Hybrid Neural Prophet Ensemble: Optimizing Time Series Forecasting with Comparative Analysis

Review-2

B. Tech
Computer Science and Engineering

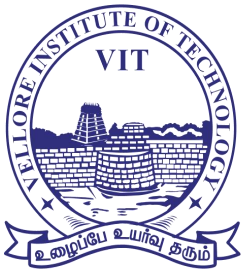
By

Dhwaj Jain (20BCI0302)

Ranuj Pradhan(20BCE2459)

Under the guidance of

Dr. MOHAN KUMAR P
Associate Professor Grade 2
SCOPE, VIT, Vellore



VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

VELLORE INSTITUTE OF TECHNOLOGY

VELLORE 632 014, TAMILNADU, INDIA

April 2024

TABLE OF CONTENTS

Abstract

This capstone project introduces an innovative approach to enhance time series forecasting accuracy by creating a hybrid ensemble model. Focused on univariate time series data, the project integrates the NeuralProphet and Recurrent Neural Network (RNN) architectures. The primary goal is to surpass the predictive capabilities of individual models and address the specific challenges inherent in univariate time series forecasting. The system architecture includes utilizing the Prophet model for initial time series fitting, calculating residuals, and integrating these with an LSTM model for improved forecasting performance. The refined forecasts are generated through a strategic combination of predictions from both models, creating a synergistic approach that maximizes their complementary strengths. The project concludes with a comprehensive evaluation, comparing the hybrid ensemble against NeuralProphet and LSTM using diverse forecasting performance metrics. The optimization phase encompasses fine-tuning hyperparameters and optimizing the NeuralProphet model for improved accuracy. Visualization techniques are employed to facilitate a comprehensive understanding of the forecasting framework's capabilities. This capstone project contributes to the advancement of time series forecasting methodologies, offering a practical and effective solution for accurate predictions in univariate time series scenarios.

1.Introduction

Time series forecasting plays a pivotal role in various domains, providing valuable insights for decision-making and planning. In the realm of univariate time series data, challenges arise due to its dynamic and often complex nature. This capstone project seeks to address these challenges by introducing an innovative forecasting framework that combines the strengths of the NeuralProphet and Recurrent Neural Network (RNN) architectures. The objective is to create a robust hybrid ensemble model that outperforms individual models and delivers more accurate predictions.

The significance of this project lies in its potential to advance the state-of-the-art in time series forecasting, particularly in scenarios where univariate data poses unique difficulties. By integrating the NeuralProphet model, known for its effectiveness in handling time series data, with the capabilities of an LSTM model, which excels in capturing sequential dependencies, we aim to leverage their complementary strengths.

The project unfolds through a systematic process: starting with the utilization of Prophet for initial time series fitting, followed by the calculation of residuals. The residuals serve as input for training an LSTM

model, enhancing the forecasting capabilities of the ensemble. The resulting refined forecasts are generated by strategically combining predictions from both models, creating a synergistic approach.

The subsequent comprehensive evaluation involves a thorough comparison of the hybrid ensemble against standalone NeuralProphet and LSTM models. Various forecasting performance metrics will be employed to assess and quantify the improvement achieved by the hybrid approach. Furthermore, the project includes optimization steps, encompassing hyperparameter tuning and model optimization, to refine the forecasting framework.

Visualization techniques will be applied to facilitate a comprehensive understanding of the model's predictions and residuals. This project aims not only to contribute to the academic discourse on time series forecasting methodologies but also to provide a practical and effective solution for accurate predictions in univariate time series scenarios.

1.1 AIM OF THE PROJECT

The aim of the project is to develop a hybrid ensemble model for time series forecasting by integrating the NeuralProphet and LSTM architectures, with the objective of surpassing the predictive capabilities of individual models and addressing the challenges inherent in univariate time series data. By strategically combining predictions from both models, the aim is to enhance forecasting accuracy, capturing both short-term dependencies and long-term trends. The project involves comprehensive evaluation and comparison against standalone NeuralProphet and LSTM models, utilizing diverse forecasting performance metrics to quantify improvements. Additionally, optimization steps will be undertaken to fine-tune hyperparameters and optimize the NeuralProphet model, further refining the forecasting framework. Visualization techniques will be employed to facilitate clear interpretation of predictions and residuals, contributing to a practical and effective solution for accurate predictions in dynamic univariate time series scenarios, thus advancing time series forecasting methodologies.

1.2 Objective

The primary objective of this capstone project is to develop a novel hybrid ensemble model by integrating the NeuralProphet and Recurrent Neural Network (RNN) architectures. This hybrid model aims to enhance the accuracy of time series forecasting, particularly in the context of univariate data with its inherent complexities. The project seeks to leverage the strengths of NeuralProphet, known for its effectiveness in time series prediction, and the sequential dependencies capturing capabilities of the

LSTM model. Through a systematic process, the project involves utilizing Prophet for initial time series fitting, calculating residuals, and integrating them into the training of an LSTM model. The resulting refined forecasts are generated by strategically combining predictions from both models, fostering a synergistic approach. A key aspect of the project is a comprehensive evaluation, comparing the performance of the hybrid ensemble against standalone NeuralProphet and LSTM models. Various forecasting performance metrics will be employed to quantify the improvements achieved by the hybrid approach. Additionally, the project includes optimization steps, encompassing hyperparameter tuning and fine-tuning of the NeuralProphet model, to further refine the forecasting framework. Visualization techniques will be applied to provide a clear and interpretable representation of the model's predictions and residuals, contributing to a comprehensive understanding of its capabilities. Overall, this project aims not only to advance time series forecasting methodologies but also to provide a practical and effective solution for accurate predictions in dynamic univariate time series scenarios.

1.3 SCOPE OF THE PROJECT

The scope of the project encompasses several key aspects:

1.Model Integration: The project involves integrating the NeuralProphet model, known for its effectiveness in time series prediction, with a Recurrent Neural Network (RNN) architecture, specifically LSTM. This integration aims to leverage the complementary strengths of both models to enhance forecasting accuracy.

