Journal of Applied Informatics and Computing (JAIC)
Vol.10, No.1, February 2026, pp. 406~416
e-ISSN: 2548-6861

Indonesian Gold Price Forecasting Using Simple and Stacked LSTM with Expanding Window

Rahmat Tegar Patriot Hari Lambang [1]*, Ifnu Wisma Dwi Prastya [2]*, Mula Agung Barata [3]*
* Department of Informatics Engineering, Faculty of Science and Technology, Universitas Nahdlatul Ulama Sunan Giri
rahmatkarorio@gmail.com [1], ifnuprastya@unugiri.ac.id [2], mula.ab26@gmail.com [3]

Article Info

Article history:

Received 2026-01-01
Revised 2026-01-23
Accepted 2026-01-30

Keyword:

Deep Learning,
Expanding Window,
Forecasting,
Gold Price,
LSTM.

ABSTRACT

This study investigates the performance of two deep learning architectures, namely Simple LSTM and Stacked LSTM, for Indonesian gold price forecasting, with a particular focus on evaluating the effect of optimizer selection and learning rate configurations. An experimental framework is implemented using daily Indonesian gold price data from 2021 to 2024. Model performance is assessed using five-fold expanding window time series cross-validation to ensure robustness and avoid data leakage. Four adaptive training optimizers (Adam, Nadam, Adamax, and RMSprop) are evaluated across three learning-rate settings as part of a systematic sensitivity analysis of training hyperparameters. The results indicate that the Simple LSTM consistently outperforms the Stacked LSTM. The best performance is achieved by the Simple LSTM using the Adam optimizer with a learning rate of 0.01, yielding an RMSE of 9.235, MAE of 7.060, and MAPE of 0.71%. These findings demonstrate that simpler architectures combined with appropriate training configurations can provide superior forecasting accuracy for volatile financial time series.

I. INTRODUCTION

Gold's role as an inflation hedge and safe haven asset has attracted worldwide attention. Several studies have investigated whether gold prices can be considered a leading indicator of security or financial stability. Gold prices are heavily influenced by macroeconomic factors, including inflation rates, supply and demand, and geopolitical issues [1]. Fluctuations in gold prices have prompted the need for predictive analysis to help investors make more accurate decisions [2].

The accuracy of gold price predictions plays a crucial role in the financial and investment sectors. Over the past few years, various deep learning approaches have been developed to improve the accuracy of time series forecasting. Traditional statistical methods often struggle to capture long-term dependencies in time series data, while architectures such as Recurrent Neural Networks (RNNs) and their derivatives have demonstrated superior capabilities in learning complex temporal patterns [3].

To address this issue, recent studies have shifted their focus to the application of deep learning models. Deep learning is known as a novel model with strong performance and the ability to create efficient data processing models [4]. One such model is Long Short-Term Memory (LSTM), a development of Recurrent Neural Networks (RNN) [5].

This algorithm addresses the primary issue of RNNs, specifically their inability to handle sequential information over extended periods (long sequences/long-term dependencies), particularly when processing time-series data [6]. LSTM has the capacity to process sequential data and store information from previous steps in the sequence, which allows it to predict future steps effectively. This characteristic makes the LSTM model very suitable for tasks involving long-term dependencies [4], [6], [7].

Several comparative studies have demonstrated performance trade-offs in time series tasks, which encourage careful selection of architectures for forecasting. Previous studies by Yudha et al., Oukhouya et al., and Tae-Yun Kim et al. have compared the performance of several models, including LSTM for forecasting [8], [9], [10]. Another study by Zahara et al. explained the importance of applying training optimizers that have been proven effective in finding the best model. The comparison was conducted on Consumer Price

Index (CPI) data and found Nadam optimization to be the superior optimizer [11].

However, the fluctuating nature of gold price data, influenced by varying global market sentiments, raises the question: Whether the relative performance of different training optimizers and learning-rate configurations remains consistent when applied to volatile gold price data? This study conducts a controlled comparative investigation focusing on optimizer and learning rate selection on gold price datasets.

Gold price data is sequential, variable, and historical. Gold prices can be used as a measure of inflation protection, as they tend to follow changes in inflation [12]. Therefore, this study aims to conduct forecasting using the LSTM method. The input data uses Indonesian gold prices (IDR) from id.investing.com. Gold price data is collected from January 1, 2021, to December 31, 2024.

The focus of this study is to assess and compare the performance of Simple LSTM and Stacked LSTM models in forecasting gold prices. Both models are widely used deep learning architectures and represent variants within the LSTM family. Accordingly, this study is confined to a comparison between these two LSTM-based architectures in order to examine the effect of architectural depth and training configuration on forecasting performance [11]. To improve prediction accuracy, this study systematically evaluates four adaptive training optimizers—Nadam, Adam, Adamax, and RMSprop—under multiple learning-rate configurations. Model performance is assessed using three evaluation metrics, namely MAE, RMSE, and MAPE, to identify an effective LSTM-based forecasting model for gold price prediction [13], [14], [15].

II. METHODOLOGY

This research method begins with data collection, followed by data pre-processing, Simple LSTM and Stacked LSTM model architecture design, data sharing, predictive model development, and model evaluation. The steps in this research can be seen in Figure 1.

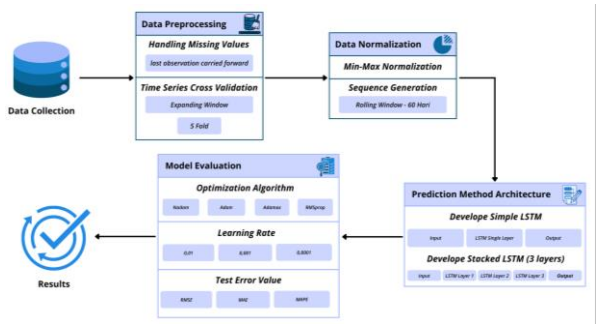

Figure 1. Flow of Research

A. Data Collection

The Indonesian gold price (IDR) data used is data obtained from id.investing.com for the period January 1, 2021 to December 31, 2024. The selection of the 2021–2024 period is further justified by its pronounced volatility and structural breaks in gold price movements. This period encompasses major global economic events, including post-pandemic recovery, inflationary shocks, monetary tightening, and geopolitical tensions, all of which significantly affected gold price dynamics. Consequently, the dataset exhibits frequent trend reversals and high variance, providing a challenging yet realistic environment for evaluating forecasting models under non-stationary conditions.

