DAT301m Lab 4: Time Series Forecasting

Trieu Vuong Nguyen
AI Development with TensorFlow
FPT University
Ho Chi Minh City, Vietnam
vuongntse182185@fpt.edu.vn

1 Introduction

This lab assignment focuses on the application of time series forecasting techniques to analyze and predict future values based on historical data. Time series forecasting plays a vital role in many real-world scenarios such as stock price prediction, weather forecasting, and demand planning. In this lab, we explore key forecasting models including traditional statistical approaches and deep learning-based methods. Through hands-on implementation, students will gain practical experience in data preprocessing, model training, and performance evaluation, preparing them for more advanced forecasting tasks in real-world applications.

2 Dataset

The `AEP_hourly.csv` file from Kaggle's *Hourly Energy Consumption* dataset contains hourly electricity usage data from American Electric Power (AEP), spanning from December 31, 2004, 01:00:00 to August 3, 2018, 00:00:00. It includes two columns: a timestamp (`Datetime`) and the corresponding electricity load in megawatts (`AEP_MW`).

This dataset provides over 13 years of hourly measurements, making it well-suited for analyzing consumption patterns and building forecasting models. Preprocessing steps involve parsing datetime values and handling missing data to ensure quality inputs for time series analysis.

3 Dataset Exploration and Preprocessing

Preprocessing To prepare the dataset for modeling, several basic preprocessing steps were applied. First, the `Datetime` column was converted to the appropriate `datetime` format using `pandas`, enabling efficient time-based indexing and operations. This ensures compatibility with time series models that rely on chronological ordering.

Next, the target variable `AEP_MW`, which represents electricity consumption in megawatts, was standardized using the `StandardScaler` from `scikit-learn`. Standardization helps improve model performance and training stability by ensuring that the input data has zero mean and unit variance, which is particularly beneficial for algorithms sensitive to feature scaling, such as neural networks.

WindowGenerator The `WindowGenerator` class is designed to facilitate time series data processing in machine learning tasks. It generates sliding windows of input and label data, which are crucial for training, validation, and testing predictive models. The class allows for flexible configuration of the window size through three parameters: `input_width`, `label_width`, and `shift`, which define the length of input sequences, label sequences, and the offset between them, respectively.

The `split_window` method extracts the corresponding input and label sequences from the dataset, ensuring that labels are selected from the correct columns (if specified). The class includes a `plot`

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

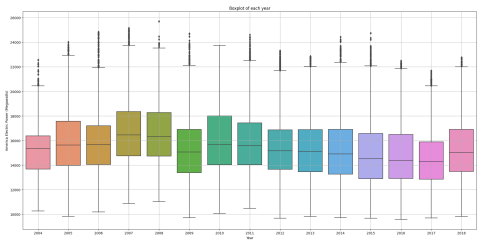

Figure 1: Boxplot of American Electric Power Consumption for Each Year.

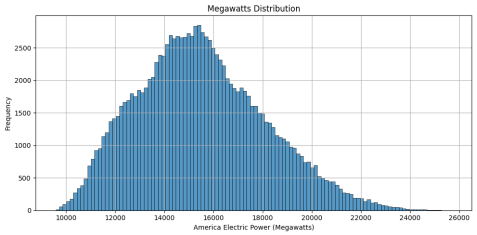

Figure 2: Distribution of American Electric Power Consumption (Megawatts).

Figure 3: Average Megawatts of American Electric Power for Each Year.

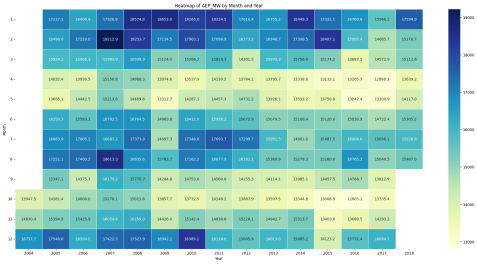

Figure 4: Heatmap of AEP (Megawatts) by Month and Year.

