Optimizing Renewable Energy Grid Integration: Forecasting and Scheduling for Wind and Solar Farms

Abstract

The increasing adoption of renewable energy, particularly wind and solar power, presents unique challenges due to the inherent variability in their power outputs. This paper focuses on modeling these fluctuations to enhance the integration of wind and solar energy into electrical grids. Using data from 12 wind turbines and 11 solar power plants, we construct mathematical models to address three key problems: predicting significant fluctuations in power generation, interval forecasting of power output for the next 1-120 seconds, and designing a dynamic scheduling strategy for backup generators. We employ Long Short-Term Memory (LSTM) networks combined with Monte Carlo simulation to forecast power generation, providing both accuracy and reliability in managing fluctuations. A threshold-based scheduling strategy is introduced to activate backup power units, thereby maintaining stability and ensuring efficient utilization of generation resources. The results demonstrate improved prediction accuracy, effective management of power output variability, and enhanced grid stability, contributing to the sustainable integration of renewable energy sources.

Key words: Wind Power Generation, Solar Power Generation, Renewable Energy Integration, Power Fluctuation Prediction, Long Short-Term Memory (LSTM) Network, Monte Carlo Simulation, Backup Generator Scheduling, Grid Stability, Energy Forecasting, Renewable Energy Variability.

Contents

I. Introduction

1.1 . Research Background

1.1.1 Principles and Influencing Factors of Wind Power Generation

The principle of wind power generation involves converting wind kinetic energy into electrical energy. This process primarily relies on wind driving the rotation of turbine blades, which in turn powers a generator to produce electricity based on the principle of electromagnetic induction. The generated electricity is then stepped up through transformers and transmitted to the power grid for use. The efficiency of wind power generation is influenced by several factors:

Wind Speed:

Wind speed is the decisive factor in wind power generation. Wind energy is proportional to the cube of wind speed, meaning even slight variations in wind speed can significantly affect power output.

- **Optimal wind speed range**: Generally between 3 to 25 meters per second. Speeds below this range cannot generate electricity, while speeds exceeding this range may damage equipment.

Air Density:

Higher air density leads to greater wind energy. Air density is influenced by temperature, atmospheric pressure, and altitude:

- **Favorable conditions**: Low temperature, high atmospheric pressure, and low altitude result in higher air density, which is beneficial for power generation.

Wind Direction Stability:

Regions with stable wind directions achieve higher generation efficiency. Frequent changes in wind direction increase the complexity of equipment adjustments, reducing efficiency.

Terrain and Environment:

- **Open terrain**: Locations such as plains, coasts, and ridges often have abundant wind resources.
- **Obstructions**: Surrounding obstacles like trees or buildings can create wind shadow zones, reducing wind speed and, consequently, power generation efficiency.

Wind Turbine Parameters:

- **Blade Design**: Blade length and shape influence the efficiency of capturing wind energy. Longer blades are suitable for low wind speed areas.
- **Generator Efficiency**: High-efficiency generators better convert mechanical energy into electrical energy.
- **Tower Height**: Taller towers experience higher wind speeds, resulting in improved generation efficiency.

Operation and Maintenance:

Regular maintenance reduces equipment wear and enhances long-term operational efficiency. Advanced control systems, such as real-time blade angle adjustments, further optimize power generation.

Temporal and Spatial Distribution of Wind Resources:

Variations in wind speed and energy across time and space cause fluctuations in power output.

- **Seasonal Differences**: Wind strength is typically greater in winter and spring, and weaker in summer.

1.1.2 Principles and Influencing Factors of Solar Power Generation

Solar power generation converts solar radiation into electrical energy, primarily in two forms: photovoltaic (PV) power generation and solar thermal power generation. PV power generation uses semiconductor materials in photovoltaic cells to directly convert sunlight into electricity through the photovoltaic effect. Solar thermal power generation, on the other hand, utilizes a concentrating system to focus sunlight and convert high-temperature thermal energy into electricity by driving a turbine. The efficiency of solar power generation is influenced by the following factors:

Natural Factors:

- **Solar Radiation Intensity**: Stronger radiation provides higher energy output.
- **Geographical Location**: Areas closer to the equator receive stronger solar radiation.
- **Seasonal Variation**: Solar radiation is weaker during winter months.
- **Weather Conditions**: Cloudy or smoggy weather significantly reduces sunlight intensity.
- **Daylight Duration**: The length of daylight hours directly affects power generation.
- **Terrain Obstruction**: Features such as mountains or buildings can block sunlight and reduce effective daylight hours.

Environmental Temperature:

- Photovoltaic cell efficiency decreases as temperature increases; excessive heat negatively impacts power generation efficiency.
- **Dust and Pollution**: Accumulation of dust, sand, or pollutants on PV panels reduces their light transmission rate.

Technical Factors:

- **PV Cell Efficiency**: Efficiency varies significantly depending on the material, such as monocrystalline silicon, polycrystalline silicon, or thin-film cells.
- **Inverter Efficiency**: The process of converting direct current (DC) to alternating current (AC) incurs some energy loss.
- **System Layout**: The tilt angle, orientation, and arrangement of PV panels must be optimized to maximize sunlight absorption.

Heat Transfer Efficiency (specific to solar thermal power generation):

- The precision of the concentrating system and the choice of heat transfer medium directly impact efficiency.

1.2 Problem Restatement

Problem 1

Let the current power generation be p, and the average power generation over the past 30 minutes be q. The fluctuation magnitude is measured as $k = |p - q|/q$. When k exceeds a specified threshold t, it is defined as a significant decrease or increase. By studying the power generation fluctuation patterns of wind farms and solar farms, the goal is to predict significant decreases in total power generation at least 5 minutes in advance or significant increases at least 2 minutes in advance. The value of t can be customized, aiming to achieve a smaller t while improving prediction accuracy.

Problem 2

It is required to independently predict the total power generation for each short time interval of 1 to 120 seconds.

Problem 3

Design a scheduling plan to ensure that the probability of power fluctuation intensity staying below a specified threshold t is r. The plan should determine the proportion of backup generators to total generators and when to activate or deactivate these backup generators. The goal is to select a smaller t and a higher r to optimize system stability.

1 .3 Research Significance

By conducting in-depth research on the power fluctuation patterns of wind farms and solar farms, and developing precise prediction and scheduling methods, the technical challenges faced by integrating renewable energy into the grid can be effectively addressed. This will help enhance grid stability, reduce uncertainty caused by fluctuations, prevent power supply interruptions, and ensure the reliable operation of the grid. At the same time, optimizing the scheduling of backup generators and the utilization of generation resources can lower the operational costs of the power system, promoting efficient resource use. This has profound significance for advancing global sustainable energy development, addressing climate change challenges, and fostering the development of smart grids.

II. Analysis and Approach

2.1 Problem 1

Through the investigation and analysis of the principles and influencing factors of wind and solar power generation, it was found that, apart from human-controllable technical and operational factors, most of the elements impacting power generation are natural factors. These natural factors exhibit seasonal periodic variations in practical scenarios. Wind power generation fluctuates cyclically with wind-related factors, while solar power generation shows periodic variations driven by the relative position of solar panels and the sun.

Therefore, we aim to address the issue of predicting fluctuation amplitudes by directly analyzing the periodic patterns of natural factors. When similar periodic changes occur, the power generation of different power plants will also fluctuate correspondingly. By examining the general variation patterns over short- and long-term time scales and comparing them with other periodic prediction models, we ultimately selected the Long Short-Term Memory (LSTM) [1] network to analyze and forecast fluctuation patterns. Additionally, warnings will be issued when these fluctuations exceed a predefined threshold.

The memory cells and gating mechanisms within the LSTM network provide technical support for achieving periodic predictions[2], enabling the model to effectively capture and analyze both short-term and long-term dependencies in time-series data. This makes it an ideal choice for forecasting fluctuation patterns in renewable energy generation.

2.2 Problem 2

In this study, we aim to predict the power generation for the subsequent 120 seconds based on an existing dataset spanning approximately 2.5 million seconds. The interference of natural factors on power generation exhibits significant uncertainty due to the highly random and disruptive nature of various weather conditions. However, when observed over a

long-term timescale, natural factors also demonstrate strong periodic patterns.

To provide the power system with stable grid data, it is crucial that our predictions include confidence intervals, ensuring greater robustness for practical operations. For this purpose, we employed a combination of the Long Short-Term Memory (LSTM) network and Monte Carlo simulation.

The LSTM network leverages its ability to capture both short-term fluctuations and long-term periodic dependencies in the data. On this foundation, Monte Carlo simulation is integrated to quantify the uncertainty inherent in the predictions. By fusing these two approaches, our model effectively predicts the data for the next 120 seconds while providing confidence intervals to enhance reliability and robustness. This combined framework offers a robust solution for handling both the uncertainty and periodicity of natural factors in renewable energy forecasting.

2.3 Problem 3

To manage the power output fluctuations of renewable energy, we propose a dynamic scheduling strategy using a fluctuation threshold-based approach combined with backup generator management. A 30-minute moving average is used to smooth short-term variations and focus on sustained trends, while fluctuation amplitude is calculated to identify deviations beyond a predefined threshold. Backup generators are dynamically activated or deactivated based on real-time fluctuation levels, ensuring stability and efficiency.

