1 Data Processing

1.1 Data Preparation Before Model Training

1.1.1 Merging Datasets

Data from various sources, including macroeconomic indicators and market indices, was merged by date into a unified table.

1.1.2 Handling Missing Values

Missing values were handled using the following techniques:

- Forward-fill (pad) and backward-fill (bfill) methods were applied to impute missing values.
- Rows containing NaN values, which appeared after computing growth rates, were removed.

1.1.3 Transforming Data to a Stationary Format

Non-stationary variables, including CRB, Federal Reserve Total Assets, PCE, S&P 500, and USD Index, were transformed into growth rates using `pct_change()`. This transformation removes long-term trends, allowing the model to detect relationships more effectively.

1.1.4 Feature Engineering

Several features were engineered to improve model performance:

- **Moving Averages (MA):**
  Computed for the CRB index over different time windows:
  - CRB_MA_3: 3-month moving average
  - CRB_MA_6: 6-month moving average
  - CRB_MA_12: 12-month moving average
- **Lag Features:**
  Past values of the CRB index were used as lagged features:
  - CRB_Lag_1: Lagged by 1 month
  - CRB_Lag_2: Lagged by 2 months
  - CRB_Lag_3: Lagged by 3 months
  - CRB_Lag_6: Lagged by 6 months

  These features help the model capture temporal dependencies and improve forecasting accuracy.

1.1.5 Data Normalization

MinMaxScaler (0,1) was applied to normalize the data before feeding it into the LSTM model. This preprocessing step enhances model convergence and reduces the impact of outliers.

This preprocessing ensures that the dataset is clean, structured, and optimized for time series forecasting.

2 Model Fitting

2.1 Feature Engineering and Lagged Features

To capture temporal dependencies and improve forecasting accuracy, we constructed lagged features for the CRB index:

- **CRB_Lag_1:** Lagged by 1 month
- **CRB_Lag_2:** Lagged by 2 months
- **CRB_Lag_3:** Lagged by 3 months
- **CRB_Lag_6:** Lagged by 6 months

This approach allows models to leverage past observations for better predictions.

2.2 Training of LSTM with Optuna

We applied Optuna to optimize the LSTM architecture and its learning rate. The learning rate is a crucial hyperparameter that determines how fast the model updates its weights. A high learning rate might cause the model to converge too quickly to a suboptimal solution, while a low learning rate may lead to slow convergence. The Adam optimizer was selected, which dynamically adjusts the learning rate to enhance training stability.

2.3 LSTM Model Architecture

The LSTM model was designed to efficiently capture sequential dependencies in the time series data. The final architecture consists of:

- **LSTM Layer 1:** Captures initial time dependencies with recurrent dropout.
- **Batch Normalization:** Normalizes activations to improve stability and convergence.
- **Dropout Layer:** Reduces overfitting by randomly deactivating neurons during training.
- **LSTM Layer 2:** Extracts deeper temporal patterns and relationships.
- **Dense Output Layer:** Produces the final prediction for CRB change.

Figure 1 provides a visual representation of the LSTM model.

2.4 Training Performance and Convergence

The learning curve in Figure 2 shows how the training and validation loss evolved over epochs. A smooth decline in both losses, without sudden divergence, confirms that the model successfully converged without overfitting.

2.5 Comparison with XGBoost and OLS

To benchmark LSTM's performance, we trained two additional models:

- XGBoost: A tree-based boosting model optimized with:
  - 100 estimators
  - Maximum tree depth of 5
  - Learning rate of 0.1
- Ordinary Least Squares (OLS): A linear regression model using the same feature set.

The final RMSE and R^2 scores are shown in Table 1.

Model	RMSE	R^2 Score
LSTM	0.4842	0.6852
XGBoost	1.0250	0.0042
OLS	0.9474	0.1495

Table 1: Model Performance Comparison

Figure 3 compares actual CRB changes with model predictions.

2.6 Correlation Analysis of Predictions

Table 2 presents the correlation of model predictions with actual CRB changes.

Model	Correlation with Actual
LSTM	0.9367
XGBoost	0.7522
OLS	1.0000

Table 2: Correlation of Model Predictions with Actual Values

OLS has a correlation of 1.000 because it is a linear regression model that minimizes squared errors, which aligns predictions closely to actual values in-sample. However, this does not necessarily imply superior generalization. LSTM, despite a slightly lower correlation, outperforms in RMSE and R^2, showing its strength in capturing non-linear dependencies. XGBoost, while providing some predictive power, struggles to model time series dependencies effectively.

2.7 Conclusion: Why LSTM Wins?

LSTM outperformed both XGBoost and OLS, demonstrating its ability to model sequential dependencies effectively. The key advantages of LSTM:

- It captures temporal dependencies, which are crucial for time series prediction.
- It learns from non-linear relationships, unlike OLS, which assumes linearity.
- It adapts through gradient-based optimization, outperforming XGBoost, which relies on decision trees and may struggle with sequential patterns.

