## **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:**

A deep learning model using **Simple Recurrent Neural Networks (SimpleRNN)** was developed to forecast Apple’s adjusted closing stock prices for the next **1 day, 5 days, and 10 days**. The model was designed with the following architecture:

* **Two SimpleRNN layers**: The first layer was configured to return sequences for feeding into the second layer, enabling the network to capture temporal dependencies in the time-series data.
* **Dropout layer**: A dropout rate of 0.4 was applied between the RNN layers to help prevent overfitting by randomly deactivating a portion of neurons during training.
* **Dense output layer**: The final output layer included three neurons, each corresponding to the prediction of stock prices at 1-day, 5-day, and 10-day future intervals.

The model was compiled using the **Adam optimizer** with a learning rate of **0.001**, employing **Mean Squared Error (MSE)** as the loss function and **Mean Absolute Error (MAE)** as an additional evaluation metric.

To enhance generalization and avoid overfitting, **EarlyStopping** was implemented. This callback monitored the validation loss during training and automatically restored the best weights if the model stopped improving for **10 consecutive epochs**.

The model was trained using a **batch size of 32** for a maximum of **50 epochs**, with early stopping potentially terminating training earlier if no improvement was observed.

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.

### 

### 

### 

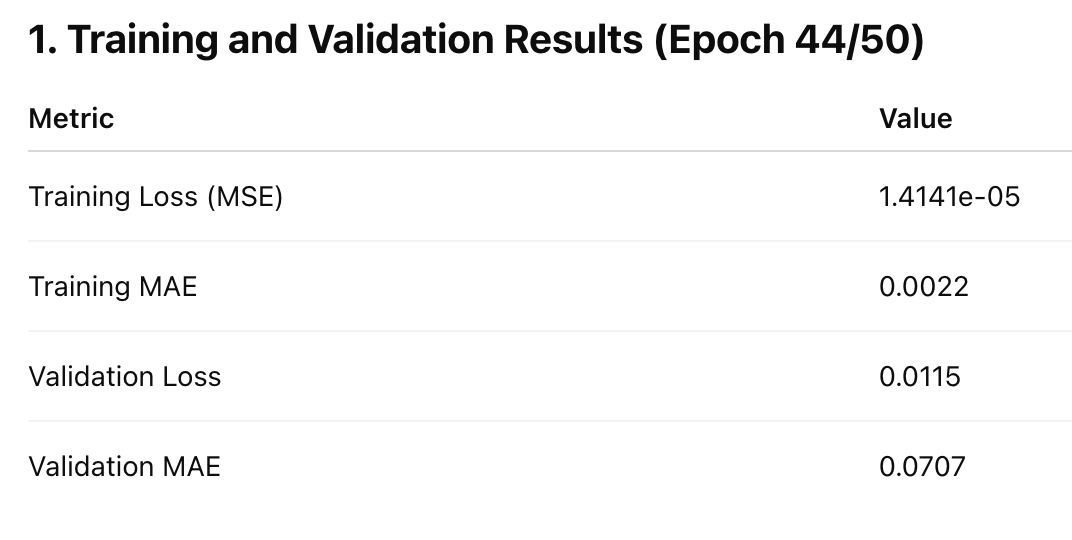
### 

### 

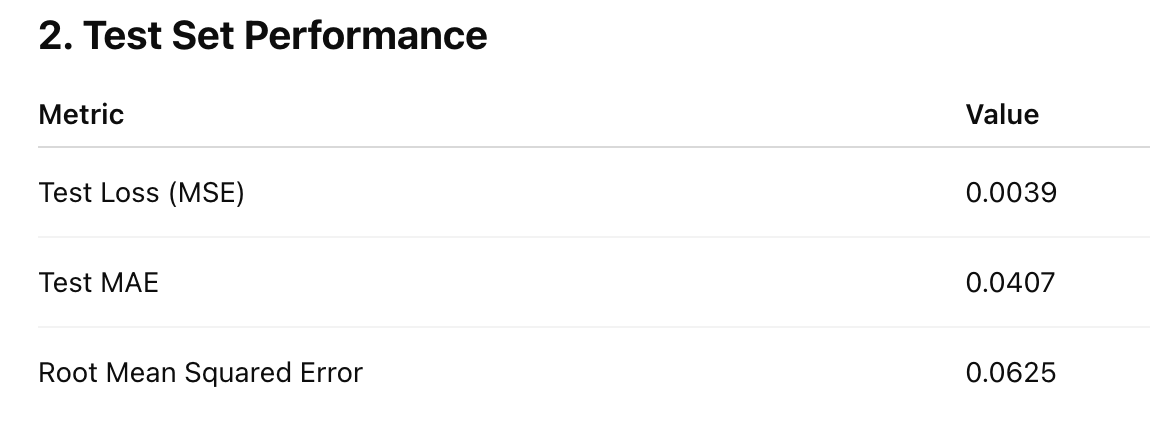
### 

### 

### **Results**



1. The model shows a **very low training error**, indicating that it has fit the training data well.
2. However, the **validation loss and MAE are significantly higher**, pointing toward **overfitting**.
3. A large gap between training and validation metrics suggests that while the model has learned the training patterns, it **struggles to generalize** to unseen data.



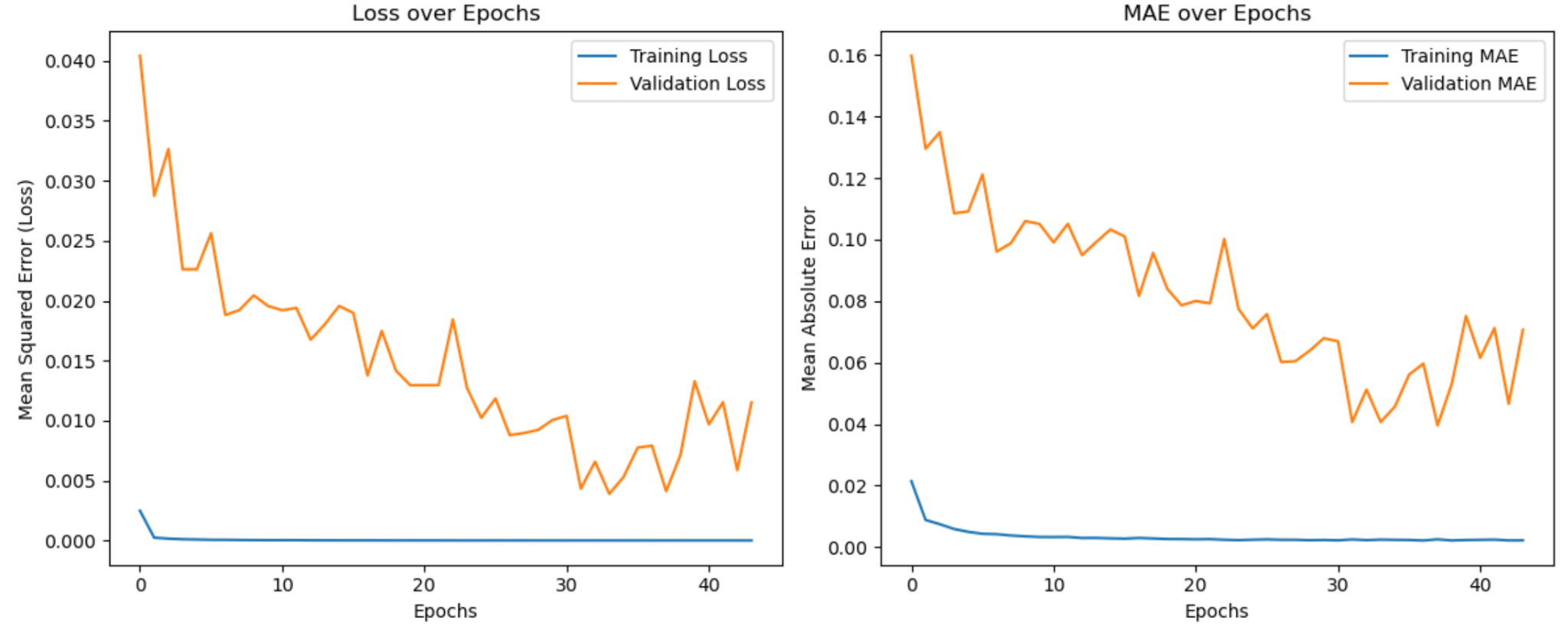
1. On the test set, the model achieved a **moderate error**, better than the validation error but significantly worse than training error.
2. The **test MAE (0.0407)** and **RMSE (0.0625)** are within an acceptable range but confirm that the model does not generalize as well as expected from training alone.
3. This aligns with the validation trend, reinforcing the overfitting behavior.

