



Siu King Sum

 **Complete Source Code and Model Output** — is available on GitHub:
 <https://github.com/siiiiiiiiiii/LSTM-with-Attention-for-Log-Return-Prediction/tree/main>

LSTM Attention-Based Stock Return Prediction Model Report

Model Architecture Overview

Layer (type)	Output Shape	Param #
input_layer_4 (InputLayer)	(None, 20, 10)	0
lstm_4 (LSTM)	(None, 20, 32)	5,504
dropout_10 (Dropout)	(None, 20, 32)	0
attention_4 (Attention)	(None, 32)	1,088
dropout_11 (Dropout)	(None, 32)	0
dense_8 (Dense)	(None, 16)	528
dropout_12 (Dropout)	(None, 16)	0
dense_9 (Dense)	(None, 1)	17

Total params: 7,137 (27.88 KB)

Trainable params: 7,137 (27.88 KB)

The model constructed in this experiment takes as input a 20-day multivariate sequence composed of 10 engineered features, which include price-based and technical indicators. This is clearly supported by the model’s input shape of `(None, 20, 10)`, as shown in the architecture summary.

The first processing layer is an LSTM with 32 hidden units, designed to extract temporal dependencies across the input sequence. The corresponding layer, `lstm_4`, produces outputs with a shape of `(None, 20, 32)`, indicating that it returns a hidden state for each of the 20 time steps.

To mitigate overfitting, a Dropout layer with a 30% rate (`dropout_10`) is applied immediately after the LSTM. This is followed by a custom-built Attention mechanism (`attention_4`), which receives the full sequence of LSTM outputs and computes importance weights for each time step. Internally, this layer applies a softmax over transformed hidden states

to generate attention scores, and then performs a weighted sum to obtain a single context vector. The output shape of the Attention layer is `(None, 32)`, and it contains 1,088 trainable parameters, as confirmed by the model summary.

Another Dropout layer with the same dropout rate (`dropout_11`) is applied after the Attention output. The resulting feature vector is passed to a Dense layer with 16 neurons, ReLU activation, and L2 regularization (with a coefficient of $1e-3$). This fully connected layer (`dense_8`) transforms the context vector into a compact representation that captures the most relevant patterns for prediction.

Finally, a linear Dense layer with a single output (`dense_9`) is used to predict the log return for the next day. This final output has a shape of `(None, 1)`.

In total, the model has 7,137 trainable parameters, making it both compact and efficient. Multiple regularization techniques are applied throughout the architecture: Dropout, L2 regularization, and training-time callbacks such as `EarlyStopping` and `ReduceLROnPlateau`. Together, these ensure the model maintains high generalization ability while avoiding overfitting.

Training Process and Loss Analysis

During training, the model utilized the Huber loss function ($\delta = 0.5$) along with the Adam optimizer (initial learning rate of $1e-3$). When the validation loss plateaued, the learning rate was halved using the `ReduceLROnPlateau` strategy, with a minimum threshold of $1e-6$. If the validation loss showed no improvement for five consecutive epochs, `EarlyStopping` was triggered and the best-performing weights were restored.

After the first epoch, the training loss was 0.0404 and the validation loss was 0.0253. By the fifth epoch, these had dropped to 0.0051 and 0.0036, respectively. Around the 50th epoch, both the training and validation losses stabilized at approximately 0.00017 (Figure 2).