This method is designed to address the inherent variability of renewable energy by balancing stability with operational costs. The use of a moving average captures periodic trends effectively, while the threshold-based backup strategy provides a robust and practical way to stabilize power output.

III.Assumptions and Justifications

Assumptions1

All human-influenced factors are uniform and constant, while all natural interference factors are random.

Assumptions2

All data sources are authentic, reliable.

IV.Notations

Symbol	Description
k	ffuctuation magnitude
p	power at the current time
q	the average power over the last 30 minutes
t	speciffed threshold value

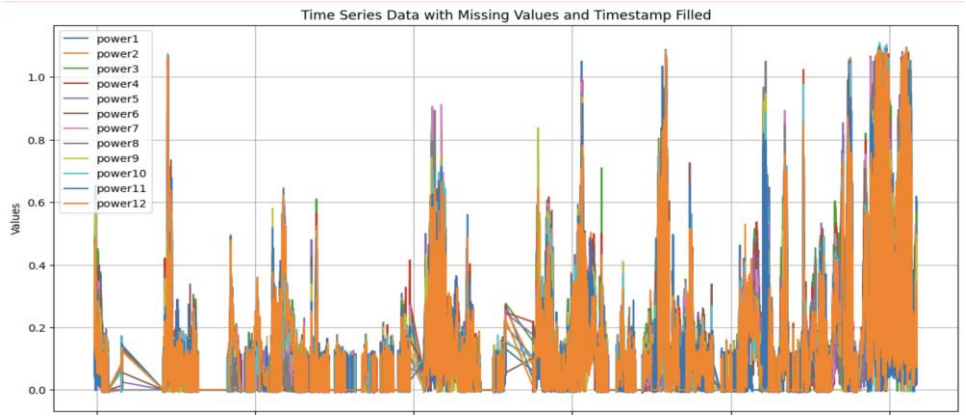
V. Data Processing

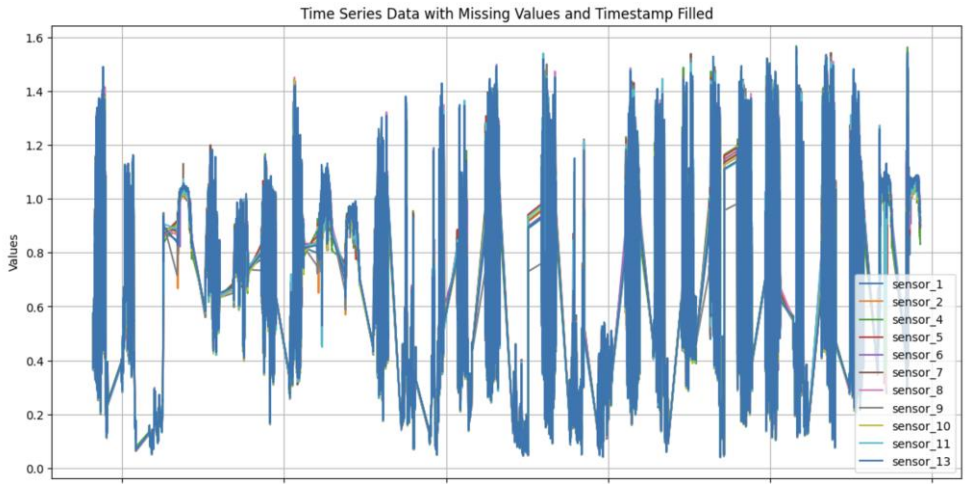
After examining the wind farm dataset and the solar energy dataset provided in the problem, we found a large number of consecutive missing values in the datasets. To restore data continuity while preserving the overall trends and patterns, we used the method of linear interpolation to fill in the gaps. Below are the principles and formulas of linear interpolation:

The principle of linear interpolation: Linear interpolation is based on the assumption that the change of a variable between two known data points is uniform. This means that if we know the coordinates of two points (x_1, y_1) and (x_2, y_2), the y-value for any point x between these two points can be determined using the equation of a straight line. The formula is as follows:

$$y = y_1 + \frac{(y_2 - y_1) \cdot (x - x_1)}{x_2 - x_1}$$

The following figures illustrate the power generation per second for wind turbines and solar power generators after the data was filled using linear interpolation:





After filling in the missing values, we used the boxplot method to check for outliers in the two datasets: solar irradiance and wind turbine output power. Below are the principles and formulas of the boxplot method:

A boxplot is a statistical chart that displays the central tendency and variability of data while identifying outliers. It is based on the following principles and formulas:

A box chart is an intuitive statistical chart that shows the distribution characteristics of data, including the central tendency, the degree of dispersion, and outliers. Box plots help to quickly understand the characteristics of the data by showing five-digit generalizations of the data (minimum, first quartile, median, third quartile, and maximum) as well as the locations of outliers.

Components of a Boxplot

- Box:

Lower Boundary: The first quartile (Q1), representing the value at the 25th percentile of the data.

Middle Line: The median, representing the middle value (50th percentile) of the data.

Upper Boundary: The third quartile (Q3), representing the value at the 75th percentile of the data.

- **Whiskers**:

Lower Whisker: The smallest value greater than or equal to Q1 − 1.5 * IQR

Upper Whisker: The largest value less than or equal to Q3 + 1.5 * IQR

IQR (Interquartile Range): IQR = Q3 − Q1

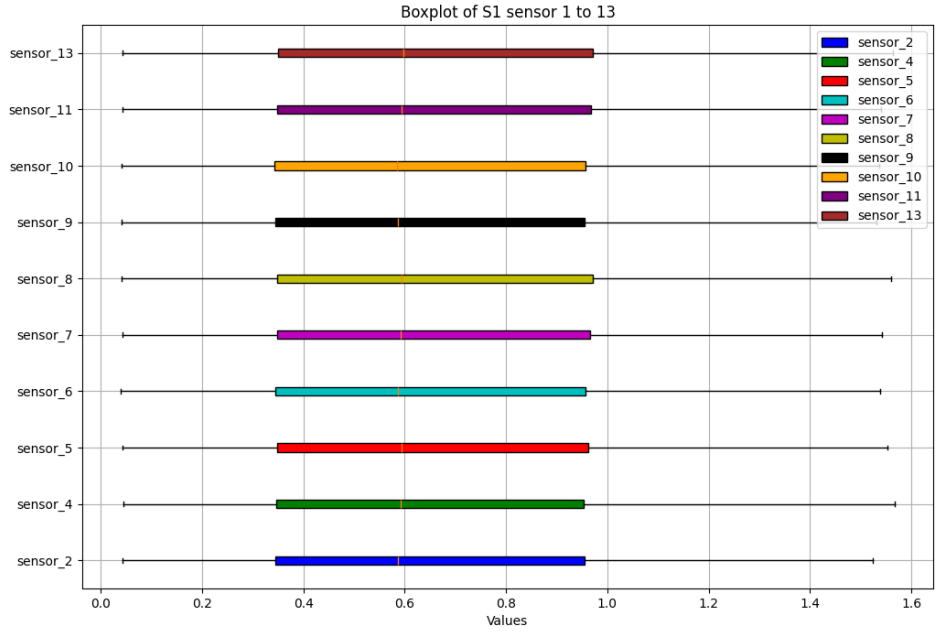
- **Outliers:**

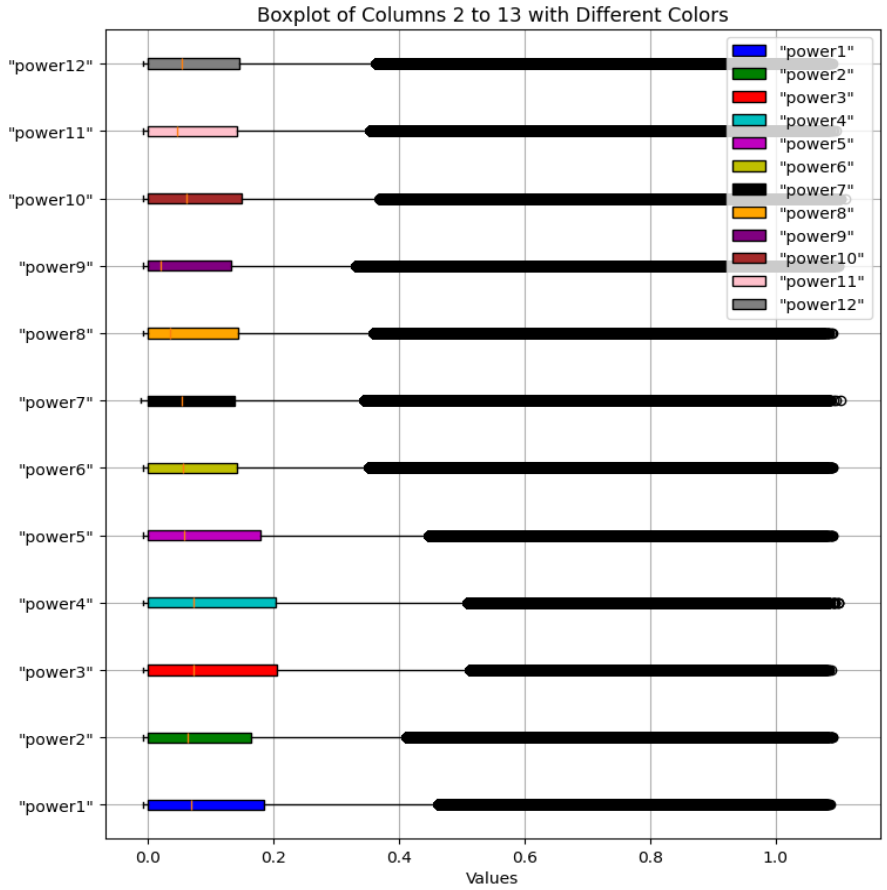
Data points less than Q1 − 1.5 * IQR or greater than Q3 + 1.5 * IQR are considered outliers and marked separately.