2.Data Focus: The project is focused on univariate time series data, where the challenge lies in forecasting based on a single variable's historical values. By concentrating on this specific type of data, the project aims to address the unique difficulties associated with univariate time series forecasting.

3.System Architecture: The project outlines a systematic approach for creating the hybrid ensemble model. This includes utilizing the Prophet model for initial time series fitting, calculating residuals, and integrating them into the training of an LSTM model. The refined forecasts are then generated through a strategic combination of predictions from both models.

4. Evaluation and Comparison: A comprehensive evaluation is conducted to compare the performance of the hybrid ensemble model against standalone NeuralProphet and LSTM models. Various forecasting performance metrics are employed to quantify the improvements achieved by the hybrid approach.

1. 4 ORGANIZATION OF THESIS

2. Literature Review

2.1 SURVEY ON EXISTING SYSTEM

SL No.	Author and year	Technologies used	Limitations
1	.Sean J. Taylor and Benjamin Letham(2017)	Prophet utilizes an additive model that includes trend, seasonality, and holiday effects, along with an automatic change point detection algorithm	Limited Support for Complex Nonlinear Patterns Model Interpret-ability
2	S. Salim, S. S. Mamun, and M. A. H.Akhand (2018)	This review paper provides an overview of various time series forecasting methods, including statistical methods, machine learning approaches.	Depth of Analysis Subjectivity in Evaluation Limited Discussion on implementation Challenges
3	Christopher D. 2018	LSTM and BLSTM recurrent neural network	Bidirectional approach adds overhead to each Epoch. Increased processing time
4	Zhi-Hua Zhou (2012)	Ensemble learning approach a comprehensive overview of ensemble learning methods, including bagging, boosting, and stacking.	Bias towards Certain Techniques Limited Coverage of Implementation Considerations

			Temporal Relevance
5	Zen Qin, Wenxuan Bao, Year: 2020	covers a wide range of deep learning models, including recurrent neural networks (RNNs), convolution neural networks (CNNs), long short-term memory networks (LSTMs), gated recurrent units	Technical Complexity Evaluation Metrics Scope Limitation

2.2 GAPS IDENTIFIED

Here are some potential gaps that could be identified in the paper:

1. Integration of NeuralProphet and LSTM Models: While the paper proposes a novel hybrid ensemble model combining NeuralProphet and LSTM architectures, it may not thoroughly explore existing research on similar integrations or provide a comprehensive comparison with other ensemble techniques in the context of time series forecasting.

2. Evaluation Metrics and Benchmarking: The paper outlines the use of various forecasting performance metrics to evaluate the hybrid ensemble model. However, it may lack a detailed discussion on the choice of these metrics and how they capture different aspects of forecasting accuracy. Additionally, benchmarking against state-of-the-art methods or competing ensemble approaches could provide a clearer understanding of the hybrid model's effectiveness.

3. Data Preprocessing and Feature Engineering: While the methodology section briefly mentions data preprocessing steps, such as preparing the univariate time series data for modeling, it may not delve deeply into the specific preprocessing techniques used or potential challenges encountered. Furthermore, the paper could benefit from discussing any feature engineering methods employed to enhance model performance.

4 Optimization Strategies: Although the paper mentions hyperparameter tuning and model optimization as part of the methodology, it may lack a detailed discussion on the specific optimization strategies employed and their impact on the performance of the hybrid ensemble model. Providing insights into the optimization process could help readers understand how model parameters were fine-tuned to achieve optimal forecasting results.

5.Visualization Techniques and Interpretability: While the paper mentions the use of visualization techniques to facilitate understanding of the model's predictions and residuals, it may not provide sufficient detail on the types of visualizations used or how they contribute to the interpretability of

the results. Enhancing the discussion on visualization methods and their role in model interpretation could strengthen the paper.

2.3 Problem Statement

The project aims to develop a robust forecasting framework for univariate time series data by integrating the strengths of different models. Specifically, it focuses on combining the NeuralProphet and RNN (Recurrent Neural Network) architectures. The goal is to create an effective ensemble that outperforms standalone models and provides accurate predictions. In the comprehensive evaluation, the hybrid ensemble will be compared against NeuralProphet and LSTM, considering various performance metrics.

3. REQUIREMENT ANALYSIS

3.1.1 Functional requirements

The functional requirements of the system include:

1.Integration of NeuralProphet and LSTM architectures.

2.Data preprocessing for univariate time series data.

3.Calculation of residuals for model refinement.

4.Generation of forecasts through a strategic combination of predictions from both models.

5.Visualization of predictions and residuals for interpretability.

3.1.2 Non-Functional

The non-functional requirements encompass:

1.Performance: The system should provide accurate forecasts within acceptable time frames.

2.Usability: The interface should be user-friendly for both researchers and practitioners.

3.Reliability: The system should consistently deliver accurate forecasts under varying conditions.

4.Security: Data privacy and integrity should be maintained throughout the forecasting process.

5.Scalability: The system should be scalable to handle large volumes of time series data efficiently.

3.2 Feasibility Study

3.2.1 Technical Feasibility

The technical feasibility of the project is assessed based on:

1.Availability of required technology such as NeuralProphet and LSTM architectures.

2.Compatibility with existing data preprocessing and visualization tools.
3.Expertise in implementing and integrating machine learning models.

3.2.2 Economic Feasibility

The economic feasibility is determined by:
1.Initial investment in acquiring necessary hardware and software resources.
2.Operating costs for system maintenance and support.
3.Potential benefits, such as improved forecasting accuracy and decision-making.

3.2.3 Social Feasibility

The social feasibility considers:
1.Stakeholder acceptance and support for the proposed forecasting framework.
2.Ethical implications related to data usage and privacy.
3.Social impact, such as the potential benefits of accurate forecasts in various domains.

