Advanced Time-Series Forecasting of Oil Prices: An Integration of ARIMA, GARCH, LSTM, Mamba Models, and Attention Mechanisms with Sentiment Analysis

Ashish Pandey

New Jersey Insttute Of Technology

Ap2934@njit.edu

Abstract

Predicting oil prices accurately has significant implications for various stakeholders in the energy sector and financial markets. This research presents an integrated framework for advanced time-series forecasting of oil prices by combining traditional econometric models, deep learning architectures, attention mechanisms, sentiment analysis, and machine learning techniques. The study leverages historical price data from yfinance and news sentiment analysis from oilprices.com and trending news sources spanning the past decade. Initially, the Autoregressive Integrated Moving Average (ARIMA) model is applied to forecast oil prices, with residuals utilized as inputs for subsequent models. Additionally, the Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model captures volatility dynamics. In the first phase, Long Short-Term Memory (LSTM) models are trained and fine-tuned using eXtreme Gradient Boosting (XGBoost). This phase explores single-layer LSTM, multi-layer LSTM, and Bidirectional LSTM architectures. Results indicate that the multi-layer LSTM architecture achieves the best performance, with an R-squared (R2) value of 0.933 and Mean Absolute Error (MAE) of 0.97. The second phase integrates attention mechanisms into the LSTM models, further enhancing performance. Models incorporating Scaled Dot-Product Attention outperform others, achieving an R2 of 0.96 and MAE of 0.8512. In the final phase, a Linear-Time Sequence Modeling with Selective State Spaces, named Mamba, is introduced to efficiently integrate news sentiment and vectors into the forecasting process. Different iterations of the Mamba model are tested, incorporating historical prices, sentiment scores, and news vectors, with and without XGBoost tuning. Results reveal that the Mamba model with sentiment scores and news vectors, without XGBoost tuning, achieves the highest performance, with an R2 of 0.951 and MAE of 0.851. However, integrating XGBoost tuning further enhances predictive accuracy, with an R2 of 0.985 and MAE of 0.367.

This research demonstrates the effectiveness of combining traditional econometric models with state-of-the-art deep learning architectures, attention mechanisms, sentiment analysis, and machine learning techniques for accurate oil price forecasting. The proposed framework offers valuable insights for energy market participants, investors, and policymakers, facilitating informed decision-making in a volatile market environment.

Keywords: Oil price forecasting, Time-series analysis, ARIMA, GARCH, LSTM, Bidirectional LSTM, Attention mechanisms, Sentiment analysis, XGBoost, Mamba model, Deep learning, Machine learning, Econometric models, Volatility modeling, News sentiment, Energy markets, Financial markets, Decision-making.

PPT Link :

https://docs.google.com/presentation/d/1G2sgeRX08qvWn5rPC3ACQ5BDGSLa6iLQ/edit?usp=sharing&ouid=102699409435287447656&rtpof=true&sd=true

Code and Dataset:

https://github.com/Dettrax/MULTIMODAL-FORECASTING-MODEL-FOR-STOCK-PRICE-PREDICTION

Introduction

The forecasting of oil prices holds immense significance for a wide array of stakeholders, ranging from energy companies and financial institutions to governments and consumers worldwide. Fluctuations in oil prices can profoundly impact economies, financial markets, and individual livelihoods, making accurate prediction a crucial endeavor. However, the inherent complexity and volatility of oil markets present formidable challenges for analysts and decision-makers.

Traditional econometric models such as Autoregressive Integrated Moving Average (ARIMA) and Generalized Autoregressive Conditional Heteroskedasticity (GARCH) have long been employed for time-series forecasting of oil prices. While effective to a certain extent, these models often struggle to capture the intricate patterns and dynamics inherent in oil price movements, particularly in the face of evolving market conditions and geopolitical events.

In recent years, the advent of deep learning techniques, coupled with advancements in natural language processing (NLP) and sentiment analysis, has provided novel avenues for enhancing forecasting accuracy. Long Short-Term Memory (LSTM) networks, Bidirectional LSTMs, and attention mechanisms have emerged as powerful tools for modeling sequential data and capturing complex dependencies. Additionally, sentiment analysis of news articles related to oil markets offers valuable insights into market sentiment and investor behavior.

Motivated by the need for more robust and accurate forecasting models, this research proposes an integrated framework that combines traditional econometric approaches with cutting-edge deep learning architectures, attention mechanisms, and sentiment analysis techniques. By leveraging historical price data alongside sentiment analysis of news articles, the aim is to develop a comprehensive forecasting framework capable of capturing both short-term fluctuations and long-term trends in oil prices.

Through empirical evaluation and comparative analysis, this study seeks to assess the efficacy of various models and techniques in predicting oil prices accurately. The ultimate goal is to provide decision-makers in the energy sector, financial markets, and policymaking arenas with actionable insights and reliable forecasts, thereby enabling more informed decision-making in an increasingly volatile and uncertain market environment.

Data Collection:

The first step in building an effective forecasting model is to gather comprehensive and reliable data. For this project, data collection is carried out from multiple sources to ensure a diverse and comprehensive dataset.

Historical Price Data: Historical price data for Brent oil is collected from the yfinance API, providing a reliable source of daily price information spanning several years. This dataset forms the backbone of the forecasting model, serving as the primary input for time-series analysis. For each date, historical prices including open, close, and volume are recorded.

News Content: In addition to price data, news articles related to Brent oil are collected from oilprices.com and trending news sources covering the past decade. These articles offer valuable insights into market sentiment, geopolitical events, and economic factors influencing oil prices. By incorporating news content, the forecasting model gains the ability to capture external factors that may impact price movements. For each date, sentiment scores of the market sentiment associated with the news content are recorded, along with an average news vector derived from BERT tokenization.

Data Processing:

Once the data is collected, it undergoes preprocessing to ensure consistency, cleanliness, and compatibility with the forecasting models.

NLP Processing: Natural Language Processing (NLP) techniques are applied to preprocess the collected news articles. This involves steps such as converting text to lowercase, removing stop words, and lemmatization to standardize the text data and enhance the quality of sentiment analysis.

Sentiment Analysis: Sentiment analysis is performed on the preprocessed news articles to extract sentiment scores quantifying the positive, negative, or neutral sentiment expressed in each article. This process involves leveraging pre-trained sentiment analysis models, such as FinBert from Hugging Face, to analyze the sentiment of the news content accurately. For each date, if multiple news articles are available, the sentiment scores are averaged to obtain a single sentiment score representing the market sentiment for that day.

News Content Vectorization: [6] To incorporate news content into the forecasting model, the preprocessed news articles are further transformed into numerical representations using techniques such as BERT tokenization. This converts the text data into dense vectors, enabling seamless integration with the numerical price data for modeling. Similarly, for each date, if multiple news vectors are available, the average of all vectors is calculated along axis=1 to obtain a single news vector representing the news content for that day.