TABLE I
GOLD PRICE DATASET (2021 – 2024)

Date	Price	...	Change
01/01/2021	867,623	...	1.16%
03/01/2021	874,168	...	0.75%
04/01/2021	867,397	...	-0.77%
...	...	...	...
27/12/2024	1,364,551		-0.46%
30/12/2024	1,357,210	...	-0.54%
31/12/2024	1,357,613	...	0.03%

In this study, data in the price column were used because prices fluctuate quite frequently, making them suitable for predictions.

B. Data Preprocessing

The data preprocessing stage aims to transform raw data into a clean, structured dataset ready for deep learning models. The quality of the processed data directly impacts the model's ability to learn patterns and produce accurate predictions. The stages involved in data preprocessing include handling missing values, expanding window cross-validation, data scaling, and sequence generation with a sliding window.

In the first stage, missing values were handled because incomplete or missing daily price data were often found during the data collection process. To address this issue, this study employed an imputation technique to fill in these missing data [16]. This approach was chosen to maintain the continuity of time-series data trends without introducing bias that might arise from using the overall average value. This process ensured that the dataset was complete and ready for the next stage.

C. Expanding Window

To ensure the data's robustness, the cross-validation technique can beapplied. Traditional cross-validation techniques are not directly applicable to time series data. Therefore, a time series cross-validation approach with a walk-forward expanding window scheme was used throughout this study [17], [18]. This method, recommended by Hewamalage et al. and Cerqueira et al., has also been successfully applied in the financial domain by Wang et al. to evaluate futures trading strategies [19], [20], [21].

Forward-validation (walk-forward / expanding window) is recommended for non-stationary time series to avoid leakage and better approximate real-world forecasting performance [20]. The implementation uses TimeSeriesSplit from the

Scikit-learn library with five folds. At each iteration, the model is trained on a subset of past data and tested on a subset of future data, simulating real-world forecasting conditions. The RMSE, MAE, and MAPE metrics from the five folds are then averaged as the final evaluation score to compare model configurations.

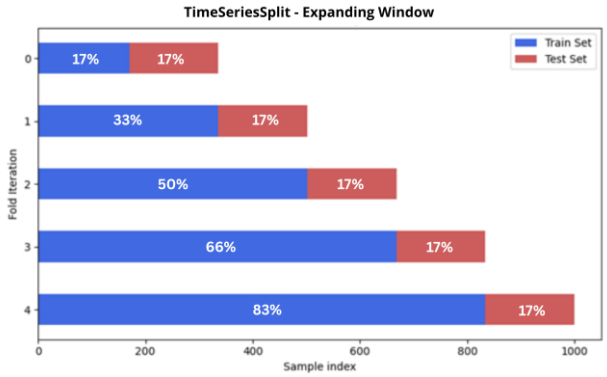

Figure 2. Using a 5-fold expanding window

D. Data Normalization

After data partitioning using the expanding window approach, the scaling process was performed using the Min–Max Normalization method with a range of [0,1] [22]. Scaling was only fitted to the training subset in each fold and applied to the testing subset using the same parameters to prevent data leakage [23]. This step aligns with the recommendations of Hewamalage et al. and Cerqueira et al., who emphasized that training subset-based normalization is a best practice for maintaining accuracy and temporal validity in non-stationary time series data such as gold prices [19], [20].

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

E. Architectural Design Model

Creating a model to predict gold prices using the Python programming language and several libraries such as Pandas, NumPy, Scikit-learn, TensorFlow, and Matplotlib. As illustrated in Figure 3, the basic structure of an LSTM cell consists of three main gates: an input gate, a forget gate, and an output gate. Each gate regulates the flow of information within the network, allowing the model to selectively remember or discard data over time. This internal mechanism allows LSTM networks to effectively capture long-term dependencies, which are crucial for time series forecasting [24].

Figure 4 shows that although a single LSTM layer can capture temporal patterns, complex financial data, such as gold prices, often has a complex dependency hierarchy. Daily patterns can form weekly trends, which then form monthly patterns. To effectively capture data representations at these multiple time scales, a deeper architecture is required. LSTM is one of the innovations of the Recurrent Neural Network (RNN) algorithm. In a typical RNN architecture, there is a network of recurrent modules consisting of layers of simple tanh functions.

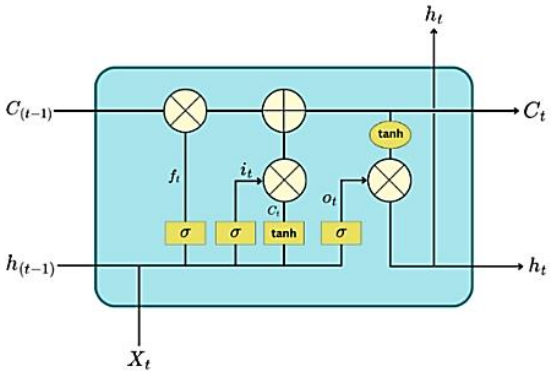

Figure 3. Basic structure of LSTM cells.

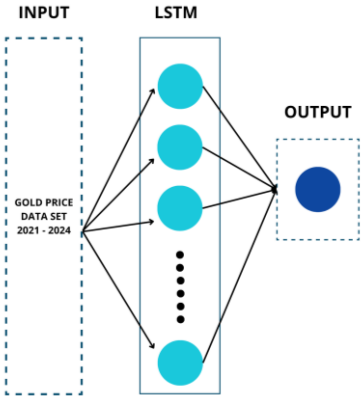

Figure 4. Simple LSTM structure [25].

The architecture of a Simple LSTM model generally consists of only one LSTM layer: an input gate that receives and processes data, a forget gate that decides which information should be discarded, and an output gate that processes all received calculations and produces the output of the LSTM cell. This architecture is relatively simple and has fewer parameters, resulting in faster training and a lower risk of overfitting.