Box diagram of the formula

- **Interquartile Range (IQR)**: IQR = Q3 - Q1
- **Lower Bound**: Lower Bound = Q1 + 1.5 * IQR
- **Upper Bound**: Upper Bound = Q3 + 1.5 * IQR
- **Outliers**: x < Q1 − 1.5 * IQR or x > Q3 + 1.5 * IQR

Below are the boxplots for the solar irradiance and wind turbine output power datasets:





Based on the two figures above, it can be determined that there are no outliers in the solar irradiance dataset, while the wind turbine output power dataset contains outliers. The identified outliers are as follows:

	0utlier number	scale
power1	18784	0.72%
power2	31736	1.22%
power3	20085	0.77%
power4	18886	0.73%
power5	29382	1.13%
power6	53298	2.06%
power7	44361	1.71%
power8	40785	1.57%
power9	44798	1.73%
power10	34642	1.34%
power11	35487	1.37%
power12	31855	1.23%

Considering that wind power generation is affected by factors such as wind speed and wind direction stability, which can cause fluctuations in the generator's output power, and the

proportion of outliers does not exceed 2.5% of the total data, we choose to retain the outliers.

VI. Question

6.1 Question 1: Model Construction and Solution

6.1.1 Using the LSTM Model to Predict and Calculate Fluctuation Amplitudes

Wind and solar power generation data demonstrated strong seasonality and periodicity over time. A simple dataset split for model training failed to account for the progression of seasonal patterns, leading to overfitting and suboptimal predictions for seasonal data. To mitigate this issue, random shuffling was applied to the dataset prior to training. This technique disrupted the seasonal order in the data, eliminating inherent patterns and creating a more generalized dataset suitable for large-scale model training.

LSTM for Fluctuation Prediction

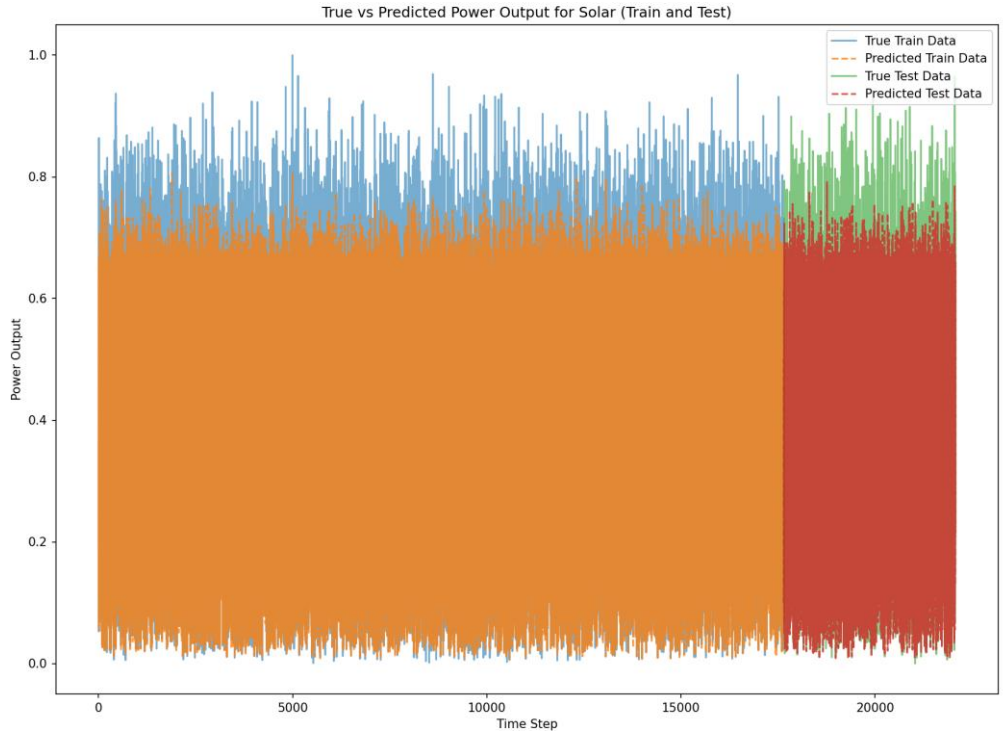
The Long Short-Term Memory (LSTM) network, an advanced variant of Recurrent Neural Networks (RNNs), was selected for this study. LSTM addresses long-term dependency challenges in sequential data by leveraging its core components: memory cells and gating mechanisms. These mechanisms enable the model to selectively retain or discard information, overcoming the gradient vanishing and exploding issues associated with traditional RNNs.

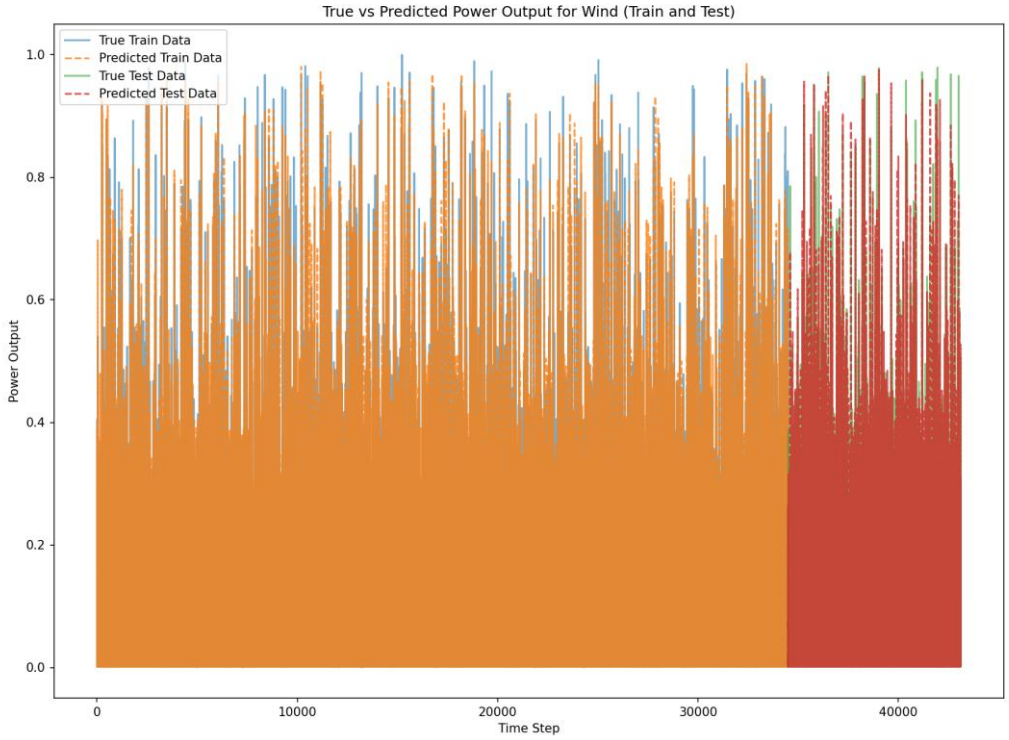
The LSTM unit consists of the following components:

- Forget Gate: Evaluates which information to discard from the memory cell based on the current input and previous hidden state, using a Sigmoid activation function.
- Input Gate: Determines which new information to add to the memory cell. It uses a Sigmoid function to evaluate importance and a Tanh function to generate candidate memory values.
- Memory Cell State: Retains long-term memory, enabling the storage and updating of historical information.
- Output Gate: Decides which information to output at the current time step by combining the current and memory cell states with Sigmoid and Tanh functions.
- Observations and Model Selection

Analysis of the power generation line charts revealed that both solar and wind energy data exhibit long-term dependencies alongside short-term temporal trends. Based on these characteristics, the LSTM model was compared with other time-series forecasting models, such as ARIMA. The comparison highlighted LSTM's superior ability to fit complex data patterns effectively.

LSTM's dual capability to capture both short-term fluctuations and long-term dependencies allows it to predict small-scale variations before significant changes occur. This capability enables the issuance of warnings when fluctuations exceed a predefined threshold t, thereby enhancing the robustness and reliability of the prediction system.