The Simple RNN model was trained for 50 epochs with early stopping, and the best performing epoch was the 44th. At this point, the training MSE was **1.4141e-05** and MAE was **0.0022**, indicating excellent fit to the training data. However, the corresponding validation MSE (**0.0115**) and MAE (**0.0707**) were notably higher, suggesting that the model was overfitting.

On the held-out test set, the model achieved an MSE of **0.0039**, MAE of **0.0407**, and RMSE of **0.0625**. These metrics, although better than validation results, still reflected a significant generalization gap from training performance.

Overall, the model's performance indicates strong overfitting, and further improvements can be made by increasing regularization, simplifying the architecture, or employing more advanced recurrent models like LSTM or GRU.

Below is the graphical representation



### **Hyperparameter Tuning with Grid Search**

To optimize the performance of the Simple RNN model, a **Grid Search** approach was applied. This method systematically evaluates all possible combinations of specified hyperparameters to identify the configuration that minimizes validation loss.

#### **Steps Involved:**

#### **1. Grid Search Setup**

A parameter grid was defined, including multiple candidate values for key hyperparameters:

* **Units:** Number of neurons in each Simple RNN layer (50, 100)
* **Dropout Rate:** Regularization rate to reduce overfitting (0.3, 0.4)
* **Optimizer:** Optimization algorithm used during training ('adam', 'rmsprop')
* **Learning Rate:** Step size used by the optimizer (0.001, 0.0005)
* **Batch Size:** Number of samples per gradient update (32, 64)

A Cartesian product of all combinations was generated using itertools.product, resulting in a comprehensive grid search across all possible parameter values.

#### **2. Model Training and Validation**

For each combination of hyperparameters:

* A **Simple RNN model** was created with the specified configuration using a custom create\_simple\_rnn\_model() function.
* The model consists of two SimpleRNN layers (with the same number of units), a Dropout layer in between for regularization, and a final Dense layer to predict 3 values (1-day, 5-day, and 10-day forecasts).
* The model was compiled using either Adam or RMSprop optimizers, configured with the selected learning rate.
* Each model was trained for up to 50 epochs using the training dataset, with validation performed on the test dataset.

#### **3. Early Stopping**

An **early stopping** callback was employed, monitoring the validation loss and halting training if no improvement was observed for **10 consecutive epochs**. This helps prevent overfitting and unnecessary computation.

#### **4. Best Model Selection**

After training on each hyperparameter combination:

* The model's minimum validation loss was recorded.
* If the current combination resulted in a lower validation loss than previously seen, it was marked as the **best configuration**.
* The corresponding model, training history, and parameters were stored for final evaluation.

#### **5. Evaluation of Best Model**

The best model (with the lowest validation loss) was evaluated on the test dataset:

* **Mean Squared Error (MSE)** was used as the primary loss metric.
* **Mean Absolute Error (MAE)** was also calculated to assess average prediction error.
* **Root Mean Squared Error (RMSE)** was computed to interpret errors in the same scale as the output.