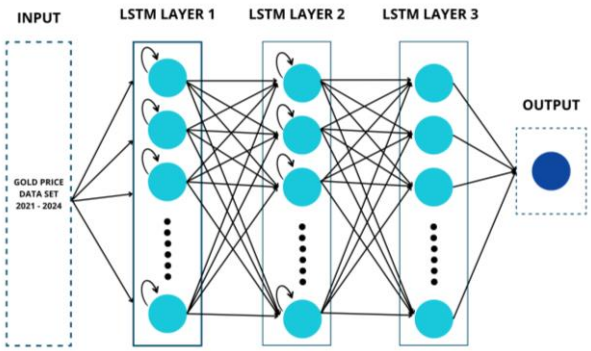

Figure 5. Stacked LSTM structure.

This study also implements the Stacked LSTM model architecture, as illustrated in Figure 5. In the Simple LSTM architecture, specifically, only the first layer of LSTM cells is directly used for prediction. Whereas in the Stacked LSTM architecture, the output of the LSTM cell in the first layer is not directly used for prediction, but becomes sequential input for the second LSTM layer, and so on for subsequent layers until the output.

This hierarchical approach allows the model to learn features at different levels of abstraction. The first layer focuses on short-term patterns in the raw data, while subsequent layers learn more complex long-term patterns from the output of the layers below. In this study, we designed a model with three LSTM layers to maximize the model's ability to learn complex dependencies in gold price data.

The architecture of the Stacked LSTM model is generally structured into three core parts of the LSTM cell, namely the input gate that plays a role in receiving data from external sources and processing the received data, the forget gate that decides which information needs to be deleted and sets the most appropriate waiting time for the next input, and the output gate that processes all calculations received and produces output from the LSTM cell. In the computation process, calculations are carried out using the following formula:

$$f_t = \sigma\left(W_f * [h_{t-1}, X_t] + b_f\right) \quad (2)$$

The forget gate is a crucial component of the Long Short-Term Memory (LSTM) architecture, controlling how much information from the previous cell state should be retained or discarded as shown in Equation (2). At each time step (t), the forget gate generates a vector (f_t) with a value between 0 and 1, obtained through a sigmoid activation function. Mathematically, the forget gate is formulated as $f_t = \sigma\left(W_f * [h_{t-1}, X_t] + b_f\right)$, where (W_f) is the weight matrix, (b_f) is the bias, , (h_{t-1}) is the hidden state from the previous time, and (X_t) is the input at time. The output (f_t) acts as a filter to determine the proportion of information in the previous cell state (C_{t-1}) that will be continued to the current cell state. A value of (f_t) close to 1 indicates that most information is retained, while a value close to 0 indicates that the information will be forgotten. This mechanism allows LSTMs to dynamically manage long-term memory and improves the model's ability to learn complex time series data dependencies.

$$i_t = \sigma(W_i * [h_{t-1}, X_t] + b_i) \quad (3)$$
$$\acute{C}_t = tanh\,(W_c * [h_{t-1}, X_t] + b_c) \quad (4)$$

The input gate functions to control the amount of new information that will be stored into the cell state at time step (t), the input gate produces an activation vector (i_t), which is calculated using the sigmoid function on a linear combination of the previous hidden state (h_{t-1}) and the current input (X_t), with the formula $i_t = \sigma(W_i * [h_{t-1}, X_t] + b_i)$. In addition, a new candidate cell state $(\acute{C}_t)$ is also created, which is generated through the tanh activation function with the formula $\acute{C}_t = tanh\,(W_c * [h_{t-1}, X_t] + b_c)$. The value of (i_t) serves as a determinant of how much contribution $(\acute{C}_t)$ will be included in the cell state. A value of (i_t) close to 1 indicates that new information is very important to store, while a value close to 0 indicates that the information is less relevant as shown in Equation (3, 4). With this mechanism, LSTM is able to selectively add new, relevant information to long-term memory, thereby increasing the accuracy in predicting complex time series data.

$$C_t = f_t * C_{t-1} + i_t * \hat{C}_t \quad (5)$$

Cell state update is a crucial step that allows LSTM to maintain long-term memory while accommodating relevant new information. In this step, the old cell state (C_{t-1}) is first modified by a forget gate (f_t), which determines the proportion of old information retained as shown in Equation (5). Next, the cell state is combined with new contributions controlled by the input gate (i_t) and the candidate cell state $(\hat{C}_t)$. Mathematically, cell state update is formulated as $C_t = f_t * C_{t-1} + i_t * \hat{C}_t$. The resulting value (C_t) acts as the LSTM cell's internal memory at a given time, which will be used in the next prediction step. This update process makes LSTM superior in learning long-term dependencies in complex time series data.

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o) \quad (6)$$
$$h_t = o_t * tanh(C_t) \quad (7)$$

The output gate plays a role in determining what information will be produced as output in the hidden state at time step (t). At this stage, the output gate produces an activation vector (o_t), which is calculated using a sigmoid function over a linear combination of the previous hidden state (h_{t-1}) and the current input (x_t), with the formula $o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$. The value of (o_t) is then used to filter the current cell state (C_t), after first being processed using the tanh activation function to limit the cell state value to the range $[-1,1]$. The final result is a hidden state (h_t) which is formulated as $h_t = o_t * tanh(C_t)$. This hidden state (h_t) not only functions as output at the current step, but will also be passed on to the next time step as input to support the learning of more complex sequential patterns as shown in Equation (6, 7). Thus, the output gate plays an important role in controlling how much internal information will be exposed to the next layer or time.

This study implemented two method architectures, namely Simple LSTM and Stacked LSTM, to compare their performance in predicting gold prices.

1. *Simple LSTM*

The Simple LSTM model architecture was designed to consist of a single layer as follows:

a.) Input Layer: Receives time-series data with an input shape of (60, 1), where 60 represents the number of historical time steps and 1 denotes a single input feature corresponding to the daily gold price.

b.) Single LSTM Layer: Consists of 50 LSTM units with return_sequences=False, such that only the final hidden state is propagated to the output layer. The LSTM layer employs the default internal activation functions, namely sigmoid functions for the input, forget, and output gates, and a tanh activation for updating the cell state.

c.) Dense Layer (Output): A fully connected Dense layer with one neuron and linear activation is used to generate a single continuous-valued prediction of the gold price for the next time step.

d.) Regularization: No dropout or additional regularization techniques are applied in the Simple LSTM model. This design choice is motivated by the relatively small number of trainable parameters in the single-layer architecture and the use of expanding window cross-validation, which inherently reduces the risk of overfitting.

