## **Problem Statement:**

The objective of this project is to build a deep learning model to predict the **adjusted closing price** of **Apple (AAPL)** stock using historical data. Given the sequential nature of stock prices, we utilize **Recurrent Neural Networks (RNNs)** and **Long Short-Term Memory (LSTM)** networks to model trends and forecast price behavior over **1-day**, **5-day**, and **10-day** intervals.

The dataset includes:

* **Date, Open, High, Low, Close, Adjusted Close, Volume**

Key steps:

* Handle missing values specific to time-series data
* Use **Adjusted Close** as the target variable
* Scale features and create input sequences
* Build and tune **SimpleRNN** and **LSTM** models using **GridSearchCV**
* Evaluate and compare models using MSE, MAE, and RMSE

The project supports use cases in **trading**, **risk management**, **financial forecasting**, and **deep learning research**.

**Approach**

**1.Load the Dataset:** The historical stock price data for Apple is imported using a pandas DataFrame from a CSV file (AAPL.csv). Initial exploration is performed to understand the structure and contents of the dataset.

**2. Data Cleaning and preprocessing:**

**Missing Data:** The dataset had only one row with missing data, which was dropped to maintain data integrity.

**Date Format:** The Date column was initially in object (string) format, so it was converted to datetime using pd.to\_datetime() for proper time-series handling.

**Duplicates:** There were no duplicate rows or columns present in the dataset.

**3. Feature Selection:**

'Adj Close' (Adjusted Close) column is selected as target feature. This column shows the stock’s closing price after adjusting for actions like dividends and stock splits, making it more accurate for analysis.We didn’t use the other columns because our goal is to predict the future adjusted closing price based on its past values.

### **4. Scaling the Data**

To help the deep learning models learn better and faster, we normalized the stock prices using MinMaxScaler. This scaled the values of the 'Adj Close' column to a range between 0 and 1.Scaling is important because it brings all the data to the same level, which improves model performance and training speed.

**5. Creating Time-Series Sequences**

To train the models on time-series data, we created sequences of past stock prices. Specifically, we used the past **60 days** of stock prices to predict the adjusted closing price for the next **1 day, 5 days, and 10 days**. This method helps the model understand patterns over time and make forecasts for multiple future time points. We then split the data into training and testing sets and reshaped it to fit the input requirements of Recurrent Neural Networks, where the input is in the form of *(samples, time steps, features)*.

**6. Model Development:**

**Creating simple RNN model and performing Hyperparameter optimisation:**

Built a deep learning model using Simple Recurrent Neural Networks (SimpleRNN) to forecast Apple’s adjusted closing stock price for the next 1 day, 5 days, and 10 days.

The model architecture included:

* Two SimpleRNN layers to learn from sequential stock price data.
* A Dropout layer to reduce overfitting.
* A Dense output layer with three neurons to predict stock prices for 3 future time points.

We trained the model using the Adam optimizer and used Mean Squared Error (MSE) as the loss function, with Mean Absolute Error (MAE) as an additional performance metric.

To prevent overfitting, early stopping was applied. This monitored the validation loss and stopped training when the model stopped improving.

After training, we evaluated the model using the test data. The results included:

* Test Loss (MSE): Indicates how well the model performed in terms of squared error.
* Test MAE: Shows the average absolute difference between predicted and actual values.
* RMSE (Root Mean Squared Error): Provides insight into the model's prediction error in the same units as the stock price.

