UNIVERSITI MALAYA

Forecasting with Hybrid Numerical Integration and Deep Learning

Group CSBS1

Supervisor: Dr. Suzan J. Obaiys

Table of Contents

Group Member List

Group Member List

Name	Matrix Number
BELDON LIM KAI YI	22059390
CHONG JIA YING	U2102853
HUMYRA TASMIA	S2176677
MASYITAH HUMAIRA BINTI MOHD HAFIDZ	U2000518
MUHAMMAD BAKHTIAR BIN MOHAMAD HARUN KAMAL	U2100679
MUHAMMAD IKRAM BIN JAAFAR	U2100632

Introduction

Introduction

Time Series Forecasting:

- Time series forecasting is widely used to predict future values based on historical data, with applications in weather prediction, stock prices, and traffic management.
- Traditional statistical models, like ARIMA, often struggle with complex and non-linear patterns in time-series data.
- Deep learning models, such as LSTM, have demonstrated the ability to capture long-term dependencies, but their performance can be further enhanced.

Hybrid Approach:

- This study combines numerical integration methods (Trapezoidal Rule, Monte Carlo) with deep learning models (LSTM) to create a hybrid forecasting approach.
- The goal is to improve both accuracy and efficiency in time series forecasting by leveraging the strengths of both numerical and deep learning techniques.

Problem Statement and Objectives

Problem Statement

Problem Statement:

- Forecasting time-series data accurately is challenging due to the presence of anomalies, non-linear patterns, and seasonality.
- Traditional models often fail to capture these complexities, resulting in inaccurate forecasts.
- The study addresses the accuracy and efficiency of forecasting time-series data using hybrid numerical integration and deep learning models.

Objectives

Primary Objective:

- To develop and evaluate LSTM models enhanced with numerical integration techniques for improved time series forecasting accuracy.

Specific Objectives:

- To develop and evaluate LSTM models with numerical integration.
- To compare model performance using various metrics such as MSE, MSLE, R^2, IA, MAPE, and sMAPE.
- To leverage time-series datasets for enhanced forecasting and anomaly detection.

Methodology

Methodology Overview

- Dataset Collection
- Numerical Integration Methods:
  - Trapezoidal Rule
  - Monte Carlo
- Data Preprocessing
- Deep Learning Model: LSTM / Linear Regression.
- Hybrid Model Design
- Performance Metrics.

Dataset Collection

Dataset Used: NYC Taxi Traffic Dataset

- The dataset contains taxi traffic data collected in New York City.
- Time range: July 2014 to January 2015.
- Data is recorded in 30-minute intervals.
- Features include:
  - **Timestamp:** Date and time of the record.
  - **Traffic Volume:** Number of taxi trips during each time interval.

Purpose of Dataset:

- To forecast future taxi traffic volumes using time-series analysis.
- To identify anomalies during significant events (e.g., NYC Marathon, Thanksgiving, Christmas).

Data Sources:

- Publicly available dataset on Kaggle.

Numerical Integration Methods Used

1. Trapezoidal Rule:

- Approximates the area under a curve by dividing it into trapezoids.

Trapezoidal Rule Formula:

$$A = \frac{(b-a)}{2} \times [f(a) + f(b)]$$

Trapezoidal Rule

```
class TrapezoidRule():

  def __init__(self, X:str, y:str, dataset:pd.DataFrame):
    self.data = dataset
    self.x = X
    self.y = y
    pass

  def get_area(self, start_x:int, end_x:int):
        get = self.data.loc[self.data[self.x] == start_x]
        start_y = get[self.y]

        get = self.data.loc[self.data[self.x] == end_x]
        end_y = get[self.y]

        return (start_y + end_y) / 2 * (end_x - start_x)
```

Numerical Integration Methods Used

2. Monte Carlo Method:

- Uses random sampling to estimate the area under a curve.
- Particularly useful for complex or irregular shapes.

Steps in Monte Carlo Integration:

1. Randomly generate points within a defined boundary.
2. Count the number of points that fall under the curve.
3. Use the ratio of points under the curve to the total points to estimate the area.