Figure 2

Epoch

```

Epoch 1/50
63/63 ————— 3s 13ms/step - loss: 0.0404 - val_loss: 0.0253 - learning_rate: 0.0010
Epoch 2/50
63/63 ————— 1s 9ms/step - loss: 0.0225 - val_loss: 0.0152 - learning_rate: 0.0010
Epoch 3/50
63/63 ————— 1s 20ms/step - loss: 0.0135 - val_loss: 0.0092 - learning_rate: 0.0010
Epoch 4/50
63/63 ————— 1s 10ms/step - loss: 0.0082 - val_loss: 0.0057 - learning_rate: 0.0010
Epoch 5/50
63/63 ————— 1s 20ms/step - loss: 0.0051 - val_loss: 0.0036 - learning_rate: 0.0010
Epoch 6/50
63/63 ————— 0s 7ms/step - loss: 0.0033 - val_loss: 0.0024 - learning_rate: 0.0010
Epoch 7/50
63/63 ————— 1s 17ms/step - loss: 0.0021 - val_loss: 0.0016 - learning_rate: 0.0010
Epoch 8/50
63/63 ————— 2s 24ms/step - loss: 0.0014 - val_loss: 0.0011 - learning_rate: 0.0010
Epoch 9/50
63/63 ————— 1s 19ms/step - loss: 9.7404e-04 - val_loss: 7.4916e-04 - learning_rate: 0.0010
Epoch 10/50
63/63 ————— 1s 11ms/step - loss: 6.8870e-04 - val_loss: 5.4522e-04 - learning_rate: 0.0010
Epoch 11/50
63/63 ————— 1s 12ms/step - loss: 5.0493e-04 - val_loss: 4.1287e-04 - learning_rate: 0.0010
Epoch 12/50
63/63 ————— 1s 8ms/step - loss: 3.8542e-04 - val_loss: 3.2621e-04 - learning_rate: 0.0010
Epoch 13/50
...
Epoch 49/50
63/63 ————— 1s 10ms/step - loss: 1.7246e-04 - val_loss: 1.7605e-04 - learning_rate: 1.0000e-06
Epoch 50/50
63/63 ————— 1s 9ms/step - loss: 1.7243e-04 - val_loss: 1.7602e-04 - learning_rate: 1.0000e-06

```

The training and validation loss curves were closely aligned throughout the process and converged at an extremely low loss level, indicating that the model had strong fitting ability without exhibiting signs of overfitting (Figure 3).

Figure 3

Train vs. Validation Loss

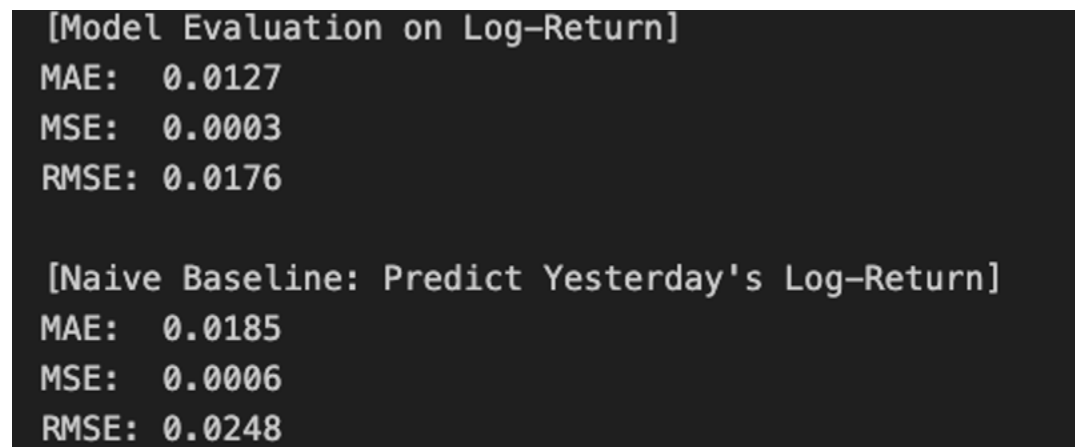


Predictive Performance and Baseline Comparison

On the test set, the model achieved a Mean Absolute Error (MAE) of 0.0127 and a Root Mean Square Error (RMSE) of 0.0176. In contrast, a simple “naive baseline” that uses the previous day's log return as the prediction yielded an MAE of 0.0185 and an RMSE of 0.0248. This demonstrates that the model outperforms the baseline by approximately 31% in MAE and 29% in RMSE (Figure 4).

Figure 4

The model outperforms the baseline by approximately 31% in MAE and 29% in RMSE

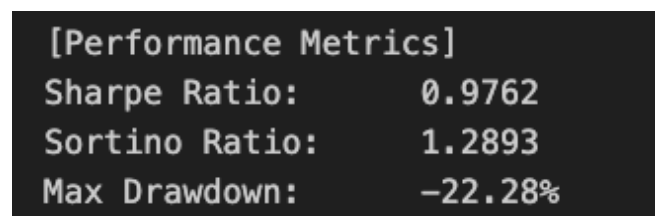


[Model Evaluation on Log-Return]	
MAE:	0.0127
MSE:	0.0003
RMSE:	0.0176
[Naive Baseline: Predict Yesterday's Log-Return]	
MAE:	0.0185
MSE:	0.0006
RMSE:	0.0248

Using a signal-based strategy—going long on predicted positive returns and short on negative ones—the model achieved an annualized Sharpe ratio of 0.9762 and a Sortino ratio of 1.2893. The maximum drawdown was around -22.28%, which is a relatively strong performance given the highly volatile nature of log-return prediction in stock prices (Figure 5).