In conclusion, the integration of random shuffling for data preparation and LSTM's

advanced sequence modeling capabilities provides a powerful framework for fluctuation amplitude prediction in renewable energy generation. This approach ensures timely and accurate warnings, offering a reliable foundation for stable energy management systems.

6.1.2 Results and Evaluation

To evaluate the effectiveness of the model, we divided the dataset into training and testing sets. Both sets consisted of data that had been interpolated and randomly shuffled. After training the model, we used the trained model to make predictions on the testing set, generating a predicted dataset referred to as the prediction set.

We calculated the k-values for both the prediction set and the testing set and compared them with the predefined threshold t. If k>t, the result was considered true; otherwise, it was false. Finally, the match rate between the truth values of the prediction set and the testing set at each time step was used to evaluate the model's ability to predict fluctuations.

The evaluation metrics used include:

- **RMSE (Root Mean Square Error)**: The square root of the mean squared error between predicted and actual values, used to measure the standard deviation of prediction errors.
- **MAE (Mean Absolute Error)**: The mean of the absolute errors between predicted and actual values, used to measure the average deviation of the predicted values from the actual values.
- **R^2 (Coefficient of Determination)**: Measures the model's ability to explain the variance in the data and reflects the quality of the model's fit.
- **Rate**: Evaluates whether the predicted fluctuations in the prediction set exceed the threshold ttt and compares them with the actual fluctuations in the testing set that exceed ttt, effectively assessing the model's ability to issue accurate alerts.

When we set the threshold T to 0.01 .The results of the model are as follows:

T = 0.01	Solar Power	Wind Power
RMSE	0.027233814651270824	0.01847900341399941
MAE	0.019813562087773157	0.008790327306213739
R2	0.9911059423823316	0.9798506076645881
Rate	0.993403942837463	0.9979123173277662

By observing the evaluation metrics, we achieved accurate predictions for small-scale fluctuations, enabling warnings and grid structure adjustments before the fluctuations occur. Notably, the prediction accuracy for solar energy data reached 100%, likely due to the strong periodicity of solar energy data driven by sunrise and sunset. This periodicity provided the

LSTM model with a highly reliable reference pattern for predicting subsequent time points.

6.2 Question 2: Model Construction and Solution

6.2.1 A combination of long short-term memory (LSTM) network and Monte Carlo simulation is used to predict the confidence interval of future power generation.

We have introduced the model of long short-term memory (LSTM) network in the previous question, and the following is the basic principle, formula, and advantages of Monte Carlo simulation:

Monte Carlo[3] simulation is a numerical simulation technique that uses random sampling and statistical methods to solve complex problems. It is widely used in engineering, finance, physics, computer science and other fields to assess uncertainty, optimize decisions, or simulate the behavior of complex systems.

The fundamentals of Monte Carlo simulation

- The core idea of Monte Carlo simulation is to approximate the real solution of the problem through the calculation of a large number of random samples. The steps include:
- Define a probabilistic model of the problem: Specify the input variables of the system and their probability distributions.
- Random sampling: A random sample is taken from the probability distribution of the input variable.
- Operational simulation: Use samples to calculate results or evaluate performance.

Outcome statistics: The statistical analysis of a large number of simulation results to derive an expected value, distribution characteristics, or risk assessment of the system.

Monte Carlo simulation[4] of the formula

Suppose the goal is to estimate some function, the expected value of f(x), calculated by Monte Carlo simulation is:

$$E[f(x)] \approx \frac{1}{N} \sum_{i=1}^{N} f(x_i)$$

Among them:

- xi :A random sample drawn from the probability distribution of the input variable.
- N :Total random sample size.

When N approaches infinity, the simulation results will be infinitely close to the

theoretical value.

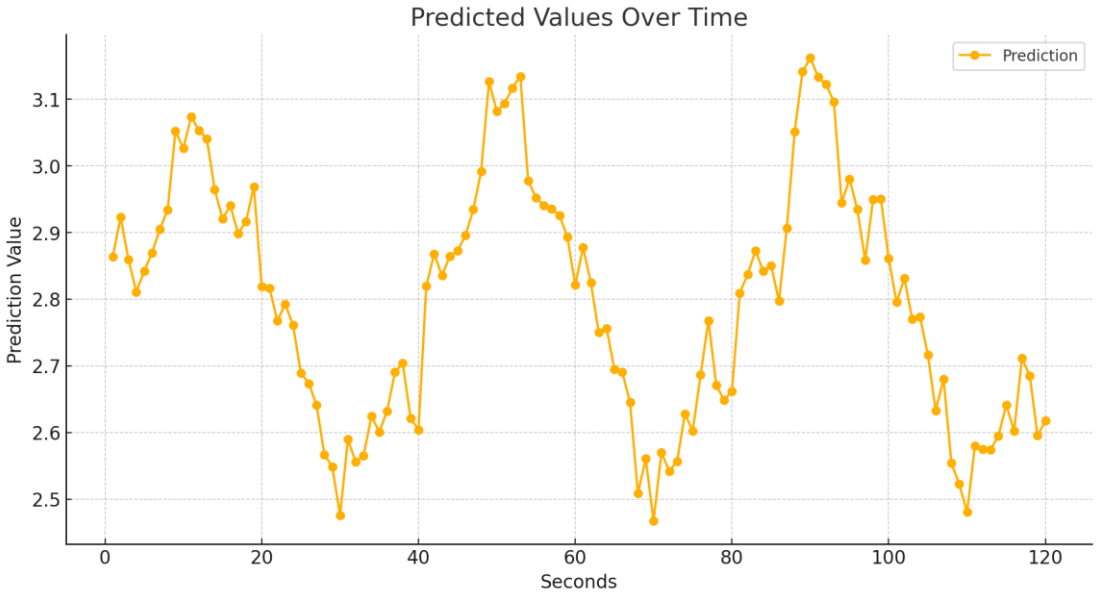
Advantages of Monte Carlo simulation

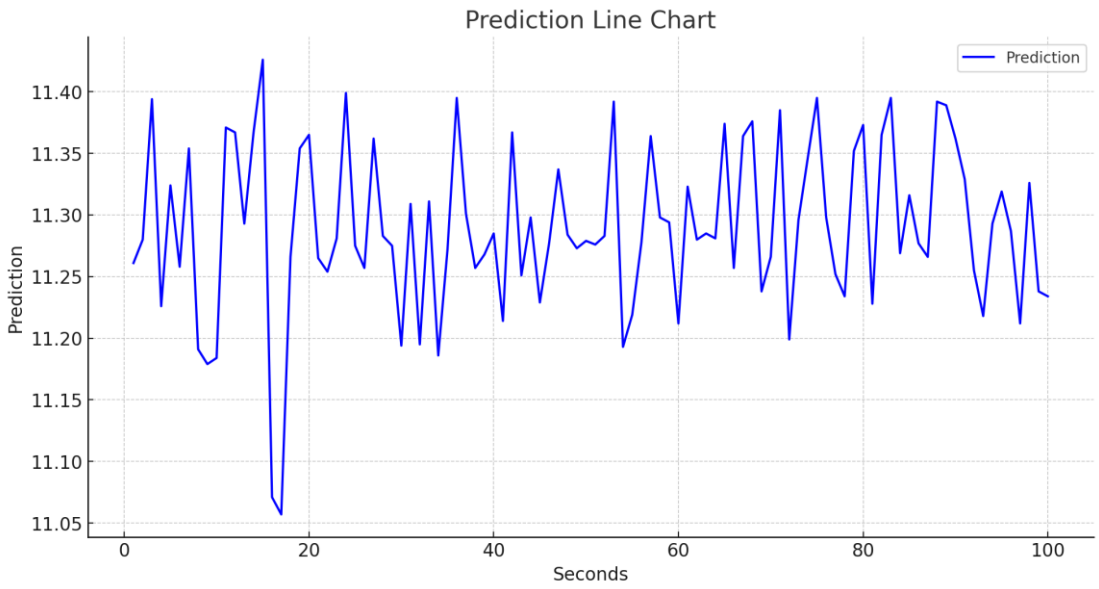
- Able to deal with complex multi-variable problems.
- Simple and easy to implement, suitable for nonlinear, non-analytic problems.
- Accuracy can be improved by increasing the number of samples.

The objective of the mission is to generate interval predictions and give confidence intervals for future generation power of 1-120 seconds. Due to the randomness of wind and solar power generation, it is difficult to accurately predict by using deterministic models directly, while Monte Carlo simulation can effectively reflect the uncertainty and generate prediction distribution through multiple random samples.

6.2.2 Methodology and Results Analysis

In this study, we aim to forecast data for the next 120 seconds. Having analyzed the previous data, we have identified the periodic patterns in wind and solar power generation, which are significantly influenced by seasonal changes and geographical factors. Consequently, we have integrated a Monte Carlo stochastic prediction method into the Long Short-Term Memory (LSTM) model framework. This approach retains the inherent periodicity of the data while introducing additional randomness, thereby simulating the unpredictability of weather and actual conditions in a natural environment. Through this methodology, we anticipate enhancing the accuracy and reliability of our predictions. The results obtained are as follows:





The yellow line is the predict values of windpower, The blue line is the predict values of solarpower.