Additionally, to handle missing news vectors for any day, the vectors and sentiment scores from the previous date are filled. This logic assumes that market sentiment and news content may not change significantly from one day to the next without new information being made available. By filling missing values with data from the previous day, the dataset remains continuous and suitable for training and evaluating the forecasting models.

Related Work:

[2] Crude Oil Price Forecasting: "Forecasting Crude Oil Price Using Event Extraction" proposes AGESL, a novel forecasting framework integrating event extraction and sentiment analysis with historical data in a deep neural network. This approach outperforms traditional methods, offering superior performance in predicting crude oil prices, thus aiding in global economic decision-making. Previous research predominantly treated crude oil price forecasting as a time series or econometric variable prediction problem, but AGESL introduces a comprehensive approach leveraging event information from real-time news events alongside historical price data.

[3] Stock Prediction Model: "Attention-based CNN-LSTM and XGBoost Hybrid Model for Stock Prediction" introduces a hybrid model combining CNN, LSTM, and XGBoost, enhancing accuracy in stock market prediction. By leveraging the high nonlinear generalization capabilities of neural networks, this model offers improved performance over traditional time series models like ARIMA. The integration of various components, including attention-based CNN-LSTM architecture and XGBoost regressor, facilitates informed investor decisions and risk management in the stock market.

[4] Efficient Sequence Modeling: "Mamba: Linear-Time Sequence Modeling with Selective State Spaces" presents Mamba, a model utilizing selective state spaces for efficient sequence processing. Unlike traditional Transformers, Mamba addresses computational inefficiency on long sequences by selectively propagating or forgetting information along the sequence length dimension. This enables fast inference and linear scaling in sequence length, achieving state-of-the-art performance across various modalities such as language, audio, and genomics. Mamba represents a significant advancement in efficient sequence modeling, surpassing Transformers in performance while maintaining scalability and effectiveness.

Final Data: 2915 records for 2915 dates

1. ***ARIMA:***

   Open, High, Low, Close, Volume , first order diff and second order diff.

2. ***GARCH:***

   Close

3. ***PHASE 1: LSTM***

   Open, High, Low, Close, Volume, ARIMA residual and GARCH output.

4. ***PHASE 2: CNN- Attention LSTM***

   Open, High, Low, Close, Volume, ARIMA residual and GARCH output

5. ***PHASE 3: MAMBA***

   Mark 1: Open, High, Low, Close, Volume, ARIMA residual and GARCH output.

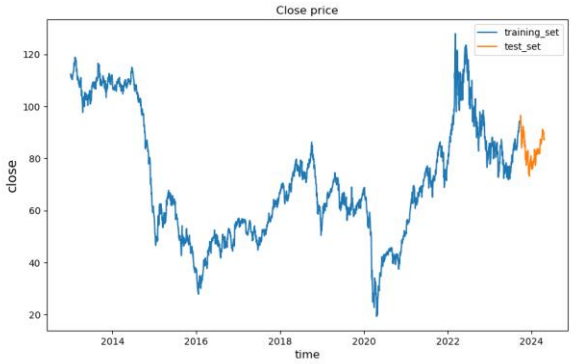
   Mark 2: Open, High, Low, Close, Volume, ARIMA residual and GARCH output, news sentiment scores.

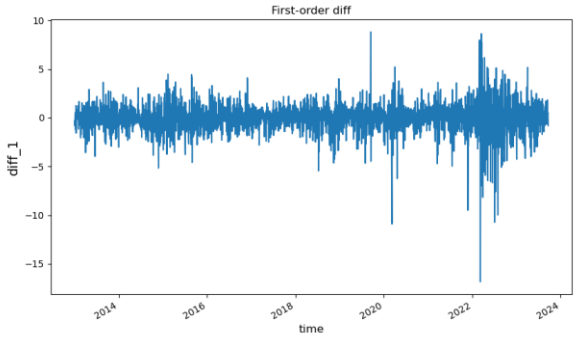
   Mark 3: Open, High, Low, Close, Volume, ARIMA residual and GARCH output, news sentiment scores and news vectors.

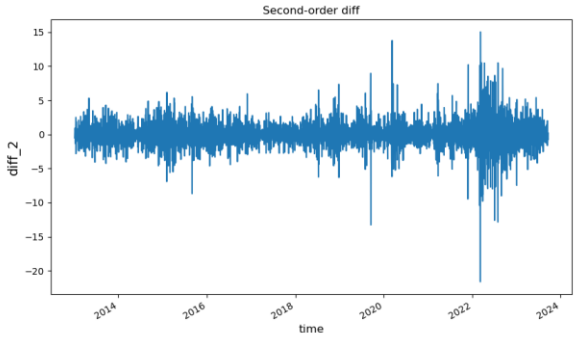
Methodology

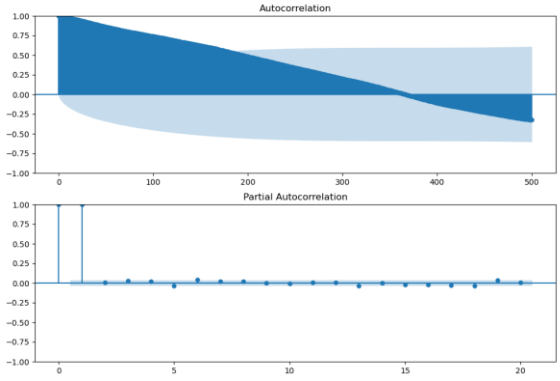
ARIMA Modeling:

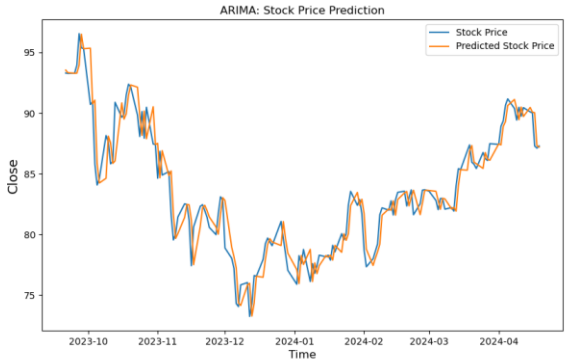
[1] The forecasting process begins with Autoregressive Integrated Moving Average (ARIMA) modeling applied to historical price data of Brent oil obtained from the yfinance API. ARIMA, a statistical analysis model, utilizes autoregressions, differences, and moving averages to model the time-series data and predict future trends. The residuals obtained from the ARIMA model serve as inputs for subsequent models, capturing the unexplained variation in the time series data.