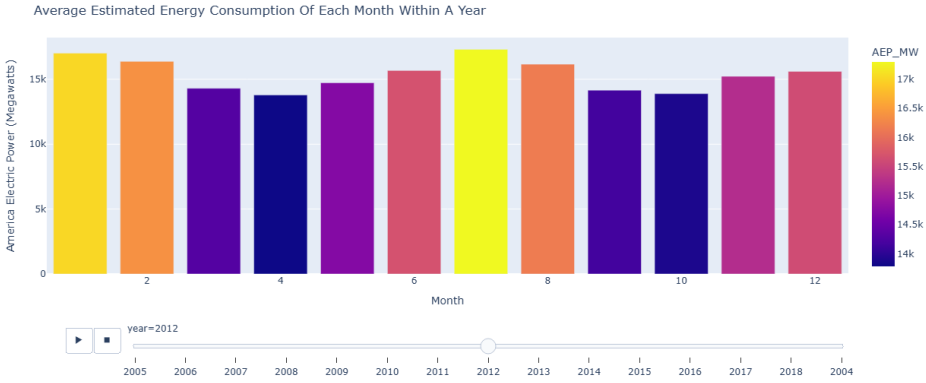

Figure 5: Average Estimated Energy Consumption of Each Month Within a Year.

method for visualizing the time series data, displaying the input data, labels, and model predictions (if available), which is valuable for evaluating model performance visually.

Moreover, the `make_dataset` method is responsible for converting the input data into a TensorFlow dataset that can be efficiently used for training and evaluation. It utilizes `timeseries_dataset_from_array` from TensorFlow, enabling seamless integration with the model training pipeline. The class also provides properties for accessing the training, validation, and test datasets, making it easy to work with the data across different stages of model development. This structure is essential for tasks such as forecasting, where the model predicts future values based on historical data.

4 Baseline Model

We evaluated two baseline models, Linear Regression and ARIMA, for forecasting growth, comparing their performance using Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE).

- **Linear Regression:**

  - MAE: 0.7002
  - RMSE: 0.9140
- **ARIMA:**
  - MAE: 1.1656
  - RMSE: 1.4631

The visualization clearly shows that the Linear Regression model (orange) closely tracks the true growth values (blue), whereas the ARIMA model (green) does not capture the fluctuations as well. Linear Regression achieved lower error metrics, making it the more effective model for this dataset in terms of forecasting accuracy.

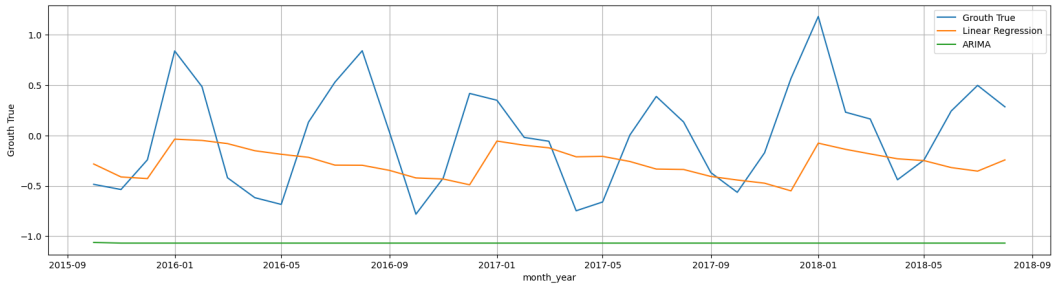

Figure 6: Forecasting results for Linear Regression and ARIMA models.

Answer Question 1

Although explicit training and validation loss curves are not provided, we can infer the following: - The Linear Regression model does not appear to exhibit overfitting, as its performance is stable with relatively low error metrics. The absence of significant overfitting indicates that the model fits the data well without excessively tailoring itself to noise. - ARIMA, however, appears to have underfitted the data. Its higher error metrics and inability to capture the fluctuations in the time series suggest that it was not able to adequately model the underlying patterns and trends.

In conclusion, the Linear Regression model exhibited better generalization, with lower errors and more accurate predictions, while the ARIMA model likely underfitted the data.

5 Deep Learning Models

Bidirectional LSTM (BiLSTM)

The **Bidirectional LSTM** model utilizes multiple layers of LSTM cells that process the input sequence in both forward and backward directions, providing richer context for each time step. This approach is particularly useful for time series forecasting tasks, where future values depend on past sequences.

Architecture:

- Three bidirectional LSTM layers with 32, 64, and 128 units respectively.
- Batch normalization and dropout to prevent overfitting.