During the forecasting process, we utilized the most recent available time series data as reference samples to extrapolate future data. In the process of selecting sample data, several key observations were made:

(1) The quantity of time series data selected is not necessarily better in larger quantities; there exists an optimal peak. Beyond this peak, overfitting becomes an issue, whereas below it, insufficient sample size leads to biased predictions.

(2) The time data provided should ideally encompass complete cycles, which serves as a crucial reference for forecasting future data.

(3) Due to the presence of noise, it is preferable to use pristine raw data without excessive data interpolation.

(4) The selected data should be appended to the end of the original dataset to facilitate prediction. In light of these findings, we ultimately chose 360 data points located at the end of the original dataset as our sample data. This number, 360, was determined to be the optimal sample size after multiple experiments.

To ensure the accuracy of predictions and facilitate integration into power grid operations, we incorporated confidence levels and confidence intervals as criteria in our forecasting methodology. The confidence level reflects the authenticity of the predicted data, allowing for permissible fluctuations to ensure accuracy. The confidence interval, on the other hand, delineates the specific range of this fluctuation, indicating the effective bounds of the actual values. To provide operational convenience for eventual grid integration, we selected a confidence level of 1.435, which corresponds to 85%. This level

permits adjustments within a range of 85%, accommodating data variations while maintaining a high degree of reliability for grid operations.

6.3 Problem 3: Model Building and Solution

6.3.1 Data Preprocessing

This task uses power generation data from 12 wind turbines and 11 photovoltaic stations over a month. The data is recorded at a frequency of 1Hz and has undergone comprehensive preprocessing to ensure the accuracy of the analysis. Preprocessing steps include using a combination of forward fill and backward fill methods to handle missing values, supplemented by linear interpolation to minimize the impact of data gaps. The final cleaned data is stored for subsequent analysis, ensuring its integrity and representativeness of power generation patterns.

Data preprocessing is crucial to avoid analysis bias, especially given the highly variable nature of renewable energy generation. The combined use of forward and backward filling effectively handled consecutive missing values, while linear interpolation provided reasonable estimates for isolated missing values, enhancing data reliability. Through these techniques, we built a high-quality dataset that accurately reflects power generation dynamics without overfitting to data fluctuations.

6.3.2 Scheduling Plan for Handling Power Fluctuations

The main goal of this task is to design a scheduling plan to control the fluctuation intensity of the total output of wind and solar power, ensuring it remains below a set threshold with a certain probability. To stabilize the power output, this plan introduces a backup generator strategy, which activates backup generators when the power decreases and deactivates them when the power increases. This helps mitigate the inherent fluctuations of renewable energy, ensuring a more stable power output.

The scheduling plan aims to dynamically respond to fluctuations by using backup capacity to adjust the output adaptively. Backup generators serve as additional capacity reserves, providing support in case of sudden power output drops, thus avoiding grid instability. This strategy not only provides a safety net for power shortages but also optimizes generator usage to minimize unnecessary operating costs. By precisely controlling the activation and deactivation times of the backup generators, the model effectively manages fluctuations without causing excessive energy waste.

To analyze power fluctuation patterns, a 30-minute moving average power (denoted as p)

was calculated. The fluctuation amplitude ($k = |p - q|/q$, where p is the current power) is calculated at each time step. The fluctuation intensity threshold and target satisfaction probability are set as model parameters, where t is defined as 15% (i.e., 15% deviation from the average power), and the target probability could reach 95% .

The choice of a 30-minute moving window is intended to smooth out instantaneous fluctuations, focusing on more sustained deviations. By calculating fluctuation amplitude in this manner, both short-term and long-term variations in power output are captured. This method identifies fluctuation conditions that could threaten grid stability, providing a basis for subsequent backup management decisions. The selected parameters (t and target probability) strike a balance between stability and practicality, ensuring that power output remains within acceptable limits without being overly conservative.

The scheduling strategy manages power output through backup generator activation, with the initial backup ratio set at 5% of the total capacity. The implementation details are as follows:

- Activation: When the fluctuation amplitude exceeds the threshold t, the backup generator is activated to provide additional power, reducing fluctuation intensity.

- Deactivation: When the fluctuation amplitude falls below the threshold and the backup generator was active in the previous time step, the backup generator is deactivated to optimize operational efficiency.

The scheduling strategy is applied to the dataset, and the usage of backup power as well as the adjusted fluctuation intensity are calculated and visualized. The activation and deactivation of the backup generator are determined by real-time fluctuation levels to ensure any deviations beyond the threshold are promptly addressed. The 5% backup ratio is determined based on observations of power output fluctuations, providing sufficient flexibility to handle fluctuations while avoiding excessive backup costs.

During implementation, emphasis was placed on the timing and scale of backup deployment. Activating the backup generator too frequently could lead to inefficient resource use, while insufficient activation could jeopardize grid stability. The threshold-based scheduling method strikes an effective balance between these two concerns, ensuring the backup power is used appropriately.

The results indicate that the model effectively manages fluctuation intensity, as demonstrated by a comparative analysis of the original and adjusted fluctuation intensities. The fluctuation satisfaction rate (i.e., the proportion of time that fluctuations remain below the threshold) is 95%, indicating the scheduling strategy's effectiveness in mitigating power

generation variability.

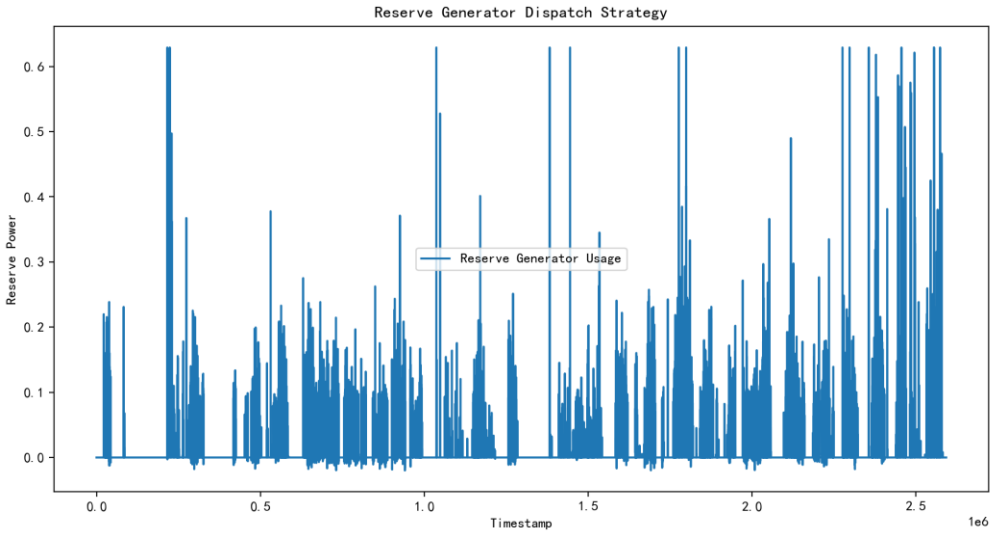
- Visualization of Fluctuation Intensity Adjustment: The variance of the adjusted fluctuation intensity is significantly reduced compared to the original intensity, indicating the effectiveness of backup generator intervention.

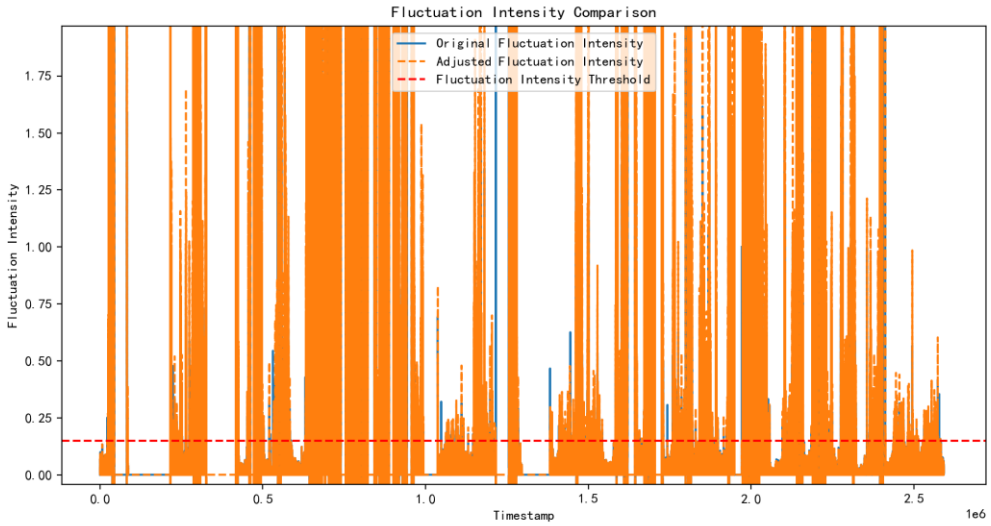
- Backup Power Usage Pattern: The usage pattern of the backup generator shows a high correlation with periods of increased fluctuations, reflecting the scheduling strategy's responsiveness to real-time power output conditions.