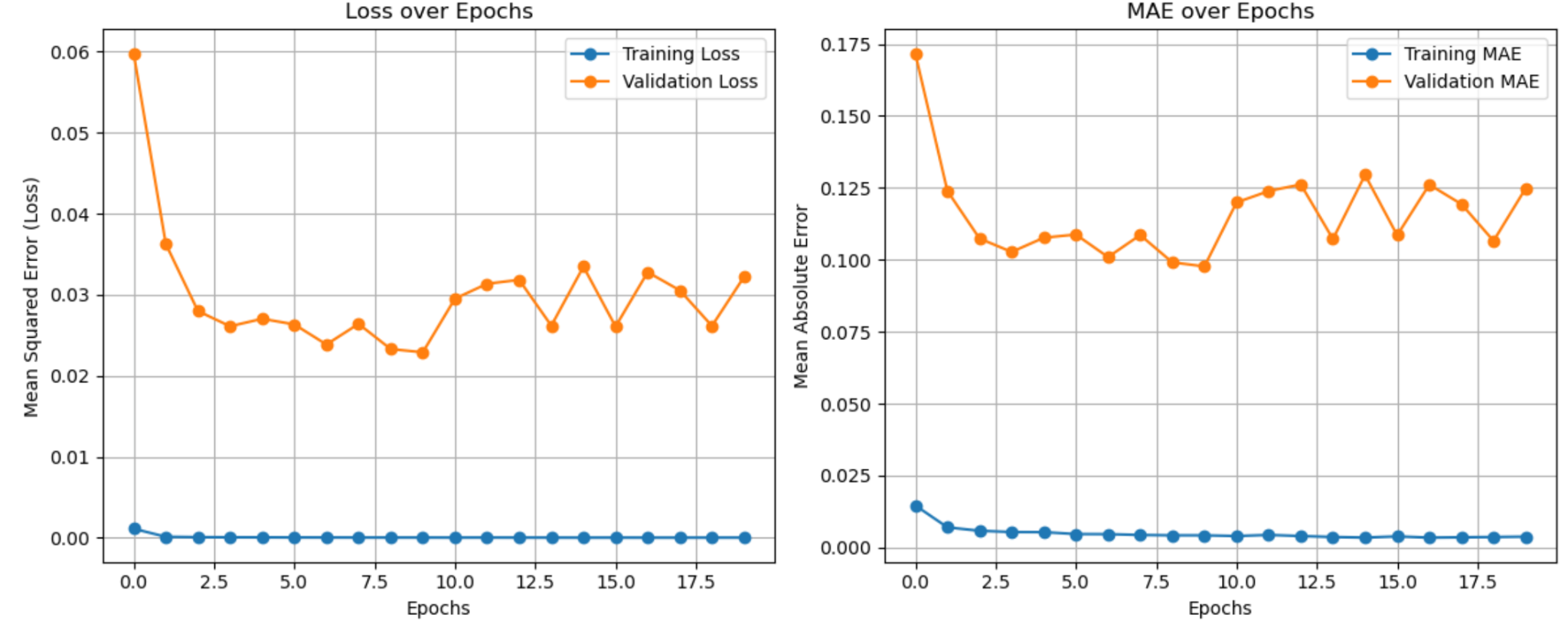
### **Results**

After training and evaluating the **Simple RNN model**, the following results were obtained:

* **Test Loss (Mean Squared Error):** 0.0229  
   This value represents how much the predicted stock prices deviated from the actual values on average, measured in terms of squared differences.
* **Test Mean Absolute Error (MAE):** 0.0977  
   This metric indicates the average absolute difference between the predicted and actual adjusted closing stock prices, which helps to understand how close the predictions are on average.
* **Root Mean Squared Error (RMSE):** 0.1513  
   RMSE measures the square root of the average squared differences between predicted and actual values. It provides an insight into how large the prediction errors are, with this value showing the error magnitude in the same scale as the stock prices.

These results suggest that while the model can predict Apple stock prices, there is still room for improvement, especially in reducing prediction error.

Below is the graphical representation



### **Hyperparameter Tuning with Randomized Search**

Randomized Search approach is applied to optimize the hyperparameters of the Simple RNN model. The goal is to find the best combination of hyperparameters (e.g., number of units, dropout rate, optimizer type, learning rate, and batch size) that minimizes the model's validation loss.