2. *Stacked LSTM*

The Stacked LSTM model architecture was designed to consist of a three layer as follows:

a.) Input Layer: Receives data in the form (60, 1), according to the predetermined 60-day time step.

b.) There are three LSTM layers:

- First LSTM Layer: Consists of 50 units with return_sequences=True so that the entire output sequence from this layer can be passed as input to the next LSTM layer. A Dropout layer with a rate of 0.3 is applied to reduce overfitting.
- Second LSTM Layer: Consists of 50 units with return_sequences=True to pass its output to the final layer. The second dropout layer is applied again with a rate of 0.3.
- Third LSTM Layer: As the final LSTM layer, this layer consists of 50 units with return_sequences=False to return the final output of the sequence, summarizing information from all time steps. The third dropout layer is applied with a rate of 0.3.

c.) Dense Layer (Output): The output from the LSTM layers is passed through a fully connected Dense layer with 16 units utilizing the ReLU activation function. Finally, the Output Layer consists of a single neuron to produce the predicted gold price value.

To ensure model convergence, the training phase utilized a batch size of 32 and was executed for 50 epochs, as depicted in the loss analysis. Furthermore, a Rectified Linear Unit (ReLU) activation function was applied in the dense layer following the LSTM layers to introduce non-linearity and enhance feature representation after temporal dependency learning [26]. While LSTM cells internally rely on sigmoid and tanh activation functions to regulate memory flow, the use of ReLU in the post-LSTM dense layer improves convergence and nonlinear feature mapping, while the LSTM internal sigmoid–tanh gating preserves recurrent stability [27], [28], [29].

F. Model Evaluation

After the data has been processed and the model architecture designed, the next step is to train the model and run computational quantitative scenarios. The model evaluation process in this study is designed in two fundamental stages. The first stage is a comparative analysis to identify the most effective training optimizer from the four tested candidates. However, the performance of a training optimizer is not solely determined by the algorithm itself but also depends heavily on its hyperparameter settings. Comparative evaluations across optimizers reveal that optimizer choice and its hyperparameters significantly affect convergence and generalization in sequential models [30].

The main focus of the first stage is to systematically test and compare the performance of various training optimizers in training the designed Simple and Stacked LSTM models. Recent empirical studies comparing adaptive optimizers report notable differences in stability and final error metrics across tasks. This process utilizes various training optimizers such as Nesterov Adam (Nadam), Adaptive Moment (Adam), Adamax, and Root Mean Square Propagation (RMSprop) [31].

Root Mean Square Propagation (RMSprop) is a training optimizer designed to train RNNs using an adaptive learning rate. This method was developed as an extension of Adagrad to address its main weakness, namely the learning rate, which tends to decrease drastically over time [32].

Adam is a training optimizer that calculates an adaptive learning rate for each parameter. In addition to storing a moving average of the squares of past gradients (like Adadelta), Adam also stores a moving average of the gradient itself, which is similar to the concept of momentum. Adam combines the advantages of handling sparse gradients well with non-stationary environments [31].

Nesterov Adam (Nadam) is a modification of the Adam algorithm that incorporates Nesterov momentum. It applies a momentum step before calculating the gradient, allowing for more intelligent updates and often accelerating convergence [7].

Adamax is a variant of the Adam algorithm based on the infinity norm. It is a simple generalization of Adam that

makes it more stable and robust, especially in some scenarios [7].

The research proceeded to the second stage, a hyperparameter sensitivity analysis, in which different learning-rate values were systematically tested for each optimizer. This approach does not aim to automatically optimize hyperparameters, but rather to empirically evaluate how variations in learning rate influence model stability and prediction accuracy. Such controlled evaluation is widely used to identify robust training configurations in deep learning for non-stationary time series [15], [33].

MAE (Mean Absolute Error) is the average of the absolute differences between predicted and actual values. This metric measures the magnitude of the average error, regardless of whether the prediction is higher or lower than the actual value.

$$MAE = \frac{1}{m}\sum_{i=1}^{m}|X_i - Y_i \qquad (8)$$

RMSE (Root Mean Square Error) is the square root of the MSE. It is one of the most commonly used metrics in forecasting evaluation.

$$RMSE = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(X_i - Y_i)^2} \qquad (9)$$

MAPE (Mean Absolute Percentage Error) measures the prediction error as a percentage of the actual value. It is the average of the percentage errors of the actual values.

$$MAPE = \frac{100\%}{m}\sum_{i=1}^{m}\left|\frac{Y_i - X_i}{Y_i}\right| \qquad (10)$$

III. Result And Discussion

This section presents the experimental results and discussion of the forecasting models evaluated in this study. To provide a comprehensive performance context, conventional statistical models, namely ARIMA and SARIMA, are first reported as baseline benchmarks. Subsequently, the performance of the Simple LSTM and Stacked LSTM models is analyzed, with particular emphasis on the effects of optimizer selection and learning-rate configurations. As explained in the methodology section, the learning rate is a crucial hyperparameter influencing convergence behavior and forecasting accuracy; therefore, a systematic sensitivity analysis was conducted across multiple learning-rate values for both LSTM architectures. The comparative results of these experiments are summarized in Tables 2, 3, and 4.

Table 2 shows the forecasting results obtained from conventional statistical models, indicating that both ARIMA and SARIMA exhibit significantly higher error values compared to the deep learning model. Across all tested configurations, ARIMA produced an RMSE value of approximately 238.000 and a MAPE value of approximately 17%, indicating limited prediction accuracy for the gold price time series.

TABLE 2
AVERAGE FORECASTING ERRORS OF THE ARIMA & SARIMA

Model	Ordo	Test Error		
		RMSE	MAE	MAPE
ARIMA	1,1,1	238.582	218.658	17.07%
	2,1,2	238.714	218.788	17.08%
	1,1,0	238.544	218.621	17.06%
	0,1,1	238.544	218.621	17.06%
SARIMA	(1,1,1)(1,0,1,5)	238.519	218.602	17.06%
	(0,1,1)(1,0,1,5)	238.977	219.048	17.10%
	(1,1,0)(1,0,1,5)	239.013	219.083	17.10%
	(2,1,1)(1,0,1,5)	238.675	218.753	17.07%

An extension of SARIMA, incorporating a seasonal component to capture potential periodic patterns, did not result in a significant reduction in forecast error. These findings imply that, even when accounting for autoregressive, moving average, and seasonal structures, linear statistical models remain inadequate to represent the complex dynamics of Indonesian gold price movements over the 2021–2024 period.