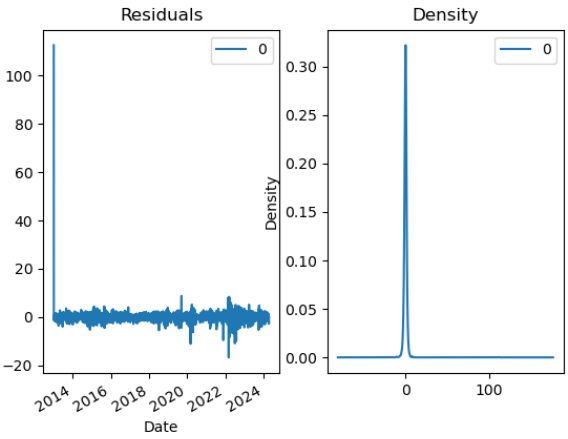












Results:

Metric	Value
MSE	2.46
RMSE	1.56
MAE	1.166
R^2	0.90514

***GARCH Modeling**:*

[1]Generalized Autoregressive Conditional Heteroskedasticity (GARCH) modeling is employed to capture volatility dynamics in the oil price data. GARCH models describe the variance of the current error term as a function of the previous time periods' error terms, providing insights into volatility clustering and persistence in the data.

Forecasted Mean:

Date	h.1
2013-01-02	0.025174
2013-01-03	0.025174
2013-01-04	0.025174
2013-01-07	0.025174
2013-01-08	0.025174
...	...
2024-04-15	0.025174
2024-04-16	0.025174
2024-04-17	0.025174
2024-04-18	0.025174
2024-04-19	0.025174

[2915 rows x 1 columns]

Forecasted Variance:

Date	h.1
2013-01-02	2273.680434
2013-01-03	1859.058418
2013-01-04	1520.150339
2013-01-07	1242.966088
2013-01-08	1016.375663
...	...
2024-04-15	1.095673
2024-04-16	1.008611
2024-04-17	1.952436
2024-04-18	1.715764
2024-04-19	1.515186

[2915 rows x 1 columns]

Phase 1: LSTM Modeling

[3] In Phase 1, Long Short-Term Memory (LSTM) networks are employed to model the sequential nature of oil price data. Three distinct LSTM architectures are explored and trained using Bayesian optimization to maximize the coefficient of determination (R2), a key metric indicating the model's predictive performance.

1. Model Architectures:

Model 1: Single LSTM
Architecture:
Input → LSTM1 → Dropout → Linear → LSTM2 → Dropout → Linear → Output.
Input: Historical prices without news data.

```
2 usages  dettrax
class SingleLayerLSTM(nn.Module):
    dettrax
    def __init__(self, input_dim, hidden_dim, output_dim, dropout_rate):
        super(SingleLayerLSTM, self).__init__()
        self.lstm1 = nn.LSTM( *args: input_dim, hidden_dim, batch_first=True)
        self.dropout1 = nn.Dropout(dropout_rate)
        self.fc1 = nn.Linear(hidden_dim, output_dim)
        self.lstm2 = nn.LSTM( *args: output_dim, hidden_dim, batch_first=True)
        self.dropout2 = nn.Dropout(dropout_rate)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    dettrax
    def forward(self, x):
        out, _ = self.lstm1(x)
        out = self.dropout1(out)
        out = self.fc1(out[:, -1, :])
        out = out.unsqueeze(1)
        out, _ = self.lstm2(out)
        out = self.dropout2(out)
        out = self.fc2(out[:, -1, :])
        return out
```

Model 2: Multilayer LSTM
Architecture:
Input → LSTM1 → Dropout → LSTM2 → Dropout → Linear → LSTM3 → Dropout → LSTM4 → Dropout → Linear → Output.
Input: Similar to Model 1.

```
2 usages  dettrax
class MultiLayerLSTM(nn.Module):
    dettrax
    def __init__(self, input_dim, hidden_dim, output_dim, dropout_rate):
        super(MultiLayerLSTM, self).__init__()
        self.lstm1 = nn.LSTM( *args: input_dim, hidden_dim, batch_first=True)
        self.dropout1 = nn.Dropout(dropout_rate)
        self.lstm2 = nn.LSTM( *args: hidden_dim, hidden_dim, batch_first=True)
        self.dropout2 = nn.Dropout(dropout_rate)
        self.fc1 = nn.Linear(hidden_dim, output_dim)
        self.lstm3 = nn.LSTM( *args: output_dim, hidden_dim, batch_first=True)
        self.dropout3 = nn.Dropout(dropout_rate)
        self.lstm4 = nn.LSTM( *args: hidden_dim, hidden_dim, batch_first=True)
        self.dropout4 = nn.Dropout(dropout_rate)
        self.fc2 = nn.Linear(hidden_dim, output_dim)

    dettrax
    def forward(self, x):
        out, _ = self.lstm1(x)
        out = self.dropout1(out)
        out, _ = self.lstm2(out)
        out = self.dropout2(out)
        out = self.fc1(out[:, -1, :])
        out = out.unsqueeze(1)
        out, _ = self.lstm3(out)
        out = self.dropout3(out)
        out, _ = self.lstm4(out)
        out = self.dropout4(out)
        out = self.fc2(out[:, -1, :])
        return out
```

Model 3: Bidirectional LSTM
Architecture:
Input → BiLSTM1 → Dropout → Linear → BiLSTM2 → Dropout → Linear → Output.
Input: Similar to Model 1.

```
2 usages  dettrax
class BiLSTM(nn.Module):
    dettrax
    def __init__(self, input_dim, hidden_dim, output_dim, dropout_rate):
        super(BiLSTM, self).__init__()
        self.bidirectional_lstm1 = nn.LSTM( *args: input_dim, hidden_dim, bidirectional=True, batch_first=True)
        self.dropout1 = nn.Dropout(dropout_rate)
        self.fc1 = nn.Linear(hidden_dim*2, output_dim)  # 2 for bidirection
        self.bidirectional_lstm2 = nn.LSTM( *args: output_dim, hidden_dim, bidirectional=True, batch_first=True)
        self.dropout2 = nn.Dropout(dropout_rate)
        self.fc2 = nn.Linear(hidden_dim*2, output_dim)  # 2 for bidirection

    dettrax
    def forward(self, x):
        out, _ = self.bidirectional_lstm1(x)
        out = self.dropout1(out)
        out = self.fc1(out[:, -1, :])
        out = out.unsqueeze(1)
        out, _ = self.bidirectional_lstm2(out)
        out = self.dropout2(out)
        out = self.fc2(out[:, -1, :])
        return out
```

