National College of Ireland

Configuration Manual for Optimizing Energy Price Forecasting with Hybrid Deep Learning

MSc in Data Analytics
Research Practicum

Shashank Rahul Shetye
Student ID: X23278510

School of Computing
National College of Ireland

Supervisor: Muslim Jameel Syed

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Shashank Rahul Shetye
Student ID:	X23278510
Programme:	Msc in Data Analytics **Year:** 2024-25
Module:	Research Practicum
Lecturer:	Muslim Jameel Syed
Submission Due Date:	15-09-2025
Project Title:	Optimizing Energy Price Forecasting with Hybrid Deep Learning
Word Count:	970 **Page Count:** 8

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Shashank Rahul Shetye
Date:	15-09-2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual for Optimizing Energy Price Forecasting with Hybrid Deep Learning

Shashank Rahul Shetye
Student ID: x23278510

1. Introduction

This manual contains the whole setup procedure with all details in a step-wise manner so that this research project can be replicated and executed. The novel idea of the research is the improved energy price prediction using a hybrid deep learning model, i.e., CNN-BiLSTM. In contrast to the CNN-BiLSTM hybrid model, the results were also compared with traditional DL models such as GRU and LSTM.

2. System Hardware Requirements

Ensure your system meets the following minimum requirements:

- **OS:** Windows 10/11, macOS 12+, or Ubuntu 20.04+ (Linux recommended)
- **CPU:** Intel Core i5 / AMD Ryzen 5 or higher (Quad-Core recommended)
- **RAM:** 16 GB Minimum (32 GB recommended)
- **Disk Space:** At least 15–20 GB free for data, dependencies, and experiment outputs

3. Software Requirements:

- **Python:** 3.8+
- **Core libraries:** pandas, numpy, scikit-learn, tensorflow, keras, matplotlib, seaborn, warnings (built-in), statsmodels.
- **Utilities:** StandardScaler, MinMaxScaler, Adam, train_test_split, PCA, mean_squared_error, mean_absolute_error, Conv1D, GRU,LSTM, Bidirectional, Dense,droput, EarlyStopping, ReduceLROnPlateau
- **Development environments:** Jupyter Notebook / VS Code / PyCharm
- **Environment Setup:**

conda create -n x23278510 python=3.9 -y
conda activate x23278510
pip install -U pip
pip install keras matplotlib numpy pandas seaborn sklearn statsmodels tensorflow warnings

4. Dataset Details

Kaggle Data Source: https://www.kaggle.com/datasets/nicholasjhana/energy-consumption-generation-prices-and-weather

4.1 Dataset Description: The dataset contains 4 years of electrical consumption/generation, prices, and weather data for Spain. Consumption/generation data was taken from ENTSOE, a public portal for Transmission Service Operator (TSO) data. Settlement prices were gathered from the Spanish TSO: Red Electric España. Weather data is provided by the Open Weather API for the five largest cities in Spain.

4.2 Dataset Justification: The "Energy Consumption, Generation, Prices and Weather" dataset featured on Kaggle was chosen for its richness of multidimensional aspects related to the energy ecosystem. Such a dataset is one that fits studies aiming at integrating consumption, supply, pricing, and external environmental factors. This is a rare dataset where these three dimensions come together, as it contains hourly records of electricity consumption, generation mix, wholesale prices, and weather factors like temperature. A multivariate structure of this kind is paramount in the construction of models that can explore various complex and nonlinear dependencies that affect price fluctuations. With hourly granularity, opportunities arise to analyze short-term volatility and peak event periods-all of which are critical in forecasting models. The data also should cover a sufficiently long time, strongly representing both seasonal and cyclical trends. The public availability of the dataset and its use in other studies lend credibility to it and allow for benchmarking, which means that the findings are more relevant for both academia and practitioners. Since the dataset provides operational as well as environmental factors affecting price behavior, it is closely aligned to the overall research aim-the optimization of energy price forecasting with hybrid deep learning-so that the models can be trained on a holistic set of real and high-quality data.