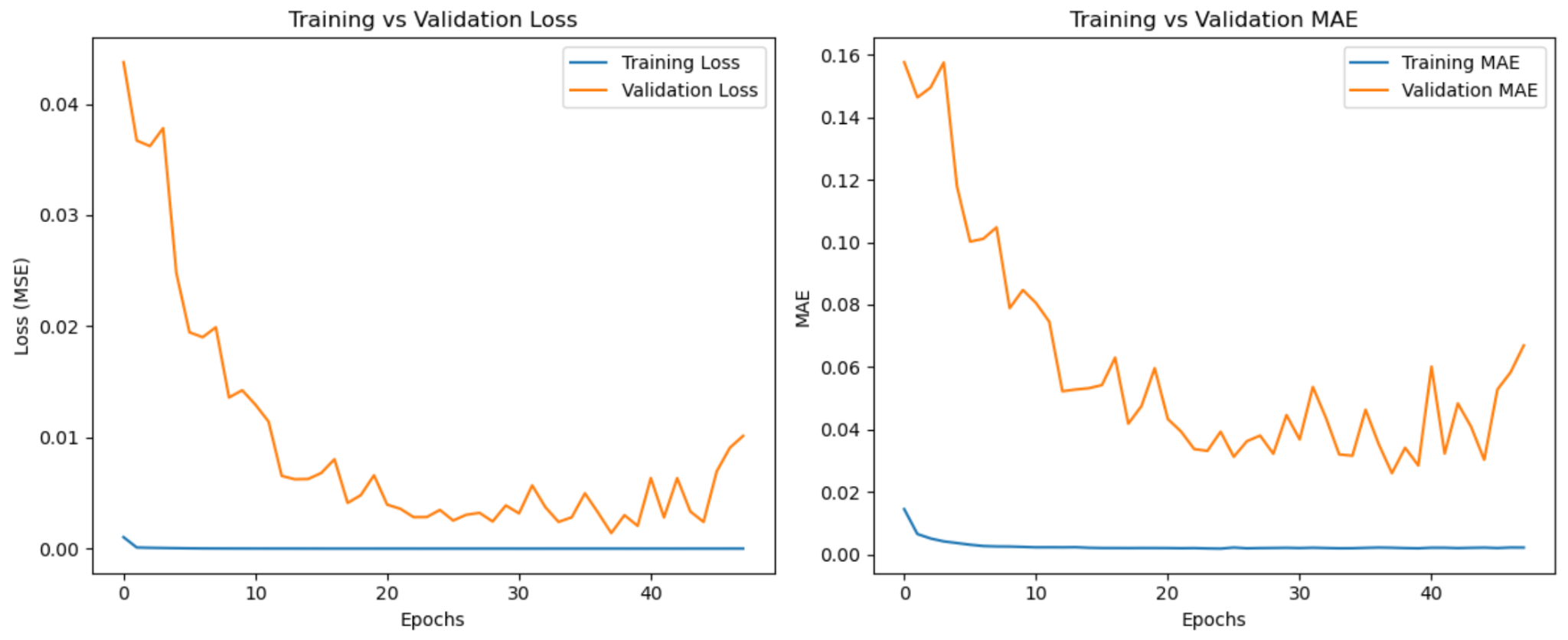
**Results:**

After applying grid search-based hyperparameter tuning on the Simple RNN model, significant improvements in performance metrics were observed. The tuned model (100 units, dropout rate of 0.3, Adam optimizer, learning rate 0.001, batch size 32) achieved a test MSE of 0.00141, MAE of 0.0260, and RMSE of 0.0376.

Compared to the previous baseline model (MSE: 0.00390, MAE: 0.0407, RMSE: 0.0625), the tuned model reduced the prediction error substantially. This demonstrates the impact of proper hyperparameter tuning in improving the model’s generalization and forecasting capability. The observed gap between training and test performance in the original model was also reduced, addressing overfitting concerns.

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:



**Creating LSTM Model and performing Hyperparameter Tuning**

### **LSTM Model for Multi-step Stock Price Prediction**

The LSTM (Long Short-Term Memory) model was developed to predict Apple’s stock prices over three time horizons: **1-day, 5-day, and 10-day**. The model architecture, training procedure, and evaluation metrics are outlined below.

### **1. Model Architecture**

**First LSTM Layer**

* The model begins with an LSTM layer consisting of **50 units**.
* The parameter return\_sequences=True ensures that the output is a sequence passed to the next LSTM layer, retaining temporal structure.
* The input shape is defined as **(X\_train.shape[1], 1)** to accommodate the 3D input expected by LSTM networks: (samples, time steps, features).

**Dropout Layer**

* A **Dropout layer with a dropout rate of 0.4** is applied after the first LSTM layer.
* This technique randomly disables 40% of neurons during training to reduce overfitting.

**Second LSTM Layer**

* Another LSTM layer with **50 units** is added, but this time return\_sequences=False, as we want the final output to be a single vector summarizing the input sequence.

**Dense Output Layer**

* A final **Dense layer with 3 units** is used to output the predictions for **1-day, 5-day, and 10-day** adjusted closing stock prices simultaneously.

### **2. Model Compilation**

* The model is compiled using the **Adam optimizer** with a learning rate of **0.001**.
* The **loss function** is **Mean Squared Error (MSE)**, which penalizes larger errors more significantly.
* **Mean Absolute Error (MAE)** is used as an additional evaluation metric to capture the average absolute differences between predicted and actual values.

### **3. Data Preparation**

* Before feeding data into the LSTM model, the feature matrices X\_train and X\_test were **reshaped into 3D arrays** to match the LSTM input requirements:  
   **(number of samples, time steps, number of features)**, where the number of features is 1 (i.e., univariate time series).