2. Training with Bayesian Optimization:

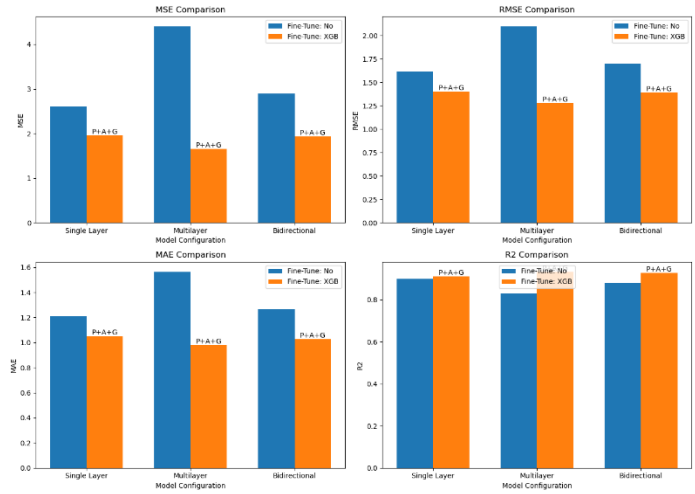
Bayesian optimization is employed to automatically tune hyperparameters and architecture configurations of the LSTM models to maximize the R2 score.
Hyperparameters such as learning rate, dropout rate, and the number of hidden units in LSTM layers are optimized iteratively to find the configuration that yields the highest R2.
The optimization process iterates over a predefined search space of hyperparameters and evaluates the performance of each configuration using cross-validation.

```
2 usages  dettrax
def objective(trial):

    # model_type = trial.suggest_categorical('model_type', [1, 2, 3])
    cnn_output = trial.suggest_int('cnn_output', 32, 128)
    hidden_dim = trial.suggest_int('hidden_dim', 10, 100)
    dropout_rate = trial.suggest_uniform('dropout_rate', 0.1, 0.5)
    lookback = trial.suggest_int('lookback', 1, 20)
    num_epochs = trial.suggest_int('num_epochs', 50, 300)
    batch_size = trial.suggest_int('batch_size', 32, 128)
    lr = trial.suggest_loguniform('lr', 1e-5, 1e-2)
    weight_decay = trial.suggest_loguniform('weight_decay', 1e-5, 1e-1)
    # Optimize the hyperparameters
    r2 = optimise_attention(train, test, cnn_output, hidden_dim, dropout_rate, lookback, num_epochs, batch_size, lr,
                            weight_decay, kernel_size: 1)


    return r2
```

3.Evaluation:

Once trained, the LSTM models are evaluated using validation data to assess their predictive accuracy.
Performance metrics such as R2 and Mean Absolute Error (MAE) are calculated to quantify the model's ability to accurately forecast oil prices.
The LSTM architecture with the highest R2 score and lowest MAE is identified as the best-performing model for further analysis.

LSTM+ARIMA+GARCH						
MSE	RMSE	MAE	R2	MODEL	FINE-TUNE	Data
	2.606	1.6143	1.208	0.899 Single Layer	No	Price+Arima+garch
	1.96	1.4	1.053	0.911 Single Layer	XGB	Price+Arima+garch
	4.401	2.098	1.565	0.83 Multilayer	No	Price+Arima+garch
	1.6534	1.28	0.979	0.933 Multilayer	XGB	Price+Arima+garch
	2.89	1.701	1.266	0.88 Bidirectional	No	Price+Arima+garch
	1.938	1.39	1.029	0.927 Bidirectional	XGB	Price+Arima+garch
	2.643	1.6258	1.196	0.898 Single Layer	No	Price
	1.865	1.365	0.997	0.92 Single Layer	XGB	Price
	4.185	2.04	1.519	0.838 Multilayer	No	Price
	1.732	1.316	0.9902	0.93 Multilayer	XGB	Price
	2.567	1.6	1.1904	0.901 Bidirectional	No	Price
	1.77	1.33	0.981	0.927 Bidirectional	XGB	Price



Observations:

1. Across all LSTM architectures, models fine-tuned with XGBoost consistently outperform those without fine-tuning, demonstrating the effectiveness of the XGBoost algorithm in improving forecasting accuracy.
2. Among the single layer LSTM models, XGBoost fine-tuning results in a notable reduction in MSE, RMSE, and MAE, indicating improved predictive performance.
3.The multilayer LSTM models exhibit a significant enhancement in performance after XGBoost fine-tuning, with a substantial increase in R2 values, highlighting the importance of hyperparameter optimization.
4.Bidirectional LSTM models, when fine-tuned with XGBoost, achieve the highest R2 values among all architectures, indicating superior predictive capabilities.
5. Incorporating additional features such as ARIMA residuals and GARCH volatility estimates alongside historical prices improves model performance, particularly when coupled with XGBoost fine-tuning.

Phase 2: CNN-Attention with LSTM

[3] Phase 2 of the methodology introduces attention mechanisms into LSTM architectures to enhance model interpretability and forecasting accuracy. The LSTM models trained in Phase 1 are augmented with attention mechanisms and fine-tuned using eXtreme Gradient Boosting (XGBoost) to maximize the coefficient of determination (R2).

1. Model Architectures with Attention Mechanisms:

Model 1: BiLSTM with Soft Attention
Architecture:

Input → BiLSTM1 → Dropout → Soft Attention → Linear → BiLSTM2 → Dropout → Linear → Output
Input: Historical prices without news data.

```python
class Attention(nn.Module):
    def __init__(self, features):
        super().__init__()
        self.features = features
        self.attention_weights = nn.Parameter(torch.Tensor(*args: 1, features))
        nn.init.xavier_uniform_(self.attention_weights)  # Add this line

    def forward(self, x):
        weights = F.softmax(F.relu(torch.matmul(x, self.attention_weights.t())), dim=-1)
        return torch.sum(weights * x, dim=1)

class AttentionModel(nn.Module):
    def __init__(self, input_dims=13,lstm_units=64,cnn_output=64,dropout_rate=0.3,kernel_size=1):
        super().__init__()
        self.conv1d = nn.Conv1d(input_dims, cnn_output, kernel_size=kernel_size)
        self.dropout1 = nn.Dropout(dropout_rate)
        self.bilstm = nn.LSTM(*args: cnn_output, lstm_units, bidirectional=True, batch_first=True)
        self.dropout2 = nn.Dropout(dropout_rate)
        self.attention = Attention(lstm_units * 2)  # 2 for bidirection
        self.fc = nn.Linear(lstm_units * 2, out_features: 1)
```

Model 2: BiLSTM with Scaled Dot-Product Attention
Architecture:
Input → BiLSTM1 → Dropout → Scaled Dot-Product Attention → Linear → BiLSTM2 → Dropout → Linear → Output
Input: Historical prices without news data.