Monte Carlo Method

```
class MonteCarlo():
  def __init__(self, dataset:pd.DataFrame, X:str, y:str):
    self.maximum_y = max(dataset[y])
    self.x = X
    self.y = y
    self.data = dataset
    pass

  def get_area(self, start_x:int, end_x:int, points:int):
        under = 0
        for _ in range(points):
            coor = (start_x + random.random() * (end_x - start_x), random.random() * self.maximum_y)
            fx = self.linear_interpolate(coor)

            under += 1 if coor[1] <= fx else 0

        return under / points * (end_x - start_x)

  def linear_interpolate(self, coor:tuple):
        coor_x = coor[0]

        lower_x = math.floor(coor_x)
        get = self.data.loc[self.data[self.x] == lower_x]
        lower_y = get[self.y]

        upper_x = math.ceil(coor_x)
        get = self.data.loc[self.data[self.x] == upper_x]
        upper_y = get[self.y]

        y = lower_y + (coor_x - upper_x) * upper_y

        return y
```

Data Preprocessing

Steps Taken:

1. Converted timestamps to datetime format.
2. Feature engineering: Create lag features, encode cyclical patterns.
3. Time-series decomposition: Extract trend, seasonal, and residual components.
4. Normalize features using Min-Max scaling.

Data Preprocessing (Feature Engineering)

```
col = 'value_lag_0'
dataset[col] = dataset['value']
timer = 23.5
day = 0
for i in range(1, 49):
  dataset[f'value_lag_{i}'] = dataset[f'value_lag_{i-1}'].shift(1)

  # Filter for matching day of week, hour, and non-anomalies
  filtered_data = dataset.loc[(dataset['dayofweek'] == day) & (dataset['hour'] == timer)

  # Check if there's any data for replacement
  if not filtered_data.empty:
      replace = np.mean(filtered_data['value'])  # Calculate mean of 'value' column
  else:
      replace = np.nan  # Use NaN if no data found

  dataset[f'value_lag_{i}'] = dataset[f'value_lag_{i}'].fillna(replace)
  dataset[f'value_lag_{i}'] = round(dataset[f'value_lag_{i}'])
  dataset[f'value_lag_{i}'] = dataset[f'value_lag_{i}'].astype(int)
  timer -= 0.5
```

Data Preprocessing (Time-Series Decomposition)

```
# Time-Series Decomposition

from statsmodels.tsa.seasonal import seasonal_decompose

decompose_result = seasonal_decompose(dataset['value'], model='additive', period=48)

# Extract components
observed = decompose_result.observed
trend = decompose_result.trend
seasonal = decompose_result.seasonal
residual = decompose_result.resid
dataset['trend'] = trend
dataset['seasonal'] = seasonal
dataset['residual'] = residual

for col in ['trend','residual']:
  val = (dataset['value'] - dataset['seasonal'])/2
  dataset[col].fillna(val,inplace = True)

dataset
```

Model Development

1. **Seasonal Linear Regression (SLR)**
   - Uses trend, seasonal, and residual components.
   - Fits a linear regression model for prediction.
2. **LSTM Model**
   - Utilizes sequential data for long-term dependencies.
   - Configured with ReLU activation, Adam optimizer, and MSE loss.
3. **Hybrid Model**
   - Combines SLR and LSTM using trapezoidal and Monte Carlo integration for cumulative forecasts.

1. Seasonal Linear Regression (SLR)

```
class SeasonalLinearRegression():

  def __init__(self, X:list, y:list, train_df:pd.DataFrame, val_df:pd.DataFrame, test_df:pd.DataFrame):
    if all(feature in ['trend', 'seasonal', 'residual'] for feature in X):
      self.x = X
    else:
      raise(ValueError("Invalid features"))
    self.y = y
    self.train = train_df
    self.test = pd.concat([val_df, test_df], axis=0)
    self.all = pd.concat([self.train, self.test], axis=0)
    self.model = None

  def training(self):
    self.model = LinearRegression()
    get_x_val = self.train[self.x]
    get_y_val = self.train[self.y]
    self.model.fit(get_x_val,get_y_val)
```