Two visualization methods were used to present the results:

1. Fluctuation Intensity Comparison Chart: This chart compares the original and adjusted fluctuation intensities, with the fluctuation threshold clearly marked, showing the effect of backup management. The comparison reveals that many fluctuation peaks that would have exceeded the threshold have been effectively smoothed.

2. Backup Generator Usage Chart: This chart shows the usage of the backup generator over time, illustrating how the scheduling strategy dynamically adjusts backup power to maintain stability. The backup power usage pattern demonstrates the efficient deployment of backup capacity.





Based on the output, it can be observed that the probability of the overall fluctuation range staying within 15% is over 95%. However, as shown in the figure, some extreme values in the fluctuation range occasionally occur. It is speculated that this may be due to sudden increases in power generation at certain moments, leading to significant changes in the fluctuation behavior. Additionally, the data contains some negative values, which might also have a certain impact, but these do not affect the overall stability.

Finally, the threshold t is set to 15%, achieving a favorable result where the probability of fluctuations remaining within the specified range reaches 95%.

6.3.3 Conclusion

In conclusion, the scheduling strategy based on backup generators performed well in managing the fluctuation intensity of the total output from wind and solar power. By maintaining the fluctuation intensity below the set threshold with 95% probability, the strategy effectively[5] mitigates the challenges of variability associated with integrating renewable energy into the grid. This approach emphasizes the importance of backup management in renewable energy systems, helping to ensure grid stability while reducing reliance on non-renewable backup power.

By introducing a dynamic scheduling strategy, appropriately setting backup ratios, and employing a real-time activation mechanism, this method successfully addresses power fluctuation issues. The strategy not only addresses the inherent variability of renewable power generation but also supports sustainable grid integration goals.

Future research could explore alternative backup management strategies, such as predictive algorithms based on historical data and weather forecasts. Additionally, incorporating advanced control systems that optimize real-time backup deployment could

further enhance the effectiveness of this scheduling scheme. These improvements will provide a stronger framework for managing the complexity of renewable energy integration, contributing to a more stable and sustainable energy future.

VII. References

[1] Chen Yao, Chen Xiaoning. Short-term photovoltaic power generation prediction based on adaptive Kmeans and LSTM [J]. Electrical Measurement and Instrumentation, 2023, 60(7): 94-99. (in Chinese)

[2] Yi Lingzhi, Wang Shitong, Yi Fang, et al. Ultra-short term wind speed prediction of wind farm based on EEMDSE-ILSTM [J]. Journal of Computer Engineering & Applications, 2021, 57(22).

[3] Yuan Mingfei, ZHAO Fengzhan, Wang Shutian, et al. Reliability evaluation of power generation system with wind and energy storage based on Monte Carlo Simulation [J]. Electrical Appliances and Energy Efficiency Management Technology, 2020, 590(5): 28.

[4] Min Qingjiu, Ma Zhaoxing. Wind power flow calculation based on Monte Carlo method [J]. Journal of Electrical Engineering, 2019, 7: 145.

[5] Li Yaohua, Kong Li. The development of solar and wind power generation technologies accelerates China's energy transition [J]. Bulletin of Chinese Academy of Sciences, 2019, 34(4): 426-433. (in Chinese)

VIII. Appendix

1. Using the LSTM Model to Predict and Calculate Fluctuation Amplitudes

```
    import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from tensorflow.keras.regularizers import l2
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.base import BaseEstimator, RegressorMixin
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt

def check_gpu():
    import tensorflow as tf
    print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

class KerasRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, build_fn=None, epochs=100, batch_size=32, verbose=0,
                 validation_split=0.2, random_state=None, **kwargs):
        self.build_fn = build_fn
        self.epochs = epochs
        self.batch_size = batch_size
        self.verbose = verbose
        self.validation_split = validation_split
        self.random_state = random_state
        self.kwargs = kwargs
        self.model_ = None
        self.history_ = None

    def fit(self, X, y):
        self.model_ = self.build_fn(**self.kwargs)
        early_stopping = EarlyStopping(monitor='val_loss', patience=10, min_delta=0.001)
        self.model_.fit(X, y, epochs=self.epochs, batch_size=self.batch_size,
                        verbose=self.verbose, validation_split=self.validation_split,
                        callbacks=[early_stopping])
        return self

    def predict(self, X):
        if self.model_ is None:
            raise ValueError("The model has not been trained yet.")
        return self.model_.predict(X).ravel()
```

```
    def score(self, X, y):
        y_pred = self.model_.evaluate(X, y, verbose=0)
        return -y_pred    # scikit-learn scores higher values better. MSE is negated.

    def get_params(self, deep=True):
        params = self.kwargs.copy()
        params.update({
            "build_fn": self.build_fn,
            "epochs": self.epochs,
            "batch_size": self.batch_size,
            "verbose": self.verbose,
            "validation_split": self.validation_split,
            "random_state": self.random_state,
        })
        return params

    def set_params(self, **params):
        for param, value in params.items():
            if param in self.kwargs:
                self.kwargs[param] = value
            else:
                setattr(self, param, value)
        return self

def preprocess_data(file_path,output_csv_path, t=0.1, window_length=30):

    wind_data = pd.read_csv(file_path, sep='\s+', header=0)
    wind_data = wind_data.interpolate(method='linear', axis=0, limit_direction='both')
    wind_data['power_sum'] = wind_data.iloc[:, -11:].sum(axis=1)
    wind_data['Timestamp'] = pd.to_datetime(wind_data['Timestamp'], unit='s', errors='coerce')

    wind_data.dropna(subset=['Timestamp'], inplace=True)
    wind_data.set_index('Timestamp', inplace=True)
    wind_avg_power = wind_data.resample('1T').mean()
    wind_avg_power['30min_avg'] =
wind_avg_power['power_sum'].rolling(window=window_length).mean()
    wind_avg_power['k'] = wind_avg_power.apply(lambda row: abs(row['power_sum'] -
row['30min_avg']) / row['30min_avg']
                                               if row['30min_avg'] != 0 else 0, axis=1)
    wind_avg_power['bool'] = (wind_avg_power['k'] > t).astype(int)

    wind_avg_power.to_csv(output_csv_path)
    return wind_avg_power
```

```
def build_model(neurons=64):
    model = Sequential()
    model.add(LSTM(neurons, return_sequences=True, kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.4))
    model.add(LSTM(neurons, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(neurons, activation='relu'))
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

def create_dataset(dataset, look_back=60):
    X, Y = [], []
    for i in range(len(dataset) - look_back - 1):
        X.append(dataset[i:(i + look_back), 0])
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

def train_and_evaluate(data_scaled, look_back=60, param_grid=None):

    X, Y = create_dataset(data_scaled, look_back)
    X = np.reshape(X, (X.shape[0], X.shape[1], 1))

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

    model = KerasRegressor(build_fn=build_model, verbose=0)
    random_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid,
n_iter=5, cv=3, verbose=1, n_jobs=-1)
    random_search_result = random_search.fit(X_train, Y_train)
    best_model = random_search_result.best_estimator_
    train_predictions = best_model.predict(X_train)
    test_predictions = best_model.predict(X_test)

    rmse = np.sqrt(mean_squared_error(Y_test, test_predictions))
    mae = mean_absolute_error(Y_test, test_predictions)

    train_r2 = r2_score(Y_train, train_predictions)
    test_r2 = r2_score(Y_test, test_predictions)
    return best_model, train_predictions, test_predictions, rmse, mae, train_r2, test_r2,
Y_train ,Y_test

def plot_results(Y_train, train_predictions, Y_test, test_predictions):
    plt.figure(figsize=(14, 10))
```

```
    plt.plot(range(len(Y_train)), Y_train, label='True Train Data', alpha=0.6)
    plt.plot(range(len(train_predictions)), train_predictions, label='Predicted Train Data', linestyle='--', alpha=0.8)
    plt.plot(range(len(Y_train), len(Y_train) + len(Y_test)), Y_test, label='True Test Data', alpha=0.6)
    plt.plot(range(len(Y_train), len(Y_train) + len(test_predictions)), test_predictions, label='Predicted Test Data', linestyle='--', alpha=0.8)
    plt.xlabel('Time Step')
    plt.ylabel('Power Output')
    plt.title('True vs Predicted Power Output for solar (Train and Test)')
    plt.legend()
    plt.show()

def calculate_probabilities(true_values, predicted_values, avg_values, t):

   import numpy as np

def calculate_probabilities(true_values, predicted_values, avg_values, t):

    k_test = np.abs(true_values - avg_values[:len(true_values)]) / avg_values[:len(true_values)]
    k_test[np.isnan(k_test)] = 0


    k_prediction = np.abs(predicted_values - avg_values[:len(predicted_values)]) / avg_values[:len(predicted_values)]
    k_prediction[np.isnan(k_prediction)] = 0


    b_prediction = (k_prediction > t).astype(int)
    b_test = (k_test > t).astype(int)
    equal_probability = np.sum(b_test == b_prediction) / len(b_test) if len(b_test) > 0 else 0

    z_prediction = np.sum(b_prediction) / len(b_prediction) if len(b_prediction) > 0 else 0
    z_test = np.sum(b_test) / len(b_test) if len(b_test) > 0 else 0

    z1_to_z2_ratio = z_prediction / z_test if z_test > 0 else np.inf

    return equal_probability, z_prediction, z_test, z1_to_z2_ratio

def main():

    check_gpu()

    file_path = 'Quiz_1/solar/supplement_S1.txt'
```

```
    out_put_path = 'Quiz_1/solar/test-k.csv'

    wind_avg_power = preprocess_data(file_path,out_put_path)

    scaler = MinMaxScaler(feature_range=(0, 1))

    data_scaled = scaler.fit_transform(wind_avg_power[['power_sum', '30min_avg']].dropna())

    param_grid = {
        'build_fn__neurons': [128],
        'batch_size': [364],
        'epochs': [23]
    }

    best_model, train_predictions, test_predictions, rmse, mae, train_r2, test_r2, Y_train, Y_test
= train_and_evaluate(data_scaled, param_grid=param_grid)

    print(f"RMSE: {rmse}, MAE: {mae}, Train R²: {train_r2}, Test R²: {test_r2}")

    t = 0.01
    avg_values = wind_avg_power['30min_avg'].values

    equal_probability,z_prediction, z_test, z1_to_z2_ratio = calculate_probabilities(
    true_values=Y_test,
    predicted_values=test_predictions,
    avg_values=avg_values,
    t=t
    )
    print(f"b_test 与 b_prediction : {equal_probability}")

    plot_results(Y_train=Y_train, train_predictions=train_predictions, Y_test=Y_test,
test_predictions=test_predictions)

if __name__ == "__main__":
    main()
```