During training, the **Mean Absolute Error (MAE)** steadily decreases, and the **Loss** also shows consistent improvement. The training and validation losses align closely, indicating that the model is generalizing well without overfitting. However, slight fluctuations in the validation error suggest room for further model refinement.

CNN + BiLSTM

The **CNN + BiLSTM** model combines a 1D convolutional layer with bidirectional LSTM layers. The convolutional layer is used for feature extraction, and the LSTM layers handle sequence modeling, making the model capable of learning both spatial and temporal patterns in time series data.

Architecture:

- A 1D convolutional layer with 64 filters and a max pooling layer to reduce dimensionality.
- Three bidirectional LSTM layers with 32, 64, and 128 units respectively.
- Batch normalization and dropout to prevent overfitting.

During training, the **MAE** and **Loss** decrease progressively, indicating successful learning. However, the validation error shows some fluctuation, and the gap between training and validation performance is slightly larger than in the BiLSTM model. This may suggest slight overfitting as the training error improves faster than the validation error.

Training Metrics Observation

Both models show a steady decrease in **Training MAE** and **Loss**, suggesting that they are learning effectively. The **Validation MAE** and **Loss** also decrease, but with some fluctuations. The **CNN + BiLSTM** model shows more variation in validation metrics, possibly indicating overfitting.

In conclusion, both models demonstrate effective learning, with the CNN + BiLSTM model requiring additional fine-tuning to improve validation performance.

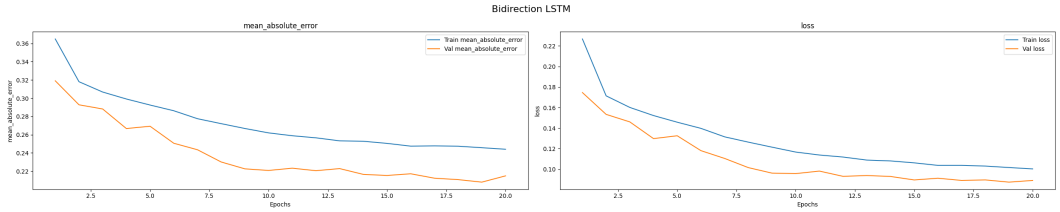

Figure 7: Training metrics for the Bidirectional LSTM model.

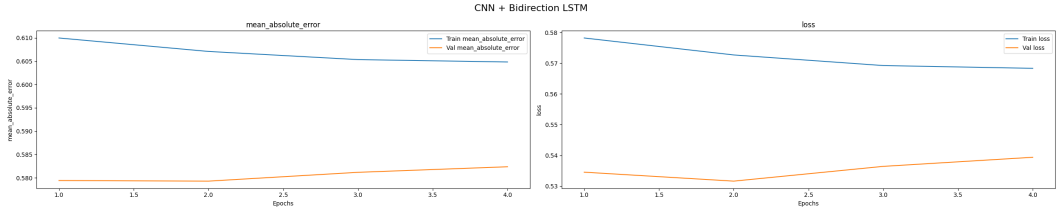

Figure 8: Training metrics for the CNN + BiLSTM model.

XGBoost

The XGBoost model was used to predict American Electric Power (AEP) consumption using time-based features derived from the ‘Datetime‘ column. Key features such as ‘hour‘, ‘dayofweek‘, ‘quarter‘, ‘month‘, ‘year‘, ‘dayofyear‘, and ‘dayofmonth‘ were extracted to capture temporal patterns.

Data Preprocessing: - The ‘Datetime‘ column was transformed into several features representing different time components, including hour, day of the week, month, and more.

Model Training: - The model was trained using the XGBoost regressor with 1000 estimators and a learning rate of 0.01, applying early stopping with a 10-round patience.

Main Features: - The model relied on time-based features such as ‘hour‘, ‘dayofweek‘, ‘month‘, and ‘year‘ to capture seasonal and cyclical patterns in the data.

Evaluation

XGBoost Model

Mean Absolute Error (MAE): 0.487
Root Mean Squared Error (RMSE): 0.627

The **XGBoost** model performed well, tracking the true growth values (blue) with minor discrepancies, particularly in peak and valley points. The forecast, represented by the orange line, is close to the true growth (blue line), providing reliable results across time steps.