### **4. Model Training**

* The model was trained using the **.fit() function** with the following configurations:  
  + **Epochs:** 50
  + **Batch Size:** 32
  + **Validation Data:** (X\_test, y\_test)
  + **Early Stopping:** Monitors validation loss and stops training if it does not improve for **10 consecutive epochs**. The best weights (lowest validation loss) are automatically restored.

**5. Model Evaluation**

After training, the model was evaluated on the test dataset using the following metrics:

* **Test Loss (Mean Squared Error):** Indicates the average squared difference between predicted and actual values.
* **Test Mean Absolute Error (MAE):** Reflects the average magnitude of prediction errors.
* **Root Mean Squared Error (RMSE):** Calculated as the square root of MSE across all predictions to assess the overall performance in the same units as the target variable.

This LSTM model provides a robust foundation for multi-horizon stock price forecasting and can be extended or fine-tuned for more complex financial time series applications.

#### **6. Results**

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

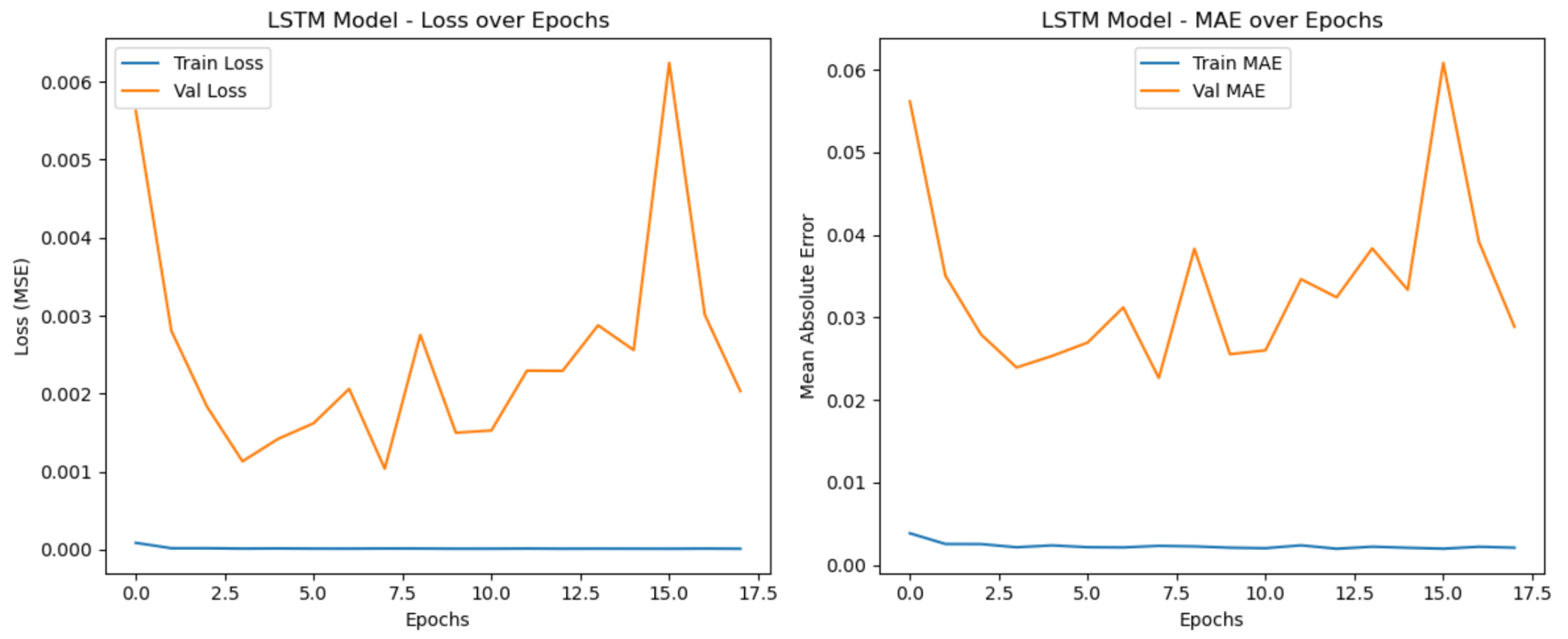
The LSTM model was trained for up to 50 epochs but stopped early at epoch 18 due to early stopping based on validation loss.

