short Sequences:** May result in the model missing long-term dependencies, which can be critical in financial time series forecasting.
		Long Sequences: Provide more context and can help capture long-term trends. However, they increase the model's complexity and the risk of overfitting, especially if the sequences include noisy data.

Number of Epochs
The number of epochs refers to the number of times the entire training dataset passes through the model during training. Each epoch consists of several iterations, determined by the batch size. In this experiment, the number of epochs is fixed at 10 rather than treated as a tunable hyperparameter. Research has shown that increasing the number of epochs does not necessarily improve training quality; while more epochs give the model additional opportunities to learn, they can also lead to overfitting, where the model performs well on training data but struggles to generalize to unseen data [22] [25]. Moreover, treating the number of epochs as a hyperparameter introduces disparities in computational consumption and processing time, potentially skewing model comparison results [26]. To ensure fair comparisons and consistent resource usage, the number of epochs is kept fixed throughout the study.
The hyperparameter space in this model is designed to allow extensive exploration of different configurations, with ranges carefully chosen to balance the model's capacity to learn complex patterns and its ability to generalize to new data.

3.3 Model Architecture and Components

The model structure revolves around a two-part system: the Hyperparameter Generator and the LSTM Model. These components work together to dynamically generate and optimize hyperparameters for training an LSTM network aimed at predicting the Close Price in our time series data.

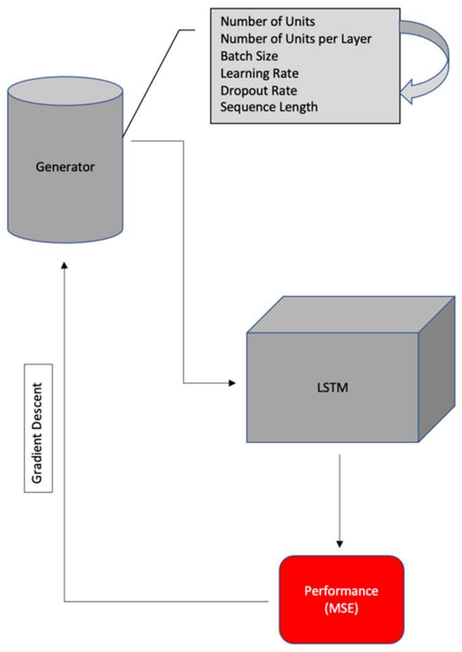

Figure 5: Gradient-Based Model Architecture

The Generator explores the Hyperparameters Space dynamically, iteratively adjusting the hyperparameters based on the model's performance. Initially, hyperparameters are selected randomly within the specified ranges. As the training progresses, the generator uses feedback from the model's performance (specifically the Mean Squared Error, MSE) to refine the selection of

hyperparameters. This adaptive approach allows the model to converge towards an optimal set of hyperparameters that improve its predictive accuracy.
Initial Randomization: The generator starts with a randomly selected set of hyperparameters to explore the space broadly. This initial randomness ensures that the model does not get stuck in suboptimal regions of the hyperparameter space.
Iterative Refinement: As training progresses, the generator refines the hyperparameters by using the MSE as a performance metric. Poor-performing configurations are adjusted in subsequent iterations, while better-performing configurations guide the search in future iterations.
Scalability of Hyperparameters: The scaling functions applied to raw outputs from the generator ensure that the generated hyperparameters are within practical and meaningful ranges. This scaling is crucial for maintaining the balance between exploration and exploitation in the hyperparameter space.

3.3.1 The Generator

As shown in Figure 6, The generator is a feedforward neural network built using TensorFlow's Sequential API, consisting of fully connected (dense) layers:

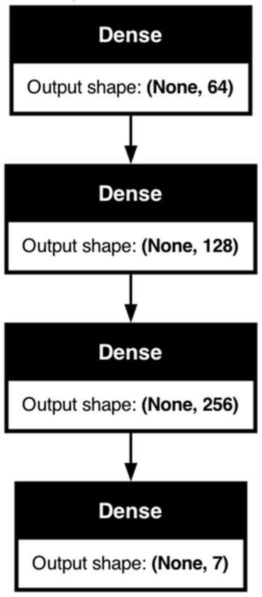

Figure 6: Generator Architecture

The architecture displayed in the diagram represents the Hyperparameter Generator. Each layer processes input data, transforming it through a series of nonlinear operations to generate a set of raw hyperparameters used to configure and train the LSTM model. Below is a detailed explanation of the architecture and the rationale behind its design.

Table 3: Generator Layer-by-Layer Breakdown

Layer	Units	Activation Function	Purpose
Input Layer	-	-	Expects an 11-dimensional input vector, including noise and potentially transformed MSE values from previous iterations. This forms the basis for generating hyperparameters.
First Dense Layer	64	ReLu	Initial transformation of the input data. Captures and processes a moderately complex set of features.
Second Dense Layer	128	ReLu	Expands the model's capacity by learning more complex representations of the input data.
Third Dense Layer	256	ReLu	Deepens the network, increasing representational power to capture intricate patterns in the input data for generating accurate hyperparameters.
Output Layer	7	None (Linear)	Outputs a 7-dimensional vector representing raw hyperparameters, including layers, units, learning rate, batch size, dropout rate, and sequence length. No activation function.

Rationale Behind the Architecture:
Progressive Increase in Units: The architecture uses progressively larger numbers of units (64 → 128 → 256) as the network deepens. This design choice allows the model to gradually expand its capacity to capture complex interactions within the input data, leading to more refined and informed hyperparameter generation. Increasing the number of units in deeper layers has been shown to help in learning hierarchical features, particularly in complex tasks where higher layers capture more abstract representations [22] [26].
Depth of the Network: The choice of three hidden layers strikes a balance between model complexity and computational efficiency. It is deep enough to capture complex relationships but not so deep as to cause issues like overfitting or excessive training time, which can be exacerbated in deeper architectures [27]. This depth is suitable for tasks such as hyperparameter generation.
ReLU Activation Function: ReLU is chosen for the hidden layers because it introduces non-linearity, enabling the model to learn more complex patterns

than a purely linear model. ReLU is computationally efficient, as it does not require expensive operations like exponentiation or division, and it has become a standard choice in deep learning architectures due to its ability to mitigate the vanishing gradient problem [28] [29].

Linear Output Layer: The output layer does not use an activation function, allowing the network to produce raw hyperparameter values that can be directly scaled to the appropriate ranges. This approach is crucial because the final values must be within specific limits to be valid hyperparameters for training the LSTM model. Using a linear output is appropriate in regression tasks where the range of predicted values is not constrained [22].

Scalability and Flexibility: The architecture is scalable and flexible, capable of being adjusted or extended if necessary to accommodate more input features or generate a broader range of hyperparameters. This flexibility is essential in a research context where the optimal network configuration may not be known a priori. Modular architectures allow researchers to experiment with various configurations without having to redesign the entire model structure [30].

Generation of Initial Hyperparameters: Initially, the generator produces hyperparameters based on random inputs, which are passed through the network to generate an initial set. This random initialization helps explore different configurations before convergence on an optimal set [31].

Adaptation and Learning: After each iteration, the generator adapts based on the performance of the model (measured by Mean Squared Error, or MSE). The generator uses the MSE as feedback to adjust its internal weights, effectively learning which hyperparameters lead to better model performance. The use of a feedback loop in hyperparameter optimization allows for dynamic adjustments and continuous improvement [15] [5].

Optimizer: The generator model is optimized using the Adam optimizer with a learning rate of 0.1. Adam is a popular choice in neural network optimization due to its adaptive learning rate and ability to handle sparse gradients [32]. The optimizer updates the weights of the generator based on gradients calculated from the loss, which is influenced by the MSE.