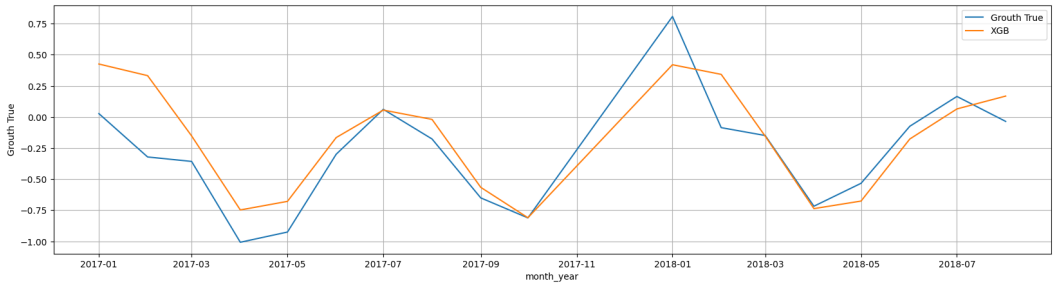

Figure 9: XGBoost Forecasting Results.

BiLSTM Model

Mean Absolute Error (MAE): 1.027
Root Mean Squared Error (RMSE): 1.287

The **BiLSTM** model captures the overall trend but fails to track sudden fluctuations in the data, leading to a higher MAE and RMSE. The model struggles to predict abrupt changes in the time series.

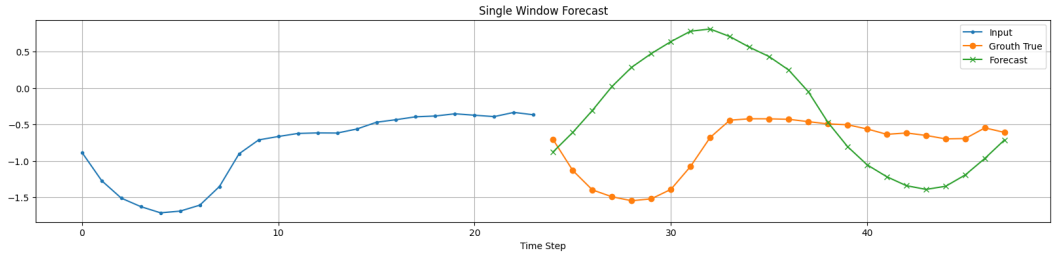

Figure 10: BiLSTM Forecasting Results.

CNN + BiLSTM Model

Mean Absolute Error (MAE): 0.901
Root Mean Squared Error (RMSE): 1.125

The **CNN + BiLSTM** model performs better than BiLSTM, with improved tracking of the true growth values (orange). While it is better at capturing broader patterns, it still does not match the precision of XGBoost.

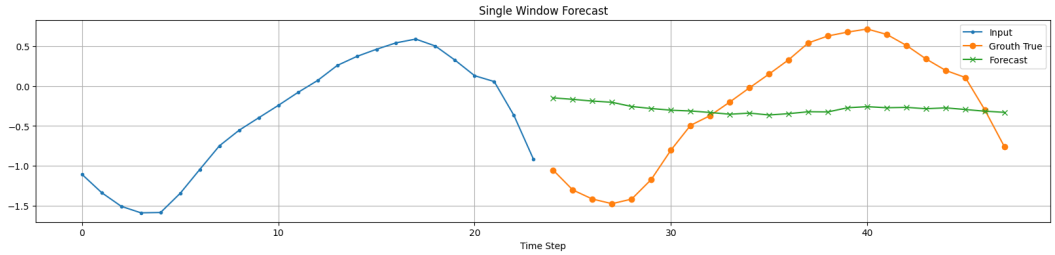

Figure 11: CNN + BiLSTM Forecasting Results.

Forecasting Plots

In the single window forecast plots, we observe the following:

- The BiLSTM and CNN + BiLSTM models show improvements over the simpler models, but they still smooth the fluctuations of the true growth, especially in the first 20-30 time steps. - XGBoost tracks the true growth more closely, with less divergence, particularly in capturing the overall trend of the time series.

In conclusion, XGBoost is the most accurate model for this dataset, while CNN + BiLSTM offers some improvement over BiLSTM, but both struggle with capturing all the fluctuations.

Anwer Question 2

Best Model for Temporal Patterns

The CNN + BiLSTM model captures temporal patterns the best, as it combines the feature extraction ability of CNN with the temporal learning of Bidirectional LSTM. The CNN layer extracts important features, while the BiLSTM layers capture both past and future dependencies, making it highly effective for time series forecasting.