4.3 Files

The jupyter notebook uses below files:

- energy_dataset.csv
- weather_features.csv

After cleaning and filtering the dataset, weather_features.csv has 35064 rows and 10 columns were used as shown in fig 4.2, whereas energy_dataset.csv has 35064 rows and 18 columns as shown in fig4.1

```
Data columns (total 18 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   time                                         35064 non-null  object
 1   generation biomass                           35064 non-null  float64
 2   generation fossil brown coal/lignite         35064 non-null  float64
 3   generation fossil gas                        35064 non-null  float64
 4   generation fossil hard coal                  35064 non-null  float64
 5   generation fossil oil                        35064 non-null  float64
 6   generation hydro pumped storage consumption  35064 non-null  float64
 7   generation hydro run-of-river and poundage   35064 non-null  float64
 8   generation hydro water reservoir             35064 non-null  float64
 9   generation nuclear                           35064 non-null  float64
 10  generation other                             35064 non-null  float64
 11  generation other renewable                   35064 non-null  float64
 12  generation solar                             35064 non-null  float64
 13  generation waste                             35064 non-null  float64
 14  generation wind onshore                      35064 non-null  float64
 15  total load actual                            35064 non-null  float64
 16  price day ahead                              35064 non-null  float64
 17  price actual                                 35064 non-null  float64
```

Fig 4.1. energy_dataset Columns

```
Data columns (total 10 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   dt_iso      35064 non-null  object
 1   temp        35064 non-null  float64
 2   temp_min    35064 non-null  float64
 3   temp_max    35064 non-null  float64
 4   pressure    35064 non-null  int64
 5   humidity    35064 non-null  int64
 6   wind_speed  35064 non-null  int64
 7   wind_deg    35064 non-null  int64
 8   rain_1h     35064 non-null  float64
 9   clouds_all  35064 non-null  int64
```

Fig 4.2. weather_features.csv Columns

These dataset are merged into file name as merged_df.csv.
After performing feature engineering on merged_df.csv dataset, following columns are generated:

- hour
- day_of_week
- month
- quarter
- is_weekend
- monthly_bill
- prev_month_bill

5. Model Configuration

5.1 Baseline Models

Before fitting the dataset into model, the dataset is **scales**, **slices into overlapping sequences.** The dataset was divided into 80% train data and 20 percent test data using code in fig 5.1

```
feature_columns = [col for col in self.data.columns
                   if col not in ['time', self.target_column]]

features = self.data[feature_columns].values
target = self.data[self.target_column].values
available_cols = [col for col in feature_columns if col in self.data.columns]
# Scaling features
feature_scaler = MinMaxScaler()
target_scaler = MinMaxScaler()

scaled_features = feature_scaler.fit_transform(features)
scaled_target = target_scaler.fit_transform(target.reshape(-1, 1)).ravel()
self.scalers['features'] = feature_scaler
self.scalers['target'] = target_scaler

X, y = [], []
for i in range(self.sequence_length, len(scaled_features)):
    X.append(scaled_features[i-self.sequence_length:i])
    y.append(scaled_target[i])

X = np.array(X)
y = np.array(y)

train_size = int(len(X) * 0.8)
```

Fig 5.1 Dataset Split

Default algorithms and hyperparameters:
LSTM Model:

```
def build_lstm_model(self, input_shape):
    model = Sequential([
        LSTM(50, return_sequences=True, input_shape=input_shape),
        Dropout(0.2),
        LSTM(50, return_sequences=False),
        Dropout(0.2),
        Dense(25, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.3),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='mse',
                  metrics=['mae'])

    return model
```

Fig 5.2. LSTM Model

GRU Model:

```
def build_gru_model(self, input_shape):
    model = Sequential([
        GRU(50, return_sequences=True, input_shape=input_shape),
        Dropout(0.2),
        GRU(50, return_sequences=False),
        Dropout(0.2),
        Dense(25, activation='relu', kernel_regularizer=l2(0.001)),
        Dropout(0.3),
        Dense(1)
    ])
    model.compile(optimizer=Adam(learning_rate=0.001),
                  loss='mse',
                  metrics=['mae'])
    return model
```

Fig 5.3. GRU Model

After these models are build, they are compiled as below:

- **Adam optimizer** (learning rate 0.001) for adaptive gradient updates.
- **MSE loss** for measuring prediction error.
- **MAE metric** for easier-to-interpret error measurement.