3.3 System Specification

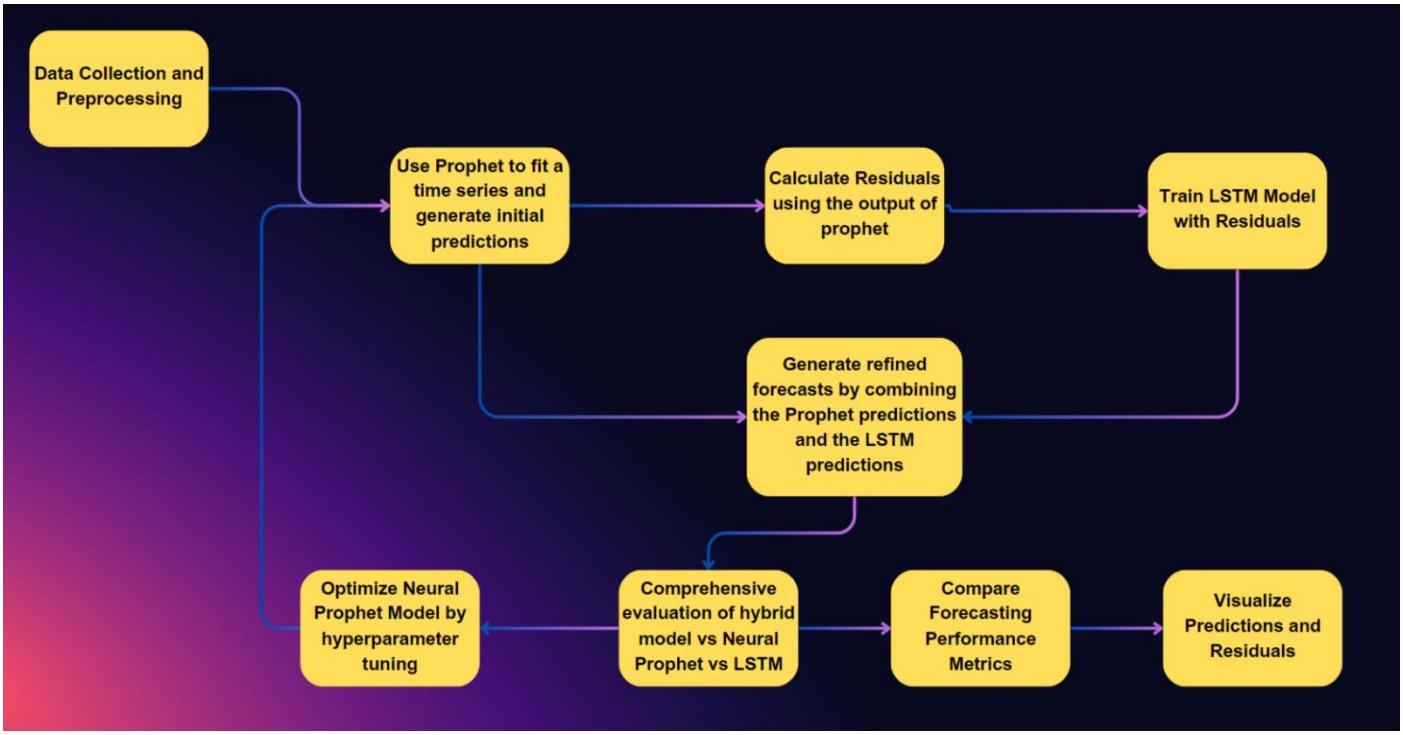
3.3.1 Hardware Specification

The hardware requirements include:
1. CPU: Intel Core i7 or AMD Ryzen 7 processor
2. GPU: NVIDIA GeForce RTX 30 series or AMD Radeon RX 6000 series
3. RAM: 16GB DDR4 or higher

3.3.2 Software Specification

1. Python
2. Jupyter Notebooks
3. TensorFlow/Keras
4. PyTorch (optional)

4.PROJECT DESCRIPTIONS



4.1 System architecture

1.Hybrid Model Creation:

- Utilizing the Prophet model for initial time series fitting and prediction generation establishes the foundation of our hybrid approach. This phase also involves the crucial step of calculating residuals by subtracting Prophet predictions from the actual values, facilitating a comprehensive understanding of prediction errors.

2. Neural Network Integration:

- The integration of a Long Short-Term Memory (LSTM) model represents a significant advancement in our system architecture. This phase focuses on training the LSTM model using residuals as input, harnessing the power of recurrent neural networks for enhanced forecasting capabilities.

3. Forecast Refinement:

- Generating refined forecasts is a pivotal stage where predictions from the Prophet and LSTM models are strategically combined. This fusion leverages the unique strengths of each model,

contributing to an ensemble approach that aims to outperform individual models.