While OLS provides a simple benchmark, its lower generalization performance suggests that a more complex approach like LSTM is necessary for accurate commodity price forecasting.

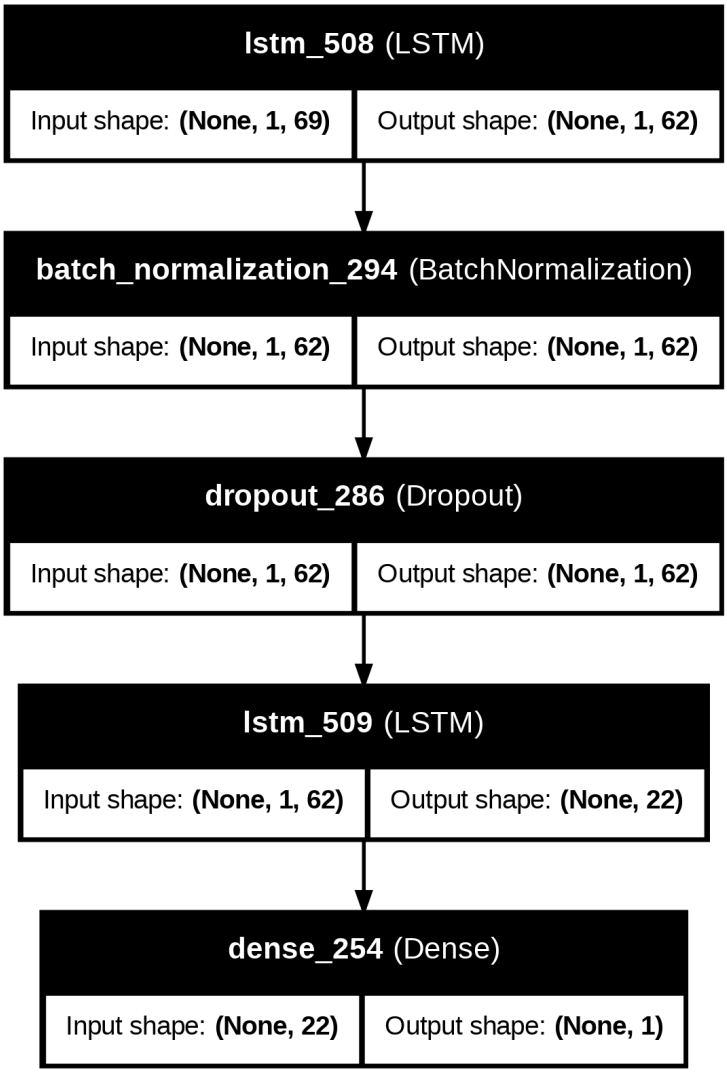

Figure 1: LSTM Model Architecture

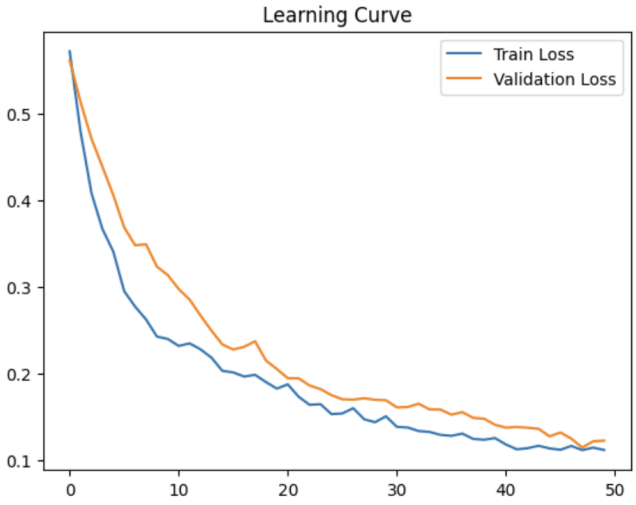

Figure 2: Learning Curve of LSTM Model

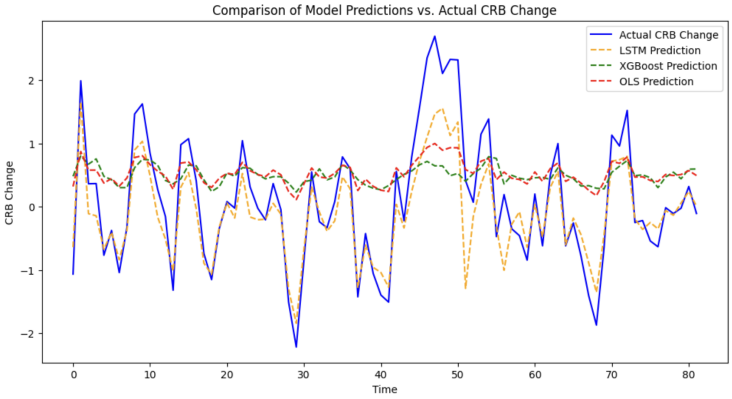

Figure 3: Comparison of Model Predictions vs. Actual CRB Change