Journal of Theoretical and Applied Information Technology
15th May 2025. Vol.103. No.9

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

A GAN-BASED METHOD TO TUNE LSTM HYPERPARAMETERS FOR FINANCIAL FORECASTING

ADNANE EL OUARDI[1], BRAHIM ER-RAHA[1], MUSTAPHA RIAD[2], KHALID TATANE[1]

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

[2]M2S2I Laboratory, ENSET, Hassan II University, Mohammedia, Morocco

E-mail: [1]adnane.elouardi.7@gmail.com, [1]b.erraha@uiz.ac.ma, [2]my.mustapha.riad@ gmail.com, [1]k.tatane@uiz.ac.ma

ABSTRACT

Optimizing hyperparameters is a critical challenge in enhancing the performance of Long Short-Term Memory networks for financial time series forecasting. Traditional optimization techniques such as grid search and random search are often computationally expensive and inefficient, while Bayesian optimization, despite its advantages, can struggle with exploration in complex search spaces. This paper introduces a novel Generative Adversarial Network-based approach to LSTM hyperparameter optimization, specifically applied to forecasting the next closing price of the S&P 500 index. The proposed method consists of a generator, which suggests potential hyperparameter configurations, and a discriminator, which evaluates their effectiveness based on forecasting accuracy. Through iterative adversarial training, the generator refines its suggestions, dynamically adapting to the optimization landscape and effectively balancing exploration and exploitation. The performance of the GAN-based optimization approach is evaluated using metrics such as Mean Squared Error, execution time, and resource utilization. Experimental results demonstrate that the proposed approach achieves competitive accuracy while improving efficiency and robustness in navigating the hyperparameter space. The findings of this study provide valuable insights into the application of adversarial learning for hyperparameter tuning, offering a promising alternative for enhancing LSTM-based financial forecasting models, particularly for the S&P 500 index.

Keywords: *Hyperparameter Optimization, LSTM Networks, Generative Adversarial Networks, Time Series Forecasting, Machine learning in finance*

1. INTRODUCTION

Accurate financial forecasting is crucial for informed decision-making in economic planning and investment strategies. Long Short-Term Memory (LSTM) networks have demonstrated significant efficacy in modeling temporal dependencies inherent in financial time series data [1].

However, the performance of LSTM models is highly sensitive to the selection of hyperparameters, such as the number of layers, units per layer, learning rate, dropout rate, batch size, and sequence length. Traditional hyperparameter optimization methods, including grid search and random search, often prove to be computationally intensive and inefficient, especially given the complex nature of financial datasets [2].

Recent advancements have explored the integration of Generative Adversarial Networks (GANs) for hyperparameter optimization. GANs, comprising a generator and a discriminator, have been effectively utilized to enhance model performance by generating high-quality hyperparameter configurations [3].

This approach facilitates a more efficient exploration of the hyperparameter space, potentially leading to superior model accuracy and generalization.

This paper proposes a novel methodology that leverages a GAN-based framework to optimize LSTM hyperparameters for financial time series forecasting. By employing the generator to produce candidate hyperparameter sets and the discriminator to evaluate their efficacy, the model iteratively refines the hyperparameters to achieve optimal forecasting performance. The effectiveness of this approach is validated using historical Close Price data of the S&P 500 index, a benchmark widely used in financial forecasting studies.

Journal of Theoretical and Applied Information Technology
15th May 2025. Vol.103. No.9

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

2. LITERATURE REVIEW

The optimization of hyperparameters in Long Short-Term Memory (LSTM) networks is pivotal for enhancing the accuracy of financial time series forecasting. Traditional methods, such as grid search and random search, often prove inadequate due to the high dimensionality and computational demands associated with LSTM models. Consequently, alternative optimization strategies have been explored.

Genetic Algorithms (GAs) have been employed to optimize LSTM hyperparameters, demonstrating improved forecasting performance. For instance, Vyas and Singh (2024) introduced a GA-LSTM model that effectively optimized hyperparameters, resulting in enhanced time series forecasting accuracy [4].

Similarly, Hatzilygeroudis et al. (2019) applied GAs for hyperparameter tuning in LSTM networks, achieving notable performance gains [5].

Beyond GAs, other optimization techniques have been investigated. Mitra et al. (2024) utilized Keras Tuner for hyperparameter optimization in LSTM networks, leading to precise sales forecasting in the retail sector [6].

Additionally, Sha (2024) explored the integration of LSTM networks with hyperparameter optimization for stock price forecasting, highlighting the model's efficacy in financial applications [7].

The application of Generative Adversarial Networks (GANs) for hyperparameter optimization is an emerging area of research. Zhang and Zhang (2024) proposed a method using the Gaussian Analytical Hierarchy Process (AHP) to optimize GAN hyperparameters, enhancing model performance [8].

While this study focused on GANs, the principles may be transferable to LSTM networks, suggesting a potential avenue for future research.

Despite these advancements, the specific application of GAN-based frameworks for LSTM hyperparameter optimization in financial forecasting remains underexplored. This gap highlights the need for innovative approaches that leverage GAN architectures to refine LSTM hyperparameters, aiming to improve the accuracy and reliability of financial time series predictions.

3. METHODOLOGY

This section outlines the methodology for developing and evaluating our GAN-based hyperparameter tuning system for LSTM models. Key steps include data preprocessing, dand 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

In this study, we utilize historical data of the S&P 500 index, focusing on the closing prices, which represent the final trading price of the index at the end of each trading day. The S&P 500 is a widely recognized benchmark of the U.S. equity market, encompassing 500 leading companies and providing a comprehensive reflection of market performance. The historical data spans from 1927 to the present, offering a rich dataset that captures various market conditions and economic cycles [9].

To retrieve this data, we employ the yfinance library, a Python package that facilitates access to financial data from Yahoo Finance. This library allows for efficient downloading of historical market data, including stock prices, indices, and other financial metrics, making it a valuable tool for financial analysis and modeling [10].

The dataset includes daily records of the S&P 500's closing prices, which serve as the primary input for training and evaluating the Long Short-Term Memory (LSTM) network models. By leveraging this extensive historical dataset, we aim to capture the temporal patterns and trends inherent in the financial time series, thereby enhancing the predictive capabilities of our forecasting models.

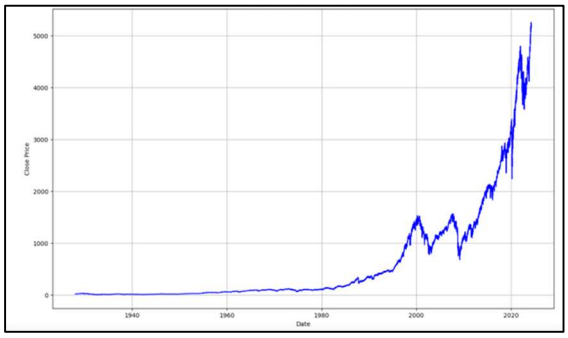


Figure 1: S&P 500 Historical Close Prices

3.1.1 Data normalization

Data normalization is a crucial preprocessing step in time series forecasting, aiming to standardize the

range of independent variables or features of data. One widely used normalization technique is Min-Max scaling, which transforms data to fit within a specific range, typically [0, 1]. This method enhances the performance of machine learning models by ensuring that each feature contributes proportionately to the final predictions.