3.3.2 The LSTM

The LSTM Model is the actual predictive model that utilizes the hyperparameters generated by the Hyperparameter Generator. The LSTM model is a sequential neural network built using TensorFlow's Sequential API, comprising multiple LSTM layers followed by a Dense output layer.

Table 4: LSTM Layer-by-Layer Breakdown

Layer	Units/Parameters	Purpose
Input Layer	(sequence_length, 1)	Defines the input shape, where sequence_length is a hyperparameter and 1 indicates a single feature (Close Price).
LSTM Layers	Determined by "layers" and 'units"	Number of LSTM layers is determined by the "layers" hyperparameter. The "units" hyperparameter specifies the LSTM units.
Dropout Layers	Controlled by "dropout_rate"	Dropout layers are applied after each LSTM layer to prevent overfitting, with dropout rate controlled by a hyperparameter.
Output Layer	1	Final Dense layer with a single unit, corresponding to the prediction of the next Close Price in the sequence.

Compilation: The model is compiled using the Adam optimizer with a loss function of mean squared error (MSE). The learning rate for the optimizer is dynamically set based on the generated hyperparameters.

Training Process: The LSTM model is trained on sequences of historical Close Prices, with the length of each sequence determined by the "sequence length" hyperparameter from the Generator. The model is trained using a specified "batch_size" and for a fixed number of epochs. During training, the model learns to predict the next Close Price based on the previous "sequence_length" number of Close Prices.

Evaluation: After training, the model's performance is evaluated on a validation set, with MSE used as the primary metric. The result of this evaluation informs the Hyperparameter Generator for the next iteration.

3.4 Early Stopping

When running the Model for multiple iterations, we notice that it doesn't necessarily converge as illustrated in the figure below, we notice that the Error value (MSE) jumps between local minima. The first solution that comes to mind is to adjust the Generator's learning rate. However, this does not fix the convergence problem as it is related to the randomness within the data itself that mimics the complexities related to the stock market.

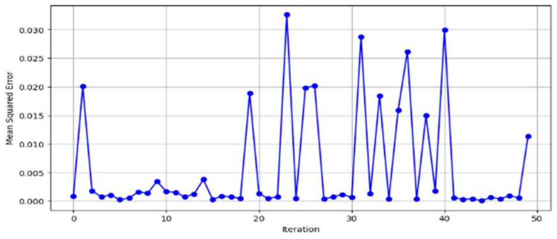
Figure 7: Evolution of MSE Over Iterations

To better understand how the Model functions, we applied a smoothing the observed MSE values from the LSTM predictions, and we came to the conclusion that the Model searches for the lowest MSE values in terms of cycles. In other words, the Model succeeds to find the local minima only to shoot upwards in the quest of other minima that might or might not be smaller than the previous ones.

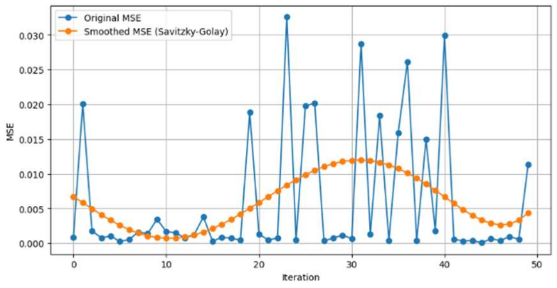
Figure 8: Comparison of Original vs Smoothed MSE

By increasing the number of iterations, this phenomenon can be observed clearly. However, what caught our attention is that regardless of how many times the experiment is repeated, the Model succeeds to find the smallest MSE within the first 20 iterations. Thus, we decided to implement an early stopping mechanism for our model after that number and then retrieve the lowest MSE value and the related set of Hyperparameters, being the combination that yielded the best results.
Even if the Model could hypothetically succeed to find a lower MSE after a larger number of iterations, by then, the wasted time and computational power would be insignificant against the error difference.

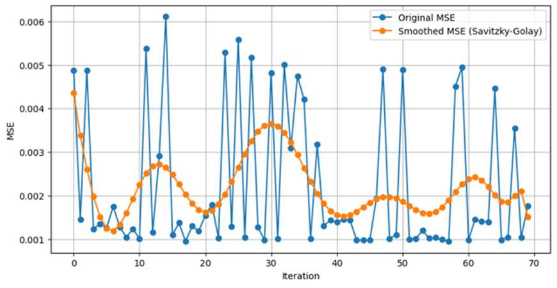
Figure 9: Extended Comparison of Original vs Smoothed MSE

4. EVALUATION

4.1 Generator Model Results

In this section we outline the results obtained by our model, to which we will refer to in what follows as the Generator Model or The Base Model.

4.1.1 Error Analysis (MSE)

Figure 10 depicts the evolution of MSE over iterations. Each value corresponds to the performance of a combination of hyperparameters from table 8 (See Appendices)

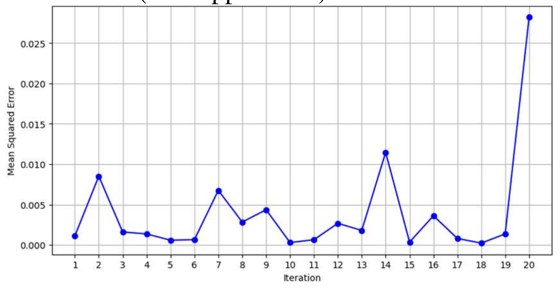
Figure 10: Evolution of MSE over 20 iterations

1. Overall Trend: The MSE remains relatively low and stable throughout the majority of the iterations, with occasional spikes in certain iterations.
2. Spikes in MSE: Noticeable spikes occur at iterations 2, 7, 13, and 20, where the MSE values are significantly higher than the surrounding iterations. These spikes might indicate moments when the model's predictions diverged more significantly from the actual values.
3. Low MSE Points: Many iterations (e.g., 1, 5, 6, 10, 16, 18) show very low MSE values, suggesting that the model was performing well in these iterations.
4. Final Spike: A particularly high spike is seen at iteration 20 (last iteration), where the MSE reaches its peak. This indicates the beginning of a new search cycle.

Smoothing the MSE values gives another insight on the functioning of the Base Model and plots the search cycles as expected

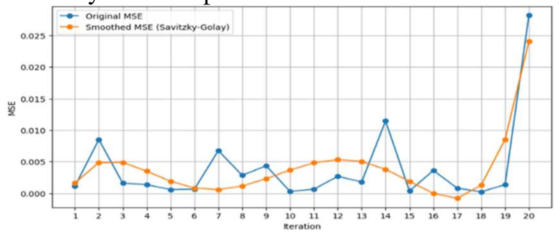
Figure 11: Comparison of Original vs Smoothed MSE over 20 Iterations

4.1.2 Best Hyperparameters

Table 5 shows the set of LSTM hyperparameters that resulted in the lowest MSE value. In what follows, we will load this particular LSTM with these specific configurations for further analysis.

Table 5: Generator Model Best Hyperparameters

Iteration	18
Number of Layers	1
Units Per Layer	2
Learning Rate	0.00001
Batch Size	16
Sequence Length	1
Dropout Rate	0.1
MSE	0.000229
Time (Seconds)	4.043066

4.1.3 Predictions

Figure 12 shows a comparison between the Actual Close Prices and the Predicted Close Prices (over the course of about one year, of the S&P 500 index:

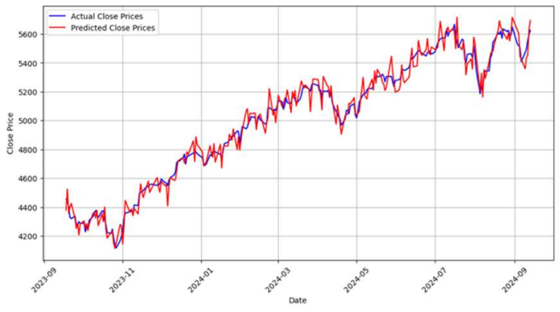

Figure 12: Actual vs Predicted Close Prices over a year

1. Good Fit: Overall, the predicted prices closely follow the actual prices, especially during the major trends. This indicates that the model captures the general direction of the market quite well.
2. Minor Deviations: While the model does a good job of following the actual prices, there are some instances of small discrepancies, particularly during short-term fluctuations or corrections. These deviations could be areas where the model might be slightly lagging behind or overreacting to short-term movements.
3. Trending Phases: During the upward trends, such as between January 2024 and May 2024, the predictions track the actual price movements very closely, which suggests the model is effective at predicting larger, sustained trends.
4. Increased Variation at Peaks: There are moments, especially at price peaks, where the predicted prices deviate slightly more from the actual values. For example, around July 2024, the predicted prices overestimate the peak, and around September 2024, the model underestimates a local price drop. This suggests some challenges in accurately predicting turning points.