The models are trained with batch size of 64 and upto to 100 epochs. Callbacks are also used to improve the training efficiency : EarlyStopping and ReduceLROnPlateau.

```
callbacks = [
    EarlyStopping(patience=10, restore_best_weights=True),
    ReduceLROnPlateau(patience=5, factor=0.5)
]
history = model.fit(
    X_train, y_train,
    batch_size=64,
    epochs=100,
    validation_data=(X_test, y_test),
    callbacks=callbacks,
    verbose=1
)

# Make predictions
y_pred = model.predict(X_test)
y_test_orig = self.scalers['target'].inverse_transform(y_test.reshape(-1, 1)).ravel()
y_pred_orig = self.scalers['target'].inverse_transform(y_pred.reshape(-1, 1)).ravel()

mse = mean_squared_error(y_test_orig, y_pred_orig)
mae = mean_absolute_error(y_test_orig, y_pred_orig)
rmse = np.sqrt(mse)
```

Fig 5.4 Prediction Code

5.2 Hybird Model (CNN-BiLSTM)

The dataset was divided into 80% train data and 20 percent test data.Before modelling, features and targets are preprocessed, then reducing feature dimensionality using PCA with n_components=0.95 (i.e. keeps enough principal components to preserve 95% of the variance in the data.) and random state as 42 and scales the ta`rget variable so that both inputs and outputs are normalized for model training.

```
feature_columns = [col for col in df.columns
                   if col not in ['time', target_column]]

features = df[feature_columns].values

print(f"Number of features before PCA: {features.shape[1]}")

scaler = MinMaxScaler()
features_scaled = scaler.fit_transform(features)

# Apply PCA for dimensionality reduction
# Keep 95% of variance
pca = PCA(n_components=0.95, random_state=42)
features_pca = pca.fit_transform(features_scaled)

print(f"Number of features after PCA: {features_pca.shape[1]}")
print(f"Explained variance ratio: {pca.explained_variance_ratio_.sum():.4f}")

# Scale target variable
target_scaler = MinMaxScaler()
target_scaled = target_scaler.fit_transform(target.reshape(-1, 1)).ravel()
```

Fig 5.5. PCA and MinMaxScalar

CNN-BiLSTM model:

```
model = Sequential([
    Conv1D(filters=32, kernel_size=3, activation='relu', padding='same',
           kernel_regularizer=l2(0.001), input_shape=(X_train.shape[1], X_train.shape[2])),
    MaxPooling1D(pool_size=2),
    Bidirectional(LSTM(32, return_sequences=False)),
    Dropout(0.3),
    Dense(64, activation='relu', kernel_regularizer=l2(0.001)),
    Dropout(0.5),
    Dense(1)
])

model.compile(optimizer=Adam(learning_rate=0.001), loss='mse', metrics=['mae'])

early_stop = EarlyStopping(monitor='val_loss', patience=2, restore_best_weights=True)
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.5, patience=3)

history = model.fit(
    X_train, y_train,
    epochs=100,
    batch_size=16,
    validation_data=(X_val, y_val),
    callbacks=[early_stop, reduce_lr]
)
```

Fig5.6 CNN-BiLSTM model

As per fig 5.6, CNN-BiLSTM Model architecture:

- **Conv1D layer**: Extracts local patterns from sequences with:32 filters, kernel size = 3, ReLU activation, padding = 'same' and L2 regularization (0.001)
- **MaxPooling1D**: Reduces sequence length by pooling over windows of size 2.
- **Bidirectional LSTM:** Processes the sequence in both forward and backward directions with 32 units, outputting the final step only.
- **Dropout (0.3)**: Reduces overfitting.
- **Dense layer**: 64 neurons, ReLU activation, L2 regularization (0.001).
- **Dropout (0.5)**: Further regularization.
- **Output Dense layer**: 1 neuron (regression output).

```
# Calculate metrics
y_pred = model.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(mse)
model_scores['CNN_BiLSTM'] = {'RMSE': rmse, 'MAE': mae}
# Inverse transform for actual metrics
y_test_orig = target_scaler.inverse_transform(y_test.reshape(-1, 1)).ravel()
y_pred_orig = target_scaler.inverse_transform(y_pred.reshape(-1, 1)).ravel()

mse_orig = mean_squared_error(y_test_orig, y_pred_orig)
mae_orig = mean_absolute_error(y_test_orig, y_pred_orig)
rmse_orig = np.sqrt(mse_orig)
```