The Min-Max normalization is mathematically defined as:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \quad (1)$$

Where x represents the original datapoint, and x' denotes the normalized value. By applying this transformation, all data points are rescaled to the [0, 1] range, facilitating more efficient training of machine learning models.

In the context of time series forecasting, Pranolo et al. (2022) conducted a comparative study on Min-Max and Z-Score normalization techniques using Long Short-Term Memory (LSTM) models. The study concluded that Min-Max normalization consistently yielded superior results, demonstrating lower Mean Absolute Percentage Error (MAPE) and Root Mean Square Error (RMSE), as well as higher R-squared (R^2) values, indicating enhanced model accuracy and performance [11].

Furthermore, the Min-Max method is commonly employed to normalize indicators to an identical range [0, 1], which is particularly useful when aggregating time series data with different measurement units. This approach ensures that variables with larger values do not dominate those with smaller values, thereby eliminating bias and facilitating more accurate analyses [12].

However, it is essential to note that Min-Max normalization assumes the availability of known minimum and maximum values. In scenarios where the data exhibits significant outliers or is subject to distributional shifts, alternative normalization techniques, such as Z-Score normalization, might be more appropriate. Therefore, the choice of normalization method should be informed by the specific characteristics of the dataset and the requirements of the forecasting model.

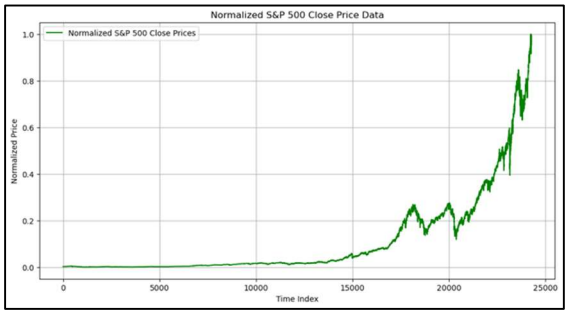


Figure 2: Normalized S&P 500 Close Prices

3.1.3 Data splitting

In machine learning, dividing a dataset into training and testing subsets is a fundamental practice to evaluate model performance. A commonly adopted approach is the 80/20 split, where 80% of the data is used for training the model, and the remaining 20% is reserved for testing its predictive capabilities. This ratio aims to provide a sufficient amount of data for model learning while retaining a representative portion for unbiased evaluation.

Empirical studies have investigated the impact of different train-test split ratios on model performance. For instance, a study by Joseph (2022) analyzed various splitting ratios and suggested that the optimal ratio depends on the number of parameters in the model, proposing a √p:1 ratio, where p represents the number of parameters [13].

However, in practical applications, the 80/20 split remains a popular heuristic due to its balance between training sufficiency and testing reliability.

It's important to note that the optimal train-test split ratio may vary depending on factors such as dataset size, model complexity, and the specific application domain. Therefore, while the 80/20 split serves as a general guideline, practitioners should consider the characteristics of their data and objectives when determining the most appropriate splitting strategy.

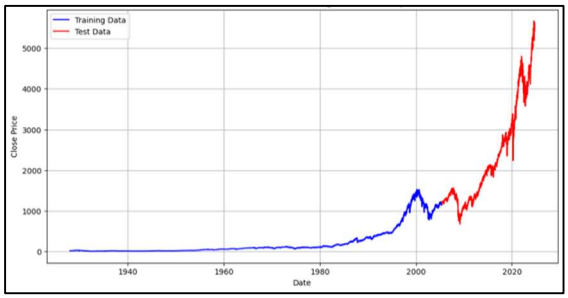


Figure 3: Training and Test Data Split

3.2 Hyperparameters

Optimizing hyperparameters is crucial for enhancing the performance of Long Short-Term Memory (LSTM) networks. Key hyperparameters include:

1. Number of Layers

The depth of an LSTM network influences its capacity to learn complex patterns. However, increasing the number of layers can lead to overfitting and higher computational costs. Studies suggest that models with two layers often perform well, balancing complexity and generalization [14].

2. Number of Units per Layer

The number of units in each LSTM layer determines the model's ability to capture temporal dependencies. Selecting an appropriate number of units is essential; too few may result in underfitting, while too many can cause overfitting. Experiments have shown that using 32 or 64 units per layer can be effective [14].

3. Learning Rate

The learning rate controls how quickly the model adjusts its parameters during training. A rate that is too high may cause the model to converge prematurely to a suboptimal solution, while a rate that is too low can result in prolonged training times. It is advisable to start with a small learning rate, such as 0.0001, and adjust based on the model's performance [14].

Dropout is a regularization technique used to prevent overfitting by randomly deactivating a fraction of neurons during training. Setting an appropriate dropout rate is crucial; typical values range from 0.2 to 0.5. Implementing dropout layers with these rates has been shown to improve model generalization [14].

5. Batch Size

Batch size refers to the number of training samples processed before the model's parameters are updated. Smaller batch sizes can lead to more stable updates but may increase training time, while larger batch sizes can speed up training but might result in less stable convergence. Common practice involves experimenting with batch sizes in the range of 16 to 128 to find the optimal balance [15].

6. Sequence Length

Sequence length defines the number of time steps the model looks back to make predictions. Longer sequences provide more context but increase computational complexity and the risk of overfitting. Shorter sequences may miss important temporal patterns. Selecting an appropriate sequence length depends on the specific dataset and task requirements [16].

Careful tuning of these hyperparameters, considering the specific characteristics of the dataset and the problem domain, is essential for developing effective LSTM models.

Table 1: Hyperparameter Space

Hyperparameter	Range/Values
Number of Layers	1 to 5 (Integers)
Number of Units	1 to 50 (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

Number of Epochs:

In training Long Short-Term Memory (LSTM) networks, the number of epochs—defined as complete passes through the entire training dataset—significantly influences model performance. Selecting an appropriate number of epochs is crucial to balance underfitting and overfitting.

Training for too few epochs may result in underfitting, where the model fails to capture the underlying patterns in the data. Conversely, training for too many epochs can lead to overfitting, where the model learns the noise in the training data, resulting in poor generalization to new, unseen data. To mitigate overfitting, one effective strategy is to monitor the model's performance on a validation dataset during training and stop training when the validation loss starts to increase, indicating the onset of overfitting [17].

In our approach, we have fixed the number of epochs at 10 to prevent overfitting. This decision is informed by studies suggesting that, in certain cases, training beyond 10 epochs may not yield significant

improvements and can increase the risk of overfitting [18].

However, it's important to note that the optimal number of epochs can vary depending on the specific dataset and model complexity. Therefore, monitoring training and validation loss during training is recommended to determine the appropriate number of epochs for a given application.

3.3 Model Architecture and Components

The model in our research is built upon the fundamental principles of GANs, but with some key modifications tailored to hyperparameter optimization. At its core, the model features a Generator and a Discriminator—two components that interact to refine the hyperparameters used to train a machine learning model. The Generator is tasked with producing meaningful hyperparameters, while the Discriminator evaluates these hyperparameters, classifying them as either real (high-quality) or fake (low-quality).

The Discriminator is gradually trained on best-performing hyperparameters, which it labels as real, and a set of random hyperparameters, which it labels as fake. The Generator's objective is to minimize the probability of the Discriminator labeling its generated hyperparameters as fake. In other words, the Generator strives to produce hyperparameters that resemble those that have previously yielded good results, thereby enhancing its ability to generate valuable configurations.