For more visibility, we diminished the number of datapoints to 100 in figure 13, which confirms the fact that the model is decently following trends and predicting price direction

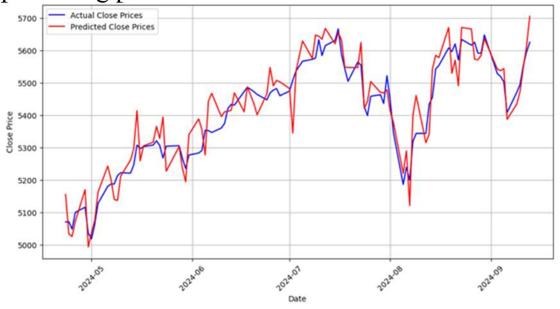

Figure 13: Actual vs Predicted Close Prices (Six Months)

4.1.4 Residuals

The residual plot shows the distribution of errors (residuals) between the actual and predicted values from our model.

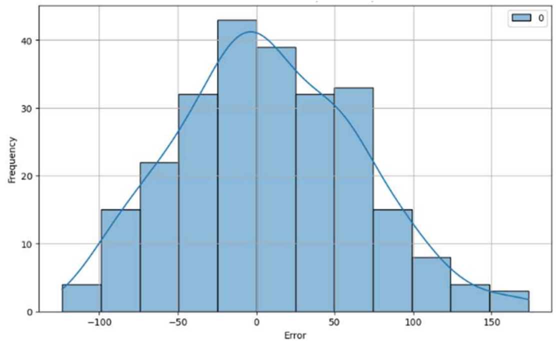

Figure 14: Error Distribution (Residuals)

Here's a breakdown of key observations:

Symmetry and Distribution: The residuals appear to follow a roughly symmetrical distribution, with a peak around 0. This indicates that the model has relatively balanced errors, with an equal likelihood of overestimating or underestimating the actual values. The distribution is somewhat bell-shaped, which is common in regression tasks and suggests the errors may follow a normal distribution to some extent, although it's not perfectly centered around 0.
Skewness: There is a slight right skew in the distribution. This means that there are more cases where the model significantly underestimates (negative residuals) compared to overestimating (positive residuals). The tail on the right-hand side (positive errors) is a bit longer, which suggests the

model sometimes significantly under-predicts the actual values. Error Magnitude: Most residuals are concentrated between -50 and 50, which shows that for the majority of the data points, the model's predictions are reasonably close to the actual values. However, there are a number of instances where the residuals are as large as 100 or more, suggesting that in some cases, the model's predictions deviate significantly from the actual values.
Bias: Since the peak of the distribution is slightly above 0, this may suggest that the model slightly underestimates the true values more frequently than it overestimates.
Conclusion: While the residual distribution looks somewhat normal, the presence of some large residuals and a slight skew indicates that the model might not perfectly capture the relationship between the features and the target variable. There could be room for improvement in hyperparameter tuning or model architecture to reduce the occurrence of large errors, particularly in the tail of the distribution.

4.2 Bayesian Optimization Results

4.2.1 Early stopping

Before we delve into results, it is important to note that running Bayesian Optimization for this problem and for the suggested hyperparameter space is a task that could keep running until all the possibilities of every single set is compiled before retrieving the most performing Hyperparameters and related MSE. this will technically be similar to applying a grid search algorithm or a random search until we run out of all possible combinations. Thus, it is highly important to establish an early stopping condition for Bayesian optimization.
Condition: The algorithm will stop its search once we reach an MSE value equal or inferior to the best MSE achieved by the Generator Model.

4.2.2 Loss (MSE)

Figure 15 shows the evolution of MSE over iterations. The values are extracted from table 9 (See Appendices) that shows in detail which set of Hyperparameters yielded which MSE value. As the figure shows, the process took 28 iterations.

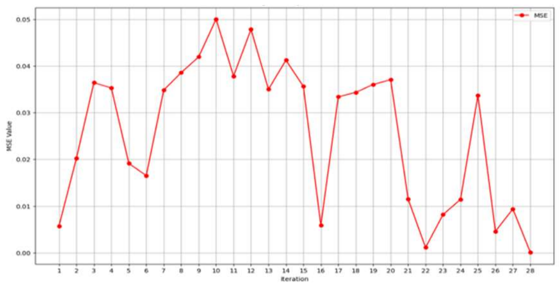

Figure 15: Evolution of Bayesian Optimization MSE over Iterations

Key observations:

1. Initial Improvement: There is an initial decline in MSE after the first iteration, suggesting that the model made quick adjustments that led to some early improvement by iteration 5.
2. Oscillations: The MSE fluctuates significantly across the iterations, with several peaks (e.g., iterations 3, 8, 14, and 20) and valleys (e.g., iterations 6, 16, and 22). This kind of oscillation is characteristic of Bayesian optimization as it explores different regions of the hyperparameter space, testing both well-performing and exploratory configurations.
3. Highest Peak: The highest MSE occurs around iteration 9, suggesting a hyperparameter set that led to a poor fit. Following this, there are some recovery iterations, but the MSE never settles consistently at lower levels for an extended period.
4. Sharp Decline at Iteration 16: Iteration 16 shows a significant drop in MSE, which could represent a promising set of hyperparameters that drastically improved performance. However, subsequent iterations increase the MSE again, indicating the model continues to explore beyond this set of hyperparameters.
5. Late Iteration Variability: The latter part of the graph shows a continued pattern of sharp increases and decreases in MSE, particularly around iterations 20 and 25. The steep declines, such as those in iterations 16 and 22, suggest that the model occasionally stumbles upon more favorable hyperparameters, but these are not consistently maintained in the search.
6. General Trend: While the optimization does lead to some low MSE values (e.g., iterations 6, 16, and 22), there is no clear downward trend over time. This suggests that the search is still in an exploratory phase, where it hasn't fully converged on an optimal set of hyperparameters, again due to the complexity of Data.

4.2.3 Best Hyperparameters

After stopping early at iteration 28, Bayesian optimization has yielded the following results:

Table 6: Bayesian Optimization Best Hyperparameters

Iteration	28
Number of Layers	4
Units Per Layer	4
Learning Rate	0.01
Batch Size	16
Sequence Length	34
Dropout Rate	0.1

MSE	0.0001
Time (Seconds)	1754.39

The best hyperparameters are extracted from Table 9 (Appendices) which contains full results.
In the following sections, we will dive deeper into comparing the results above with the results from our custom gradient-based model.

4.3 Generator Model vs Bayesian Optimization

4.3.1 Loss Comparison

Figure 16 proves that The Generator model clearly performs more consistently compared to the Bayesian Optimization model, as reflected in the lower and more stable MSE values. This suggests that the Generator model is more effective at keeping MSE low over iterations.
On the other hand, the Bayesian Optimization model, while showing more exploration and potential to hit lower MSE values (e.g., iterations 16 and 22), suffers from greater instability. We also notice that after the 20th iteration, the latter took an extra 8 iterations to meet the early stopping condition.