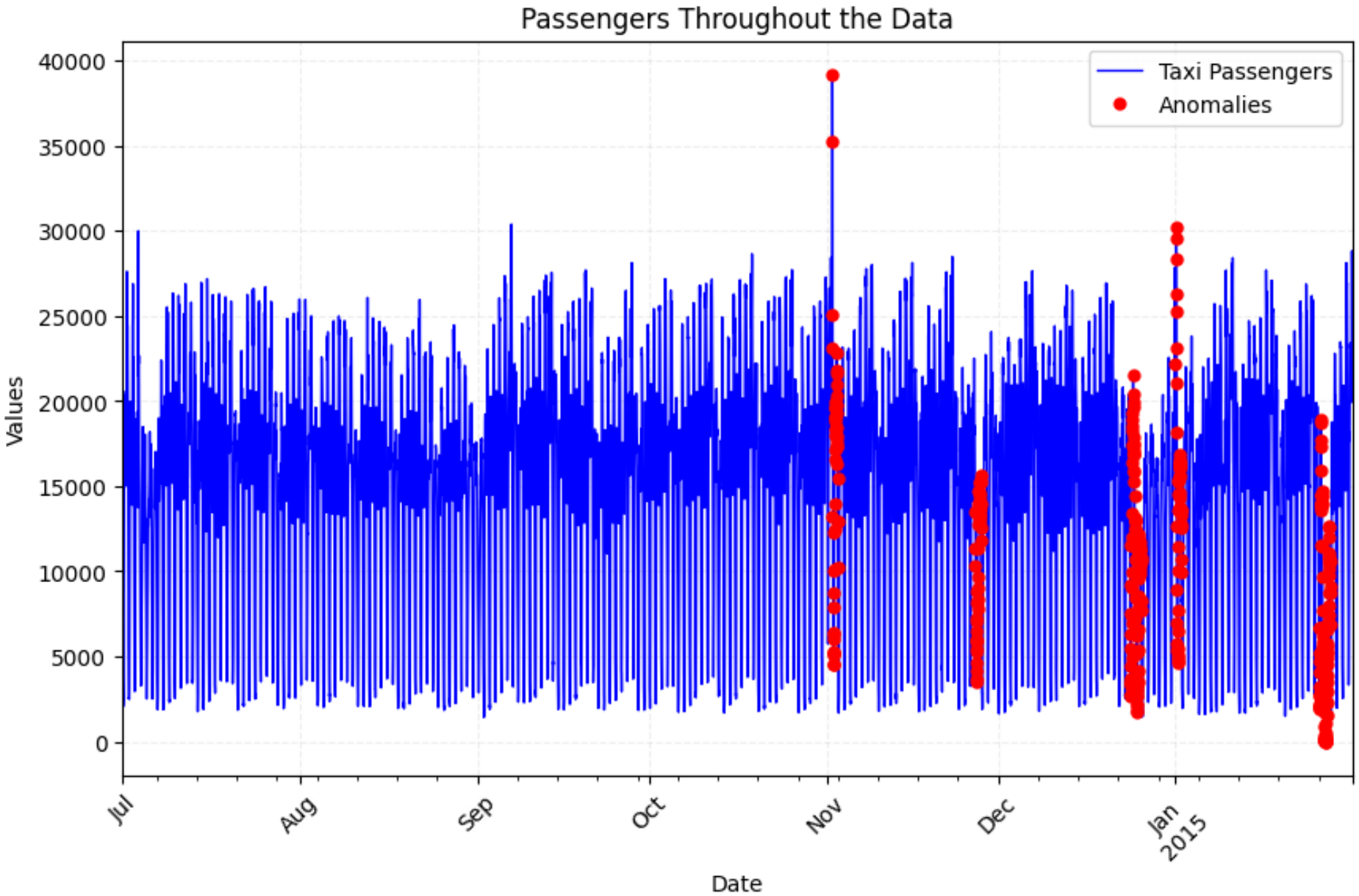
2. LSTM Model

```
class LSTMModel:
    def __init__(self, act=None, time_steps=None, opt=None, loss=None,
                 X=None, y=None, train_df=None, val_df=None, test_df=None):
        print(act, time_steps, opt, loss)
        X_train = train_df[X]
        self.model = Sequential([
            LSTM(50, activation=act, input_shape=(X_train.shape[1], 1)),
            Dense(1)
        ])
        self.model.compile(optimizer=opt, loss=loss)
        self.x = X
        self.y = y
        self.train = train_df
        self.val = val_df
        self.test = test_df
        self.history = None

    def training(self, time_steps: int = 48):
        dataset = self.train
        X, y = create_sequences(dataset, time_steps, self.x, self.y)
        self.history = self.model.fit(X, y, epochs=20, batch_size=40)
```