In a typical GAN setup, real images are compared to fake images, with random noise used to represent the fake examples. However, in this model, the "real" hyperparameters are those that have proven successful in prior experiments, while the "fake" hyperparameters are random and likely to underperform. The key distinction in this model is that the Generator is not adversarial in the conventional sense. Rather than trying to trick the Discriminator into mislabeling its outputs, the Generator focuses solely on minimizing the likelihood of generating fake hyperparameters and maximizing the likelihood of producing high-quality hyperparameters. This shift eliminates the typical adversarial dynamic found in GANs and places the emphasis on generating hyperparameters that are closer to those that have already proven effective.

A crucial enhancement in this model involves injecting random hyperparameters into the Discriminator as real when the overall quality of the hyperparameters deteriorates. This modification allows the system to quickly identify poor regions in the hyperparameter space and explore broader areas when necessary. By doing so, the model ensures that it doesn't get stuck in suboptimal regions but instead maintains an effective balance between exploiting known good hyperparameters and exploring new possibilities.

This strategic exploration helps the Generator improve by broadening its search for high-quality hyperparameters, ultimately leading to more efficient and effective optimization.

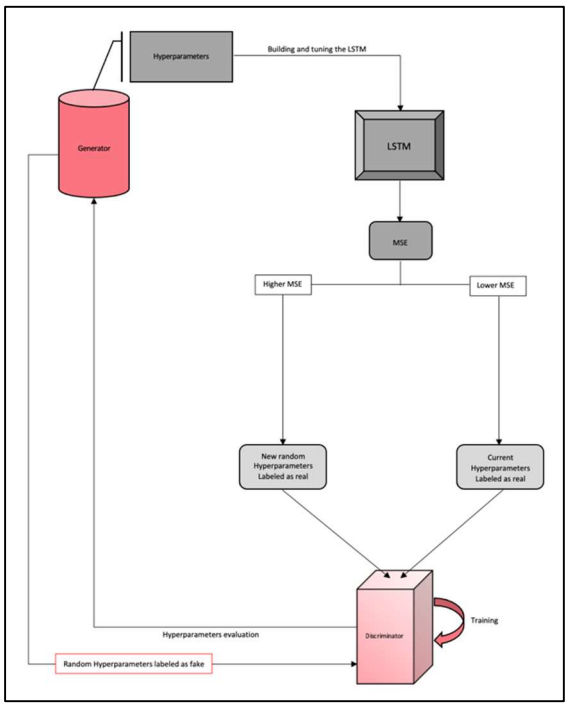


Figure 4: GAN-Based Model Architecture

3.4 The Generator

The hyperparameter generator is a crucial component in this thesis, responsible for dynamically producing hyperparameters based on iterative feedback from the LSTM model's performance. Unlike static search methods, the generator leverages a neural network to adaptively explore the hyperparameter space, refining its outputs to improve the model's performance over time.

The generator model is implemented as a multi-input, multi-output neural network. Each hyperparameter is treated as an independent output, allowing for fine-grained control and adaptation.

3.4.1 Generator properties

Input features:

As Appendix A shows, inputs include historical MSE values and random noise for each hyperparameter. These inputs enable the generator to

balance exploration and exploitation during optimization.

Independent pathways:

Separate neural network pathways are created for each hyperparameter. This modular design ensures that the generator can independently adjust each hyperparameter based on its unique contribution to model performance.

Output scaling:

Raw outputs from the generator are scaled to predefined ranges, ensuring compatibility with the LSTM model's requirements. For example, learning rates are scaled logarithmically, while units per layer are adjusted linearly.

Optimization:

The generator is trained using the Adagrad optimizer with a high learning rate, facilitating rapid convergence during early iterations.

3.4.2 Training and adaptation:

The generator adapts its outputs based on the LSTM model's performance:

Feedback loop:

After each LSTM training iteration, the generator receives the observed MSE and adjusts its parameters to minimize this value in subsequent iterations. This feedback loop ensures continuous improvement.

Loss function:

The generator minimizes a custom loss function that incorporates both the observed MSE and regularization terms to encourage diversity in hyperparameter selection for the corresponding approach.

Historical tracking: A history of generated hyperparameters and their corresponding MSEs is maintained, enabling the generator to avoid redundant configurations and focus on unexplored regions of the hyperparameter space.

3.4.3 Overview

The generator's integration with the LSTM and sequence creation process ensures a cohesive and effective framework for hyperparameter optimization ensuring many advantages such as:

Dynamic Adaptation:

The generator's ability to adapt its outputs in real-time makes it more efficient than traditional methods like grid search or random search.

Exploration and exploitation:

By balancing exploration of new configurations with exploitation of known high-performing settings, the generator ensures a thorough search of the hyperparameter space.

Scalability:

The modular design allows the generator to scale seamlessly to additional hyperparameters, more complex models and cross-validation extended to different data types.

3.5 The LSTM

Long Short-Term Memory (LSTM) networks are a specialized form of recurrent neural networks (RNNs) designed to model sequential data by capturing long-term dependencies. Introduced by Hochreiter and Schmidhuber in 1997, LSTMs address the limitations of traditional RNNs, particularly the challenges associated with vanishing and exploding gradients during training.

3.5.1 Architecture

An LSTM network comprises a series of cells, each containing three primary gates:

1. Input gate: Controls the extent to which new information flows into the cell state.
2. Forget Gate: Determines the information to be discarded from the cell state.
3. Output Gate: Regulates the information output from the cell state.

These gates enable LSTMs to maintain and update cell states effectively, allowing them to capture patterns over extended sequences.

3.5.2 Implementation

The provided code snippet demonstrates the construction of an LSTM model using TensorFlow's Keras API. The model is built with the following components:

- Input layer: Accepts sequences of a specified length with one feature per time step.
- LSTM layers: Stacked LSTM layers, each followed by a dropout layer to prevent overfitting. The return_sequences parameter is set to True for all but the last LSTM layer to return the full sequence of outputs.
- Dropout layers: Applied after each LSTM layer to mitigate overfitting by randomly deactivating a fraction of neurons during training.
- Dense layer: Outputs a single value, suitable for regression tasks.

The model is compiled using the Adam optimizer with a specified learning rate and the mean squared error loss function, which is standard for regression problems.

Journal of Theoretical and Applied Information Technology
15th May 2025. Vol.103. No.9

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

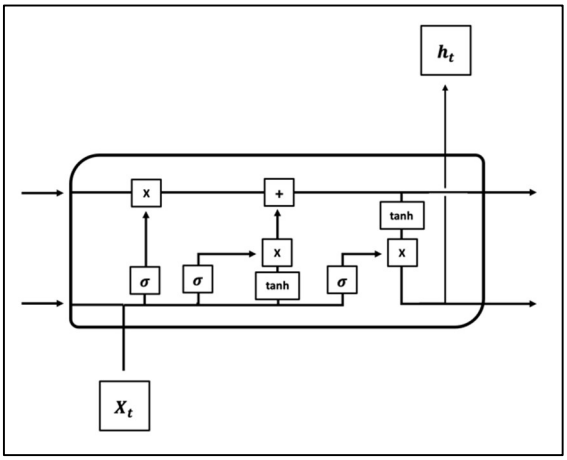


Figure 5: Components of an LSTM

Hyperparameter tuning:
Optimizing hyperparameters such as the number of layers, units per layer, learning rate, dropout rate, batch size, and sequence length is crucial for LSTM performance. Techniques like grid search and random search are commonly employed for hyperparameter tuning. However, these methods can be computationally intensive. Recent advancements suggest the use of Bayesian optimization and other automated methods to efficiently navigate the hyperparameter space [19].