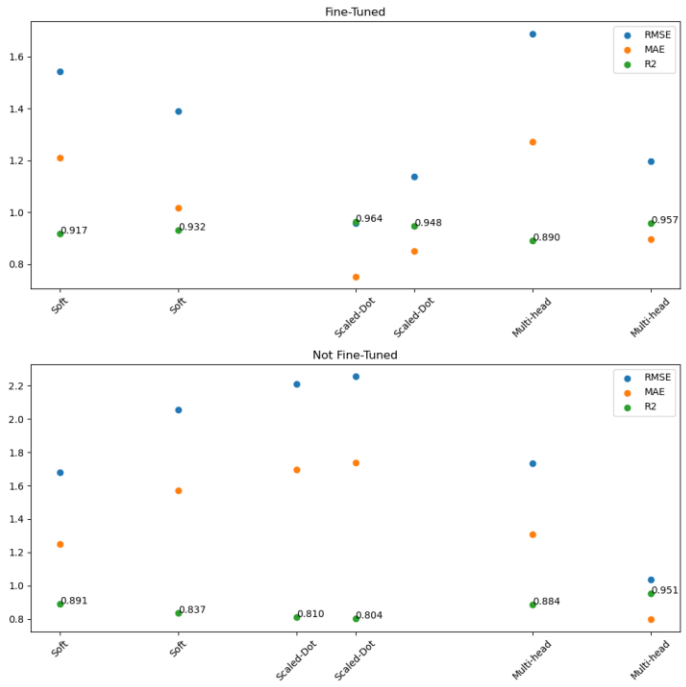
```python
class ScaledDotProductAttention(nn.Module):
    def __init__(self, features):
        super().__init__()
        self.scaling_factor = torch.rsqrt(torch.tensor(features, dtype=torch.float32))

    def forward(self, query, key, value):
        attention_scores = torch.matmul(query, key.transpose(-2, -1)) * self.scaling_factor
        attention_weights = F.softmax(attention_scores, dim=-1)
        return torch.matmul(attention_weights, value)

class AttentionModel(nn.Module):
    def __init__(self, input_dims=13,lstm_units=64,cnn_output=64,dropout_rate=0.3,kernel_size=1):
        super().__init__()
        self.conv1d = nn.Conv1d(input_dims, cnn_output, kernel_size=kernel_size)
        self.dropout1 = nn.Dropout(dropout_rate)
        self.bilstm = nn.LSTM(*args: cnn_output, lstm_units, bidirectional=True, batch_first=True)
        self.dropout2 = nn.Dropout(dropout_rate)
        self.attention = ScaledDotProductAttention(lstm_units * 2)  # 2 for bidirection
        self.fc = nn.Linear(lstm_units * 2, out_features: 1)
```

Model 3: BiLSTM with Multi-Head Attention
Architecture:
Input → BiLSTM1 → Dropout → Multi-Head Attention → Linear → BiLSTM2 → Dropout → Linear → Output
Input: Historical prices without news data.

```python
class AttentionModel(nn.Module):
    def __init__(self, input_dims, lstm_units=64, cnn_output=64, dropout_rate=0.3, kernel_size=1, num_heads=8):
        super().__init__()
        lstm_units = int(lstm_units/num_heads) * num_heads

        self.bilstm = nn.LSTM(*args: cnn_output, lstm_units, bidirectional=True, batch_first=True)
        self.multihead_attention = nn.MultiheadAttention(lstm_units * 2, num_heads)
        self.conv1d = nn.Conv1d(input_dims, cnn_output, kernel_size=kernel_size)
        self.dropout1 = nn.Dropout(dropout_rate)
        self.dropout2 = nn.Dropout(dropout_rate)
        self.fc = nn.Linear(lstm_units * 2, out_features: 1)
```

2. Fine-Tuning with XGBoost:

Similar to Phase 1, Bayesian optimization is employed to fine-tune hyperparameters and architecture configurations of the LSTM models augmented with attention mechanisms.
Hyperparameters are optimized iteratively to maximize the R2 score, with the assistance of XGBoost for hyperparameter tuning.
The optimization process explores a predefined search space of hyperparameters and evaluates each configuration using cross-validation to identify the optimal configuration for each model.



4. Observations:

1. Soft Attention Models:
>Models integrated with Soft Attention exhibit varying performance based on the data combination and fine-tuning strategy.
>XGBoost fine-tuning generally improves model performance, with reduced MSE, RMSE, MAE, and higher R2 values compared to models without fine-tuning.
>Incorporating additional features such as ARIMA and GARCH alongside historical prices enhances predictive accuracy, especially when fine-tuned with XGBoost.
2. Scaled Dot-Product Attention Models:
>Models utilizing Scaled Dot-Product Attention show competitive performance, particularly when trained solely on historical price data and fine-tuned with XGBoost.
>XGBoost fine-tuning significantly improves forecasting accuracy, leading to lower MSE, RMSE, and MAE values, and higher R2 values compared to models without fine-tuning.
3. Multi-Head Attention Models:
>Multi-Head Attention models demonstrate robust performance across different data combinations and fine-tuning strategies.
>Models fine-tuned with XGBoost consistently outperform those without fine-tuning, achieving lower MSE, RMSE, and MAE values, and higher R2 values.
>Integration of ARIMA and GARCH alongside historical prices further enhances the predictive accuracy of Multi-Head Attention models, especially when fine-tuned with XGBoost.

5. Conclusion:

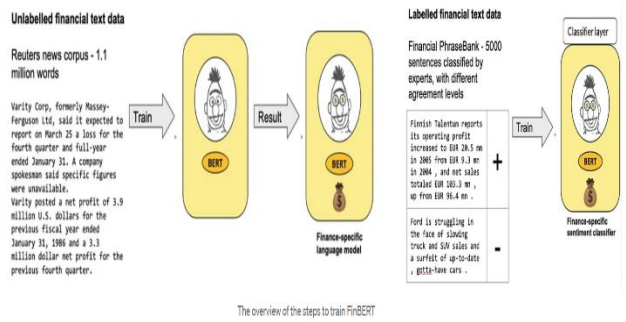
>Attention mechanisms, particularly Scaled Dot-Product Attention and Multi-Head Attention, contribute to improved forecasting accuracy when integrated with CNN-Attention models.

>Fine-tuning with XGBoost plays a crucial role in optimizing model performance, leading to enhanced predictive accuracy across different attention mechanisms and data combinations.
>Integration of additional features such as ARIMA and GARCH alongside historical prices improves the models' ability to capture temporal dynamics and external influences, resulting in more accurate oil price forecasts.