Fig 5.7 Calculation Metrics for CNN-BiLSTM Model

Then evaluating a trained CNN-BiLSTM model's performance, both on scaled data and on the original unscaled data.

Metrics calculated:

- **MAE** (Mean Absolute Error)
- **RMSE** (Root Mean Squared Error = root of MSE)

6. Evaluation and Results

6.1 Evaluation Metrics (for all models):

- **RMSE**
- **MEA**

6.2 Final Results:

Models	RMSE	MEA
LSTM	5.99	4.47
GRU	5.75	4.27
CNN-BiLstm	5.18	4.10

Fig 6.1 Model Results

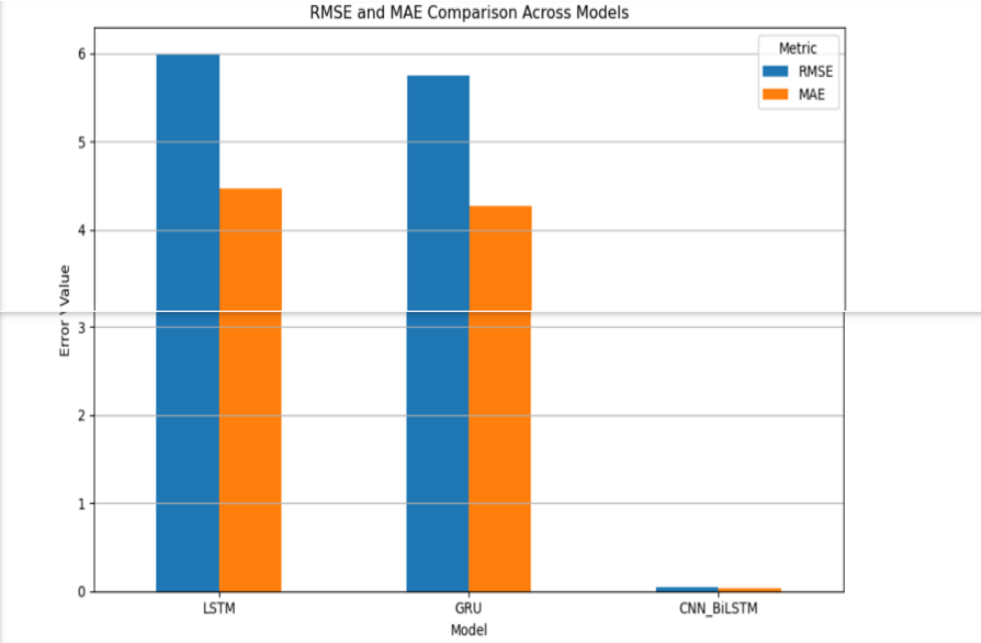

Fig 6.2 Model Comparison Graph

7. Conclusion

This manual walks you through the complete setup and configuration needed to replicate the research project *Optimizing Energy Price Forecasting with Hybrid Deep Learning*. The study uses both traditional deep learning and hybrid architectures, focusing on a CNN-BiLSTM model to improve energy price forecasting. Users are encouraged to experiment with parameter tuning, alternative feature combinations, and extended architectures to further boost performance. For troubleshooting, check error logs and refer to the documentation of the relevant libraries.

References

Python Official Docs: https://www.python.org/doc/

TensorFlow: https://www.tensorflow.org/learn

Scikit-learn: https://scikit-learn.org/

Keras: https://keras.io/getting_started/

Seaborn: https://seaborn.pydata.org/

Matplotlib: https://matplotlib.org/

Dataset:https://www.kaggle.com/datasets/nicholasjhana/energy-consumption-generation-prices-and-weather