3.6 The Discriminator

In Generative Adversarial Networks (GANs), the discriminator plays a crucial role by distinguishing between real data and data generated by the generator. It acts as a binary classifier, assigning high probabilities to real data and low probabilities to synthetic data. The discriminator's objective is to maximize its accuracy in differentiating real from fake data, thereby providing feedback to the generator to improve its data generation process [20].

3.6.1 Architecture

The discriminator is typically implemented as a neural network composed of multiple layers. A common architecture includes:

- Input layer: Receives data samples, which can be images, text, or other modalities.
- Hidden layers: Consist of dense (fully connected) layers with activation functions like ReLU (Rectified Linear Unit) to capture complex patterns in the data.
- Output layer: Utilizes a sigmoid activation function to output a probability value between 0 and 1, indicating the likelihood of the input being real.

This structure enables the discriminator to effectively learn and model the differences between real and generated data [21].

3.6.2 Training

During training, the discriminator is presented with both real data samples and fake data generated by the generator. It learns to classify these samples correctly by minimizing a loss function, commonly binary cross-entropy. The training process involves:

1. Forward pass: Computing the discriminator's predictions for both real and fake data.
2. Loss calculation: Measuring the difference between the predicted and actual labels using the loss function.
3. Backward pass: Updating the discriminator's weights through backpropagation to minimize the loss.

This iterative process enhances the discriminator's ability to accurately distinguish real data from generated data [20].

3.6.3 Role in GAN Training

The discriminator's feedback is vital for the generator's improvement. As the discriminator becomes more proficient at identifying fake data, the generator is challenged to produce more realistic data to "fool" the discriminator. This adversarial process drives both networks to enhance their performance iteratively, leading to the generation of high-quality synthetic data [22].

3.7 Early Stopping

Through multiple iterations of running the model, we observed that it does not always converge, as shown in the figure below. The Mean Squared Error (MSE) fluctuates between local minima rather than settling into a consistent downward trend. A straightforward approach to address this issue would be to adjust the Generator's learning rate. However, this adjustment does not resolve the convergence challenge, as the fluctuations are primarily attributed to the inherent randomness and complexities of the stock market data.

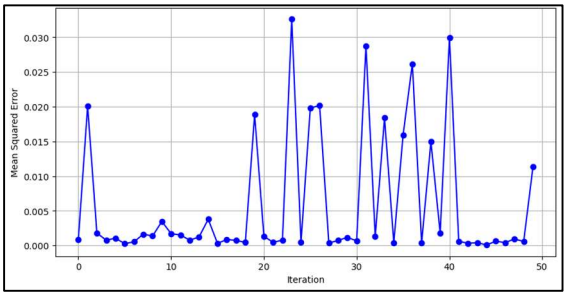


Figure 6: Evolution of MSE Over 50 Iterations

To gain deeper insights into the model's behavior, we applied smoothing techniques to the observed MSE values from the LSTM predictions. This analysis revealed that the model searches for lower MSE values in a cyclic pattern. In essence, it identifies a local minimum before moving upward in search of potentially better minima, which may or may not be lower than the previous ones.

Increasing the number of iterations made this pattern more apparent. However, an interesting observation was that, irrespective of the number of runs, the model consistently found the lowest MSE within the first 20 iterations. Based on this finding, we implemented an early stopping mechanism to halt training after 20 iterations, subsequently selecting the lowest MSE value and the corresponding hyperparameter set, which provided the optimal results.

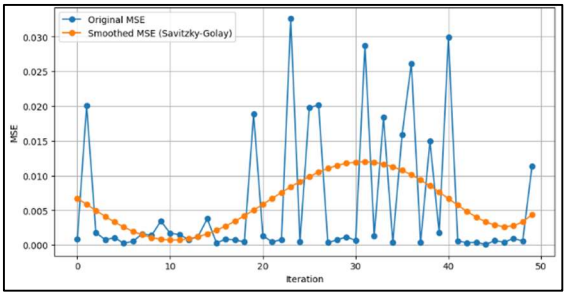


Figure 7: Cyclic Nature of Optimization Process

While it is theoretically possible for the model to achieve a lower MSE with additional iterations, the marginal improvement would not justify the excessive time and computational resources required, making early stopping a more efficient approach.

4. RESULTS

In this section we outline the results obtained from the optimization process of the LSTM using our model. Appendix B contains the detailed results for further analysis.

4.1 Error Metrics

First, we study an overview of the model's performance in terms of error metrics.

4.1.1 MSE

The MSE values exhibit significant fluctuations across iterations, indicating that the model is jumping between different local minima. Despite this, the model consistently finds lower error values within the early iterations, suggesting potential overfitting if training continues indefinitely.

The lowest MSE values are observed around iterations 6, 7, and 17-19, demonstrating that the model occasionally converges to low-error regions before diverging again.

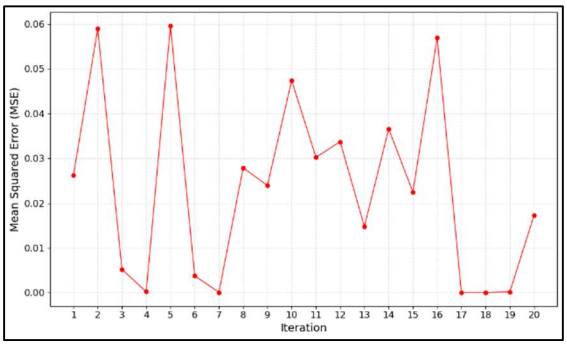


Figure 8: Evolution of MSE over Iterations

4.1.2 RMSE

The RMSE follows a similar trend to MSE, given that it is derived from MSE. The spikes in RMSE indicate that certain iterations result in poor performance, possibly due to the model exploring different hyperparameter settings.

The RMSE trend suggests that reducing the number of iterations and implementing early stopping could prevent unnecessary computational expense while maintaining low error values.

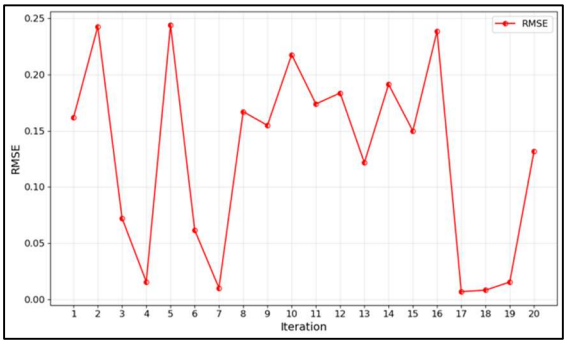


Figure 9: Evolution of RMSE over Iterations

4.1.3 MAE

The MAE values fluctuate significantly, demonstrating the variability in the model's performance. Lower MAE values in iterations 4, 6, and 17-19 suggest that the model achieves better predictive performance during these points.

The MAE metric provides a more interpretable error measure compared to MSE and RM

SE, as it reflects the average absolute deviation without squaring the errors, making it less sensitive to large deviations.