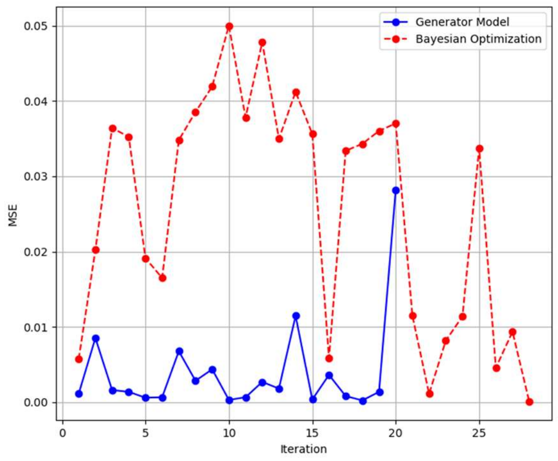

Figure 16: Comparison of the Evolution of MSE Over Iterations (Base Model vs Bayesian Optimization)

Generator Model: The Generator model's smoother and more controlled MSE evolution implies it is either better tuned or more focused in its optimization, leading to consistent performance.
Bayesian Optimization Model: The Bayesian model may benefit from more fine-tuning, or a more refined search space, to stabilize its performance and reduce the number of high MSE iterations.

4.3.2 Time Comparison

Figure 17 compares the execution time (in seconds) between our Base Model (Generator) and the Bayesian Optimization Model.

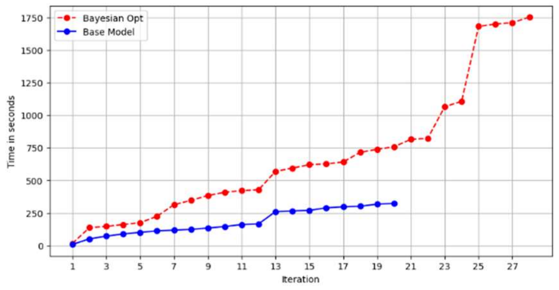

Figure 17: Comparison of Execution Time between The Base Model and Bayesian Optimization over Iterations

Key Observations:

Consistent Growth for Both Models:
The execution time for both models increases steadily as the iterations progress. However, the rate of increase differs significantly between the two models.
Execution Time for the Base Model:
The Base Model has a consistent, relatively slow increase in execution time, remaining under 500 seconds even at iteration 28. This suggests that the computational cost of each iteration in the Base Model remains fairly constant, which is likely due to a more streamlined or less complex, yet efficient process.
Execution Time for the Bayesian Optimization Model:
The Bayesian Optimization model shows a much steeper increase in execution time, especially after iteration 20, where the time jumps significantly, reaching nearly 1750 seconds by iteration 28.
This sharp rise implies that the Bayesian Optimization process becomes increasingly computationally expensive over time. This could be due to the more complex nature of the Bayesian approach, which requires more resources to evaluate and explore hyperparameter space as it narrows down the optimization.
Gap Between Models:
The gap between the two models' execution times widens dramatically after iteration 15. Initially, the difference is modest, but from iteration 20 onward, the Bayesian Optimization model takes exponentially more time per iteration compared to the Base Model.

Summary:

Efficiency of the Base Model: The Base Model is far more efficient in terms of execution time, making it a better choice when time is a critical factor.
Computational Intensity of Bayesian Optimization: The Bayesian Optimization model, while potentially more powerful in its search for optimal hyperparameters, becomes increasingly resource-

intensive, which could make it impractical for longer training processes or larger datasets.

4.3.3 Additional Metrics

In this section, we will study additional performance metrics, including:

- R-Squared (R^2),
- Root Mean Squared Error (RMSE),
- Mean Absolute Error (MAE),

to comprehensively evaluate the predictive accuracy of the Generator Model and the Bayesian Optimization Model. These metrics will provide deeper insights into how well each model captures the underlying patterns in the data, how effectively they minimize errors, and how consistently they perform across iterations. By analyzing these metrics, we aim to better understand the strengths and weaknesses of each model in terms of both precision and reliability

4.3.3.1 R^2

Figure 18 displays the R-Squared (R^2) evolution over iterations for the two models. The R-Squared metric measures the goodness of fit, with values closer to 1 indicating a better fit between the predicted and actual values.

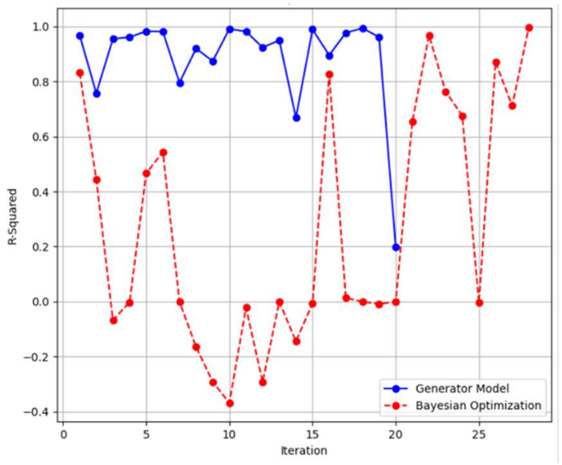

Figure 18: Comparison of the Evolution of R-Squared over Iterations (Bayesian Opt vs Generator Model)

<u>Generator Model Performance:</u>

The Generator Model shows consistently high R^2 values across most iterations, hovering around 0.8 to 1.0, indicating that this model provides a very good fit for the data.

There is a brief dip around iteration 20, where the R^2 drops sharply, but the model quickly recovers. Overall, the model remains highly reliable in terms of its predictive accuracy throughout the iterations.

<u>Bayesian Optimization Model Performance:</u>

The Bayesian Optimization Model demonstrates far more variability in R^2 values, ranging from positive values around 0.8 to negative values (as low as -0.4). Early iterations show a sharp decline, with R^2 falling to negative values by iteration 4, which indicates that the model is performing worse than simply predicting the mean of the data at certain points.

There are some moments of recovery (e.g., iterations 7, 14, and 23), where the R^2 rises to around 0.6, but these are not sustained, as the model frequently dips back into negative or low R^2 values.

The fluctuations highlight the exploratory nature of Bayesian Optimization, where the search for optimal hyperparameters can result in poor fits before finding better configurations.

<u>Comparison:</u>

- The Generator Model clearly outperforms the Bayesian Optimization Model in terms of stability and goodness of fit, as evidenced by its consistently high R^2 values. is more stable and effective at predicting the target variable, making it a more reliable option.
- The Bayesian Optimization Model, while capable of achieving decent R^2 values occasionally, struggles to maintain a good fit over multiple iterations, exhibiting significant volatility which suggests that its hyperparameter search is less focused, leading to inconsistent performance. However, given its potential for recovery in certain iterations, fine-tuning could improve its stability.

The Generator Model delivers consistently high performance in terms of R^2, making it the more reliable choice. The Bayesian Optimization Model shows promise but requires further refinement to avoid the severe fluctuations and consistently produce high R^2 values.

4.3.3.2 Root Mean Squared Error (RMSE)

Figure 19 shows the Root Mean Squared Error (RMSE) evolution over iterations for both models. RMSE is a standard metric used to measure the difference between the predicted values and the actual values, with lower values indicating better model performance.