TABLE 3
AVERAGE FORECASTING ERRORS OF THE SIMPLE LSTM

Optimizer	*LR*	Test Error		
		RMSE	MAE	MAPE
Adam	0.01	9.235	7.060	0.71%
Adamax	0.01	10.314	7.801	0.79%
Nadam	0.01	10.535	8.146	0.80%
Adam	0.001	11.490	8.690	0.86%
RMSprop	0.001	11.902	9.498	0.93%
Nadam	0.001	12.821	9.698	0.96%
RMSprop	0.01	13.497	11.388	1.10%
Adamax	0.001	13.974	10.782	1.09%
RMSprop	0.0001	17.350	13.446	1.33%
Nadam	0.0001	17.759	13.543	1.34%
Adamax	0.0001	19.587	15.014	1.50%
Adam	0.0001	24.453	19.686	1.87%

Table 3 presents the average forecasting errors of the Simple LSTM model across different optimizer and learning-rate configurations. The results demonstrate that both optimizer choice and learning-rate setting influence model performance. Among all evaluated configurations, the Adam optimizer with a learning rate of 0.01 achieves the lowest RMSE, MAE, and MAPE values, indicating superior

forecasting accuracy and stable convergence behavior. As the learning rate decreases, the forecasting errors consistently increase across all optimizers, suggesting slower convergence and reduced adaptability to rapid price fluctuations. These findings indicate that the Simple LSTM architecture is highly sensitive to learning-rate selection, and that an appropriately chosen learning rate is critical for achieving optimal performance in volatile gold price forecasting.

TABLE 4
AVERAGE FORECASTING ERRORS OF THE STACKED LSTM

Optimizer	*LR*	Test Error		
		RMSE	MAE	MAPE
Adam	0.01	12.063	9.469	0.92%
Nadam	0.01	12.946	10.342	0.99%
Adamax	0.01	13.518	10.418	1.04%
Adam	0.001	14.202	10.571	1.06%
Adamax	0.001	15.660	11.935	1.20%
RMSprop	0.001	17.734	14.816	1.40%
Nadam	0.001	18.829	15.482	1.46%
RMSprop	0.01	19.789	15.679	1.57%
Adam	0.0001	23.790	18.452	1.79%
Nadam	0.0001	23.853	19.337	1.89%
Adamax	0.0001	33.680	26.630	2.54%
RMSprop	0.0001	38.748	32.928	2.88%

Table 4 reports the forecasting performance of the Stacked LSTM model under the same optimizer and learning-rate configurations. Although the Adam optimizer with a learning rate of 0.01 again produces the lowest error values within this architecture, the overall forecasting accuracy of the Stacked LSTM remains inferior to that of the Simple LSTM model. In particular, deeper architectures exhibit higher RMSE, MAE, and MAPE values, especially when combined with smaller learning rates. This pattern suggests that the increased model complexity amplifies sensitivity to hyperparameter settings and may lead to unstable generalization when applied to highly volatile gold price data.

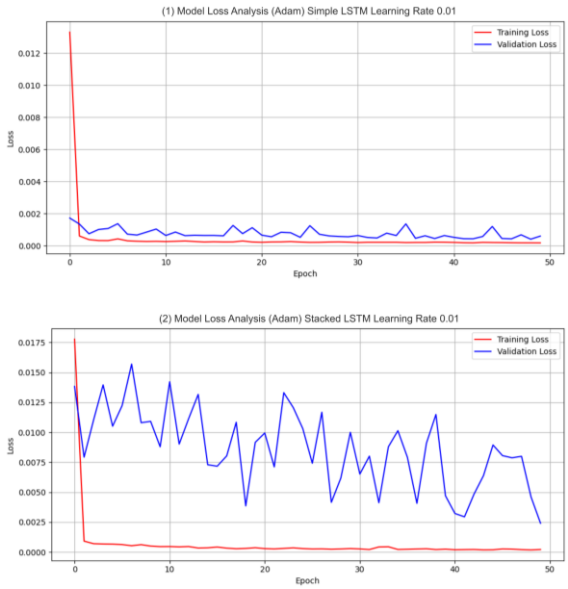

Figure 6. Illustrate the time-to-time Loss Analysis.

Figure 6 presents the training and validation loss curves for the best-performing configuration (Adam optimizer with a learning rate of 0.01) using both Simple LSTM and Stacked LSTM models. In the Simple LSTM model, the training and validation loss curves decrease smoothly and remain closely aligned, indicating stable learning and good generalization without significant overfitting. In contrast, the Stacked LSTM shows a sharp decrease in training loss while the validation loss fluctuates and remains relatively high, especially toward the later training stages. This divergence between training and validation loss suggests that the deeper architecture fits the training data well but struggles to generalize to unseen data, which is a typical indication of overfitting in complex neural networks.

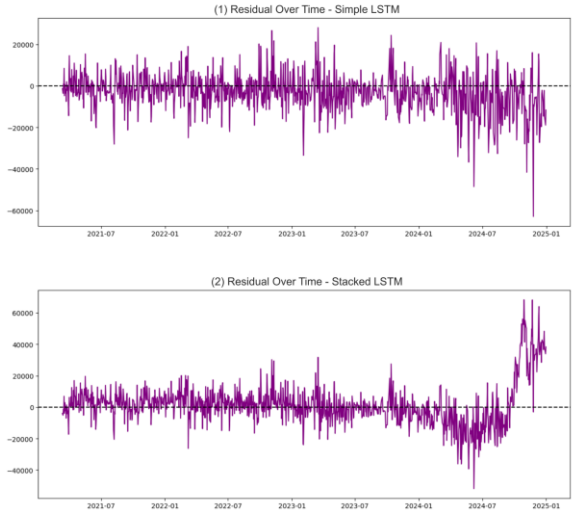

Figure 7. Shows the residual over time (1) Simple LSTM and (2) Stacked LSTM.

The model error consistency analysis is visualized through the residual plot in Figure 7. In graph (1), the Simple LSTM residual fluctuates randomly around the zero line from the beginning to the end of the period, indicating that the model is able to capture the data trend with minimal bias. This contrasts with graph (2) of the Stacked LSTM, which, although stable at the beginning, experiences a drastic error spike at the end of the data period (end of 2024). These extreme negative and positive spikes indicate that the deeper architecture actually experiences significant lag or delay in response when faced with dynamic changes in gold price trends in the future.