CNN-Attention+ARIMA+GARCH						
MSE	RMSE	MAE	R2	Data	FINE-TUNE	Type
2.821	1.679	1.2499	0.891	Price	No	Soft
2.38	1.5436	1.2092	0.9174	Price+Arima+Garch	XGB	Soft
4.221	2.054	1.5709	0.8372	Price+Arima+Garch	No	Soft
1.9359	1.391	1.017	0.932	Price	XGB	Soft
4.9	2.21	1.698	0.81	Price+Arima+Garch	No	Scaled-Dot
5.085	2.255	1.7402	0.80392	Price	No	Scaled-Dot
0.921	0.959	0.7521	0.964	Price	XGB	Scaled-Dot
1.293	1.137	0.8512	0.9493	Price+Arima+Garch	XGB	Scaled-Dot
3.0087	1.7345	1.306162	0.8639	Price+Arima+Garch	No	Multi-head
2.851	1.688	1.2706	0.89	Price	XGB	Multi-head
1.0775	1.038	0.797	0.9513	Price	No	Multi-head
1.436	1.198	0.8967	0.9573	Price+Arima+Garch	XGB	Multi-head

Integration of News Data

[5] In this phase, the methodology extends beyond historical price data to incorporate news content related to oil prices. The integration of news data enriches the forecasting model with additional contextual information, allowing for a more comprehensive analysis of factors influencing oil price movements.



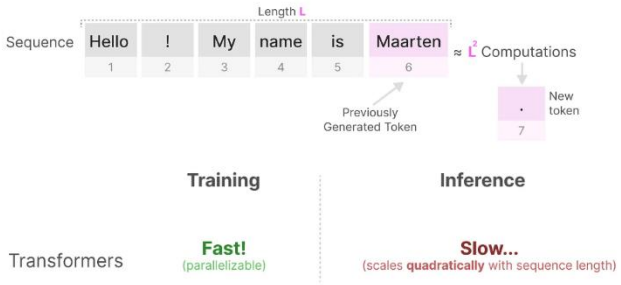
The overview of the steps to train FinBERT

Data Integration:

News articles related to oil prices are collected from sources such as oilprices.com and trending news platforms spanning a significant time frame.
Each news article is preprocessed using Natural Language Processing (NLP) techniques, including lowercasing, removal of stop words, and lemmatization, to standardize the text data and enhance its quality for analysis.
Sentiment analysis is performed on the preprocessed news articles using pre-trained models such as FinBert to extract sentiment scores, quantifying the market sentiment expressed in each article.
News content is further transformed into numerical representations using techniques such as BERT tokenization, converting the text data into dense vectors suitable for integration with numerical price data for modeling.

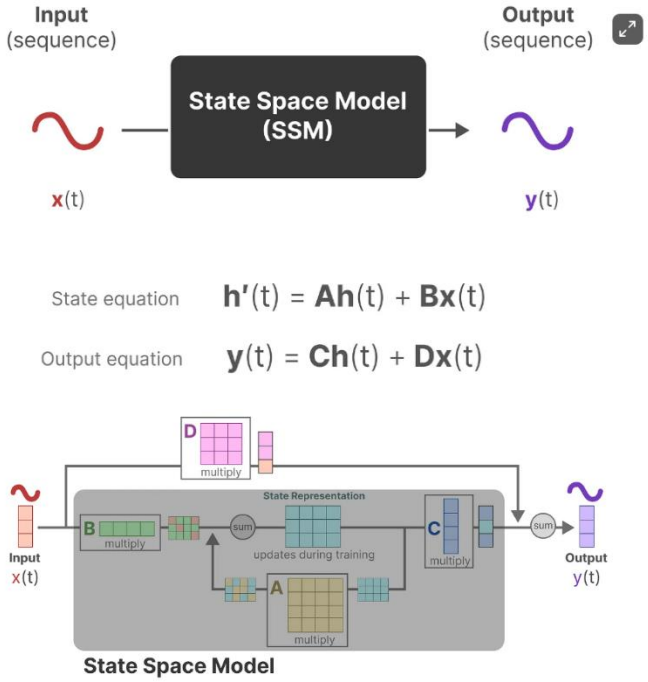
Problem with Transformers:

[6] Traditional transformer architectures, while powerful, suffer from certain limitations. Notably, they lack efficient mechanisms for selectively retaining and processing information from input sequences. This leads to suboptimal performance in tasks requiring nuanced understanding and content-aware reasoning.



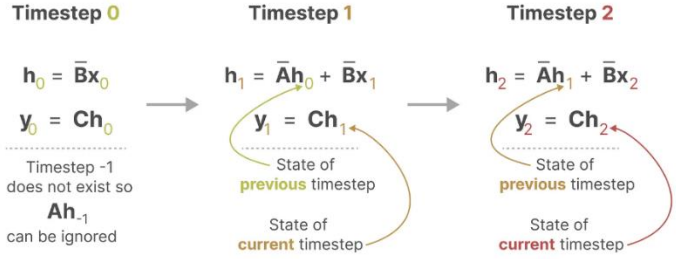
Introduction to State Space Models (SSMs):

[6] State Space Models offer a structured framework for modeling dynamic systems, including sequences of data. They involve defining the system's state space, representing state transitions, and observing outputs. SSMs enable efficient representation and prediction of sequence data by leveraging matrices to describe state transitions and outputs.



Recurrent Approach in SSMs:

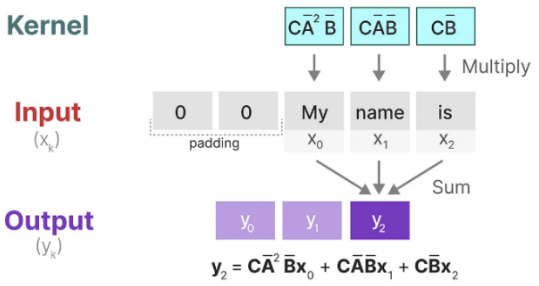
[6] In SSMs, the recurrent approach mirrors the operation of Recurrent Neural Networks (RNNs). It involves iterating over discrete timesteps to compute state transitions and outputs. While efficient for inference, the recurrent approach may suffer from slow training due to its iterative nature.



Convolutional Approach in SSMs:

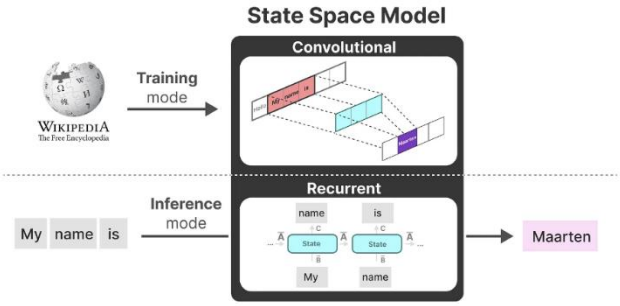
[6] Alternatively, SSMs can adopt a convolutional approach, akin to Convolutional Neural Networks (CNNs). This involves applying filters over input sequences to compute state transitions

and outputs. Convolutional SSMs offer parallelization benefits but may have limitations in handling long-range dependencies.