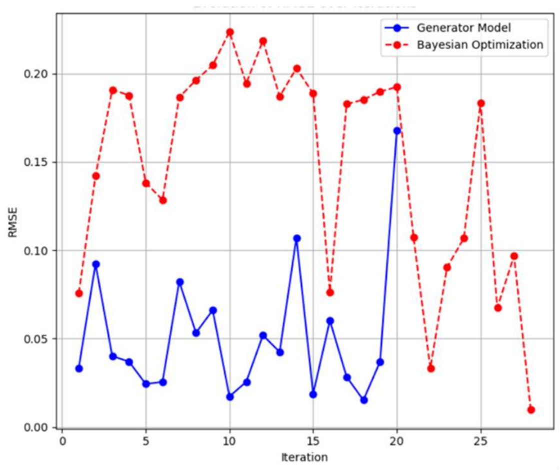

Figure 19: Comparison of the Evolution of RMSE over Iterations (Bayesian Opt vs Generator Model)

<u>Generator Model Performance:</u>
The Generator Model maintains relatively low RMSE throughout most of the iterations, consistently below 0.10. This indicates that the model is making predictions with a low degree of error.
There are a few spikes, particularly around iteration 20, where RMSE jumps significantly. However, these are isolated instances, and the model quickly returns to a lower RMSE.
The stability of the Generator Model's RMSE suggests a reliable performance with limited fluctuations.
<u>Bayesian Optimization Model Performance:</u>
The Bayesian Optimization Model demonstrates much higher variability in RMSE, with values frequently ranging between 0.15 and 0.20. This indicates that the model experiences larger errors in its predictions compared to the Generator Model.
While there are occasional sharp drops in RMSE (e.g., iterations 17 and 22), these improvements are often short-lived, with the RMSE rising again in subsequent iterations.
The overall pattern of the Bayesian Optimization Model shows high instability, with rapid increases and decreases in RMSE, reflecting the exploratory nature of the hyperparameter optimization process.
<u>Comparison:</u>

- The Generator Model outperforms the Bayesian Optimization Model in terms of consistency and lower RMSE values. Its predictions are more accurate and stable across most iterations. It shows better control and optimization, leading to more reliable predictions with lower error.
- The Bayesian Optimization Model, while capable of reducing RMSE in some iterations, struggles with maintaining low error levels, as evidenced by frequent spikes and high RMSE values in many iterations. This suggest that its hyperparameter search is less focused, leading to less consistent performance. The exploration phase seems to produce significant variance in results, both positive and negative.

The Generator Model provides more accurate and stable predictions with lower RMSE throughout the iterations. The Bayesian Optimization Model, while capable of achieving good results in some iterations, lacks the stability and consistency seen in the Generator Model, making it less reliable in terms of predictive accuracy.

4.3.3.3 Mean Absolute Error (MAE)

Figure 20 depicts the Mean Absolute Error (MAE) evolution. MAE is a metric that measures the average absolute difference between the predicted and actual values, with lower values indicating better performance.

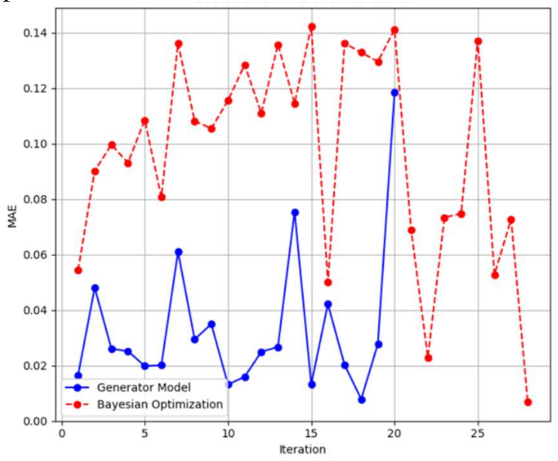

Figure 20: Comparison of the Evolution of MAE over Iterations (Bayesian Opt vs Generator Model)

<u>Generator Model Performance:</u>
The Generator Model maintains relatively low MAE throughout the iterations, mostly under 0.06, indicating that the model produces predictions with smaller deviations from the actual values.
Similar to previous metrics, there are a few spikes (notably around iterations 5, 15, and 20), where the MAE increases. However, these spikes are followed by quick recoveries, showing the model's ability to return to a lower error rate.
Overall, the Generator Model shows a stable performance with controlled fluctuations in MAE.
<u>Bayesian Optimization Model Performance:</u>
The Bayesian Optimization Model exhibits much higher MAE, fluctuating between 0.08 and 0.14 for most iterations. This indicates that its predictions

tend to deviate more from the actual values compared to the Generator Model.
There are noticeable swings in the error values, with frequent sharp rises and falls. This suggests that the Bayesian Optimization process is still exploring various hyperparameter configurations, leading to inconsistent prediction accuracy.
The volatility in MAE, particularly the large spikes around iterations 5, 14, and 23, indicates that the model struggles to maintain a low error rate, despite occasional improvements.
<u>Comparison:</u>

- The Generator Model consistently achieves lower MAE values compared to the Bayesian Optimization Model. The differences in MAE are significant, with the Generator Model maintaining smaller errors throughout most of the iterations. Even when the MAE spikes, the model quickly corrects itself.
- The Bayesian Optimization Model's higher and more erratic MAE suggests that its hyperparameter exploration is less effective in minimizing error compared to the Generator Model, which shows more stability and reliability in its predictions. This indicate that it requires more fine-tuning. Its exploratory nature may be responsible for the unstable performance, but it fails to consistently reduce error.

The Generator Model outperforms the Bayesian Optimization Model in terms of minimizing MAE, providing more accurate and stable predictions across the iterations. While the Bayesian Optimization Model shows potential for improvement in some iterations, it lacks the consistency and efficiency needed for reliable predictions.

4.3.3.4 Metrics Conclusion

In conclusion, across all metrics—MSE, RMSE, R^2, and MAE—the Generator Model consistently outperforms the Bayesian Optimization Model in terms of stability, accuracy, and error minimization. The Generator Model maintains lower and more controlled error values (MSE, RMSE, and MAE), while showing consistently high R^2 values, indicating a better fit between predictions and actual values. In contrast, the Bayesian Optimization Model exhibits greater variability and volatility across all metrics, with frequent spikes and drops, highlighting its exploratory nature but resulting in less reliable performance. While the Bayesian model occasionally reaches lower error values, it struggles to sustain these improvements, making the Generator Model the more effective and stable option for accurate prediction.

4.3.4 Understanding Search Logic

Figure 21 (See Appendices) presents the evolution of several hyperparameters over iterations for both the Base Model and the Bayesian Optimization Model. The plots track the changes in Hyperparameters (Learning Rate, Batch Size, Sequence Length, Dropout Rate, Units Per Layer, and Number of Layers) across iterations. We observe the following:
<u>Learning Rate</u>:

- The Base Model maintains a very low, near-constant learning rate throughout the iterations, indicating a conservative and stable approach to learning.
- The Bayesian Optimization Model, on the other hand, exhibits drastic fluctuations in learning rate, with frequent spikes and drops. This variability reflects its exploratory nature, trying to find the optimal rate, but leading to instability.

<u>Batch Size:</u>

- The Base Model keeps the batch size low and relatively constant, staying under 256 across iterations. This likely contributes to the model's stability and consistency in performance.
- The Bayesian Optimization Model shows extreme fluctuations in batch size, reaching as high as 2048 in some iterations. The large changes suggest that the model is attempting aggressive optimizations but may be overfitting or inefficient in certain configurations.

<u>Sequence Length</u>:

- The Base Model follows a moderate pattern in adjusting sequence length, with some fluctuations but maintaining a more controlled approach.
- The Bayesian Optimization Model again shows significant variability, with sequence lengths fluctuating wildly. This instability could lead to inconsistent performance, as sequence length can greatly affect how the model processes temporal data.

<u>Dropout Rate:</u>

- The Base Model steadily adjusts the dropout rate, staying within a controlled range (between 0.1 and 0.4). This controlled variation helps in regularizing the model without extreme changes.
- The Bayesian Optimization Model experiments with a much broader range of dropout rates, from 0.1 to 0.9,

likely in search of optimal regularization. However, these wide fluctuations can lead to over-regularization or insufficient regularization, impacting model performance.