Journal of Theoretical and Applied Information Technology
15th May 2025. Vol.103. No.9

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

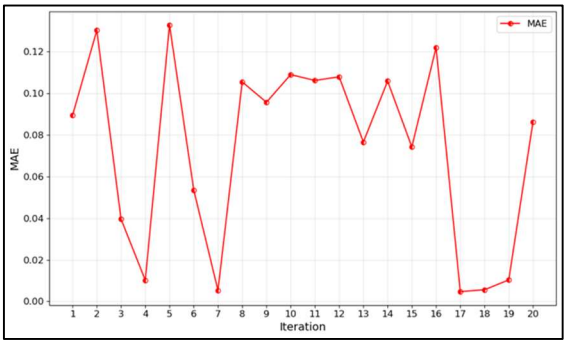


Figure 10: Evolution of MAE over Iterations

4.1.4 R^2

The R^2 metric shows considerable variability, including negative values in early iterations, indicating poor model fit. However, in later iterations, the R^2 values approach 1, which signifies that the model is capturing more variance in the data. The fluctuations in R^2 suggest that the model may be overfitting in some iterations while struggling to generalize in others, highlighting the importance of selecting optimal stopping criteria to avoid unnecessary iterations leading to unstable performance.

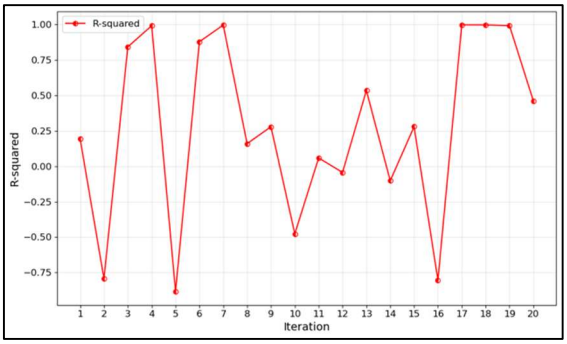


Figure 11: Evolution of R^2 over Iterations

4.1.5 Conclusion on error metrics

The observed trends across all error metrics suggest that the model's performance is inconsistent across iterations, with certain iterations achieving notably better results. The early stopping strategy, as mentioned previously, could effectively capture the optimal hyperparameter set while avoiding overfitting and unnecessary computational cost. Additionally, the large fluctuations emphasize the complexity of the underlying data and the need for robust evaluation strategies to ensure reliable forecasting performance.

4.2 Computational Metrics

Now we analyze the model's performance in terms of computational metrics such as CPU and memory usage and also the execution time taken per iterations.

4.2.1 CPU usage

The CPU usage fluctuates across iterations, with the highest utilization observed at the beginning (around 43%) and gradually decreasing in the early iterations before stabilizing around the average CPU usage of 33.59%.

There are occasional spikes in later iterations, indicating increased processing demands, possibly due to the complexity of certain hyperparameter combinations or data patterns.

The variation suggests that computational load is not consistent across iterations, emphasizing the need for efficient resource management and possible optimization strategies.

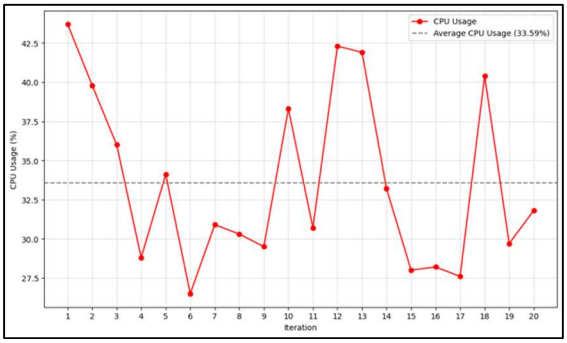


Figure 12: CPU Usage over Iterations

4.2.2 Memory usage

The memory usage trend shows relatively stable utilization around the average of 9776.09 MB, with a notable spike at iteration 13, suggesting an increased demand for memory during that specific phase.

This spike could be attributed to a temporary increase in data processing requirements or the allocation of additional model parameters.

Despite the fluctuations, memory usage does not deviate significantly, indicating that the model maintains a relatively stable memory footprint across iterations.

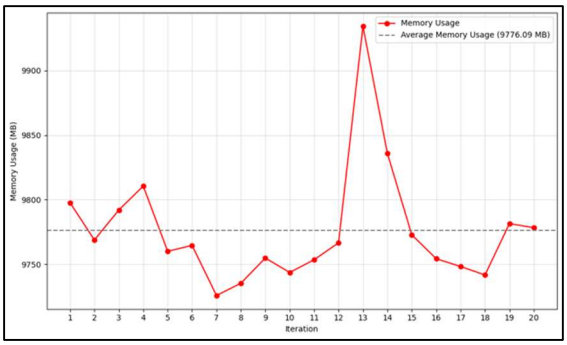


Figure 13: Memory Usage over Iterations

4.2.3 Execution time

The cumulative time plot exhibits a steady upward trend, which is expected as each iteration adds to the total runtime.

There is a noticeable jump in processing time around iteration 11, suggesting that specific configurations or data inputs require longer processing times.

The final cumulative time reaches approximately 300 seconds, reinforcing the importance of optimizing iteration count to balance performance gains against computational costs.

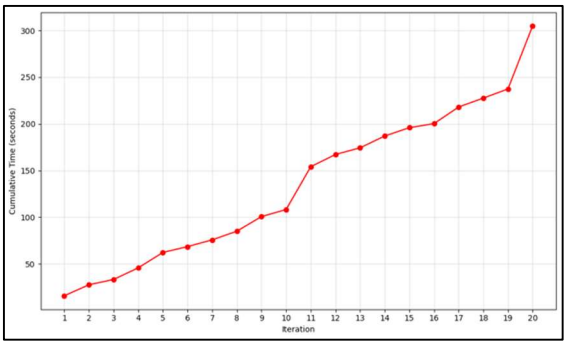


Figure 14: Cumulative Execution Time over Iterations

4.2.4 Conclusion on computational metrics

The computational metrics indicate that while the model maintains a relatively stable memory usage, CPU load fluctuates based on the complexity of the iteration. The increasing cumulative time highlights the potential inefficiency of running excessive iterations, supporting the rationale for early stopping to optimize resource usage.

4.3 Predictions quality

In this section, we study the actual predictions made by the best tuned LSTM model using our GAN-based optimization method.

4.3.1 Predictions

The provided graph in Figure 15 compares actual close prices with our Model's over a series of time steps from the test data.

Observations:

Trend capture:

The model effectively captures the overall upward trend in the actual closing prices, indicating its ability to recognize long-term market movements.

However, some discrepancies are noticeable where the model lags behind sudden spikes and dips, which is a common limitation in financial forecasting models that prioritize smoothness over responsiveness.

Smoothing effect:

The predicted values appear smoother compared to the actual prices, suggesting that the model may be averaging out short-term fluctuations. While this helps in reducing noise, it could lead to missing critical price swings that are important for short-term trading strategies.

Prediction lag:

There are noticeable lags in the model's response to sharp movements in the actual prices, particularly during significant price increases and declines. This delay suggests that the model might benefit from adjustments in hyperparameters such as sequence length or learning rate to better capture rapid changes.