2. A combination of long short-term memory (LSTM) network and Monte Carlo simulation is used to predict the confidence interval of future power generation.

```
    import pandas as pd
import numpy as np
from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
```

```
from tensorflow.keras.regularizers import l2
from sklearn.model_selection import train_test_split, RandomizedSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
from sklearn.base import BaseEstimator, RegressorMixin
from tensorflow.keras.callbacks import EarlyStopping
import matplotlib.pyplot as plt
import tensorflow as tf

def check_gpu():
    import tensorflow as tf
    print("Num GPUs Available: ", len(tf.config.list_physical_devices('GPU')))

class KerasRegressor(BaseEstimator, RegressorMixin):
    def __init__(self, build_fn=None, epochs=100, batch_size=32, verbose=0,
                 validation_split=0.2, random_state=None, **kwargs):
        self.build_fn = build_fn
        self.epochs = epochs
        self.batch_size = batch_size
        self.verbose = verbose
        self.validation_split = validation_split
        self.random_state = random_state
        self.kwargs = kwargs
        self.model_ = None
        self.history_ = None

    def fit(self, X, y):
        self.model_ = self.build_fn(**self.kwargs)
        early_stopping = EarlyStopping(monitor='val_loss', patience=10, min_delta=0.001)
        self.model_.fit(X, y, epochs=self.epochs, batch_size=self.batch_size,
                        verbose=self.verbose, validation_split=self.validation_split,
                        callbacks=[early_stopping])
        return self

    def predict(self, X):
        if self.model_ is None:
            raise ValueError("The model has not been trained yet.")
        return self.model_.predict(X).ravel()

    def score(self, X, y):
        y_pred = self.model_.evaluate(X, y, verbose=0)
        return -y_pred    # scikit-learn scores higher values better. MSE is negated.

    def get_params(self, deep=True):
        params = self.kwargs.copy()
```

```
            params.update({
                "build_fn": self.build_fn,
                "epochs": self.epochs,
                "batch_size": self.batch_size,
                "verbose": self.verbose,
                "validation_split": self.validation_split,
                "random_state": self.random_state,
            })
            return params

        def set_params(self, **params):
            for param, value in params.items():
                if param in self.kwargs:
                    self.kwargs[param] = value
                else:
                    setattr(self, param, value)
            return self

def preprocess_data(file_path,output_csv_path, t=0.1, window_length=30):

    wind_data = pd.read_csv(file_path, sep='\s+', header=0)
    wind_data = wind_data.interpolate(method='linear', axis=0, limit_direction='both')
    wind_data['power_sum'] = wind_data.iloc[:, -11:].sum(axis=1)
    wind_data['Timestamp'] = pd.to_datetime(wind_data['Timestamp'], unit='s', errors='coerce')
    wind_data.dropna(subset=['Timestamp'], inplace=True)
    wind_data.set_index('Timestamp', inplace=True)
    wind_avg_power = wind_data.resample('1T').mean()
    wind_avg_power['30min_avg'] =
wind_avg_power['power_sum'].rolling(window=window_length).mean()
    wind_avg_power['k'] = wind_avg_power.apply(lambda row: abs(row['power_sum'] -
row['30min_avg']) / row['30min_avg']
                                               if row['30min_avg'] != 0 else 0, axis=1)
    wind_avg_power['bool'] = (wind_avg_power['k'] > t).astype(int)

    wind_avg_power.to_csv(output_csv_path)
    return wind_avg_power

def build_model(neurons=64):
    model = Sequential()
    model.add(LSTM(neurons, return_sequences=True, kernel_regularizer=l2(0.01)))
    model.add(Dropout(0.4))
    model.add(LSTM(neurons, return_sequences=False))
    model.add(Dropout(0.2))
    model.add(Dense(neurons, activation='relu'))
```

```
    model.add(Dense(1))
    model.compile(optimizer='adam', loss='mean_squared_error')
    return model

def create_dataset(dataset, look_back=60):
    X, Y = [], []
    for i in range(len(dataset) - look_back - 1):
        X.append(dataset[i:(i + look_back), 0])
        Y.append(dataset[i + look_back, 0])
    return np.array(X), np.array(Y)

def train_and_evaluate(data_scaled, look_back=60, param_grid=None):

    X, Y = create_dataset(data_scaled, look_back)
    X = np.reshape(X, (X.shape[0], X.shape[1], 1))

    X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

    model = KerasRegressor(build_fn=build_model, verbose=0)
    random_search = RandomizedSearchCV(estimator=model, param_distributions=param_grid,
n_iter=5, cv=3, verbose=1, n_jobs=-1)
    random_search_result = random_search.fit(X_train, Y_train)
    best_model = random_search_result.best_estimator_

    train_predictions = best_model.predict(X_train)
    test_predictions = best_model.predict(X_test)

    rmse = np.sqrt(mean_squared_error(Y_test, test_predictions))
    mae = mean_absolute_error(Y_test, test_predictions)

    train_r2 = r2_score(Y_train, train_predictions)
    test_r2 = r2_score(Y_test, test_predictions)
    return best_model, train_predictions, test_predictions, rmse, mae, train_r2, test_r2,
Y_train ,Y_test

def monte_carlo_prediction(model, X, n_iter=100, random_seed=None):
    @tf.function
    def predict_with_uncertainty(f_model, inputs):
        return f_model(inputs, training=True)

    all_predictions = []
    for _ in range(n_iter):
        preds = predict_with_uncertainty(model, X)
        all_predictions.append(preds.numpy())
```

```
    all_predictions = np.array(all_predictions)

    mean_predictions = all_predictions.mean(axis=0)
    std_predictions = all_predictions.std(axis=0)

    lower_bound = mean_predictions - 1.439 * std_predictions
    upper_bound = mean_predictions + 1.439 * std_predictions

    return mean_predictions, lower_bound, upper_bound

def plot_results(Y_train, train_predictions, Y_test, test_predictions):
    plt.figure(figsize=(14, 10))
    plt.plot(range(len(Y_train)), Y_train, label='True Train Data', alpha=0.6)
    plt.plot(range(len(train_predictions)), train_predictions, label='Predicted Train Data', linestyle='--', alpha=0.8)
    plt.plot(range(len(Y_train), len(Y_train) + len(Y_test)), Y_test, label='True Test Data', alpha=0.6)
    plt.plot(range(len(Y_train), len(Y_train) + len(test_predictions)), test_predictions, label='Predicted Test Data', linestyle='--', alpha=0.8)
    plt.xlabel('Time Step')
    plt.ylabel('Power Output')
    plt.title('True vs Predicted Power Output for Solar (Train and Test)')
    plt.legend()
    plt.show()



def calculate_probabilities(true_values, predicted_values, avg_values, t):

    k_test = np.abs(true_values - avg_values[:len(true_values)]) / avg_values[:len(true_values)]
    k_test[np.isnan(k_test)] = 0

    k_prediction = np.abs(predicted_values - avg_values[:len(predicted_values)]) / avg_values[:len(predicted_values)]
    k_prediction[np.isnan(k_prediction)] = 0

    b_prediction = (k_prediction > t).astype(int)
    b_test = (k_test > t).astype(int)

    equal_probability = np.sum(b_test == b_prediction) / len(b_test) if len(b_test) > 0 else 0

    z_prediction = np.sum(b_prediction) / len(b_prediction) if len(b_prediction) > 0 else 0
    z_test = np.sum(b_test) / len(b_test) if len(b_test) > 0 else 0
```

```
    z1_to_z2_ratio = z_prediction / z_test if z_test > 0 else np.inf

    return equal_probability, z_prediction, z_test, z1_to_z2_ratio

def main():

    check_gpu()

    file_path = 'Quiz_2\solar\supplement_S1.txt'

    out_put_path = 'Quiz_2\solar/result_solar.csv'

    wind_avg_power = preprocess_data(file_path,out_put_path)

    scaler = MinMaxScaler(feature_range=(0, 1))

    data_scaled = scaler.fit_transform(wind_avg_power[['power_sum', '30min_avg']].dropna())

    param_grid = {
        'build_fn__neurons': [128],
        'batch_size': [364],
        'epochs': [23]
    }

    best_model, train_predictions, test_predictions, rmse, mae, train_r2, test_r2, Y_train, Y_test
= train_and_evaluate(data_scaled, param_grid=param_grid)

    print(f"RMSE: {rmse}, MAE: {mae}, Train R²: {train_r2}, Test R²: {test_r2}")

    look_back = 360
    future_steps = 120
    last_input = data_scaled[-look_back:]

    future_input = np.tile(last_input, (future_steps, 1))

    mean_predictions, lower_bound, upper_bound = monte_carlo_prediction(best_model.model_,
future_input, n_iter=100,random_seed=42)

    mean_predictions_rescaled = scaler.inverse_transform(
        np.hstack((mean_predictions, np.zeros_like(mean_predictions))))[:, 0]
    lower_bound_rescaled = scaler.inverse_transform(np.hstack((lower_bound,
np.zeros_like(lower_bound))))[:, 0]
    upper_bound_rescaled = scaler.inverse_transform(np.hstack((upper_bound,
```

```
np.zeros_like(upper_bound))))[:, 0]

    print("result in future 120s:")
    for i in range(future_steps):
        print(
            f"{i + 1}s - : {mean_predictions_rescaled[i]}, upper and lower: ({lower_bound_rescaled[i]}, {upper_bound_rescaled[i]})")

    plot_results(Y_train=Y_train, train_predictions=train_predictions, Y_test=Y_test, test_predictions=test_predictions)

    t = 0.01
    avg_values = wind_avg_power['30min_avg'].values

    equal_probability,z_prediction, z_test, z1_to_z2_ratio = calculate_probabilities(
    true_values=Y_test,
    predicted_values=test_predictions,
    avg_values=avg_values,
    t=t
    )

    plot_results(Y_train=Y_train, train_predictions=train_predictions, Y_test=Y_test, test_predictions=test_predictions)
if __name__ == "__main__":
    main()
```