Figure 5

Performance



[Performance Metrics]	
Sharpe Ratio:	0.9762
Sortino Ratio:	1.2893
Max Drawdown:	-22.28%

From the predicted curve, the model captures small fluctuations across most periods. However, it still struggles to fully model extreme jumps, indicating that it performs better at smoothing return trends than at forecasting sudden spikes (Figure 6).

Figure 6

Log-Return Prediction



Key Code Interpretation

In the data preprocessing stage, the model first downloads Google's (GOOG) stock data from Yahoo Finance, including open, high, low, close, and volume. It then computes daily log returns and adds a total of 10 features, including a 7-day moving average, 14-day Relative Strength Index (RSI), Moving Average Convergence Divergence (MACD), 5-day momentum, and 10-day volatility.

After feature normalization, a sliding window function reshapes the time series into model input tensors with a shape of (number of samples, 20, 10), representing sequences of 20 days and 10 features, along with corresponding target log returns.

In the model-building function `build_model`, the LSTM layer includes L2 regularization to prevent excessively large weights. The custom Attention layer performs a linear transformation on each time step's hidden state, calculates attention scores, and aggregates the sequence into a weighted sum based on importance. The dense layer also includes L2 regularization and is followed by Dropout to enhance generalization.

During training, two key callbacks—ReduceLROnPlateau and EarlyStopping—ensure dynamic learning rate adjustment and automatic restoration of the best-performing weights.

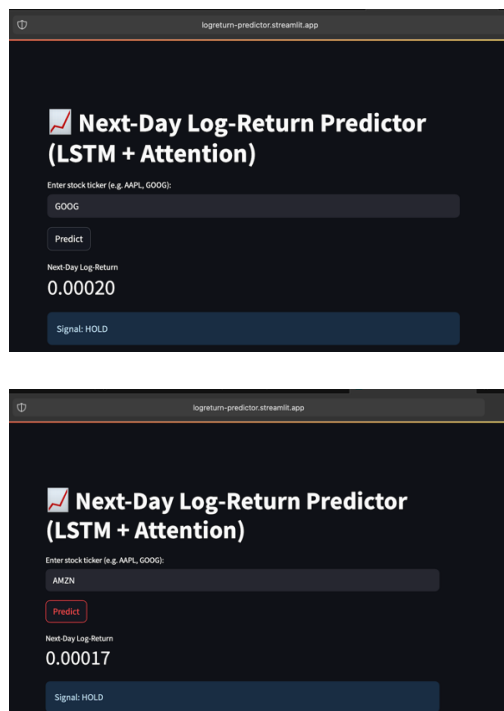
Overall, the model effectively handles the noisy nature of log return sequences by combining multiple regularization techniques with an attention mechanism. It demonstrates a significant performance advantage over the naive baseline in both error metrics and risk-adjusted return indicators.

Deployment Attempt and Limitation in Practical Use

At the final stage of this project, a Streamlit web application was developed to deploy the trained model and provide interactive predictions. However, the deployment revealed a critical limitation in the model's practical usability. Specifically, the predicted log-returns were consistently around 0.000xx, regardless of the input conditions. This overly conservative behavior is likely the result of the model being trained with MSE-based objectives and strong regularization, which prioritize minimizing average error rather than making bold directional forecasts.

Figure 7

Streamlit



As a consequence, while the model performed well in evaluation metrics and beat the naive baseline, it ultimately failed to deliver actionable signals in real-world scenarios. The predicted values were too small and flat to generate meaningful trading decisions or distinguishable long/short signals. This highlights a key lesson in financial modeling: performance metrics alone are insufficient—the model must also be interpretable and practically useful, especially when integrated into decision-making systems such as trading dashboards.