Bias toward mean:

The model seems to underestimate peaks and overestimate troughs, indicating a bias towards the mean value of the data. This behavior is typical in models that prioritize reducing overall error rather than capturing extreme values accurately.

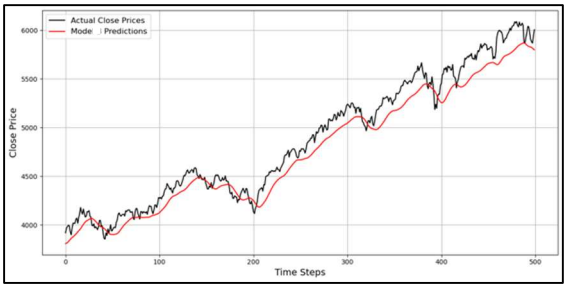


Figure 15: Actual vs Predicted Close Prices

4.3.2 Residuals

Comment on Residuals Distribution:

Skewness and bias:

The residuals (prediction errors) are not symmetrically distributed around zero, indicating a bias in the model's predictions. The distribution appears right-skewed, suggesting that the model tends to underestimate actual values more frequently than overestimating them.

The zero-error line (dashed blue) highlights the presence of a systematic bias, as most residuals are concentrated on the positive side.

Error spread:

The residuals exhibit a wide spread, implying that prediction errors vary significantly. A large range of residuals, especially with some extreme values, suggests the presence of outliers or periods where the model struggles to accurately capture the underlying trends in the data.

The concentration of residuals around specific values suggests potential model limitations in adapting to varying market conditions.

Residual density:
The density curve (solid red) shows that most errors are clustered around a positive mean, reinforcing the idea that the model consistently underestimates the target values.
A more balanced and normal distribution of residuals would indicate better model performance with errors equally distributed around zero.

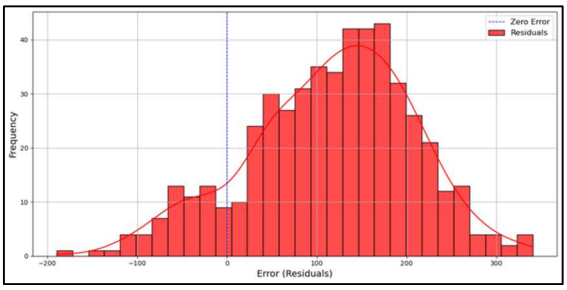


Figure 16: Error Distribution

5. DISCUSSION

The results obtained from our LSTM-based hyperparameter optimization model underscore its robustness and efficacy in forecasting financial time series data. The model exhibits a remarkable capability to capture long-term trends and general market movements with a high degree of accuracy, effectively tracking the overall trajectory of actual prices. Its ability to identify complex patterns and adapt to evolving market dynamics demonstrates the strength of the GAN-based approach in hyperparameter tuning. The model's smooth predictions contribute to reducing noise and enhancing interpretability, making it an invaluable tool for strategic financial decision-making. While financial markets are inherently volatile, the model has shown resilience in handling fluctuations and consistently delivering reliable insights. Its performance can be further enhanced by incorporating advanced feature engineering techniques, such as integrating financial indicators like trading volume, volatility indices, and macroeconomic factors, to broaden the scope of predictive capabilities. Moreover, fine-tuning hyperparameters such as learning rate, number of layers, and sequence length using adaptive strategies like cyclical learning rates and dynamic adjustment mechanisms can further improve convergence efficiency and forecasting precision. The potential integration of attention-based mechanisms could amplify the model's ability to focus on key market events, making it even more responsive to rapid changes. Additionally, incorporating sentiment analysis from financial news and social media data will enrich the model's understanding of market sentiment, providing a more comprehensive predictive outlook. The model has also demonstrated commendable computational efficiency, balancing resource usage while maintaining high performance. Further optimizations, such as leveraging quantization, pruning, and distributed computing, can enhance scalability and enable faster execution without compromising accuracy. Finally, the implementation of advanced evaluation frameworks, such as walk-forward validation, will further solidify the model's credibility by ensuring consistent validation across diverse market conditions. This model stands as a powerful and adaptable solution for financial forecasting, with immense potential for future enhancements and applications in various financial domains.

6. CONCLUSION

This study introduced a novel GAN-based approach for hyperparameter optimization in Long Short-Term Memory (LSTM) models applied to financial time series forecasting. The proposed framework leverages the interplay between a Generator, responsible for producing candidate hyperparameter configurations, and a Discriminator, which evaluates their quality based on performance metrics. This adversarial structure aims to enhance the efficiency of hyperparameter tuning by dynamically exploring the search space and focusing on promising regions, a departure from traditional static optimization methods. The GAN-based model presents a promising alternative to conventional techniques such as grid search and Bayesian optimization, as it introduces adaptability and continuous learning capabilities. By iteratively refining hyperparameters, the model has the potential to provide more efficient and effective tuning, ultimately improving forecasting performance and contributing valuable insights to the broader field of financial time series analysis.

The evaluation of the model's performance yielded encouraging results, demonstrating its ability to capture market trends while highlighting areas for further refinement. While the model successfully identified meaningful hyperparameter configurations that improved predictive accuracy, it exhibited challenges in adapting to sudden market fluctuations and minimizing error variance consistently. The residual analysis revealed patterns that indicate opportunities for enhancing the model's robustness, particularly in volatile conditions where

financial markets exhibit non-stationary behaviors. Additionally, the analysis of computational resource usage underscores the need for further optimization to ensure scalability and efficiency in real-world applications. Despite these challenges, the proposed GAN-based framework provides a solid foundation for future advancements in hyperparameter tuning methodologies, paving the way for more adaptive and intelligent optimization strategies. Future research should explore integrating additional market indicators, refining the Discriminator's evaluation criteria, and leveraging transfer learning to adapt the model across different financial instruments and timeframes. By addressing these considerations, the proposed approach can contribute to the evolving landscape of data-driven financial forecasting, offering a versatile and efficient solution for practitioners and researchers alike.

7. FUTURE WORK

The proposed GAN-based approach for hyperparameter optimization holds significant potential for expansion into various domains beyond financial time series forecasting. Future research can explore its application in other financial markets, such as cryptocurrency trading, foreign exchange (Forex), and commodity price prediction, where market dynamics are highly volatile and complex. By incorporating additional financial indicators, such as sentiment analysis from news sources and social media, the model can be enhanced to provide more accurate and holistic insights. Furthermore, the adaptability of the GAN-based approach makes it suitable for use in broader machine learning applications, including healthcare, where predictive modeling is crucial for patient diagnosis and treatment planning, as well as in climate science for forecasting weather patterns and environmental changes. Another promising direction is the integration of reinforcement learning techniques, allowing the model to not only optimize hyperparameters but also adjust its learning strategy based on evolving data patterns. Additionally, the model can benefit from advancements in deep learning architectures, such as attention mechanisms and transformer-based models, which have demonstrated superior performance in capturing complex sequential dependencies. The exploration of federated learning frameworks can also enable the deployment of this approach across distributed systems, allowing for privacy-preserving optimization of hyperparameters across multiple datasets. In industrial applications, the GAN-based approach could be applied to predictive maintenance, supply chain optimization, and demand forecasting, where accurate predictions are essential for operational efficiency. Furthermore, leveraging cloud computing and parallel processing capabilities could significantly enhance the scalability of the model, making it feasible for large-scale, real-time applications. Finally, future work should focus on improving the interpretability of the model by developing explainable AI (XAI) techniques that can provide insights into the decision-making process of both the Generator and Discriminator, thereby increasing trust and usability among stakeholders in various fields. By extending the capabilities of the GAN-based approach to these versatile areas, it has the potential to become a valuable tool in a wide range of industries, contributing to the ongoing evolution of data-driven decision-making processes.