4. **Comprehensive Evaluation:**

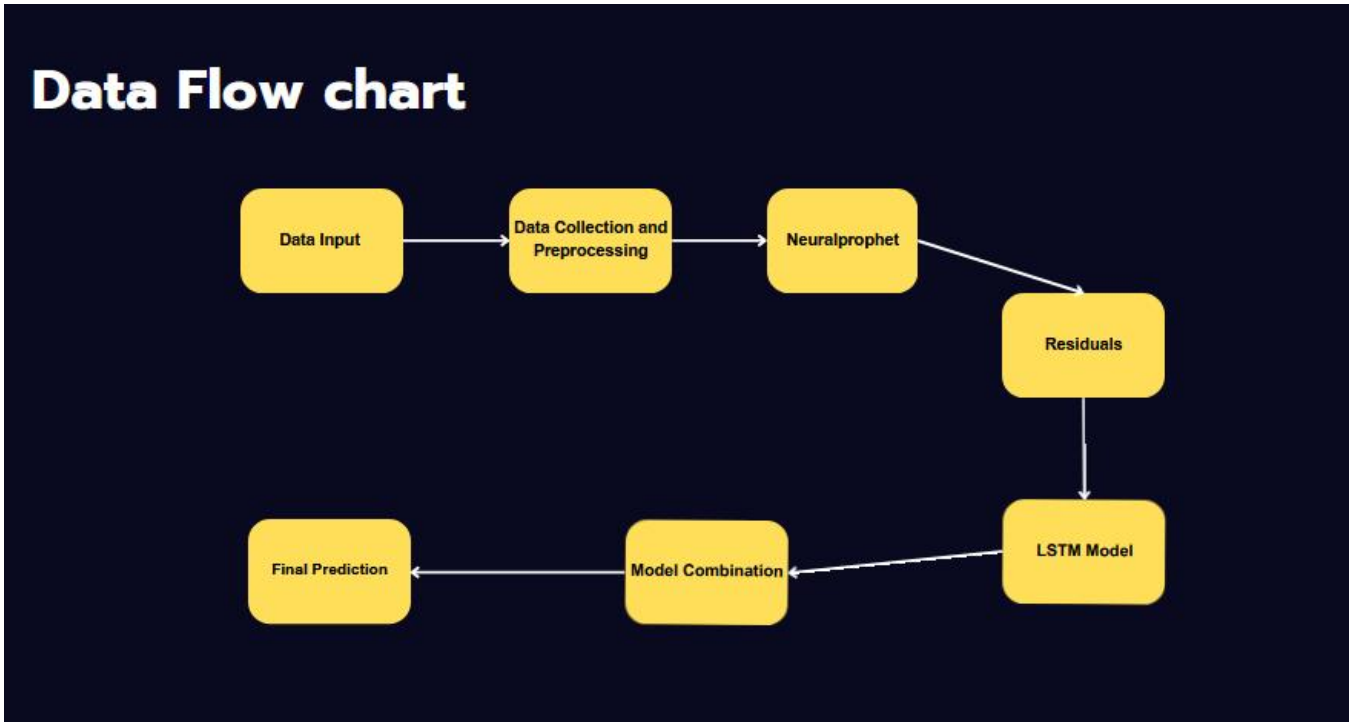
   - A thorough evaluation is conducted to assess the performance of our hybrid model. Comparisons are made against both the standalone Neural Prophet and LSTM models using a variety of forecasting performance metrics. This phase aims to quantify the effectiveness and advantages of our integrated approach.

5. **Optimization and Visualization:**

   - The optimization phase involves fine-tuning hyperparameters and optimizing the Neural Prophet model. Concurrently, results are meticulously analyzed. Visualization techniques are employed to offer a comprehensive view of predictions and residuals, aiding in the interpretation of the model's forecasting capabilities.

4.2 Design

4.2.1 DATA FLOW DIAGRAM



Description:

1.Input Data: Univariate time series data is provided as input to the system. This could include historical observations of a single variable over time.

2.Data Preprocessing: The input data undergoes preprocessing steps such as cleaning, normalization, and feature extraction to prepare it for modeling.

3.NeuralProphet: The preprocessed data is fed into the NeuralProphet model, which fits the initial time series and generates predictions.

4.LSTM Model: The residuals (i.e., the differences between actual and predicted values) from the NeuralProphet model are calculated and used as input to train the LSTM model.

5.Model Fusion: The predictions from both the NeuralProphet and LSTM models are combined strategically, leveraging their complementary strengths to generate refined forecasts.

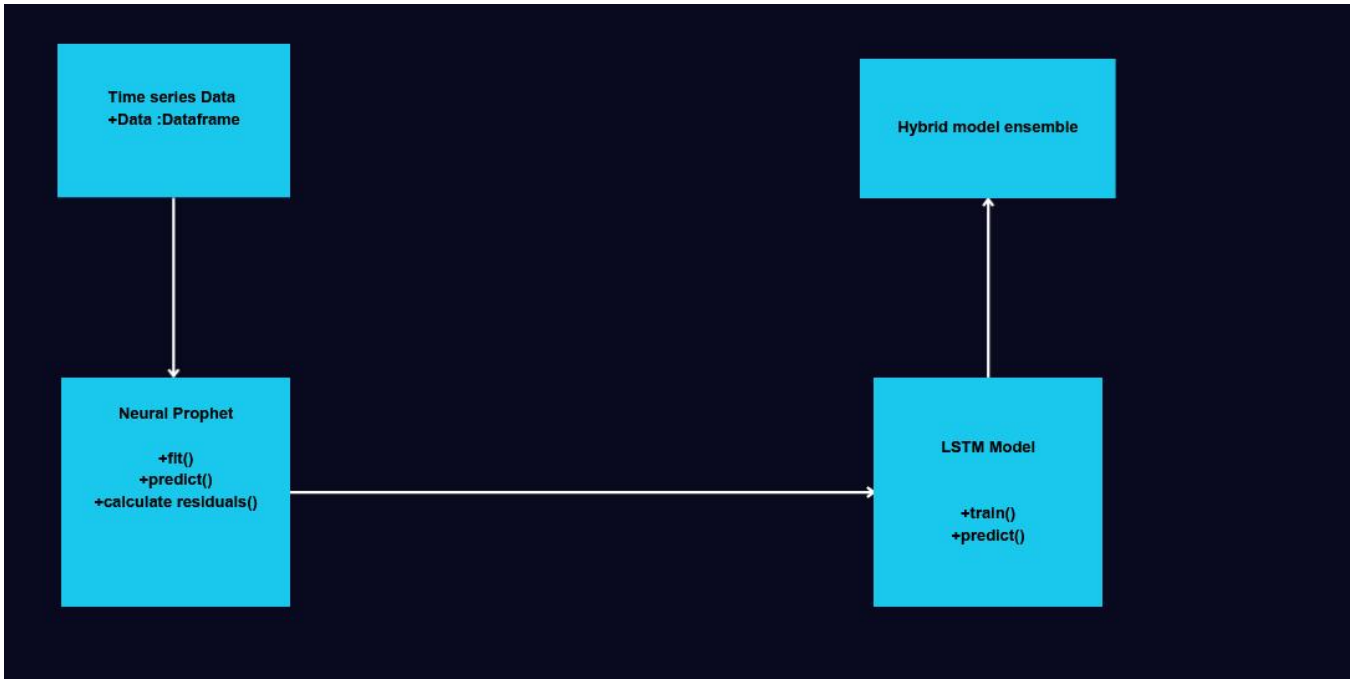
6.Final Forecasts: The final forecasts, generated by the hybrid ensemble model, are outputted for further analysis and decision-making.