Pros and Cons

XGBoost:

- **Pros:** Efficient, fast, handles missing values.
- **Cons:** Does not capture temporal dependencies effectively.

BiLSTM:

- **Pros:** Learns sequential patterns well, captures both past and future dependencies.
- **Cons:** Slower to train, does not extract features as well as CNN.

CNN + BiLSTM:

- **Pros:** Combines feature extraction and temporal learning, captures complex patterns.
- **Cons:** Requires more computational resources, prone to overfitting.

6 Encoder Decoder Models

LSTM Encoder + Attention Decoder

The **LSTM Encoder + Attention Decoder** model adopts a sequence-to-sequence framework, using an LSTM encoder to process the input sequences and an LSTM decoder for generating the output sequences. The attention mechanism enhances the model's ability to focus on relevant parts of the input sequence during prediction, effectively capturing long-term temporal dependencies.

Architecture:

- Encoder LSTM with 64 units, returning both sequences and states.
- Decoder LSTM with 64 units, initialized with encoder states.
- Attention layer to generate context vectors by selectively weighting encoder outputs.
- TimeDistributed Dense layer to produce final forecasts.

During training, both Mean Absolute Error (MAE) and Loss decrease steadily. Training and validation losses show close alignment, with minor fluctuations in validation performance, indicating effective generalization with potential for further refinement.

Transformer

The **Transformer** model leverages self-attention mechanisms to capture complex temporal patterns effectively without sequential constraints. Utilizing positional encoding, this architecture excels

at modeling long-range dependencies in time series data and can process sequences in parallel, improving computational efficiency.

Architecture:

- Positional encoding to incorporate sequence order information.
- Multi-head self-attention mechanism to focus simultaneously on multiple aspects of the sequence.
- Feed-forward network with two Dense layers (128 and 64 units) for complex pattern extraction.
- Layer normalization to stabilize and accelerate training.

Throughout training, the Transformer demonstrates consistent improvement in MAE and Loss. The architecture exhibits efficient convergence and effectively handles long-range dependencies, although it demands higher computational resources compared to recurrent-based models.

Training Metrics Observation

Both the Transformer and LSTM Encoder + Attention Decoder models show a steady decrease in Training Mean Absolute Error (MAE) and Loss, indicating effective learning throughout the training process. However, there are notable differences in how the models perform on the validation set.

The Transformer model demonstrates a consistent decrease in both training MAE and Loss. However, the validation metrics show some fluctuations, especially in the Validation MAE and Validation Loss, suggesting that the model is slightly overfitting at certain points. Despite this, the model achieves relatively stable performance, with the training and validation curves staying close after the initial few epochs.

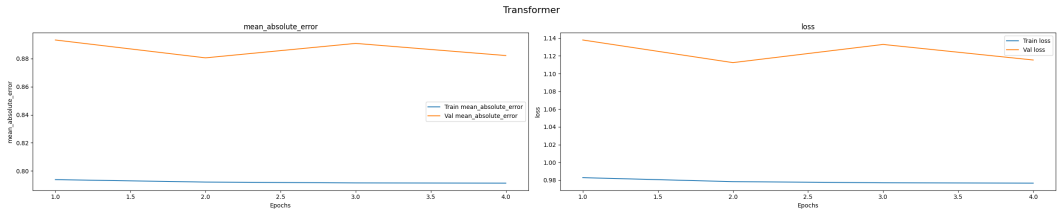

Figure 12: Transformer Model Training and Validation Metrics.

The LSTM Encoder + Attention Decoder model shows a similar trend, with both training MAE and Loss steadily decreasing. However, the validation MAE and Loss exhibit more noticeable fluctuations compared to the Transformer model, particularly after a few epochs. This suggests that the model may require further fine-tuning to better generalize on the validation set. The model does improve over time but shows some instability in the later epochs.