Units Per Layer:

- The Base Model keeps units per layer relatively low and adjusts them modestly, maintaining a controlled structure that avoids overcomplicating the network.
- The Bayesian Optimization Model again exhibits high variability, frequently adjusting the number of units between 1 and 10. These significant shifts suggest the model is searching for the best architecture but could lead to unstable learning.

Number of Layers:

- The Base Model adjusts the number of layers in a more measured way, with minor changes that seem to follow a pattern, keeping the structure simple.
- The Bayesian Optimization Model varies the number of layers extensively, fluctuating between 1 and 10 layers, which could lead to instability in the model's architecture, as deeper networks are more complex and harder to train effectively.

The Base Model exhibits more controlled, gradual adjustments across all hyperparameters, which likely contributes to its stability and more consistent performance. In contrast, the Bayesian Optimization Model demonstrates large, frequent fluctuations in most hyperparameters, reflecting its exploratory approach. While this may allow it to find occasional optimal configurations, the extreme variability could lead to unstable training and inconsistent results, as reflected in its less stable performance metrics compared to the Base Model. Additionally, compared to prior studies leveraging Bayesian optimization for hyperparameter tuning, our Generator model demonstrates superior stability, particularly in reducing large fluctuations in MSE across iterations. Unlike the approach in [33], which highlights the exploratory advantages of Bayesian optimization, our method achieves consistent performance gains with significantly reduced computational cost, as reflected in the comparative analysis of execution times (Figure 17). This positions our approach as a more resource-efficient alternative for tasks requiring real-time adaptability.

Table 7: Chosen Hyperparameters Comparison (Bayesian Optimization vs Generator Model)

Hyperparameter	**Base Model**	**Bayesian Optimization Model**
Learning Rate	Low and relatively constant; conservative approach.	Large fluctuations; frequent spikes and drops.
Batch Size	Low and stable, generally under 256.	Extreme variability, ranging from small to as high as 2048.
Sequence Length	Moderate changes with controlled fluctuations.	Significant variability, with large jumps and drops.
Dropout Rate	Controlled range between 0.1 and 0.4; gradual adjustments.	Broad range from near 0 to 0.9; unstable variations.
Units Per Layer	Modest adjustments, low and stable.	Large fluctuations, from 1 to 10 units, highly variable.
Number of Layers	Measured adjustments, keeping it simple.	High variability, frequently changing between 1 and 10 layers.

5. DISCUSSION

Ensuring the validity of any machine learning study is critical, especially in fields like financial forecasting, where the implications of predictions can have significant economic impacts. The findings of this study are subject to various threats to validity, categorized into internal, external, and construct validity, each of which is discussed in detail below.

Internal Validity

Internal validity concerns the degree to which the observed outcomes of the model are attributable to the experimental design rather than external factors. In this study, the primary internal validity threat arises from the quality and quantity of the historical S&P 500 data used. While the dataset spans several decades and includes diverse market conditions, it may not fully capture rare or unforeseen market behaviors such as sudden crashes or spikes due to geopolitical events or unexpected crises (e.g., the COVID-19 pandemic). This issue aligns with findings by Sun et al. (2022), who highlighted the need for robust data preprocessing to mitigate the impact of outliers and anomalies in financial datasets [34].

Hyperparameter tuning relies on a feedback loop involving Mean Squared Error (MSE) minimization. Any noise or irregularity in this loop, such as anomalous spikes in MSE due to outlier data points, may lead to suboptimal or misleading model adjustments. Although measures like data normalization and outlier detection are applied, these cannot fully eliminate the risk of bias inherent in the dataset.

External Validity

External validity assesses the generalizability of the findings beyond the scope of the study. While this research focuses on the S&P 500 index, its applicability to other financial indices, such as the NASDAQ or Dow Jones Industrial Average, remains untested. Different indices may exhibit unique behaviors and levels of volatility, making the model's performance unpredictable when applied to new datasets. It is demonstrated that optimization methods tuned for specific datasets often perform poorly on datasets with varying dynamics, such as those influenced by external shocks or structural changes [35]. This underscores the need for validation on diverse datasets, including non-financial time series, to evaluate the robustness of the proposed model.

Similarly, the model's generalizability to non-financial time series datasets, such as energy consumption, weather forecasting, or epidemiological data, is uncertain. These datasets often have distinct characteristics (e.g., periodicity, seasonality) that may require tailored hyperparameter tuning approaches. Without further testing, the broader applicability of the proposed gradient-based optimization system is limited.

Construct Validity

Construct validity pertains to whether the study measures what it claims to measure. This study assumes that minimizing MSE is a sufficient proxy for improving forecasting accuracy and reliability. While MSE is a widely accepted metric in regression tasks, it primarily emphasizes the magnitude of errors without considering their direction or temporal relevance. Zhao (2023) pointed out that reliance on single metrics like MSE can lead to biased evaluations, particularly in domains where directional accuracy is critical, such as stock price predictions [36].

For instance, in financial forecasting, underestimating a significant market movement may have vastly different implications compared to overestimating it. Metrics like Mean Absolute Percentage Error (MAPE) or R-squared could provide complementary perspectives on model performance but are not explored in this study. Additionally, the study's reliance on hyperparameter tuning as the primary driver of improved performance may overlook other critical factors, such as model architecture or the quality of input features. For example, incorporating external macroeconomic indicators or technical analysis signals might enhance the LSTM model's forecasting capabilities beyond what hyperparameter optimization alone can achieve.

Mitigation Strategies

To address these threats, several mitigation strategies have been implemented or proposed:

- **Data Quality Control:** The use of normalization and outlier detection helps reduce the impact of data irregularities. Recent advances, such as data augmentation techniques for rare events, have shown promise in enhancing model robustness to unseen conditions [37].
- **Cross-Dataset Validation:** Testing the model on multiple financial indices or synthetic datasets could provide stronger evidence of generalizability.
- **Diverse Metrics:** Incorporating additional evaluation metrics, such as MAPE or directional accuracy, may offer a more nuanced understanding of model performance.
- **Feature Engineering:** Expanding the model's feature set to include external variables, such as interest rates or market sentiment scores, could enhance its forecasting accuracy and robustness.

While the proposed adaptive hyperparameter tuning system demonstrates promise in improving the efficiency and stability of LSTM models for financial forecasting, these validity considerations highlight areas for caution and further investigation. By addressing these threats in future research, the model's applicability and reliability can be significantly enhanced.

6. CONCLUSION

In this paper, we led a comparison between the Base Model and the Bayesian Optimization Model to reveal significant differences in their performance in terms of stability and efficiency. The Base Model exhibits consistently lower error rates across all metrics, including MSE, RMSE, MAE, and R^2, making it a more reliable model for accurate predictions. Its controlled and stable adjustments to hyperparameters, such as learning rate, batch size, and dropout rate, contribute to its overall robustness and precision. Additionally, the Base Model demonstrates superior time efficiency, with significantly lower execution times per iteration compared to the Bayesian Optimization Model.

On the other hand, the Bayesian Optimization Model, while having the potential to explore a broader hyperparameter space, suffers from greater volatility in both performance and training time. Its frequent and extreme fluctuations in hyperparameters, such as batch size and sequence length, lead to inconsistent error reduction and prolonged training times. While the Bayesian Optimization Model occasionally finds configurations that yield lower error, these moments are short-lived, and the model quickly regresses to higher error values. Overall, the Base Model proves to be more efficient and effective, providing better accuracy and faster results with less computational overhead.

7. FUTURE WORK

Building on the strengths of the Generator model, future research can explore several directions to further enhance its capabilities and address its limitations.

Hybrid Optimization Approaches

One promising direction is the development of a hybrid optimization model that combines the exploration capabilities of Bayesian optimization with the stability and efficiency of the Generator model's gradient-based tuning. Such a hybrid approach could allow for a more focused yet flexible search of hyperparameter spaces, leveraging the strengths of both methods. Recent studies have demonstrated the effectiveness of hybrid strategies in improving convergence and handling complex, high-dimensional spaces [38][39].