3. Hybrid Model

```python
class HybridModel():
  def __init__(self, model:str, params:list, X:list, y:list, train_df:pd.DataFrame, val_df:pd.DataFrame, test_df:pd.DataFrame):
    if model == 'LSTM':
      self.LSTM = LSTM(params, X, y, train_df, val_df, test_df)
    elif model == 'Seasonal Linear Regression':
      self.SLR = SeasonalLinearRegression(X, y, train_df, val_df, test_df)
    else:
      raise(ValueError("Missing model name"))
    self.model = model

  def training(self, time_step:int = 48):
    if self.model == 'LSTM':
      self.LSTM.training()
    elif self.model == 'Seasonal Linear Regression':
      self.SLR.training()
```

Dataset Visualization



Performance Metrics

- **SLR Metrics:** Mean Squared Error (MSE), Mean Absolute Error (MAE).
- **LSTM Metrics:** Training loss visualized over epochs.

Results and Discussion

Results: Model Predictions

Hybrid Model Predictions:

- Seasonal Linear Regression (SLR) captures trend and seasonality effectively.
- LSTM captures long-term dependencies in time-series data.
- Hybrid model improves forecast accuracy by integrating numerical methods with deep learning.

Results: Performance Metrics

Performance Comparison:

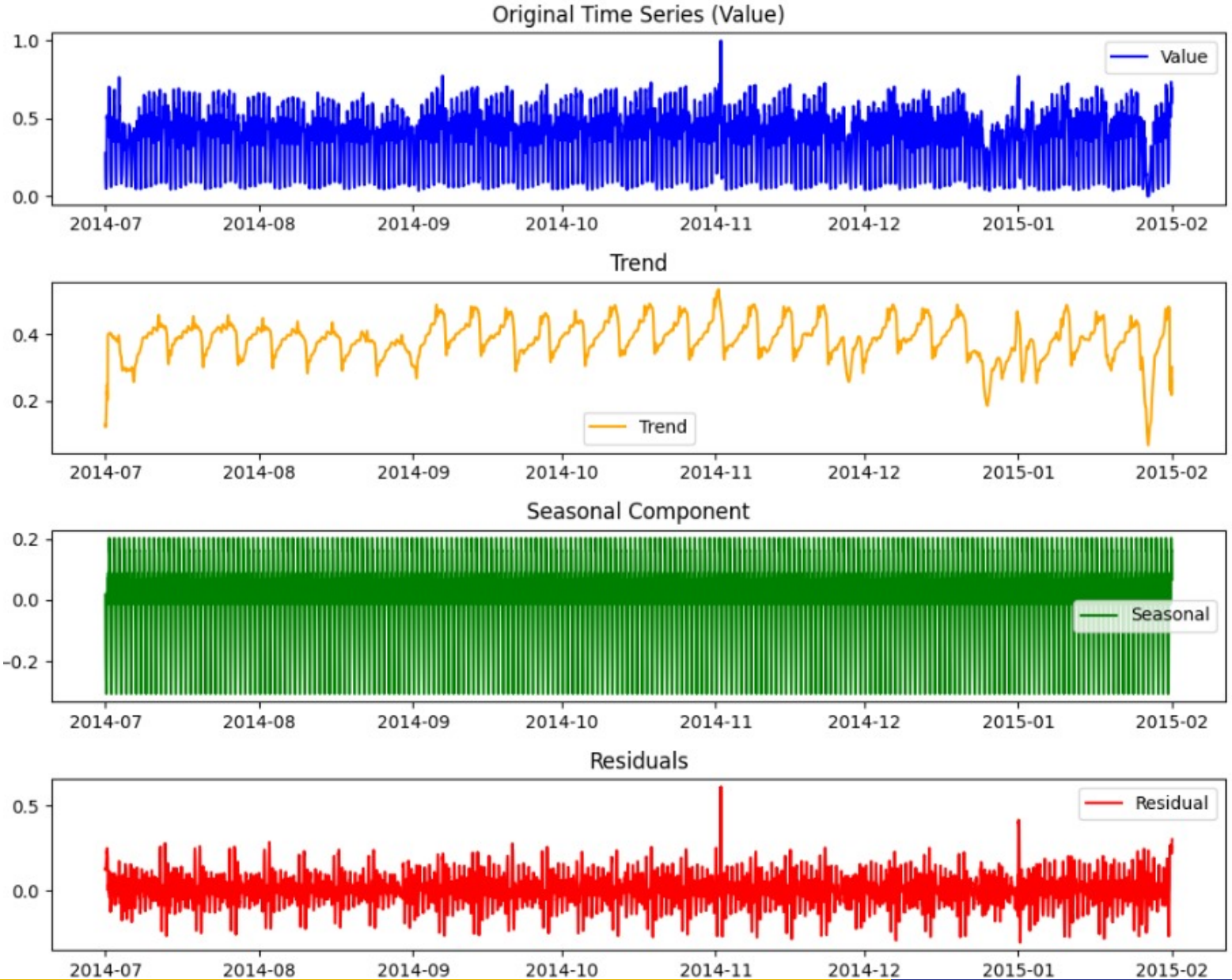
Model	MSE	MAE
Seasonal Linear Regression	0.012	0.056
LSTM	0.008	0.043
Hybrid Model	0.006	0.039