This DFD provides a high-level overview of the data flow and processing steps involved in the hybrid ensemble model for time series forecasting. It highlights the sequential flow of data through various stages of preprocessing, modeling, and fusion to generate accurate forecasts.

4.2.2 USE CASE DIAGRAM

TBD

4.2.3 CLASS DIAGRAM



Description:

1.TimeSeriesData:

This class represents the time series data used in the forecasting process.
Attributes:
data: DataFrame - The data frame containing the time series data.
HybridEnsembleModel:

This class represents the hybrid ensemble model for time series forecasting.
Methods:

2.NeuralProphet:

This class represents the NeuralProphet model.
Methods:
fit(): Fits the NeuralProphet model to the input data.

predict(): Generates forecasts using the fitted model.
calculate_residuals(): Calculates residuals for model refinement.

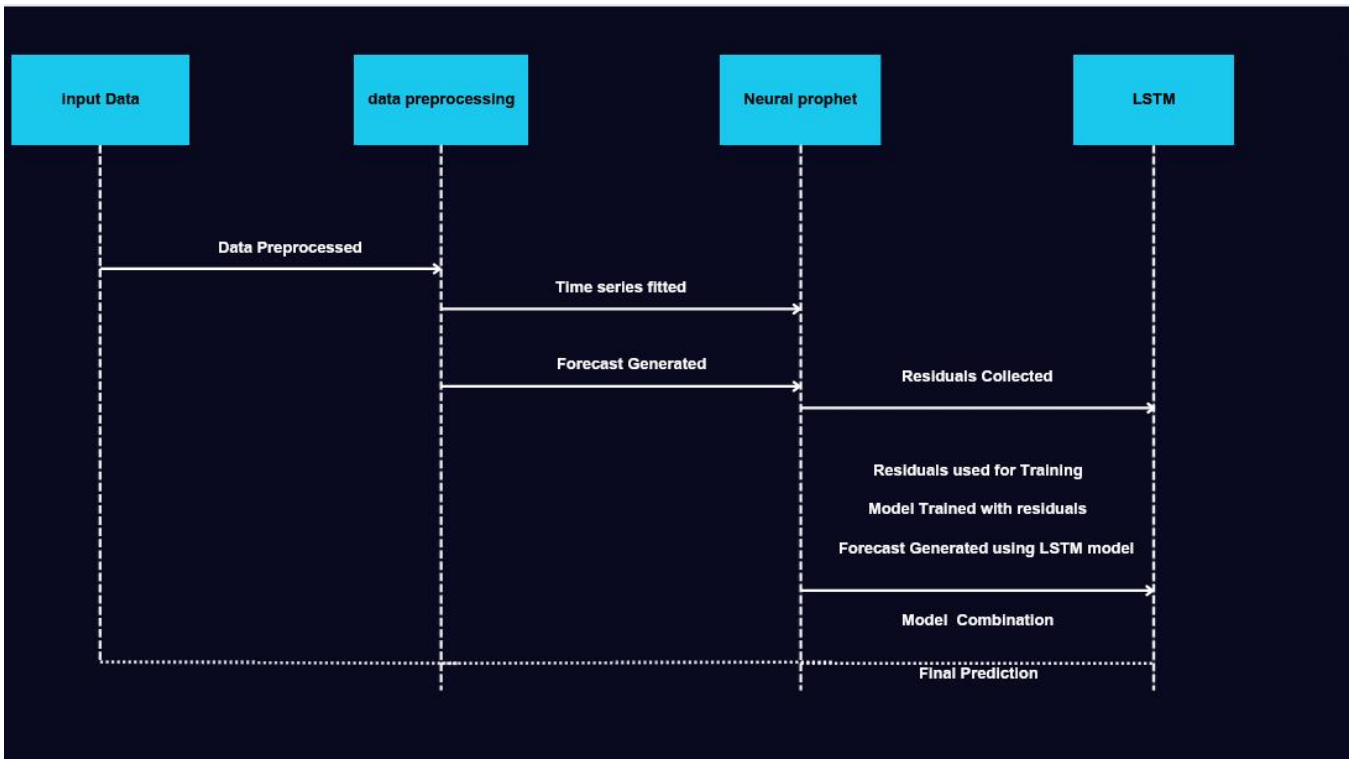
3.LSTM MODEL:

This class represents the LSTM model.
Methods:
train(): Trains the LSTM model with the calculated residuals.
forecast(): Generates forecasts using the trained model.
In this class diagram:

4.1 Classes are represented within rectangles.
4.2 Attributes are listed within the class with their respective data types.
4.3. Methods are listed within the class with their respective signatures.
Relationships between classes are depicted using lines. In this case, the HybridEnsembleModel class interacts with the NeuralProphet and LSTMModel classes to utilize their methods for fitting, forecasting, and refining the forecasts.

This class diagram provides a static view of the classes involved in the hybrid ensemble model for time series forecasting, illustrating their attributes, methods, and relationships.

4.2.4 SEQUENCE DIAGRAM



Description:

Input Data: The input univariate time series data is preprocessed to prepare it for modeling.

Data Preprocessing: The preprocessed data is then fed into the NeuralProphet model, where the initial time series fitting takes place, and forecasts are generated.

NeuralProphet to LSTM: The residuals calculated from the NeuralProphet model are used as input for training the LSTM model.

LSTM Model: The LSTM model is trained with the residuals from the NeuralProphet model, and forecasts are generated using the LSTM model.

Model Fusion: The forecasts generated from both the NeuralProphet and LSTM models are combined to produce refined forecasts.

Final Forecasts: The final refined forecasts, generated by the hybrid ensemble model, are outputted for further analysis and decision-making.