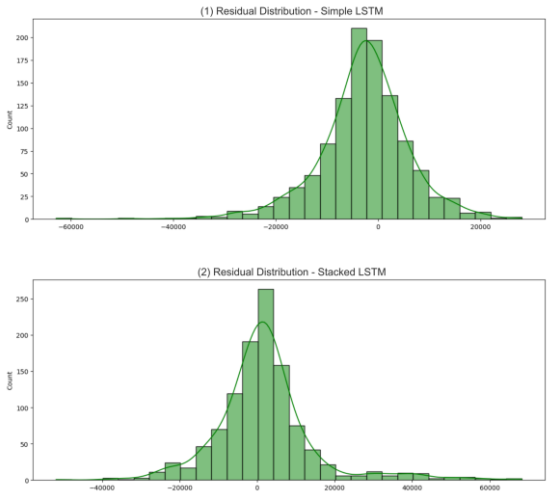

Figure 8. Shows the residual distribution.

Statistical validation of the errors is strengthened by Figure 8, which displays the histograms of the residual distributions of the best-performing configuration of both models. The Simple LSTM model (1) produces a symmetrical bell-shaped distribution curve centered on zero, confirming that most predictions have a high level of accuracy with minimal error. On the other hand, although the residual distribution of the Stacked LSTM (2) is also close to normal, it shows a wider tail spread. This confirms that the Stacked model has a higher frequency of large prediction errors than the Simple model, making it less reliable for precision forecasting.

TABLE 5
RESULTS OF THE WILCOXON SIGNED RANK TEST FOR MODEL COMPARISON

	Optimizer	p-value
Simple LSTM	Adam vs Nadam	0.18750
	Adam vs Adamax	0.18750
	Adam vs RMSprop	0.06250
Stacked LSTM	Adam vs Nadam	0.00001
	Adam vs Adamax	0.00001
	Adam vs RMSprop	0.00078

The model comparison results presented in Table 5 show that, for the Simple LSTM model, all pairwise comparisons between the Adam optimizer and the other optimizers yield p-values greater than 0.05, indicating that the observed differences in forecasting errors are not statistically significant. In contrast, the Stacked LSTM model exhibits statistically significant differences across all optimizer comparisons, as reflected by p-values below 0.05, suggesting that optimizer selection plays a more important role in deeper LSTM architectures. However, statistical significance only indicates the presence of performance differences and does not imply superiority; therefore, the direction of model performance is determined based on quantitative error metrics, namely RMSE, MAE, and MAPE, which consistently show that the Adam optimizer produces the lowest forecasting errors in this study.

Based on the comparison of the results of table 3 and table 4, the results show that Simple LSTM gets results with smaller values, which means the accuracy of the results is close to the actual value, which can be seen in table 3, that Adam optimizer with a learning rate of 0.01 is superior and has an RMSE value of 9.235, MAE: 7.060, and MAPE: 0.71%. Meanwhile, for the results of Stacked LSTM which can be seen in table 4, the results show that Adam optimizer with a learning rate of 0.01 has an RMSE value of 12,063, MAE: 9,469, and MAPE: 0.92%, which is greater than the Simple LSTM model.

Figure 9. Actual vs predicted price chart from Simple LSTM model all optimizer.

Figure 10. Actual vs predicted price graph of the best Simple LSTM model.

Figures 9 and 10 present a direct comparison between the actual gold prices and the predicted values generated by the Simple LSTM model. The close alignment between the predicted curve and the actual price trajectory indicates that

the model successfully captures both the overall trend and short-term fluctuations of the gold price time series. Minor deviations are primarily observed during periods of abrupt price changes, reflecting the inherent volatility of financial markets. Overall, this visualization supports the quantitative results in Table 3, where the Simple LSTM achieved the lowest RMSE, MAE, and MAPE values among all evaluated configurations.

Figure 11. 60-day prediction results of the best Simple LSTM optimizer adam with a learning rate of 0.01.

Figure 11 illustrates the 60-day forecasting results produced by the best-performing Simple LSTM configuration using the Adam optimizer with a learning rate of 0.01. The forecasted trajectory shows a smooth continuation of the historical price pattern, indicating that the model maintains temporal consistency beyond the training horizon. This visualization demonstrates the model's capability for short-term forward-looking prediction, complementing the historical actual-versus-predicted comparison.

Figure 12. Actual vs predicted price chart from Stacked LSTM model all optimizer.

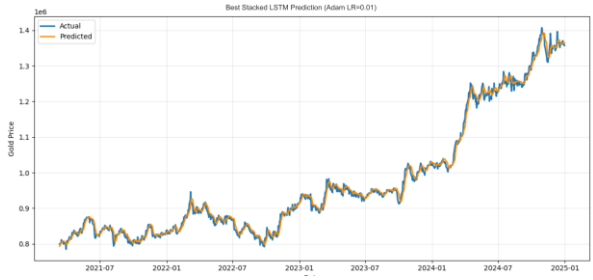

Figure 13. Actual vs predicted price graph of the best Stacked LSTM model.

Figures 12 and 13 compare the actual gold prices and the predictions generated by the Stacked LSTM model. The prediction curves are able to follow the general trend of the actual price movement; however, noticeable deviations are observed, particularly during periods of sharp price fluctuations. These discrepancies indicate that while the Stacked LSTM captures long-term patterns, it is less effective in adapting to sudden changes in volatile gold price time series, resulting in reduced prediction accuracy compared to the Simple LSTM model.

Figure 14. 60-day prediction results of the best Stacked LSTM optimizer adam with a learning rate of 0.01.

Figure 14 presents the 60-day forecasting results generated by the best-performing Stacked LSTM configuration using the Adam optimizer with a learning rate of 0.01. The forecasting curve generally follows the long-term direction of the historical gold price trend, indicating that the model is capable of capturing global temporal patterns. However, compared to the Simple LSTM forecast, the Stacked LSTM exhibits higher variability and a smoother delayed response to recent price movements. This behavior suggests that the deeper architecture, while effective in learning hierarchical temporal representations, is less adaptive to short-term fluctuations in volatile gold price data. These characteristics are consistent with the higher RMSE, MAE, and MAPE values reported in Table 4, confirming that increasing model depth does not necessarily lead to improved forecasting accuracy in this context.