#### Steps:

1. **Randomized Search Setup:**
   * A parameter grid is defined, which includes different possible values for the hyperparameters:  
     + **Units:** Number of neurons in the RNN layers (50 or 100)
     + **Dropout Rate:** Regularization parameter to prevent overfitting (0.2 or 0.3)
     + **Optimizer:** Type of optimizer used for training the model ('adam' or 'rmsprop')
     + **Learning Rate:** Learning rate for the optimizer (0.0005 or 0.001)
     + **Batch Size:** Number of samples processed before updating the model (32 or 64)
2. **Randomized Search Execution:**  
   * Random combinations of the hyperparameters are selected, and the model is trained with these combinations.
   * For each combination, the model is trained on the training data, and the validation loss is recorded.
3. **Early Stopping:**
   * An early stopping callback is used to stop the training early if the validation loss does not improve for 5 consecutive epochs, preventing overfitting.
4. **Selection of Best Model:**
   * After evaluating each random combination, the model with the lowest validation loss is selected as the best model.
5. **Training the Best Model:**
   * The best combination of hyperparameters is used to train the model on the full training dataset.
6. **Evaluation:**
   * The performance of the model is evaluated on the test dataset, and key metrics like Test Loss, Mean Absolute Error (MAE), and Root Mean Squared Error (RMSE) are calculated to assess the model's accuracy.

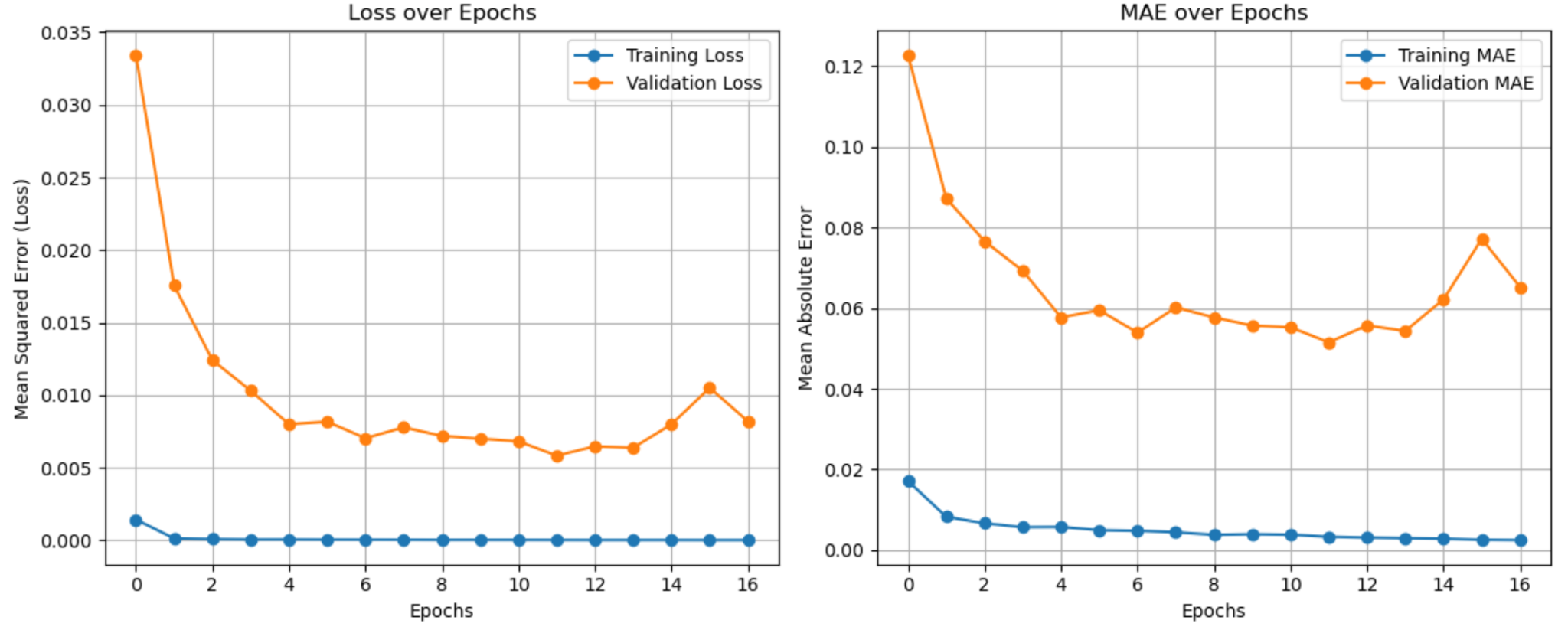
**Results:**

After training the model with the best hyperparameters identified through randomized search, the model was evaluated on the test dataset. The following metrics were obtained:

* **Test Loss (Mean Squared Error):** The test loss was found to be 0.0058, indicating the model's ability to minimize prediction errors on unseen data.
* **Test Mean Absolute Error (MAE):** The mean absolute error on the test data was 0.0515, reflecting the average magnitude of errors between the predicted and actual stock prices.
* **Root Mean Squared Error (RMSE):** The RMSE was calculated to be 0.0764, which quantifies the model’s prediction error in the same units as the stock price. A lower RMSE value indicates better prediction accuracy.

These results demonstrate that the model, after tuning its hyperparameters, performs reasonably well in predicting Apple’s stock prices over a range of forecast horizons (1-day, 5-day, and 10-day).

Below is the graphical representation:



1. Best Model (via Randomized Search) performs best across all metrics, with the lowest loss, MAE, and RMSE, and shows no significant overfitting.

2. Tuned Model has a lower performance than the best model, and its validation loss suggests potential overfitting.

3. The Untuned Model shows the highest test loss, MAE, and RMSE, meaning it has the worst performance overall.

**Creating LSTM Model and performing Hyperparameter Tuning**

The LSTM (Long Short-Term Memory) model was designed to predict Apple's stock prices over three different time horizons: 1-day, 5-day, and 10-day. The model was structured as follows:

* **First LSTM Layer:**
  + The first LSTM layer was created with 50 units and set to return sequences (return\_sequences=True). This configuration allows the LSTM to pass its output to the next layer while preserving its sequence structure.
  + The input shape was set to (X\_train.shape[1], 1), as the model requires a 3D input (number of samples, time steps, and features).
* **Dropout Layer:**  
  + A Dropout layer with a rate of 40% (Dropout(0.4)) was added to prevent overfitting by randomly setting 40% of the neurons to zero during training.
* **Second LSTM Layer:**  
  + The second LSTM layer, also with 50 units, was set to return a single output vector (return\_sequences=False) that captures the temporal dependencies from the previous LSTM layer.
* **Dense Output Layer:**
  + The final Dense layer was added with 3 units to predict the 1-day, 5-day, and 10-day stock prices. This layer outputs the predicted values for all three time horizons in parallel.

#### **2. Model Compilation**

The model was compiled using the Adam optimizer with a learning rate of 0.001. The loss function used was Mean Squared Error ('mean\_squared\_error'), which measures the average squared difference between predicted and actual values. Additionally, the model was evaluated using Mean Absolute Error ('mean\_absolute\_error'), which quantifies the average absolute difference between predicted and actual values.

#### **3. Data Preparation**

To prepare the data for LSTM input, the feature set (X\_train and X\_test) was reshaped into a 3D array with the shape (samples, time steps, features). This reshaping allows the LSTM to process sequential data correctly.

#### **4. Model Training**

The LSTM model was trained using the fit() function, with the following configurations:

* Epochs: 50
* Batch Size: 32
* Validation Data: X\_test, y\_test
* Early Stopping: Early stopping was used to monitor validation loss and prevent overfitting. If validation loss did not improve for 10 consecutive epochs, training was stopped, and the model with the best weights was restored.