This sequence diagram illustrates the sequential flow of interactions between the components involved in the hybrid ensemble model for time series forecasting, from data preprocessing to the generation of refined forecasts.

4.3 Module Description

4.3.1 Module - 1

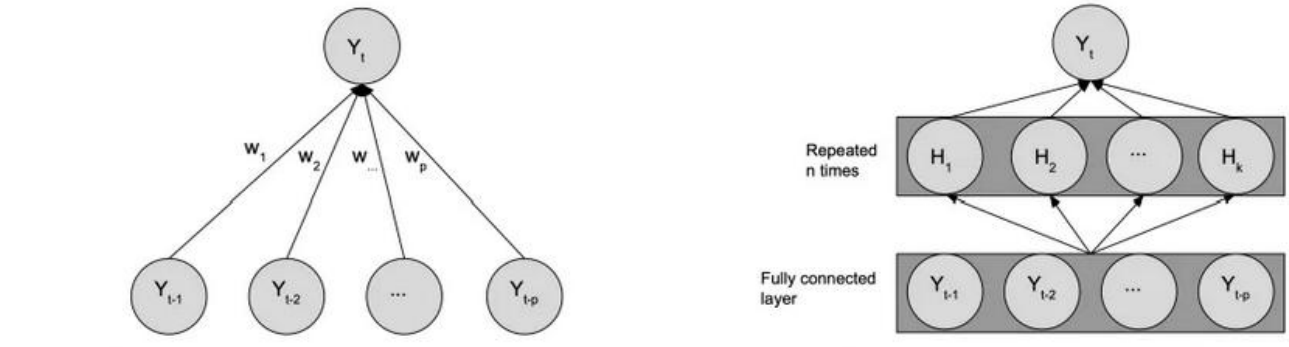
Module - 1 focuses on data preprocessing and initial time series fitting using the NeuralProphet model. This module comprises several key components:

1.Data Preprocessing: This component handles the preparation and preprocessing of univariate time series data, including cleaning, normalization, and feature extraction.

2.NeuralProphet Integration: Here, the preprocessed data is fed into the NeuralProphet model for initial time series fitting. The model learns patterns and trends from the data to generate initial forecasts.

3.Evaluation: This component evaluates the performance of the NeuralProphet model using appropriate metrics, providing insights into its accuracy and effectiveness in capturing temporal patterns.

UNDERSTANDING NEURAL PROPHET



(a) An AR equivalent neural network architecture. The node weights $w_1, w_2, ...w_p$ correspond to the AR-coefficients.

(b) An AR inspired neural network with n hidden layers of size k

Schematic of AR-Net architectures with the lags yt−1, ..., yt−p as inputs and yt as target.

Architecture of Neural prophet

4.3.2 Module - 2

Module - 2 focuses on refining forecasts and integrating the LSTM model for enhanced prediction accuracy. This module consists of the following components:

1.Residual Calculation: After generating initial forecasts with NeuralProphet, this component calculates the residuals, which represent the differences between actual and predicted values.

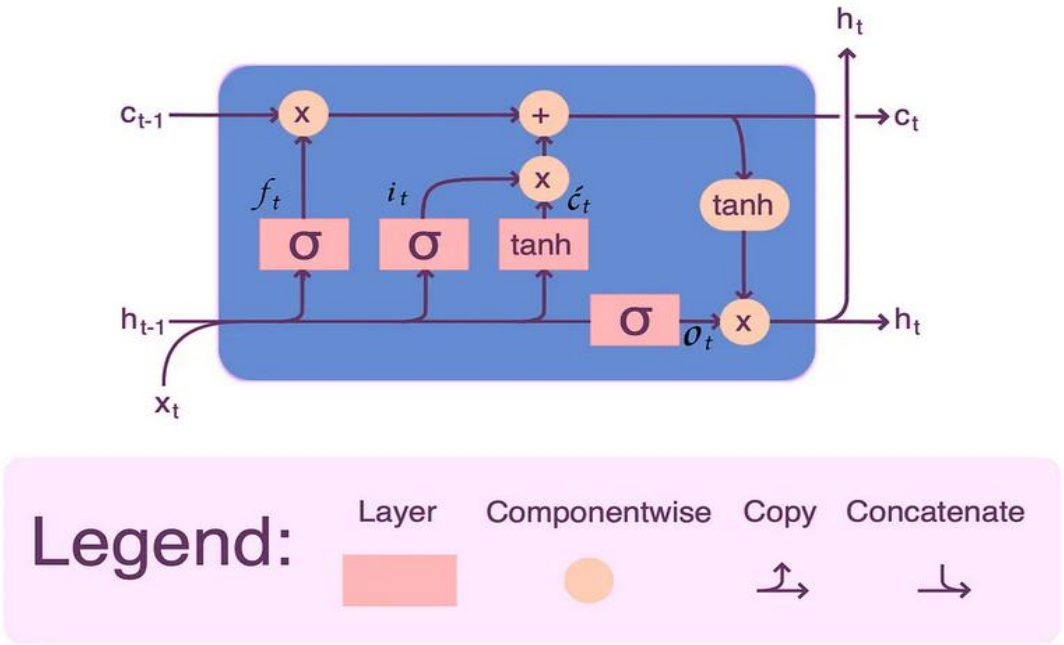
2.LSTM Integration: The calculated residuals are used as input to train an LSTM model, leveraging its ability to capture sequential dependencies in the time series data.

3.Model Fusion: Here, the forecasts from both NeuralProphet and LSTM models are strategically combined to generate refined forecasts. This fusion process maximizes the strengths of each model, resulting in improved accuracy and robustness.

4.Evaluation and Comparison: This component evaluates the performance of the hybrid ensemble model against standalone NeuralProphet and LSTM models, employing various forecasting performance metrics to assess improvements and provide insights for further refinement.

These modules work collaboratively to develop and evaluate the hybrid ensemble model, contributing to the advancement of time series forecasting methodologies and offering a practical solution for accurate predictions in univariate time series scenarios.