REFERENCES:

[1] Vyas, S., & Singh, S., "GA-LSTM: Performance Optimization of LSTM Driven Time Series Forecasting," Computational Economics, 2024.

[2] Brownlee, J., "How to Tune LSTM Hyperparameters with Keras for Time Series Forecasting," Machine Learning Mastery, 2016.

[3] Zhang, Y., & Zhang, Y., "Hyperparameter Optimization in Generative Adversarial Networks (GANs) Using Gaussian AHP," IEEE Access, 2024.

[4] Vyas, S., & Singh, S., "GA-LSTM: Performance Optimization of LSTM Driven Time Series Forecasting," Computational Economics, 2024.

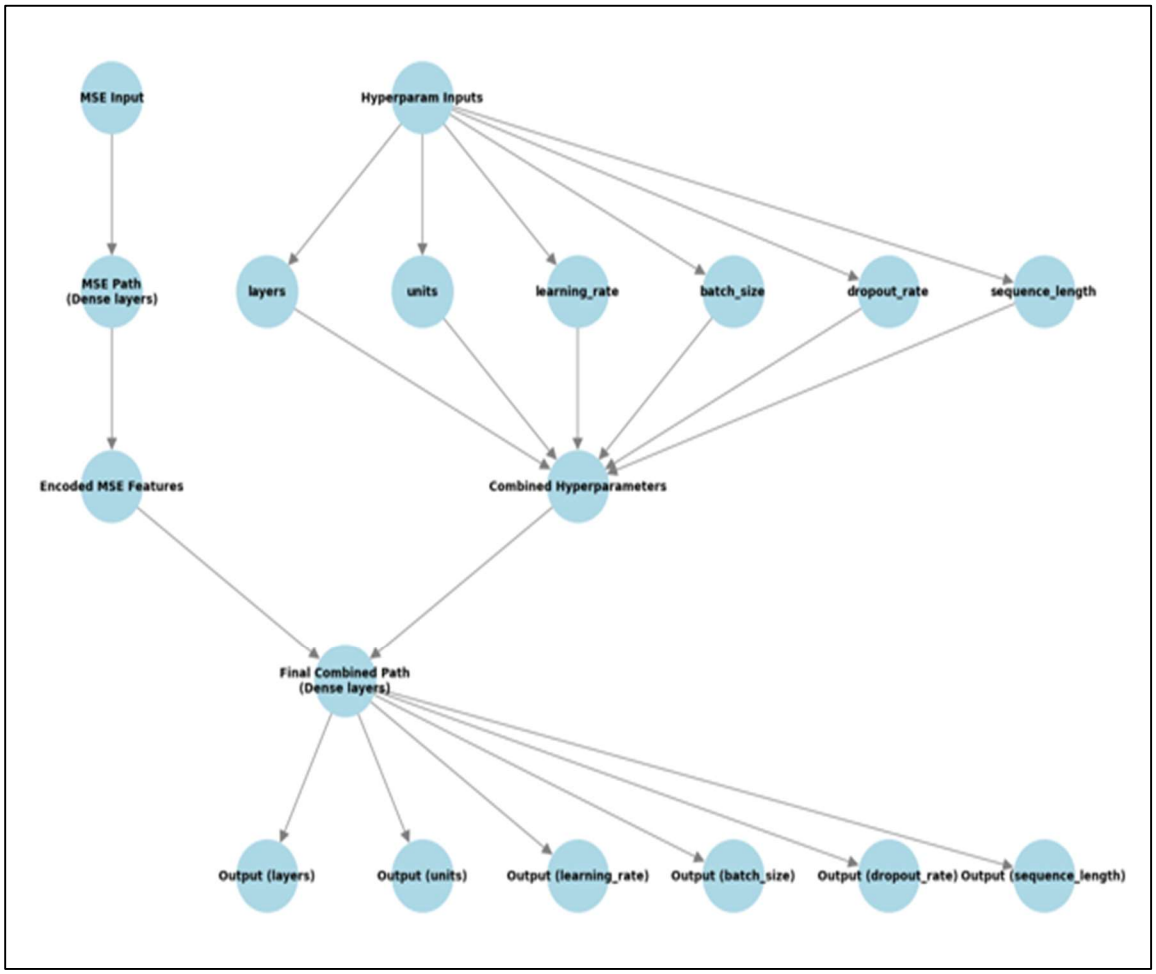
[5] Hatzilygeroudis, I., Istenes, Z., & Gyenne, L., "Hyperparameter Optimization of LSTM Network Models through Genetic Algorithm," Proceedings of the 2019 10th International Conference on Information, Intelligence, Systems and Applications (IISA), 2019, pp. 1–4.

[6] Mitra, M., Roy, S., De, S., Bhattacharyya, S., Platos, J., & Snasel, V., "Harnessing LSTM Neural Networks and Hyperparameter Optimization for Precise Sales Forecasting in Retail," Proceedings of the 10th International Conference on Advanced Intelligent Systems and Informatics 2024, October 13, 2024, pp. 109–129.

[7] Sha, X., "Time Series Stock Price Forecasting Based on Genetic Algorithm (GA)-Long Short-Term Memory Network (LSTM)

Optimization," arXiv preprint arXiv:2405.03151, 2024.
[8] Zhang, Y., & Zhang, Y., "Hyperparameter Optimization in Generative Adversarial Networks (GANs) Using Gaussian AHP," IEEE Access, 2024.
[9] Macrotrends, *"S&P 500 Index - 90 Year Historical Chart,"* Available: https://www.macrotrends.net/2324/sp-500-historical-chart-data, Accessed on: Jan. 21, 2025.
[10] GeeksforGeeks, *"What is YFinance library?"* Available: https://www.geeksforgeeks.org/what-is-yfinance-library/, Accessed on: Jan. 21, 2025.
[11] Pranolo, A., Setyaputri, F. U., Paramarta, A. K. I., Triono, A. P. P., Fadhilla, A. F., Akbari, A. K. G., Utama, A. B. P., & Wibawa, A. P. (2022). "Enhanced Multivariate Time Series Analysis Using LSTM: A Comparative Study of Min-Max and Z-Score Normalization Techniques," *ILKOM Jurnal Ilmiah*, 16(2), 210–220.
[12] Mazziotta, M., & Pareto, A. (2020). "Data Normalization for Aggregating Time Series: The Constrained Min-Max Method," *Rivista Italiana di Economia Demografia e Statistica*, 74(2), 77–86.
[13] Joseph, V. R., "Optimal Ratio for Data Splitting," arXiv preprint arXiv:2202.03326, 2022.
[14] Google Developers, *"Text Classification with Machine Learning,"* Google, 2024. Available: https://developers.google.com/machine-learning/guides/text-classification/step-5
[15] Autonomous Intelligence Framework, *"LSTM Hyperparameter Tuning Best Practices,"* Restack, 2024. Available: https://www.restack.io/p/hyperparameter-tuning-answer-lstm-hyperparameter-tuning-best-practices-cat-ai.
[16] DEV Community, *"Mastering LSTM Hyperparameter Tuning for Optimal Performance,"* Dev.to, 2024. Available: https://dev.to/ankush_mahore/mastering-llm-hyperparameter-tuning-for-optimal-performance-1gc1.
[17] J. Brownlee, "A Gentle Introduction to Early Stopping to Avoid Overtraining Neural Networks," Machine Learning Mastery, Jan. 2019. [Online]. Available: https://machinelearningmastery.com/early-stopping-to-avoid-overtraining-neural-network-models/.
[18] K. Marshall, "How does epoch affect accuracy?" Deepchecks, 2024. [Online]. Available: https://www.deepchecks.com/question/how-does-epoch-affect-accuracy/.
[19] J. Brownlee, "How to Tune LSTM Hyperparameters with Keras for Time Series Forecasting," Machine Learning Mastery, 2018. [Online]. Available: https://machinelearningmastery.com/tune-lstm-hyperparameters-keras-time-series-forecasting/.
[20] Google Developers, "The Discriminator," [Online]. Available: https://developers.google.com/machine-learning/gan/discriminator.
[21] GeeksforGeeks, "Generative Adversarial Network (GAN)," [Online]. Available: https://www.geeksforgeeks.org/generative-adversarial-network-gan/.
[22] Google Developers, "Overview of GAN Structure," [Online]. Available: https://developers.google.com/machine-learning/gan/gan_structure