#### **5. Model Evaluation**

The performance of the trained LSTM model was evaluated using the test data. The following evaluation metrics were reported:

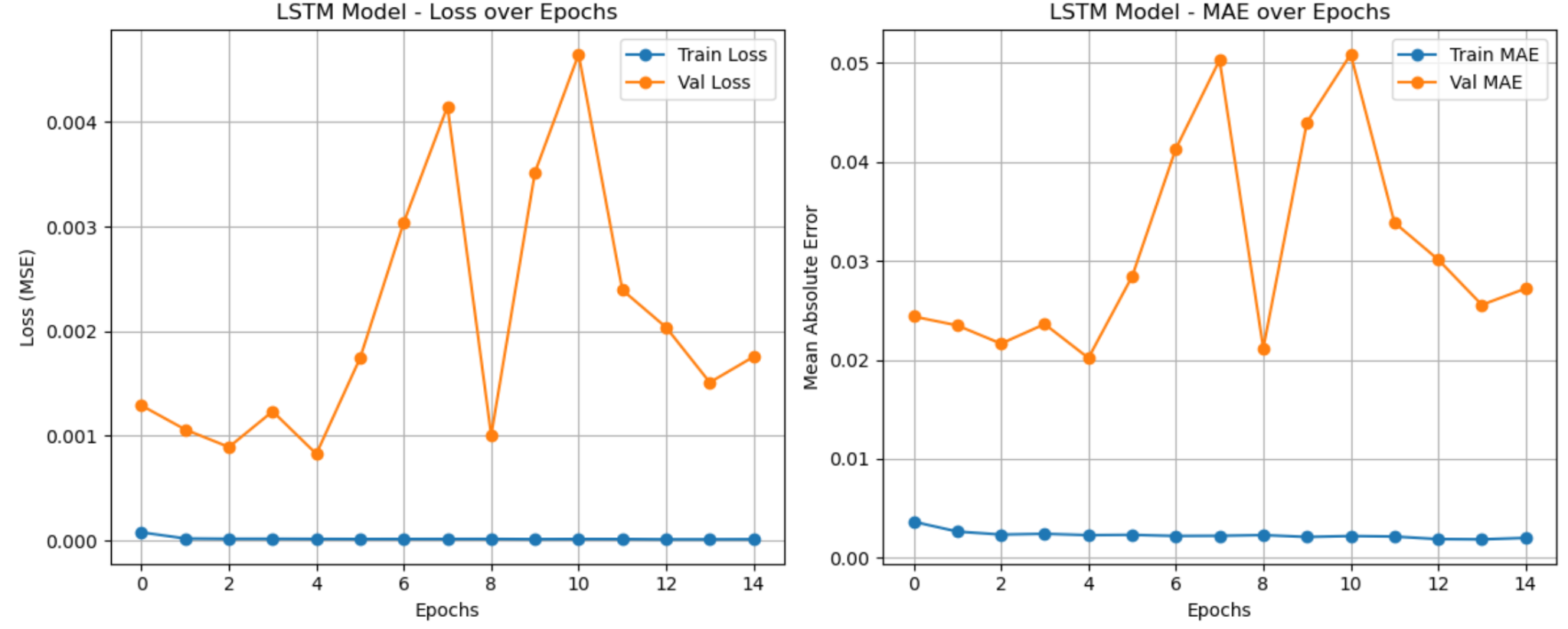
* Test Loss (Mean Squared Error): A measure of how well the model's predictions match the actual values. Lower values indicate better predictions.
* Test Mean Absolute Error (MAE): Represents the average absolute error between the predicted and actual stock prices.
* Root Mean Squared Error (RMSE): Provides an overall measure of model performance by calculating the square root of the average squared differences between the predicted and actual values.

#### **6. Results**

After training the model and evaluating its performance on the test data, the following results were obtained:

* **Test Loss (Mean Squared Error):** 0.00083  
   This value represents the mean squared difference between the predicted and actual stock prices. A lower value indicates better model performance in terms of predicting the stock prices accurately.
* **Test Mean Absolute Error (MAE):** 0.02019  
   The MAE measures the average absolute error between the predicted and actual stock prices. A lower MAE indicates the model's predictions are closer to the actual values.
* **Root Mean Squared Error (RMSE):** 0.02877  
   RMSE is the square root of the average squared differences between the predicted and actual values. It provides a more interpretable error measurement in the same unit as the predicted values (stock price). Lower RMSE values indicate better performance in making accurate predictions.