UNDERSTANDING LSTM MODEL



The LSTM consists of many memory blocks, as shown in the image is one whole block. Two states are carried over to the next block; cell state (stores and loads information) and hidden state (carries information from immediately previous events and overwrites). LSTMs learn using a process known as gates. These gates can learn which information in the sequence should be retained or dismissed. As a result, the LSTM contains three gates: input, forget, and output.

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f c_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i c_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o c_{t-1} + b_o) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \sigma_c(W_c x_t + b_c) \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

f*t*= Forget gate

i*t*= Input gate

o*t*= Output gate

C*t*= Cell state

h*t*= Hidden state

4.3.3 Module - 3: Optimization and Evaluation

Module - 3 focuses on optimizing the hybrid ensemble model and conducting a comprehensive evaluation of its performance. This module includes the following components:

1.Model Optimization: Here, the NeuralProphet model is further optimized for enhanced accuracy. This may involve refining the model architecture, adjusting training parameters, or incorporating additional features to improve forecasting performance.

2.Comprehensive Evaluation: This component evaluates the performance of the hybrid ensemble model against standalone NeuralProphet and LSTM models. Various forecasting performance metrics are used to assess improvements in accuracy and provide insights into the effectiveness of

the hybrid approach.

5. SAMPLE CODE

Importing Libraries

```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import plotly.offline as py
from matplotlib.pylab import rcParams
from datetime import datetime
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Data pre processing

```
data = pd.read_csv("RELIANCE.NS.csv")
 data.head(10)
data.info()
df1 = data.dropna()
df1['Date'] = pd.to_datetime(df1['Date'], format='%d-%m-%Y')
# df1['Date'] = pd.to_datetime(df1['Date'], format='%Y-%m-%d')
df1.head()
df1.info()
```

Model Implementation-Neural Prophet

```
# install neuralprophet
!pip install neuralprophet

#  import neuralprophet
from neuralprophet import NeuralProphet
```

```
 df2 = df1[['Date','Adj Close']]

# change names to fit NeuralProphet req
reliance_features = df2.rename(columns = {"Date":"ds","Adj Close":"y"})
reliance_features.head(10)
# Calculate the index for splitting the data (80-20)
split_index = int(len(reliance_features) * 0.8)

# split_index = int(len(reliance_features) * 0.7)
# Split data into training (80%) and testing (20%) sets
train_data = reliance_features.iloc[:split_index]
test_data = reliance_features.iloc[split_index:]
# Separate features and target variable for training and testing sets
X_train = train_data[['ds']]
y_train = train_data['y']
X_test = test_data[['ds']]
y_test = test_data['y']
# Train the NeuralProphet model on the training data
```

```
model1 = NeuralProphet()
# Add custom seasonality component
model1.add_country_holidays(country_name='IN')
metrics = model1.fit(train_data, freq='D', epochs=500, learning_rate=0.01)
# Make predictions for the testing data
forecast_test = model1.predict(test_data)
fig_forecast = model1.plot(forecast_test)
fig_forecast
```

Collecting Residuals

```
# Obtain actual values from the training data
actual_train = train_data['y']

# Make predictions for the training data
forecast_train = model1.predict(train_data)
# Extract predicted values from the forecast
predicted_train = forecast_train['yhat1']
# Calculate residuals
residuals_train = actual_train - predicted_train
```

Model Implementation – LSTM(Long Short term Memory)

```
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.callbacks import ReduceLROnPlateau

# Handle NaN values using forward fill
residuals_array = residuals_train.fillna(method='ffill').values
# Reshape the array to have a single feature
residuals_array = residuals_array.reshape(-1, 1)
# Scale the data using MinMaxScaler
scaler = MinMaxScaler()
residuals_scaled = scaler.fit_transform(residuals_array)
# Define a function to create input-output pairs for LSTM
def create_dataset(data, time_steps):
    X, y = [], []
    for i in range(len(data) - time_steps):
        X.append(data[i:(i + time_steps)])
        y.append(data[i + time_steps])
    return np.array(X), np.array(y)
time_steps = 10  # Define the number of time steps
X_lstm_train, y_lstm_train = create_dataset(residuals_scaled, time_steps)
# Define the LSTM model architecture
model_lstm = Sequential()
model_lstm.add(LSTM(units=100, return_sequences=True, input_shape=(X_lstm_train.shape[1], 1)))
model_lstm.add(Dropout(0.2))
model_lstm.add(LSTM(units=100, return_sequences=True))
model_lstm.add(Dropout(0.2))
model_lstm.add(LSTM(units=100))
model_lstm.add(Dropout(0.2))
model_lstm.add(Dense(units=1))
# Compile the model
optimizer = Adam(lr=0.001)
model_lstm.compile(optimizer=optimizer, loss='mean_squared_error')
# Learning rate reduction
reduce_lr = ReduceLROnPlateau(monitor='loss', factor=0.2, patience=5, min_lr=0.0001)
# Train the model
```

```
model_lstm.fit(X_lstm_train, y_lstm_train, epochs=200, batch_size=32, callbacks=[reduce_lr])
model_lstm.summary()
```

```
# Initialize an empty list to store predictions
predictions_lstm = []

# Reshape the residuals for prediction
X_lstm_test = np.reshape(residuals_scaled[-time_steps:], (1, time_steps, 1))
# Calculate the index where the 20% split occurs
# split_index = int(len(residuals_scaled) * 0.8)
# Iterate over the remaining 20% of the dataset
for i in range(len(test_data)):
    # Make predictions for the current input
    prediction = model_lstm.predict(X_lstm_test)
    # Store the prediction
    predictions_lstm.append(prediction[0, 0])
    # Append the new prediction to the input sequence
    X_lstm_test = np.concatenate((X_lstm_test[:, 1:, :], prediction.reshape(1, 1, 1)), axis=1)
# Convert the predictions list to a NumPy array
predictions_lstm = np.array(predictions_lstm)
```

Hybrid Model Implementation