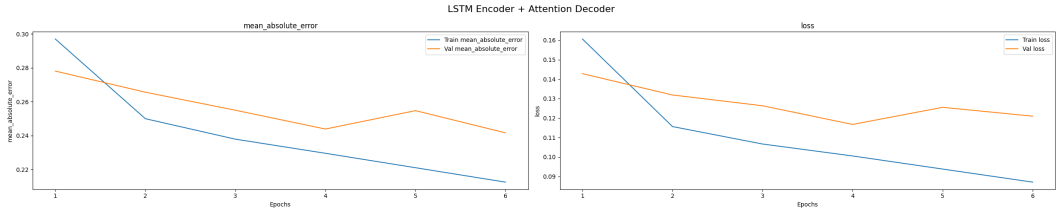

Figure 13: LSTM Encoder + Attention Decoder Training and Validation Metrics.

Evaluation

LSTM Encoder + Attention Decoder

Mean Absolute Error (MAE): 1.051
Root Mean Squared Error (RMSE): 1.318

The LSTM Encoder + Attention Decoder model captures the overall trend in the time series data, but struggles to track sudden fluctuations. This results in a higher MAE and RMSE compared to the Transformer model. The model does not predict abrupt changes well, and while the attention mechanism helps focus on important parts of the sequence, the performance could be further refined.

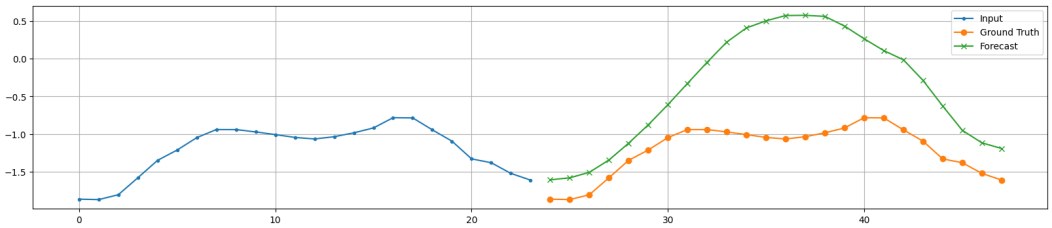

Figure 14: LSTM Encoder + Attention Decoder Forecasting Results.

Transformer

Mean Absolute Error (MAE): 0.854
Root Mean Squared Error (RMSE): 1.025

The Transformer model performs better than the LSTM Encoder + Attention Decoder, with a lower MAE and RMSE. The Transformer captures the overall trend more effectively and handles long-range dependencies better, especially in comparison to the LSTM model. Although the Transformer model demonstrates good performance, there is still some fluctuation in the validation metrics, indicating potential overfitting.

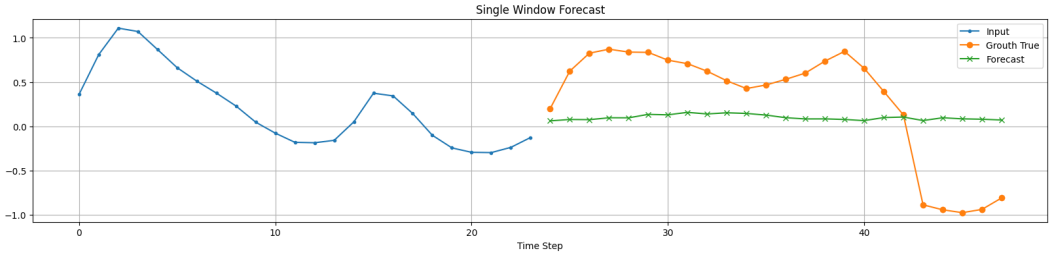

Figure 15: Transformer Forecasting Results.

Forecasting Plots

In the single window forecast plots, we observe the following: - The LSTM Encoder + Attention Decoder model shows steady learning, but the fluctuations in the forecast are still not captured as well as in the Transformer model. - The Transformer model follows the true growth values more closely, especially in capturing the overall trend and reducing the divergence in the forecast.

Comparison

The following table summarizes the performance of the models based on Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE):

Model	MAE	RMSE
BiLSTM	1.027	1.287
CNN + BiLSTM	0.901	1.125
LSTM Encoder + Attention	1.051	1.318
Transformer	0.854	1.025

Table 1: Comparison of Models based on MAE and RMSE.

Key Insights

- Transformer achieves the best performance with the lowest MAE and RMSE, capturing complex temporal patterns. - CNN + BiLSTM outperforms BiLSTM, with a significant reduction in both MAE and RMSE. - LSTM Encoder + Attention Decoder performs well but has the highest error values, indicating room for improvement. - BiLSTM has the highest error metrics, suggesting it struggles with capturing fluctuations and complex patterns.