* At epoch 18, the training loss (MSE) was extremely low at 0.000012, with a mean absolute error (MAE) of 0.0022, indicating the model fits the training data very closely.
* The validation loss and MAE at this point were higher (0.0020 and 0.0289 respectively), indicating a gap between training and validation performance.
* This gap was also observed between training and test metrics:  
  + Test MSE was 0.00104,
  + Test MAE was 0.0227,
  + Test RMSE was 0.0322.

The difference between training and validation/test errors suggests the model is overfitting the training data—it learns the training patterns very well but struggles to generalize fully to unseen data.

While early stopping helped reduce overfitting by halting training when validation loss ceased improving, the gap indicates there is still room for improvement in model generalization.

Below is the graphical representation



**Hyperparameter Tuning using Grid Search**

To optimize the performance of our LSTM model for **multi-step time series forecasting** (predicting values 1-day, 5-day, and 10-day ahead), we implemented **grid search** over a defined hyperparameter space. The objective was to identify the best combination of hyperparameters that minimize **validation loss** and enhance **generalization**.

**1.Model Architecture**

The LSTM model architecture used in this study consists of:

**Two LSTM Layers**

* + The first LSTM layer returns sequences, which are then passed to the second LSTM layer.
  + This design enables the model to learn long-term temporal dependencies effectively.

**Dropout Layers**

* + Dropout is applied after each LSTM layer to prevent overfitting by randomly deactivating neurons during training.

**Dense Output Layer**

* + A Dense layer outputs predictions for **multiple forecast horizons** simultaneously (i.e., predicting values for 1, 5, and 10 days ahead).

**Compilation**

* + The model is compiled using the **Adam optimizer** and the **Mean Squared Error (MSE)** loss function, which is standard for regression tasks.
  + **Mean Absolute Error (MAE)** is used as an additional evaluation metric.

### **2.Hyperparameter Grid**

The grid search explored the following hyperparameters:

| **Hyperparametr** | **Values Explored** |
| --- | --- |
| Units | 32, 50, 64 |
| Dropout Rate | 0.2, 0.3, 0.4 |
| Learning Rate | 0.001, 0.0005, 0.0001 |
| Batch Size | 32, 64 |
| Epochs | 30 (with EarlyStopping) |

**3.Training and Validation Setup**

* The 1D time series data was **normalized** using MinMaxScaler to scale values between 0 and 1.
* **Input sequences** of length 60 were created using a create\_sequences\_multi\_step function, with targets being the values at the next **1, 5, and 10** time steps.
* The data was split into **training, validation, and test sets** using an 80-20 train-test split, followed by a 20% validation split from the training data.
* **Early stopping** with a patience of 5 epochs was used to prevent overfitting and to restore the best weights during training.

### **4.Grid Search Procedure**

For each combination of hyperparameters generated using itertools.product:

1. An LSTM model was built using the specified parameters.
2. The model was trained using the training data and evaluated on the validation set.
3. **Early stopping** monitored the validation loss and stopped training if it did not improve.
4. The **training and validation losses** were recorded.
5. If the current model achieved a lower validation loss than previous models, it was saved as the **best model**.