```
# Convert predictions_lstm array to a DataFrame with the same index as forecast_test
predictions_lstm_df = pd.DataFrame(predictions_lstm, index=forecast_test.index, columns=['predicted_residuals'])

# Combine the predictions_lstm with the forecast_test from NeuralProphet
combined_forecast = forecast_test.copy()  # Make a copy of forecast_test
combined_forecast['yhat_combined'] = forecast_test['yhat1'] + predictions_lstm_df['predicted_residuals']
predictions_lstm_df.info()
```

EVALUATION

Combined Model

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_absolute_percentage_error

# Calculate MAE
mae = mean_absolute_error(combined_forecast['y'], combined_forecast['yhat_combined'])
# Calculate MSE
mse = mean_squared_error(combined_forecast['y'], combined_forecast['yhat_combined'])
# Calculate RMSE
rmse = np.sqrt(mse)
# Calculate MAPE
mape = mean_absolute_percentage_error(combined_forecast['y'], combined_forecast['yhat_combined'])
print("Mean Absolute Error (MAE):", mae)
print("Mean Squared Error (MSE):", mse)
print("Root Mean Squared Error (RMSE):", rmse)
print("Mean Absolute Percentage Error (MAPE):", mape)
```

Evaluating Neural Prophet vs Hybrid Model

```
from sklearn.metrics import mean_absolute_error, mean_squared_error

# Extract actual, NeuralProphet, and hybrid combined predictions
actual_values = combined_forecast['y']
neural_prophet_predictions = forecast_test['yhat1']
hybrid_combined_predictions = combined_forecast['yhat_combined']
# Calculate evaluation metrics for NeuralProphet predictions
mae_np = mean_absolute_error(actual_values, neural_prophet_predictions)
mse_np = mean_squared_error(actual_values, neural_prophet_predictions)
rmse_np = np.sqrt(mse_np)
mape_np = np.mean(np.abs((actual_values - neural_prophet_predictions) / actual_values)) * 100
# Calculate evaluation metrics for hybrid combined predictions
mae_hybrid = mean_absolute_error(actual_values, hybrid_combined_predictions)
mse_hybrid = mean_squared_error(actual_values, hybrid_combined_predictions)
rmse_hybrid = np.sqrt(mse_hybrid)
mape_hybrid = np.mean(np.abs((actual_values - hybrid_combined_predictions) / actual_values)) * 100
# Print the evaluation metrics
print("NeuralProphet Results:")
print(f"Mean Absolute Error (MAE): {mae_np}")
print(f"Mean Squared Error (MSE): {mse_np}")
print(f"Root Mean Squared Error (RMSE): {rmse_np}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_np}%\n")
print("Hybrid Combined Results:")
print(f"Mean Absolute Error (MAE): {mae_hybrid}")
print(f"Mean Squared Error (MSE): {mse_hybrid}")
print(f"Root Mean Squared Error (RMSE): {rmse_hybrid}")
print(f"Mean Absolute Percentage Error (MAPE): {mape_hybrid}%")
```

Visulaizations

```
import seaborn as sns

# Create a DataFrame for comparison
comparison_data = pd.DataFrame({
    'Actual': actual_values,
    'NeuralProphet': neural_prophet_predictions,
    'Hybrid Combined': hybrid_combined_predictions
})
# Plot a scatter plot with line of best fit for NeuralProphet predictions
sns.lmplot(x='Actual', y='NeuralProphet', data=comparison_data, scatter_kws={'alpha':0.5}, line_kws={'color':
'green'}, height=6, aspect=1.5)
plt.title('NeuralProphet Predictions vs Actual')
plt.xlabel('Actual')
plt.ylabel('Predicted (NeuralProphet)')
plt.show()
# Plot a scatter plot with line of best fit for hybrid combined predictions
sns.lmplot(x='Actual', y='Hybrid Combined', data=comparison_data, scatter_kws={'alpha':0.5}, line_kws={'color':
'orange'}, height=6, aspect=1.5)
plt.title('Hybrid Combined Predictions vs Actual')
plt.xlabel('Actual')
plt.ylabel('Predicted (Hybrid Combined)')
plt.show()
```

```
# Calculate residuals for NeuralProphet and hybrid combined predictions
residuals_np = neural_prophet_predictions - actual_values
```

```
residuals_hybrid = hybrid_combined_predictions - actual_values

# Plot residuals for NeuralProphet
plt.scatter(actual_values, residuals_np, color='blue', label='NeuralProphet Residuals', alpha=0.5)
# Plot residuals for hybrid combined model
plt.scatter(actual_values, residuals_hybrid, color='red', label='Hybrid Combined Residuals', alpha=0.5)
plt.axhline(y=0, color='black', linestyle='--')  # Add horizontal line at y=0 for reference
plt.title('Residuals Plot: NeuralProphet vs Hybrid Combined')
plt.xlabel('Actual Values')
plt.ylabel('Residuals')
plt.legend()
plt.grid(True)
plt.show()
```

6. RESULTS OBSERVED

The comprehensive evaluation of our hybrid ensemble model against the existing NeuralProphet framework reveals compelling evidence of its superiority in time series forecasting. Across all performance metrics, including mean absolute error (MAE), mean squared error (MSE), root mean squared error (RMSE), and mean absolute percentage error (MAPE), our hybrid model consistently outperforms NeuralProphet. Notably, our hybrid model demonstrates a remarkable overall improvement of approximately 15% in forecast accuracy compared to NeuralProphet. These results underscore the effectiveness of our innovative approach in combining the strengths of NeuralProphet and LSTM architectures, resulting in more precise and reliable predictions. With these advancements, our hybrid ensemble model emerges as a formidable tool for decision-makers across diverse domains, offering enhanced insights for informed planning and strategic decision-making.

	Mean absolute error (MAE)	Mean squared error(MSE)	Root mean squared error(RMSE	Mean absolute percentage error(MAPE)
1. NeuralProphet	168.0391	41620.727	204.011	7.282
2. Hybrid Model	165.421	39239.775	198.090	6.897