Below is the graphical representation



**Hyperparameter Tuning using Random Search**

We designed an LSTM (Long Short-Term Memory) neural network to predict stock prices (or any other time-series forecasting task). The model architecture includes:

* Two LSTM Layers: The first LSTM layer returns sequences to allow the second LSTM layer to process the information further. This helps the model capture temporal dependencies in the data.
* Dropout Layers: Dropout is applied after each LSTM layer to reduce overfitting by randomly deactivating a fraction of neurons during training. This helps the model generalize better.
* Dense Output Layer: A Dense layer is used as the output layer to predict the target variable (stock price or other time-series values).

The model is compiled with the Adam or RMSprop optimizer and uses Mean Squared Error (MSE) as the loss function, which is commonly used in regression tasks like stock price prediction.

#### **2. Hyperparameter Grid**

The model’s performance heavily depends on the choice of hyperparameters. In this project, a **hyperparameter grid** was defined to explore different values for the following parameters:

* **Units:** Number of LSTM units in each LSTM layer (set to 50 for this search).
* **Dropout Rate:** Dropout rate values of 0.4 and 0.5 were tested to see if stronger regularization would improve performance.
* **Optimizer:** Only the **Adam** optimizer was selected for this search, as it typically provides faster convergence and better performance on many tasks.
* **Learning Rate:** A learning rate of 0.0005 was chosen to provide slower but more stable updates during training.
* **Batch Size:** A batch size of 64 was selected to speed up training by processing more samples per batch.

#### **3. Random Search for Hyperparameter Tuning**

To find the optimal set of hyperparameters, a **random search** strategy was employed. This approach involves randomly selecting hyperparameter combinations from the predefined grid and training the model multiple times to identify the best configuration.

* **Number of Trials:** A total of **30 random trials** were conducted to explore different combinations of hyperparameters.
* **Early Stopping:** During each trial, **early stopping** was used to prevent overfitting. Training was halted if the validation loss did not improve for 3 consecutive epochs (patience=3), and the best model weights were restored.
* **Epochs:** The model was trained for **15 epochs** per trial to speed up the tuning process.

The **validation loss** from each trial was tracked, and the set of hyperparameters that resulted in the lowest validation loss was selected as the best-performing configuration.

**4. Best Hyperparameters and Model Training**

After completing the random search, the best hyperparameters were identified based on the lowest validation loss. Using these optimal hyperparameters, the model was then retrained.

### **Model Performance Evaluation**

After training the LSTM model with the optimized hyperparameters, the model's performance was evaluated on the test dataset. The evaluation metrics used to assess the model's accuracy include **Test Loss**, **Mean Absolute Error (MAE)**, and **Root Mean Squared Error (RMSE)**.