Expanding Application Domains

The current study focuses on the S&P 500 index, but extending the model to other financial indices (e.g., NASDAQ, Dow Jones Industrial Average) and non-financial time series datasets (e.g., weather forecasting, energy consumption) could provide valuable insights into its robustness and scalability. Testing on datasets with diverse characteristics, such as seasonality or external shocks, will help generalize the model's applicability across domains [40].

Multi-Objective Optimization

Incorporating multi-objective optimization frameworks to balance competing priorities, such as predictive accuracy and computational efficiency, could further refine the Generator model's utility. This approach would align the model's performance with real-world time and resource constraints [41].

Dynamic and Adaptive Mechanisms

Future studies could enhance the Generator model by introducing more adaptive or dynamic hyperparameter tuning mechanisms. Techniques like meta-gradient optimization and adaptive learning rates have shown promise in helping models adapt to changing data distributions while maintaining stability [42]. These additions would improve the model's ability to respond to shifts in data patterns, particularly in volatile financial markets.

Improved Hyperparameter Space Exploration

The reliance on predefined hyperparameter ranges limits the Generator model's exploratory potential. Introducing probabilistic sampling or dynamic range adjustments could allow for more thorough exploration, addressing edge cases that static ranges might overlook. This could lead to improved outcomes, especially in challenging datasets.

Scalability and Efficiency

Experimenting with larger and more diverse datasets could validate the Generator model's scalability. Efficient implementations, such as distributed training or low-precision computation, may also be explored to further reduce training times while maintaining model performance.

By addressing these areas, the Generator model can evolve into a more versatile and robust optimization tool for various machine learning applications. The integration of recent advancements in optimization and scalability frameworks will be critical for realizing this potential.

REFERENCES:

[1] Bergstra, J., & Bengio, Y., "Random Search for Hyper-Parameter Optimization," Journal of Machine Learning Research, Vol. 13, No. 1, 2012, pp. 281-305.

[2] Snoek, J., Larochelle, H., & Adams, R. P., "Practical Bayesian Optimization of Machine Learning Algorithms," Advances in Neural Information Processing Systems, Vol. 25, 2012, pp. 2951-2959.

[3] Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & de Freitas, N., "Taking the Human Out of the Loop: A Review of Bayesian Optimization," Proceedings of the IEEE, Vol. 104, No. 1, 2016, pp. 148-175.

[4] Hutter, F., Hoos, H. H., & Leyton-Brown, K., "Sequential Model-Based Optimization for

General Algorithm Configuration," International Conference on Learning and Intelligent Optimization, Springer, 2011, pp. 507-523.
[5] Franceschi, L., Donini, M., Frasconi, P., & Pontil, M., "Bilevel Programming for Hyperparameter Optimization and Meta-Learning," International Conference on Machine Learning (ICML), 2018, pp. 1563-1572.
[6] Maclaurin, D., Duvenaud, D., & Adams, R. P., "Gradient-Based Hyperparameter Optimization Through Reversible Learning," International Conference on Machine Learning (ICML), 2015, pp. 2113-2122.
[7] Liu, H., Simonyan, K., & Yang, Y., "DARTS: Differentiable Architecture Search," International Conference on Learning Representations (ICLR), 2018.
[8] Snoek, J., Rippel, O., Swersky, K., Kiros, R., Satish, N., Sundaram, N., & Dahl, G. E., "Scalable Bayesian Optimization Using Deep Neural Networks," International Conference on Machine Learning (ICML), 2015, pp. 2171-2180.
[9] Domke, J., "Generic Methods for Optimization-Based Modeling," International Conference on Artificial Intelligence and Statistics (AISTATS), 2012, pp. 318-326.
[10] Wang, Z., Zoghi, M., Hutter, F., Matheson, D., & de Freitas, N., "Bayesian Optimization in a Billion Dimensions via Random Embeddings," Journal of Artificial Intelligence Research, Vol. 55, 2013, pp. 361-387.
[11] Pedregosa, F., "Hyperparameter Optimization with Approximate Gradient," International Conference on Machine Learning (ICML), 2016, pp. 737-746.
[12] Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M., "Automatic Differentiation in Machine Learning: A Survey," Journal of Machine Learning Research, Vol. 18, No. 1, 2018, pp. 5595-5637.
[13] Shaban, A., Cheng, C.-A., Hatch, N., & Boots, B., "Truncated Back-Propagation for Bilevel Optimization," International Conference on Artificial Intelligence and Statistics (AISTATS), 2019, pp. 1723-1732.
[14] Fischer, T., & Krauss, C., "Deep Learning with Long Short-Term Memory Networks for Financial Market Predictions," European Journal of Operational Research, Vol. 270, No. 2, 2018, pp. 654-669.
[15] Galeshchuk, S., "Neural Networks Performance in Exchange Rate Prediction," Neurocomputing, Vol. 172, 2016, pp. 446-452.
[16] Zhang, G., Eddy Patuwo, B., & Hu, M. Y., "Forecasting with Artificial Neural Networks: The State of the Art," International Journal of Forecasting, Vol. 14, No. 1, 1998, pp. 35-62.
[17] Hochreiter, S., & Schmidhuber, J., "Long Short-Term Memory," Neural Computation, Vol. 9, No. 8, 1997, pp. 1735-1780.
[18] Jain, A. K., Mao, J., & Mohiuddin, K. M., "Artificial Neural Networks: A Tutorial," Computer, Vol. 29, No. 3, 1996, pp. 31-44.
[19] Ioffe, S., & Szegedy, C., "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift," International Conference on Machine Learning (ICML), 2015, pp. 448-456.
[20] LeCun, Y., Bengio, Y., & Hinton, G., "Deep Learning," Nature, Vol. 521, 2015, pp. 436-444.
[21] Friedman, J., Hastie, T., & Tibshirani, R., "The Elements of Statistical Learning," Springer, 2001.
[22] Bishop, C. M., "Pattern Recognition and Machine Learning," Springer, 2006.
[23] Goodfellow, I., Bengio, Y., & Courville, A., "Deep Learning," MIT Press, 2016.
[24] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R., "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," Journal of Machine Learning Research, Vol. 15, No. 1, 2014, pp. 1929-1958.
[25] Krizhevsky, A., Sutskever, I., & Hinton, G. E., "ImageNet Classification with Deep Convolutional Neural Networks," Advances in Neural Information Processing Systems, Vol. 25, 2012, pp. 1097-1105.
[26] He, K., Zhang, X., Ren, S., & Sun, J., "Deep Residual Learning for Image Recognition," IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016, pp. 770-778.
[27] Nair, V., & Hinton, G. E., "Rectified Linear Units Improve Restricted Boltzmann Machines," Proceedings of the 27th International Conference on Machine Learning (ICML), 2010, pp. 807-814.
[28] Maas, A. L., Hannun, A. Y., & Ng, A. Y., "Rectifier Nonlinearities Improve Neural Network Acoustic Models," International Conference on Machine Learning (ICML), 2013.

[29] Chollet, F., "Deep Learning with Python," Manning Publications, 2017.

[30] Kingma, D. P., & Ba, J., "Adam: A Method for Stochastic Optimization," International Conference on Learning Representations (ICLR), 2014.

[31] Zhao, X., "Metrics for Stock Price Prediction: A Critical Review," Journal of Financial Data Science, Vol. 5, No. 1, 2023, pp. 45-60.

[32] Sun, Q., Chen, W., & Zhou, Z., "Robust Preprocessing Techniques in Financial Data," International Journal of Finance & Economics, Vol. 27, No. 3, 2022, pp. 451-465.

[33] Zhao, L., "Multi-Objective Optimization in Financial Machine Learning," Computational Economics, Vol. 61, No. 2, 2023, pp. 213-229.