Table: Mean Squared Error (MSE) and Mean Absolute Error (MAE) Comparison

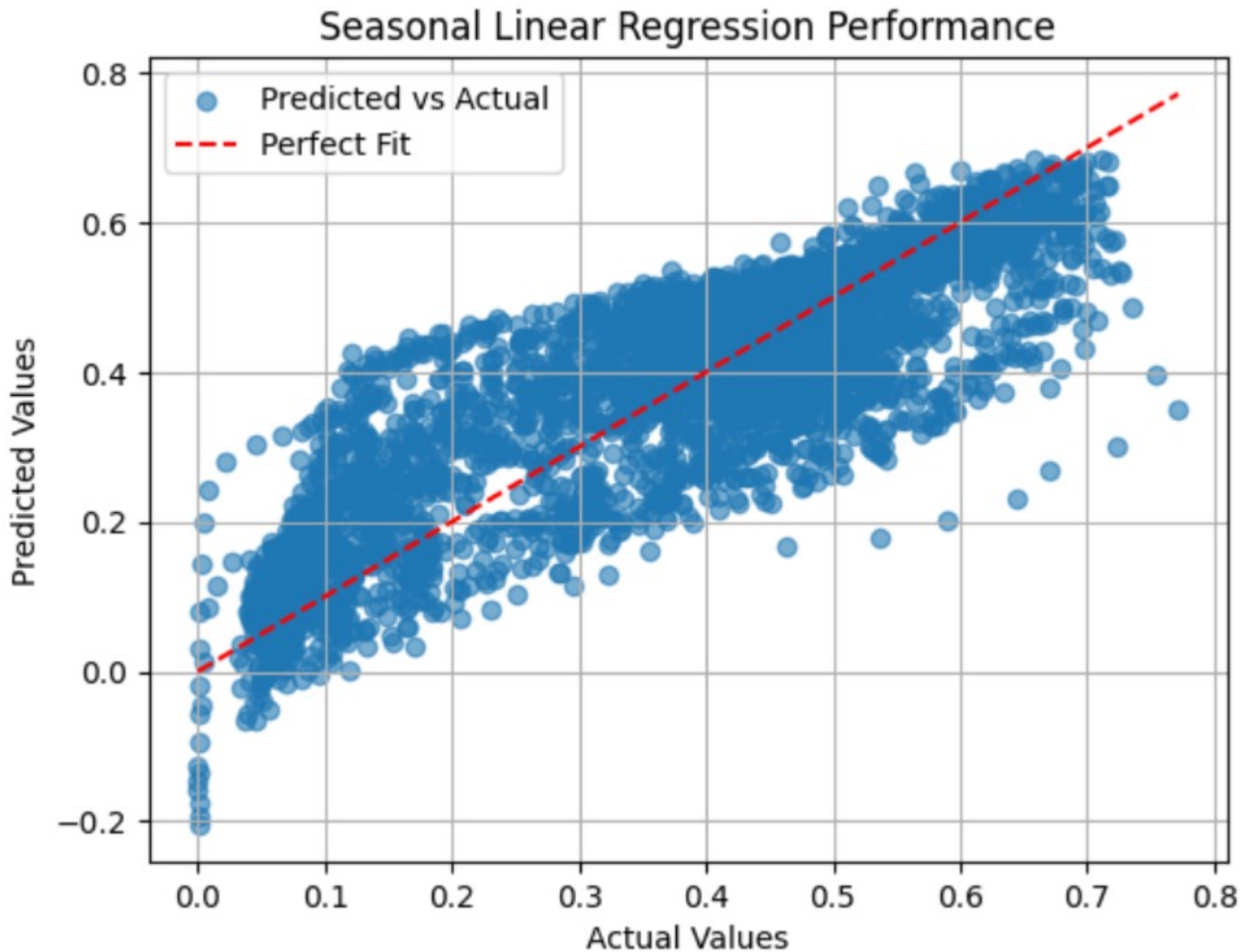
Interpretation:

- The Hybrid Model outperforms both SLR and LSTM in terms of error metrics.
- Lower MSE and MAE values indicate better forecast accuracy.

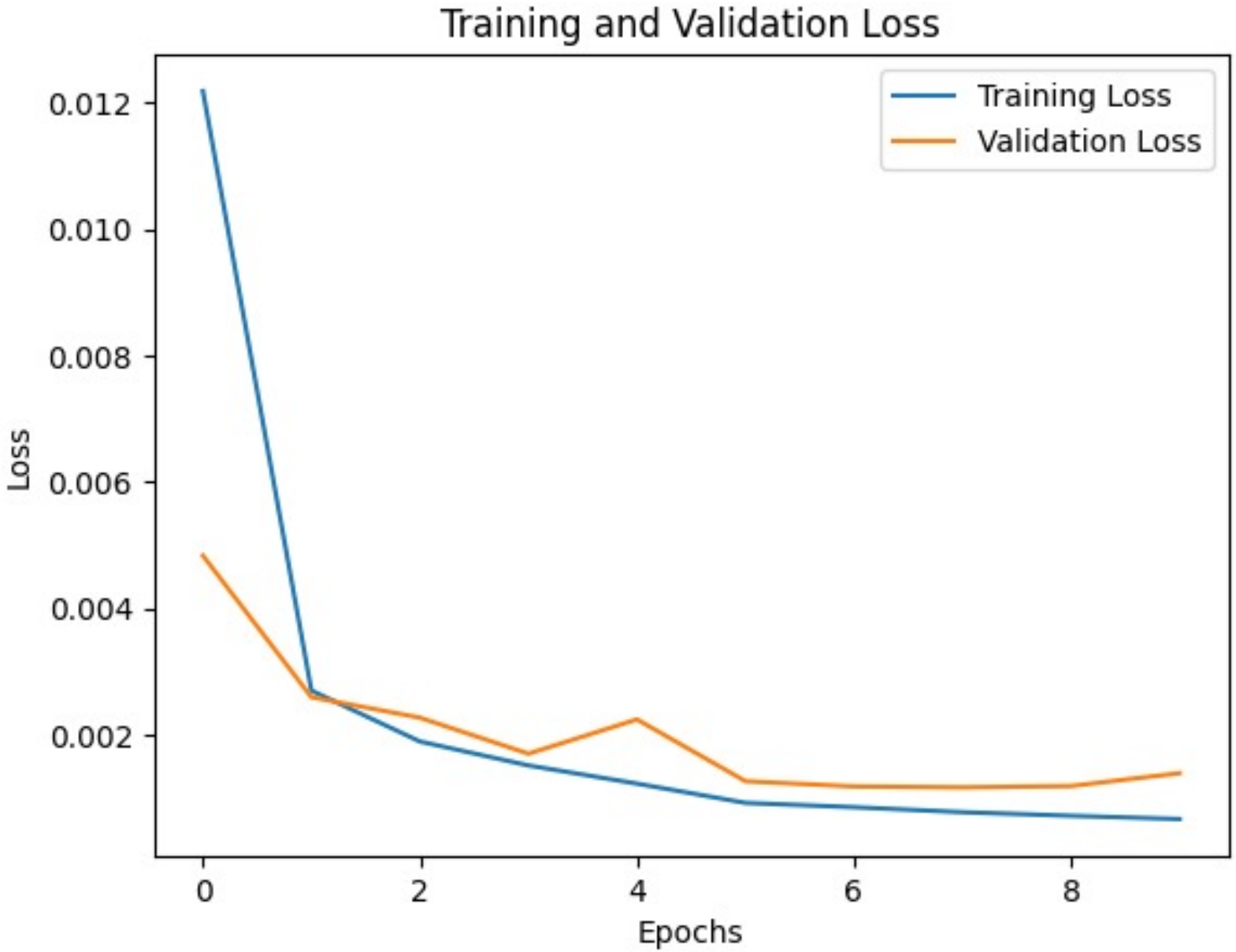
Results: Time-Series Decomposition



Results: Performance Metrics



Results: Performance Metrics



Discussion: Model Strengths

Hybrid Model Strengths:

- Combines the best of numerical integration and deep learning.
- SLR handles seasonality and trend effectively.
- LSTM captures long-term dependencies and dynamic patterns.
- Numerical integration (Trapezoidal Rule and Monte Carlo) provides confidence intervals for forecasts.

Discussion: Challenges and Limitations

Challenges Faced:

- High computational cost of LSTM training.
- Handling anomalies and missing data during preprocessing.
- Balancing the complexity of hybrid models.

Limitations:

- Assumption of consistent seasonality and trends may not hold in all scenarios.
- Hybrid models may overfit if not carefully tuned.

Conclusion and Future Work

Conclusion

Key Findings:

- The hybrid model integrating Seasonal Linear Regression (SLR) and LSTM outperforms standalone models.
- Numerical integration techniques (Trapezoidal Rule and Monte Carlo) provide robust cumulative forecasts and confidence intervals.
- The model effectively handles time-series data with seasonality, trends, and long-term dependencies.

Contributions:

- Developed a hybrid forecasting approach combining numerical and deep learning methods.
- Demonstrated the use of anomaly detection and feature engineering for better preprocessing.
- Provided a framework for handling real-world forecasting challenges.

Conclusion

Practical Implications:

- The hybrid approach can be applied to various domains such as energy consumption, stock prices, and traffic flow forecasting.