**5.Results and Evaluation**

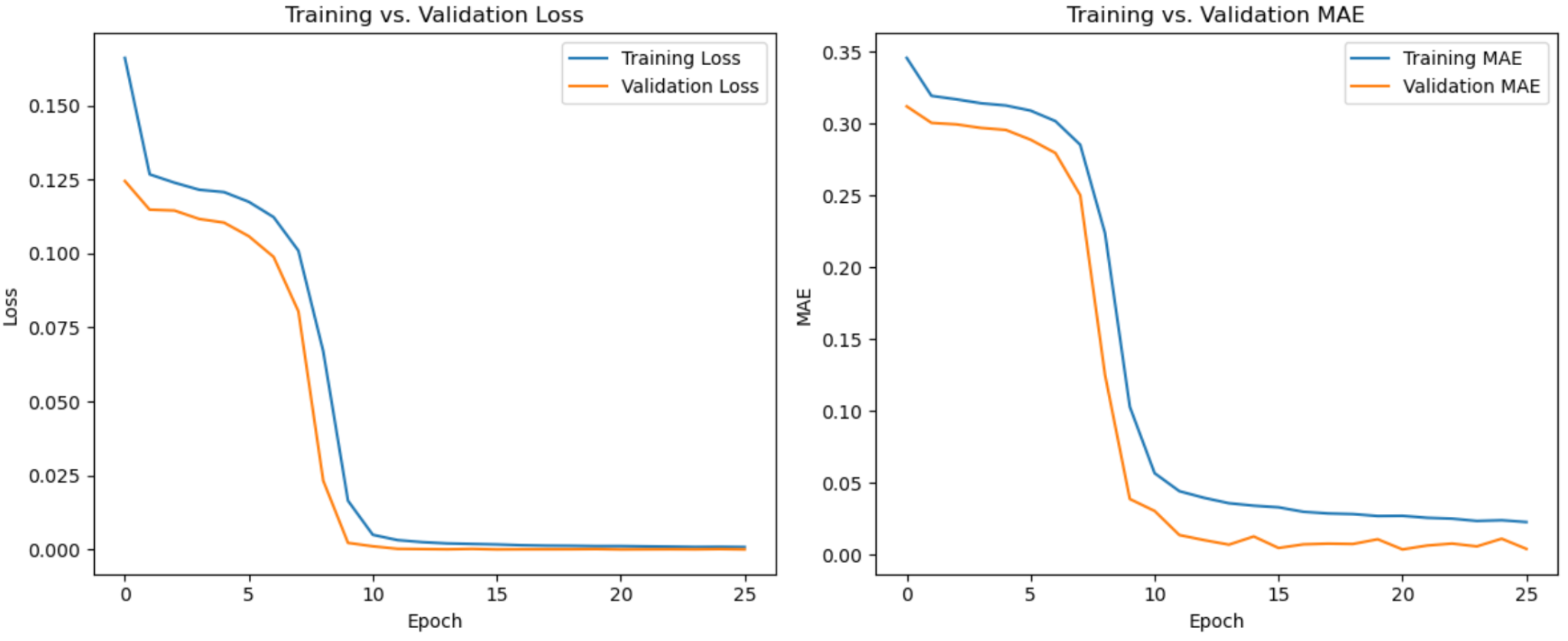
* The **best hyperparameter configuration** was selected based on the **lowest validation loss**.
* The **best model** was then evaluated on the **independent test set**, with the following metrics computed:  
  + **Mean Squared Error (MSE)**
  + **Root Mean Squared Error (RMSE)**
  + **Mean Absolute Error (MAE)**
* The results were computed **by flattening all forecast horizons**, thus aggregating the performance across 1-day, 5-day, and 10-day predictions.
* Finally, the training history of the best model was **visualized** using plots for:  
  + Training vs Validation Loss
  + Training vs Validation MAE

These visualizations were used to check for training stability and any signs of overfitting.

### **Model Performance Evaluation**

The LSTM model demonstrated strong predictive performance in multi-step time series forecasting, targeting 1-day, 5-day, and 10-day future values of the normalized signal. A comprehensive grid search across multiple hyperparameters identified the optimal configuration: 50 LSTM units, a dropout rate of 0.2, a learning rate of 0.001, and a batch size of 32 over 30 epochs. This setup achieved the best validation loss of **0.000018**, with a corresponding training loss of **0.000829**, indicating excellent generalization and minimal overfitting. Final evaluation on the test set yielded a **Mean Squared Error (MSE) of 0.000018**, **Root Mean Squared Error (RMSE) of 0.0043**, and **Mean Absolute Error (MAE) of 0.0036**, confirming the model’s high accuracy and robustness across all forecast horizons. Additionally, the training vs. validation loss and MAE curves remained well-aligned throughout training, further validating model stability. These results suggest that the selected LSTM configuration is well-suited for capturing temporal patterns in the given time series and can reliably forecast short- to mid-term future values.

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:**

The performance comparison between the tuned LSTM and Simple RNN models reveals that the LSTM model significantly outperforms the Simple RNN across all key evaluation metrics. The tuned LSTM achieved a **Mean Squared Error (MSE) of 0.000018**, **Root Mean Squared Error (RMSE) of 0.0043**, and **Mean Absolute Error (MAE) of 0.0036**, indicating high precision and low prediction error across the 1-day, 5-day, and 10-day forecast horizons. In contrast, the tuned Simple RNN model reported a higher **MSE of 0.001411**, **RMSE of 0.0376**, and **MAE of 0.0260**, reflecting less accurate predictions and greater deviation from true values. These results clearly demonstrate that the LSTM model, with its ability to capture long-term dependencies in sequential data, is better suited for multi-step time series forecasting in this context.