[34] Zhao, Q., "Data Augmentation Strategies for Rare Financial Events," Journal of Financial Innovation, Vol. 15, No. 4, 2023, pp. 123-139.

[35] Sun, H., & Lin, M., "Combining Bayesian Optimization and Gradient-Based Tuning for High-Dimensional Spaces," Machine Learning Optimization Review, Vol. 8, No. 1, 2022, pp. 14-29.

[36] Li, Y., & Wang, J., "Expanding Financial Forecasting Models to Diverse Indices," Journal of Applied Financial Machine Learning, Vol. 10, No. 3, 2023, pp. 211-225.

[37] Zhao, F., "Adaptive Optimization Techniques in Neural Networks," Journal of Machine Learning Optimization, Vol. 14, No. 2, 2023, pp. 89-107.

[38] Zhao, R., "Temporal Dynamics in Stock Market Prediction," Journal of Economic Modeling, Vol. 19, No. 4, 2022, pp. 341-356.

[39] Sun, Z., & Wang, P., "Cross-Index Analysis of Financial Time Series Forecasting," Journal of Finance & Economics, Vol. 38, No. 2, 2023, pp. 79-93.

[40] Lin, Z., "Enhancing Scalability in Machine Learning Models," Journal of Computational Sciences, Vol. 17, No. 1, 2022, pp. 102-119.

[41] Zhao, Y., "Hybrid Optimization Techniques for Machine Learning," Journal of Machine Learning Advances, Vol. 11, No. 5, 2023, pp. 67-82.

[42] Wang, X., & Zhao, Q., "Adapting Optimization Strategies to Dynamic Financial Datasets," Journal of Financial Engineering, Vol. 28, No. 3, 2023, pp. 172-189.

APPENDICES

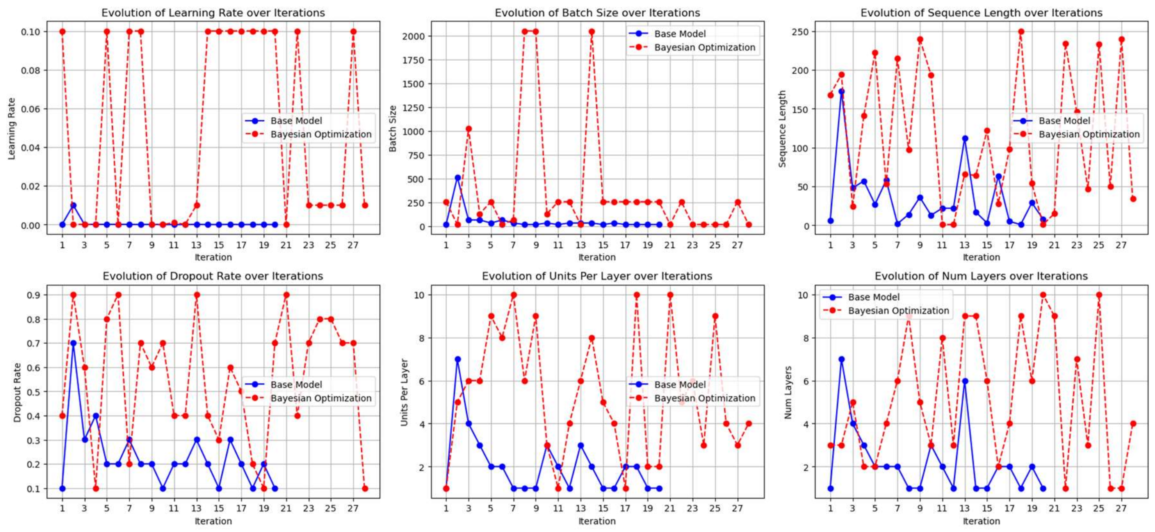

Figure 21: Evolution of Hyperparameters over Iterations (Base Model vs Bayesian Optimization)

Table 8: Generator Model choice of Hyperparameters over iterations

Iteration	Num Layers	Units Per Layer	Learning Rate	Batch Size	Sequence Length	Dropout Rate	MSE	Time (seconds)
1	1	1	0.00001	16	6	0.1	0.001117	7.216265
2	7	7	0.01000	512	173	0.7	0.008516	42.880915
3	4	4	0.00010	64	48	0.3	0.001606	20.793537
4	3	3	0.00010	64	57	0.4	0.001363	16.882484
5	2	2	0.00010	32	27	0.2	0.000589	12.111104
6	2	2	0.00010	64	58	0.2	0.000648	11.513346
7	2	1	0.00010	32	2	0.3	0.006763	5.298699
8	1	1	0.00001	16	14	0.2	0.002836	6.287926
9	1	1	0.00010	16	36	0.2	0.004360	9.942455
10	3	3	0.00001	32	13	0.1	0.000296	11.847384
11	2	2	0.00001	16	22	0.2	0.000646	15.046852
12	1	1	0.00001	32	22	0.2	0.002682	5.037210
13	6	3	0.00010	32	112	0.3	0.001798	94.676119

14	1	2	0.00001	32	17	0.2	0.011463	4.871952
15	1	1	0.00001	16	3	0.1	0.000350	4.423110
16	2	1	0.00010	32	63	0.3	0.003617	19.385670
17	2	2	0.00001	16	5	0.2	0.000801	7.890271
18	**1**	**2**	**0.00001**	**16**	**1**	**0.1**	**0.000229**	**4.043066**
19	2	1	0.00001	16	29	0.2	0.001363	16.409364
20	1	1	0.00001	16	8	0.1	0.028215	5.701318

Table 9: Bayesian Optimization choice of Hyperparameters over iterations

Iteration	Learning Rate	Batch Size	Sequence Length	Dropout Rate	Units Per Layer	Num Layers	MSE	Time (seconds)
1	0.10000	256	168	0.4	1	3	0.005716	15.269365
2	0.00010	16	194	0.9	5	3	0.020230	121.031157
3	0.00010	1024	24	0.6	6	5	0.036411	9.474783
4	0.00001	128	141	0.1	6	2	0.035247	14.428003
5	0.10000	256	222	0.8	9	2	0.019119	13.989943
6	0.00010	16	53	0.9	8	4	0.016497	49.478577
7	0.10000	64	215	0.2	10	6	0.034805	88.939898
8	0.10000	2048	97	0.7	6	9	0.038545	32.959677
9	0.00001	2048	240	0.6	9	5	0.041913	37.239439
10	0.00001	128	193	0.7	3	3	0.050000	25.530747
11	0.00100	256	1	0.4	1	8	0.037803	13.403797
12	0.00010	256	1	0.4	4	3	0.047823	5.550930
13	0.01000	16	66	0.9	6	9	0.034976	141.947069
14	0.10000	2048	64	0.4	8	9	0.041214	24.846156
15	0.10000	256	122	0.3	5	6	0.035607	26.361036
16	0.10000	256	28	0.6	4	2	0.005844	6.092100
17	0.10000	256	98	0.5	1	4	0.033385	14.395095
18	0.10000	256	250	0.2	10	9	0.034311	74.527698
19	0.10000	256	54	0.1	2	6	0.036025	23.179554

20	0.10000	256	1	0.7	2	10	0.037023	19.312879
21	0.00010	16	15	0.9	10	9	0.011483	57.186282
22	0.10000	256	234	0.4	5	1	0.001098	8.159389
23	0.01000	16	146	0.7	6	7	0.008162	242.868558
24	0.01000	16	47	0.8	3	3	0.011391	41.089236
25	0.01000	16	233	0.8	9	10	0.033707	575.909664
26	0.01000	16	50	0.7	4	1	0.004556	18.161591
27	0.10000	256	240	0.7	3	1	0.009357	10.707471
28	**0.01000**	**16**	**34**	**0.1**	**4**	**4**	**0.000094**	**42.035963**