The prediction results with a very low average error rate, namely RMSE: 9.235, MAE: 7.060 and MAPE of 0.71%, indicate that the proposed LSTM model has potential practical applications in the context of investment decision making and financial policy. Operationally, the prediction error rate below 1% indicates that the average prediction deviation is relatively small when compared to actual gold price fluctuations, so this model can be used as a tool in planning short to medium-term investment strategies with controlled prediction risks.

The finding that the Simple LSTM consistently outperforms the Stacked LSTM highlights that increased architectural depth does not necessarily lead to better forecasting accuracy for volatile financial time series. Although the Stacked LSTM theoretically offers higher representational capacity, the experimental results indicate that the deeper architecture is more prone to overfitting. This behavior is evident from the divergence between training and validation loss as well as the higher variability observed in the

residual analysis, suggesting that the Stacked LSTM tends to capture noise and short-lived patterns in the training data rather than generalizable temporal structures.

Furthermore, the relative performance difference can be explained by the interaction between model complexity and dataset characteristics. The daily gold price dataset from 2021–2024, while volatile, remains limited in size for effectively training deep multi-layer architectures. Under such conditions, the simpler architecture of the Simple LSTM, with fewer trainable parameters, achieves a more favorable bias–variance trade-off and demonstrates more stable generalization performance. These results suggest that for non-stationary financial time series with moderate data availability, simpler LSTM architectures may provide more reliable and robust forecasts than deeper stacked models.

In an investment context, accurate gold price predictions have the potential to provide additional information for making buy or sell decisions, particularly during periods of high market volatility. Predictive information with a low error rate can help investors anticipate price changes without relying entirely on conventional technical indicators, thereby improving the timing of market entry and exit.

From an economic policy perspective, stable gold price projections can also serve as a reference in monitoring global and domestic commodity market dynamics, as well as in designing fiscal or monetary policies that consider the impact of safe-haven asset price movements on investor and market participant responses. However, these predictions should be viewed as a decision-making tool and not solely the basis for policy, given that external factors such as global macroeconomic conditions and market shocks can significantly impact gold prices.

IV. CONCLUSION

This study concludes that the Simple LSTM model trained using the Adam optimizer and a learning rate of 0.01 delivers the most effective performance for predicting Indonesian gold prices from 2021 to 2024. The model achieved the lowest error values with RMSE of 9235, MAE of 7060, and MAPE of 0.71%, outperforming the deeper Stacked LSTM architecture across all evaluation metrics.

The main contribution of this research lies in demonstrating that simpler deep learning architectures, when combined with carefully evaluated optimizer and learning-rate configurations, can outperform deeper models for volatile financial time series. This study highlights the importance of systematic experimental evaluation rather than relying solely on model depth or default training settings.

The insights from this study provide practical guidance for analysts, investors, and researchers in choosing efficient and computationally feasible forecasting models. Future work may incorporate additional macroeconomic indicators, employ hybrid or ensemble models, or expand the dataset to further enhance prediction stability and accuracy.

REFERENCES

[1] S. Sathyanarayana dan T. Mohanasundaram, "The Surge in Gold Price Volatility: Macroeconomic Drivers, Geopolitical Risk, and Market Dynamics," *IRA-International J. Manag. Soc. Sci. (ISSN 2455-2267)*, vol. 21, no. 1, hal. 15, Apr 2025, doi: 10.21013/jmss.v21.n1.p2.

[2] H. M. Zangana dan S. Ramadan, "Deep Learning - based Gold Price Prediction : A Novel Approach using Time Series Analysis," *J. Sist. Informasi*, vol. 13, hal. 2581–2591, 2024, doi: https://doi.org/10.32520/stmsi.v13i6.4651.

[3] I. D. Mienye, T. G. Swart, dan G. Obaido, "Recurrent Neural Networks : A Comprehensive Review of Architectures , Variants , and Applications," *Information*, hal. 1–34, 2024, doi: https://doi.org/10.3390/info15090517.

[4] X. Kong *et al.*, *Deep learning for time series forecasting : a survey*, vol. 16, no. 7. Springer Berlin Heidelberg, 2025. doi: 10.1007/s13042-025-02560-w.

[5] N. Chen, "Exploring the development and application of LSTM variants," *Appl. Comput. Eng.*, vol. 0, hal. 103–107, 2024, doi: 10.54254/2755-2721/53/20241288.

[6] S. Hochreiter dan J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, hal. 1735–1780, 1997, doi: 10.1162/neco.1997.9.8.1735.

[7] B. J. Kim dan I.-W. Nam, "A Review of Hybrid LSTM Models in Smart Cities," *Processes*, vol. 13, no. 7, hal. 1–46, 2025, doi: 10.3390/pr13072298.

[8] H. Oukhouya dan E. Khalid, "A comparative study of ARIMA, SVMs, and LSTM models in forecasting the Moroccan stock market," *Int. J. Simul. Process Model.*, vol. 20, no. 2, hal. 125–143, 2023, doi: 10.1504/IJSPM.2023.136481.

[9] Y. R. Madhika, Kusrini, dan T. Hidayat, "Gold Price Prediction Using the ARIMA and LSTM Models," *Sinkron*, vol. 8, no. 3, hal. 1255–1264, 2023, doi: 10.33395/sinkron.v8i3.12461.

[10] T.-Y. Kim, H.-S. Yun, H.-M. Yoon, dan S.-J. Lee, "Comparative Analysis and Validation of LSTM and GRU Models for Predicting Annual Mean Sea Level in the East Sea: A Case Study of Ulleungdo Island," *Appl. Sci.*, vol. 15, no. 20, hal. 1–19, 2025, doi: 10.3390/app152011067.

[11] S. Zahara, Sugianto, dan M. B. Ilmiddafiq, "Prediksi Indeks Harga Konsumen Menggunakan Metode Long Short Term Memory (LSTM) Berbasis Cloud Computing," *J. RESTI (Rekayasa Sist. dan Teknol. Informasi)*, vol. 3, no. 3, hal. 357–363, 2019, doi: 10.29207/resti.v3i3.1086.

[12] Mahendra, M. M. Amalia, dan H. Leon, "Analisis Pengaruh Suku Bunga, Harga Minyak Dunia, Harga Emas Dunia Terhadap Indeks Harga Saham Gabungan Dengan Inflasi Sebagai Variabel Moderating Di Indonesia," *Owner*, vol. 6, no. 1, hal. 1069–1082, 2022, doi: 10.33395/owner.v6i1.725.