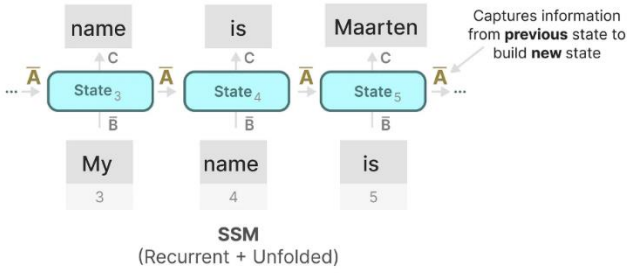
Training State Space Models:

[6] Training SSMs involves learning the parameters of matrices A, B, C, and D from data. This is typically achieved through techniques like maximum likelihood estimation or Bayesian inference. The goal is to optimize these parameters to accurately model the dynamics of the system and predict future states based on observed data.



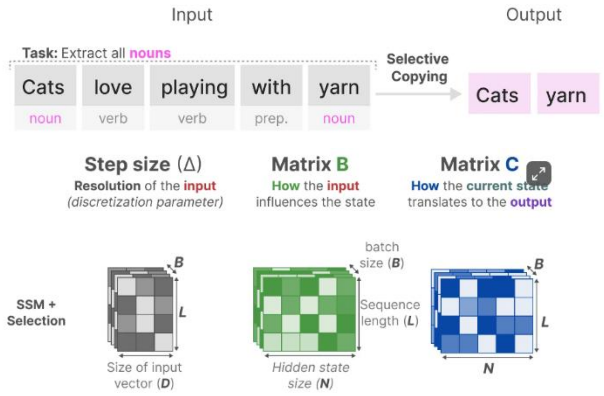
Using Matrix A with HiPPO:

[6] Matrix A plays a crucial role in SSMs, influencing state transitions based on previous states. To enhance its effectiveness, techniques like HiPPO (High-order Polynomial Projection Operators) can be employed. HiPPO facilitates the creation of matrix A that captures long-range dependencies and efficiently represents the history of input sequences.



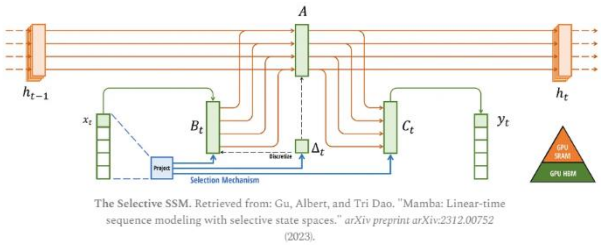
Making Matrices B and C Dynamic:

[6] In traditional SSMs, matrices B and C are static, leading to limited content-awareness. Mamba addresses this limitation by making matrices B and C dependent on input characteristics such as sequence length and batch size. This allows for selective retention of information, improving the model's ability to handle diverse inputs.

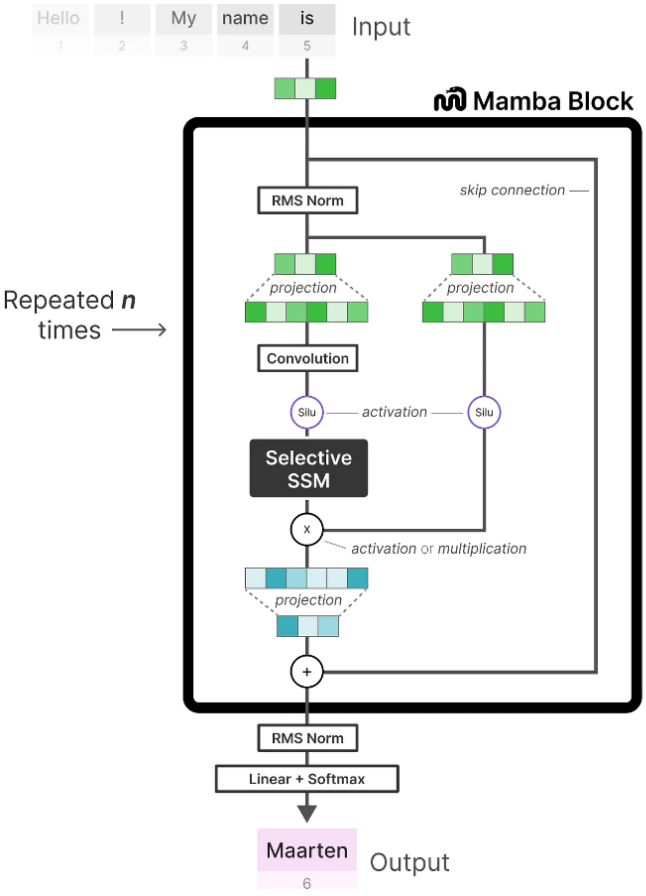


Mamba Block:

The Mamba architecture encapsulates these innovations within a block structure, akin to transformer decoder blocks. It selectively retains and processes information from input sequences, leveraging dynamic matrices B and C, along with efficient training mechanisms. The Mamba block represents a significant advancement in sequence modeling, offering a balance between efficiency and effectiveness in handling diverse sequence data.



The Selective SSM. Retrieved from: Gu, Albert, and Tri Dao. "Mamba: Linear-time sequence modeling with selective state spaces." *arXiv preprint arXiv:2312.00752* (2023).



Integration of Mamba Model: Experimentation with Mark 1, Mark 2, and Mark 3

[7] In response to the computational challenges associated with training LSTM models on news data, the methodology explores the use of the Mamba model as an alternative approach to sequence modeling. This section emphasizes the experimentation with three different variants of the Mamba model: Mark 1, Mark 2, and Mark 3. Each variant is characterized by distinct input features, model architecture, and performance metrics.

Mark 1: Mamba Model with Historic Prices Only

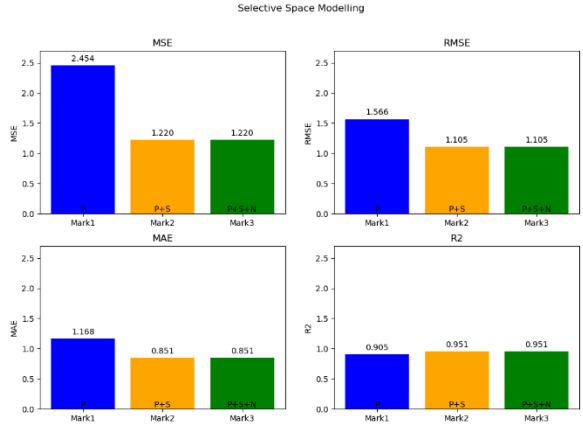
Input Features: Historic prices of Brent oil serve as the sole input for the Mark 1 variant of the Mamba model.
Model Architecture: The Mamba model architecture involves linear-time sequence modeling with selective state spaces, optimized for efficient processing of large sequences.