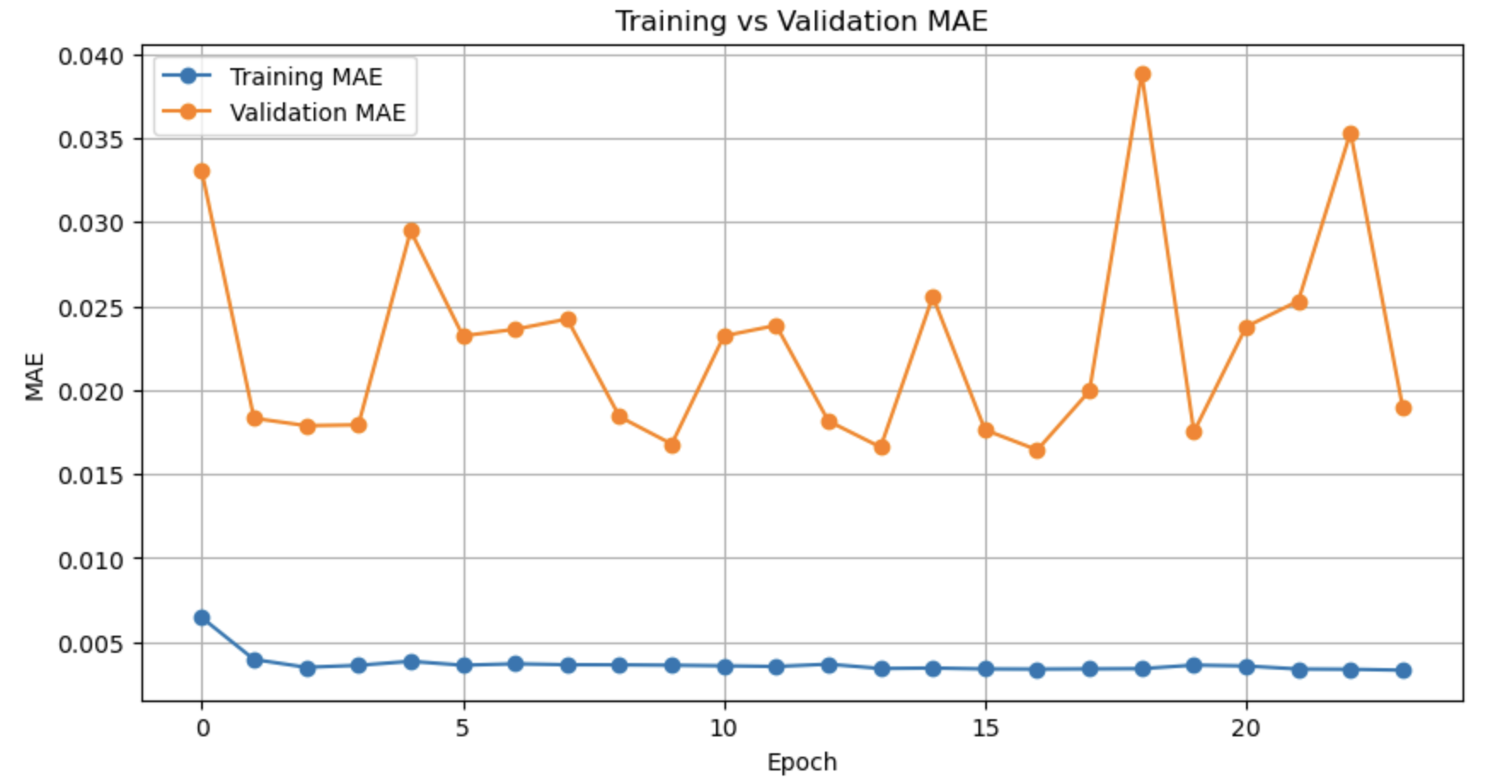
The results are as follows:

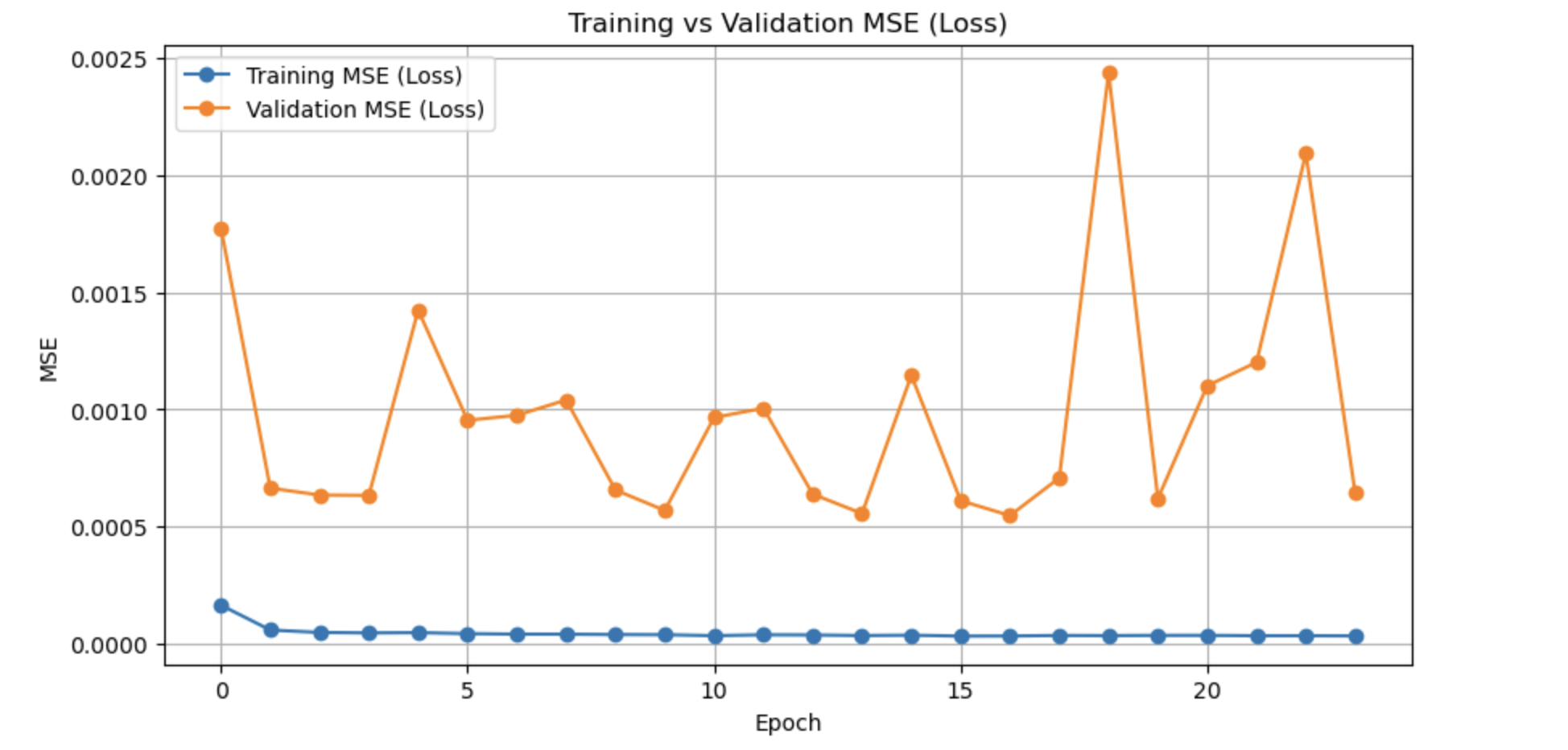
* **Test Loss (Mean Squared Error):** The model achieved a **test loss** of **0.0003929986269213259**, which indicates the average squared difference between predicted and actual values. A lower test loss value signifies better model performance, as the model's predictions are closer to the true values.
* **Test Mean Absolute Error (MAE):** The **Test MAE** was found to be **0.013554777018725872**. MAE represents the average magnitude of the errors between predicted and actual values, without considering the direction of the error. The smaller the MAE, the better the model's accuracy in predicting the target variable.
* **Root Mean Squared Error (RMSE):** The **RMSE** value was **0.019824194058721697**, which provides the square root of the average squared differences between predicted and actual values. RMSE is sensitive to large errors and penalizes them more heavily compared to MAE. A lower RMSE value indicates that the model has made predictions closer to the true values.

### **Interpretation of Results**

* **Test Loss (MSE)** and **RMSE** are both quite low, suggesting that the model has learned the underlying patterns in the data effectively and can make accurate predictions.
* **Test MAE** further confirms that the model's predictions are, on average, within a small margin of error, indicating good performance.

Below is the Graphical Representation





1. The model (after hyperparameter tuning) is performing better in terms of loss, MAE, and RMSE.

2. There is no clear sign of overfitting as both the test and validation losses are quite low and similar in both models.

The use of early stopping likely helped prevent overfitting by ensuring the model didn't train too long without improvement.

**Conclusions:**

1. Tuned LSTM Model performs better than the Tuned SimpleRNN Model across all metrics (Test Loss, Test MAE, and RMSE), suggesting that the LSTM model is more suited for time-series prediction tasks in this context.

2. The SimpleRNN Model shows signs of higher prediction errors, as reflected by the higher loss, MAE, and RMSE values.