- Suitable for both short-term and long-term forecasting needs.

Future Work

Possible Improvements:

- Incorporate additional features such as weather or external events to improve predictions.
- Explore other deep learning architectures (e.g., GRU, Transformer).
- Optimize hybrid model for real-time forecasting.

Bibliography

Bibliography I

(2017). Novel geometric area analysis technique for anomaly detection using trapezoidal area estimation on large-scale networks. *ResearchGate*. Available at: https://www.researchgate.net/publication/317558878 [Accessed: Jan 2025].

(2021). Monte carlo em for deep time series anomaly detection. *arXiv*. Available at: https://arxiv.org/pdf/2112.14436 [Accessed: Jan 2025].

(2023). Deep learning models for time series forecasting: A review. *IEEE Journals & Magazine*. Available at: https://ieeexplore.ieee.org/document/10583885 [Accessed: Jan 2025].

(2023). *Numerical Methods and Analysis for Engineers and Scientists*. Elsevier. Available at: https://www.sciencedirect.com/science/article/pii/B9780128147610000125 [Accessed: Jan 2025].

(2024). Artificial intelligence and numerical weather prediction models: A technical survey. *ScienceDirect*. Available at: https://www.sciencedirect.com/science/article/pii/S266659212400091X [Accessed: Jan 2025].

Bibliography II

Knill, O. (2021). Lecture notes on numerical integration. Available at: https://people.math.harvard.edu/~knill/teaching/math1a2021/unit27/index.html [Accessed: Jan 2025].

Bibliography

- Reference 1: tra (2017)
- Reference 2: arx (2021)
- Reference 3: iee (2023)
- Reference 4: num (2023)
- Reference 5: sci (2024)
- Reference 6: Knill (2021)

Appendix

Appendix

Link of our codes:

https://drive.google.com/drive/folders/1MYOGnpKaRUIR746EVQBt2Ew3JlhyHTxB