[13] S. M. Hasanat *et al.*, "Enhancing Load Forecasting Accuracy in Smart Grids: A Novel Parallel Multichannel Network Approach Using 1D CNN and Bi-LSTM Models," *Int. J. Energy Res.*, vol. 2024, 2024, doi: 10.1155/2024/2403847.

[14] Y. Ye, "Improved Gold Price Prediction Based on the LSTM-ARIMA Hybrid Model," *Appl. Comput. Eng.*, vol. 165, no. 1, hal. 56–65, 2025, doi: 10.54254/2755-2721/2025.ld24518.

[15] M. R. Nurhambali, Y. Angraini, dan A. Fitrianto, "Implementation of Long Short-Term Memory for Gold Prices Forecasting," *Malaysian J. Math. Sci.*, vol. 18, no. 2, hal. 399–422, 2024, doi: 10.47836/mjms.18.2.11.

[16] L. O. Joel, W. Doorsamy, dan B. S. Paul, "A Review of Missing Data Handling Techniques for Machine Learning," *Int. J. Innov. Technol. Interdiscip. Sci.*, vol. 5, no. 3, hal. 971–1005, 2022, doi: https://doi.org/10.15157/IJITIS.2022.5.3.971-1005.

[17] K. Oskam, *Driving Change : Time Series Forecasting For Electric Vehicle Adoption In The Netherlands*. 2024.

[18] R. S. Santos, M. A. Ponti, dan K. R. Rodrigues, "Analyzing college student dropout risk prediction in real data using walk-forward validation," *Intell. Syst.*, hal. 291–305, 2023.

[19] H. Hewamalage, C. Bergmeir, dan K. Bandara, "Recurrent Neural Networks for Time Series Forecasting: Current status and future

directions," *Int. J. Forecast.*, vol. 37, no. 1, hal. 388–427, 2021, doi: 10.1016/j.ijforecast.2020.06.008.

[20] I. M. Vitor Cerqueira, Luis Torgo, *Evaluating time series forecasting models: an empirical study on performance estimation methods*, vol. 109, no. 11. Springer US, 2020. doi: 10.1007/s10994-020-05910-7.

[21] S. Wang, K. Li, Y. Chen, dan X. Tang, "VIX constant maturity futures trading strategy: A walk-forward machine learning study," *PLoS One*, vol. 19, no. 4 April, hal. 1–22, 2024, doi: 10.1371/journal.pone.0302289.

[22] P. J. M. Ali, "Investigating the Impact of Min-Max Data Normalization on the Regression Performance of K-Nearest Neighbor with Different Similarity Measurements," *ARO-The Sci. J. Koya Univ.*, vol. 10, no. 1, hal. 85–91, 2022, doi: 10.14500/aro.10955.

[23] L. Sasse *et al.*, "On Leakage in Machine Learning Pipelines," *J. Big Data*, 2023, doi: https://doi.org/10.1186/s40537-025-01193-8.

[24] A. Pranolo *et al.*, "Enhanced Multivariate Time Series Analysis Using LSTM: A Comparative Study of Min-Max and Z-Score Normalization Techniques," *Ilk. J. Ilm.*, vol. 16, no. 2, hal. 210–220, 2024, doi: 10.33096/ilkom.v16i2.2333.210-220.

[25] A. Suominen, "Deep Learning for Cryptocurrency Price Prediction-LSTM Model Performance Comparison and Evaluation Deep Learning for Cryptocurrency Price Prediction-LSTM Model Performance Comparison and Evaluation," 2024.

[26] H. Huang, Z. Wang, Y. Liao, W. Gao, C. Lai, dan X. Wu, "Ecological Informatics Improving the explainability of CNN-LSTM-based flood prediction with integrating SHAP technique," *Ecol. Inform.*, vol. 84, no. July, 2024, doi: https://doi.org/10.1016/j.ecoinf.2024.102904.

[27] J. Yuan, H. Dengxin, W. Yufeng, Y. Xueting, D. Huige, dan Y. Qing, "Attention mechanism based CNN-LSTM hybrid deep learning model for atmospheric ozone concentration prediction," *Sci. Rep.*, hal. 1–16, 2025, doi: https://doi.org/10.1038/s41598-025-05877-2 1.

[28] S. Ghimire, R. C. Deo, D. Casillas-pérez, dan S. Salcedo-sanz, "Deep learning CNN-LSTM-MLP hybrid fusion model for feature optimizations and daily solar radiation prediction," *Measurement*, vol. 202, no. August, hal. 111759, 2022, doi: 10.1016/j.measurement.2022.111759.

[29] A. Grenyer, O. Schwabe, J. A. Erkoyuncu, dan Y. Zhao, "Multistep prediction of dynamic uncertainty under limited data," *CIRP J. Manuf. Sci. Technol.*, vol. 37, hal. 37–54, 2022, doi: 10.1016/j.cirpj.2022.01.002.

[30] W. Riyadi dan Jasmir, "Comparative Analysis of Optimizer Effectiveness in GRU and CNN-GRU Models for Airport Traffic Prediction," *J. Ilm. Tek. Elektro Komput. dan Inform.*, vol. 10, no. 3, hal. 580–593, 2024, doi: 10.26555/jiteki.v10i3.29659.

[31] B. Radomirovic *et al.*, "Optimizing long-short term memory neural networks for electroencephalogram anomaly detection using variable neighborhood search with dynamic strategy change," *Complex Intell. Syst.*, vol. 10, no. 6, hal. 7987–8009, 2024, doi: 10.1007/s40747-024-01592-z.

[32] R. Elshamy, O. Abu-Elnasr, M. Elhoseny, dan S. Elmougy, "Improving the efficiency of RMSProp optimizer by utilizing Nestrove in deep learning," *Sci. Rep.*, vol. 13, no. 1, hal. 1–16, 2023, doi: 10.1038/s41598-023-35663-x.

[33] R. Taylor, V. Ojha, I. Martino, dan G. Nicosia, "Sensitivity Analysis for Deep Learning : Ranking Hyper-parameter Influence," *Int. Conf. Tools with Artif. Intell.*, hal. 512–516, 2021, doi: 10.1109/ICTAI52525.2021.00083.