3, Data preprocessed

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt

input_txt_path = r"C:\Users\yan\Desktop\supplement_S1.txt"
output_csv_path = r"C:\Users\yan\Desktop\supplement_S1_new.csv"
def process_large_txt_to_csv(input_path, output_path):
    try:
        df = pd.read_csv(input_path, sep=r"\s+", header=0, low_memory=False)
        df.replace(["NA", "", " "], np.nan, inplace=True)
        first_column_name = df.columns[0]
        df[first_column_name] = pd.to_numeric(df[first_column_name], errors='coerce')
        df = df.dropna(subset=[first_column_name])

        df[first_column_name] = pd.to_datetime(df[first_column_name], unit='s', origin='unix')
```

```
            for col in df.columns:
                if col != first_column_name and df[col].dtype.kind in 'biufc':
                    df[col] = df[col].ffill()
                    df[col] = df[col].bfill()
                    df[col] = df[col].interpolate(method='linear', limit_direction='both')

            df.to_csv(output_path, index=False)

            detect_and_visualize_outliers(df)

        except Exception as e:
    def detect_and_visualize_outliers(df):
        try:
            numeric_columns = df.select_dtypes(include=[np.number]).columns.tolist()
            colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'orange', 'purple', 'brown']
            plt.figure(figsize=(12, 8))

            for i, col in enumerate(numeric_columns[1:12]):
                if col != df.columns[0]:
                    Q1 = df[col].quantile(0.25)
                    Q3 = df[col].quantile(0.75)
                    IQR = Q3 - Q1

                    lower_bound = Q1 - 1.5 * IQR
                    upper_bound = Q3 + 1.5 * IQR
                    outliers = df[(df[col] < lower_bound) | (df[col] > upper_bound)]
                    print(f"{col} : {len(outliers)}")
                    plt.boxplot(df[col].dropna(), vert=False, patch_artist=True,
                                boxprops=dict(facecolor=colors[i % len(colors)]),
positions=[i])

            plt.yticks(range(0, len(numeric_columns[1:12])), numeric_columns[1:12])
            plt.title('Boxplot of S1 sensor 1 to 13')
            plt.xlabel('Values')
            plt.grid(True)
            plt.legend(numeric_columns[1:12], loc='upper right')
            plt.show()

        except Exception as e:
            print(f"{e}")

process_large_txt_to_csv(input_txt_path, output_csv_path)
```

3. import matplotlib.pyplot as plt

```
import pandas as pd
import numpy as np

plt.rcParams['font.sans-serif'] = ['SimHei']
plt.rcParams['axes.unicode_minus'] = False

file_path = r"W1_power_test_filled.csv"
power_data = pd.read_csv(file_path)

power_data.columns = power_data.columns.str.replace("", '').str.strip()

power_data = power_data.ffill().bfill()  响
power_data = power_data.interpolate(method='linear', axis=0, limit=5, limit_direction='both')

output_path = r"W1_power_test_filled.csv"
power_data.to_csv(output_path, index=False)


filled_data = pd.read_csv(output_path)
filled_data['Timestamp'] = range(1, len(filled_data) + 1)
filled_data.to_csv(output_path, index=False)


power_data["Timestamp"] = range(1, len(power_data) + 1)


power_data = power_data.replace("NA", np.nan).astype(float)

power_data["TotalPower"] = power_data.iloc[:, 1:].sum(axis=1)

T = 1800
t_threshold = 0.15
r_target = 0.95
reserve_ratio = 0.05
total_capacity = power_data["TotalPower"].max()
reserve_capacity = total_capacity * reserve_ratio

power_data["AvgPower"] = power_data["TotalPower"].rolling(window=30,
min_periods=1).mean()

power_data["Fluctuation"] = abs(power_data["TotalPower"] - power_data["AvgPower"]) /
power_data["AvgPower"]

def scheduling_strategy(data, reserve_capacity, t_threshold):
```

```
    reserve_used = np.zeros(len(data))
    fluctuation_adjusted = data["Fluctuation"].copy()
    total_power_adjusted = data["TotalPower"].copy()

    for t in range(len(data)):
        if data["Fluctuation"].iloc[t] > t_threshold:
            reserve_needed = (data["Fluctuation"].iloc[t] - t_threshold) *
data["AvgPower"].iloc[t]
            reserve_used[t] = min(reserve_capacity, reserve_needed)
            total_power_adjusted.iloc[t] += reserve_used[t]
        elif t > 0 and reserve_used[t - 1] > 0 and data["Fluctuation"].iloc[t] < 0:
            reserve_needed = (t_threshold - data["Fluctuation"].iloc[t]) *
data["AvgPower"].iloc[t]
            reserve_used[t] = min(reserve_capacity, reserve_needed)
            total_power_adjusted.iloc[t] -= reserve_used[t]

        if data["AvgPower"].iloc[t] != 0 and not np.isnan(data["AvgPower"].iloc[t]):
            fluctuation_adjusted.iloc[t] = abs(total_power_adjusted.iloc[t] -
data["AvgPower"].iloc[t]) / \
                                           data["AvgPower"].iloc[t]
        else:
            fluctuation_adjusted.iloc[t] = 0

    return reserve_used, fluctuation_adjusted

power_data["ReserveUsed"], power_data["AdjustedFluctuation"] =
scheduling_strategy(power_data, reserve_capacity, t_threshold)

satisfaction_rate = np.mean(power_data["AdjustedFluctuation"] <= t_threshold)

# Visualization Results
plt.figure(figsize=(12, 6))
plt.plot(power_data["Timestamp"], power_data["Fluctuation"], label="Original Fluctuation
Intensity")
plt.plot(power_data["Timestamp"], power_data["AdjustedFluctuation"], label="Adjusted
Fluctuation Intensity", linestyle="--")
plt.axhline(y=t_threshold, color="red", linestyle="--", label="Fluctuation Intensity Threshold")
plt.legend()
plt.title("Fluctuation Intensity Comparison")
plt.xlabel("Timestamp")
plt.ylabel("Fluctuation Intensity")
plt.show()

plt.figure(figsize=(12, 6))
```

```
plt.plot(power_data["Timestamp"], power_data["ReserveUsed"], label="Reserve Generator Usage")
plt.title("Reserve Generator Dispatch Strategy")
plt.xlabel("Timestamp")
plt.ylabel("Reserve Power")
plt.legend()
plt.show()
```