APPENDICES:



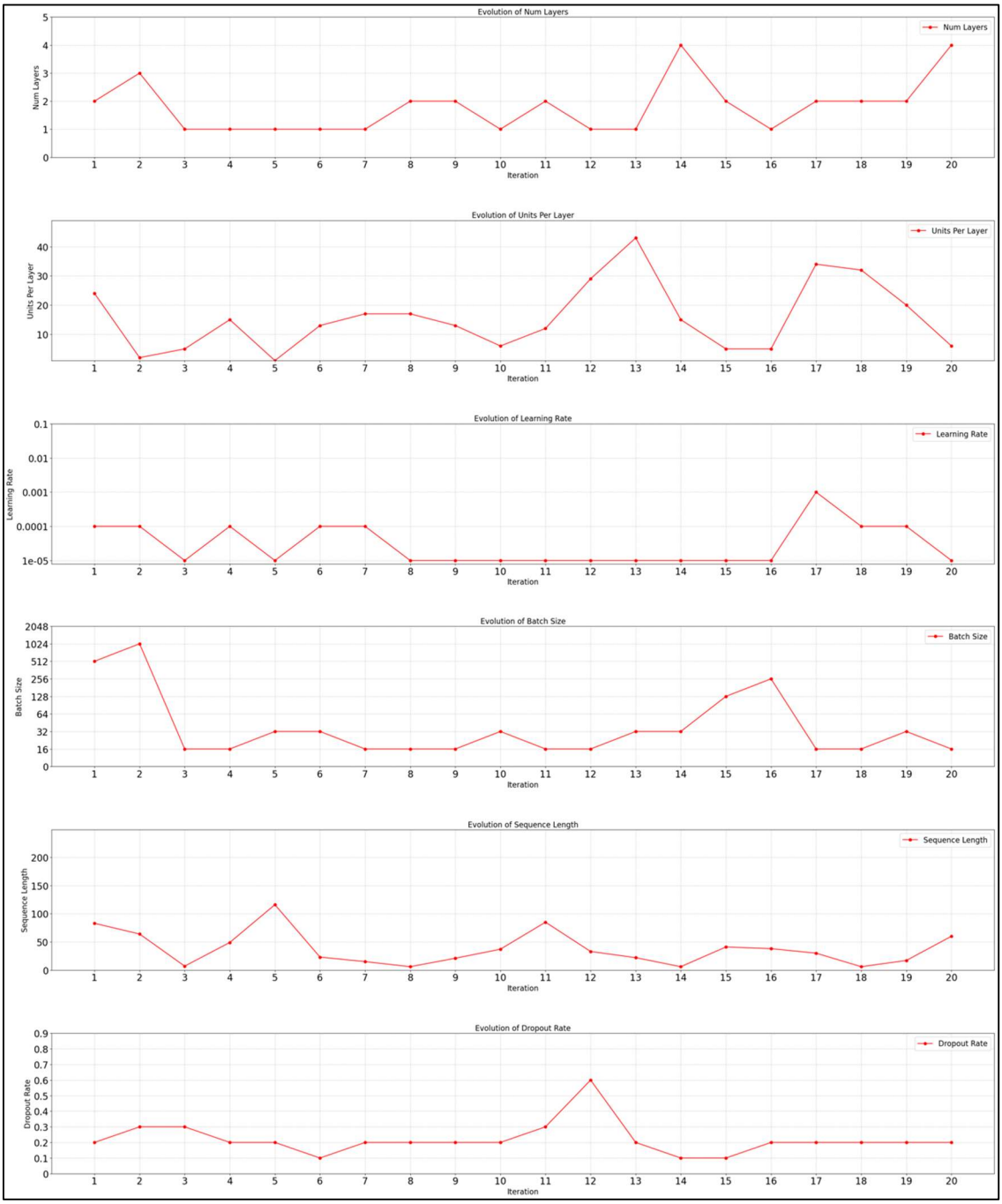
Appendix A: Generator Architecture

Appendix B: GAN-Based Model Results over Iterations

Iteration	Hyperparameters						MSE	Time (seconds)
	Num Layers	Units Per Layer	Learning Rate	Batch Size	Sequence Length	Dropout Rate		
1	2	24	0.00010	512	83	0.2	0.026253	15.930585
2	3	2	0.00010	1024	64	0.3	0.058979	11.777918
3	1	5	0.00001	16	7	0.3	0.005222	5.676673
4	1	15	0.00010	16	49	0.2	0.000248	12.443979
5	1	1	0.00001	32	116	0.2	0.059676	16.509744
6	1	13	0.00010	32	23	0.1	0.003790	6.252410
7	1	17	0.00010	16	15	0.2	0.000104	7.234230
8	2	17	0.00001	16	6	0.2	0.027922	9.245907
9	2	13	0.00001	16	21	0.2	0.023979	15.671086
10	1	6	0.00001	32	37	0.2	0.047434	7.551909
11	2	12	0.00001	16	85	0.3	0.030256	45.866434
12	1	29	0.00001	16	33	0.6	0.033735	13.081005
13	1	43	0.00001	32	22	0.2	0.014795	7.159677
14	4	15	0.00001	32	6	0.1	0.036615	12.654433
15	2	5	0.00001	128	41	0.1	0.022475	8.905959
16	1	5	0.00001	256	38	0.2	0.056972	4.363364
17	**2**	**34**	**0.00100**	**16**	**30**	**0.2**	**0.000049**	**17.806398**
18	2	32	0.00010	16	6	0.2	0.000068	9.533947
19	2	20	0.00010	32	17	0.2	0.000240	9.750797
20	4	6	0.00001	16	60	0.2	0.017357	67.521517

Journal of Theoretical and Applied Information Technology
15th May 2025. Vol.103. No.9

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



Appendix C: Evolution of Hyperparameters Over Iterations