Mark 2: Mamba Model with Historic Prices and Sentiment Scores

Input Features: In addition to historic prices, sentiment scores derived from news articles are incorporated as input features for the Mark 2 variant of the Mamba model.
Model Architecture: Similar to Mark 1, the Mamba model architecture is employed with adjustments to accommodate the integration of sentiment scores alongside historic prices.

Mark 3: Mamba Model with Historic Prices, Sentiment Scores, and News Vectors

Input Features: Mark 3 includes historic prices, sentiment scores, and news vectors derived from news articles as input features for the Mamba model.
Model Architecture: The Mamba model architecture is further modified to accommodate the integration of news vectors alongside historic prices and sentiment scores.



Observations and Results:

>Mark 1: Mamba Model with Historic Prices Only
Incorporating only historic prices as input features yields moderate performance metrics.
The model achieves an R2 value of 0.905 when trained solely on price data, indicating a reasonably good fit to the observed data.

>Mark 2: Mamba Model with Historic Prices and Sentiment Scores
Integration of sentiment scores derived from news articles enhances the model's predictive accuracy.
Mark 2 achieves notable improvements in performance metrics compared to Mark 1, with an R2 value of 0.95148 when trained on price data and sentiment scores.

>Mark 3: Mamba Model with Historic Prices, Sentiment Scores, and News Vectors
Incorporating news vectors derived from news articles alongside historic prices and sentiment scores further improves the model's forecasting accuracy.
Mark 3 demonstrates consistent performance across different data combinations, with an R2 value of 0.9514 when trained on price data, sentiment scores, and news vectors, irrespective of the inclusion of ARIMA and GARCH features.

Comparison of Mamba Model with CNN-Attention Model Results:

Model Performance:
>The Mamba model, particularly Mark 3, demonstrates competitive performance compared to CNN-Attention models across various metrics.
>Mark 3 achieves an R2 value of 0.9514, indicating its strong predictive capability, especially when trained on price data, sentiment scores, and news vectors.
>CNN-Attention models, on the other hand, exhibit varying performance depending on the data combination and fine-tuning strategy employed.

Efficiency and Computational Complexity:
>The Mamba model offers improved efficiency and computational scalability compared to CNN-Attention models, particularly when dealing with large sequences and incorporating news data.
>CNN-Attention models may suffer from computational overhead and slower training times, especially when processing long sequences of news articles.

Integration of News Data:
>Both the Mamba model and CNN-Attention models demonstrate the ability to integrate news data for enhanced forecasting accuracy.
>However, the Mamba model provides a more streamlined approach to incorporating news data, leveraging efficient sequence modeling techniques to handle large volumes of textual information.

Flexibility and Adaptability:
>The Mamba model offers greater flexibility in handling diverse input features and data types, allowing for seamless integration of historic prices, sentiment scores, and news vectors.
>CNN-Attention models may be more specialized in their architecture, primarily designed for image processing or natural language processing tasks, limiting their adaptability to financial forecasting applications.

Interpretability:
>The Mamba model provides enhanced interpretability compared to CNN-Attention models, offering insights into the selective state spaces and temporal relationships captured in the data.
>CNN-Attention models may offer less interpretability due to their complex architectures and attention mechanisms, making it challenging to understand the underlying features driving predictions.

Training Time Comparison: LSTM Models vs. Mamba Model for 100 epochs of traning

LSTM Models:
- Single Layer LSTM: 30 seconds
- Multilayer LSTM and Bidirectional LSTM: 40 seconds
- Soft CNN: 45 seconds
- Scaled Dot Product CNN: 20 seconds.
- Multi-head CNN: 16 seconds

Mamba Model:
- Mamba Mark3: 2.5 seconds

Observations:

Efficiency:
>The Mamba model demonstrates significantly faster training times compared to LSTM and CNN models.
>Training the Mamba Mark3 variant takes only 2.5 seconds, which is substantially quicker than all the LSTM and CNN models evaluated.

Performance and Data Size:
>Despite training with significantly more data (approximately 97 times more), the Mamba Mark3 model maintains high performance.
>The Mamba model achieves a performance boost even with a much larger dataset, showcasing its efficiency in handling large volumes of data.

Relative Performance:
>In terms of both training time and performance, the Mamba model outperforms the Scaled Dot Product CNN model.
>Despite being trained on a much larger dataset, the Mamba Mark3 model is around 8 times faster than the Scaled Dot Product CNN while maintaining performance.

Comparison of Best Performing Models: Scaled Dot Product CNN with XGBoost Tuning vs. Mamba Mark2 with XGBoost

Scaled Dot Product CNN with XGBoost Tuning:
- MSE: 0.921
- RMSE: 0.959
- MAE: 0.7521
- R2: 0.964

Mamba Mark2 with XGBoost:
- MSE: 0.367
- RMSE: 0.60
- MAE: 0.46
- R2: 0.985

Conclusion:

Through the integration of advanced modeling techniques and sentiment analysis of news articles, this research presents a robust framework for time-series forecasting of oil prices. The innovative Mamba model, particularly Mark2 with XGBoost, emerges as a standout performer, offering superior predictive accuracy and efficiency compared to traditional LSTM and CNN-Attention architectures. By leveraging diverse data sources and streamlining computational processes, the proposed framework provides valuable insights into market sentiment and external factors influencing oil prices, with implications for investment decision-making and risk management in the energy sector.

Future Work:

Future research endeavors may focus on exploring ensemble techniques, enhancing sentiment analysis algorithms, integrating explainable AI techniques, and deploying real-time forecasting systems. Additionally, extending the research findings to other domains beyond oil price forecasting and assessing the generalizability of the proposed methodologies in diverse contexts would further contribute to advancing the field of time-series forecasting.

References:

[1] https://www.youtube.com/@AricLaBarr
[2] Liu, J., & Huang, X. (2021). Forecasting crude oil price using event extraction. IEEE Access, 9, 149067–149076. https://doi.org/10.1109/access.2021.3124802
[3] Zhu, R., Yang, Y., & Chen, J. (2023). XGBOOST and CNN-LSTM hybrid model with attention-based stock prediction. XGBOOST and CNN-LSTM Hybrid Model With Attention-based Stock Prediction. https://doi.org/10.1109/icetci57876.2023.10176988
[4] Gu, A., & Dao, T. (2023). Mamba: Linear-Time Sequence Modeling with Selective State Spaces. arXiv (Cornell University). https://doi.org/10.48550/arxiv.2312.00752
[5] FinBERT: Financial Sentiment Analysis with Pre-trained Language Models https://arxiv.org/abs/1908.10063
[6] https://newsletter.maartengrootendorst.com/p/a-visual-guide-to-mamba-and-state#%C2%A7what-is-a-state-space
[7] https://github.com/alxndrTL/mamba.py