National College of Ireland

Configuration Manual for Solar-Net: Leveraging Transformers for Enhanced Solar Power Prediction

MSc Research Project
MSc in Data Analytics

AKIF ROSHAN SHAIK
Student ID: 23204231

School of Computing
National College of Ireland

Supervisor: Vladimir Milosavljevic

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name: ……. ……Akif Roshan Shaik……

Student ID: ……23204231……..……

Programme: ……MSc in Data Analytics…… **Year:** …2024-2025…..

Module: ……MSc Research Project…….……

Lecturer: ……Vladimir Milosavljevic…….……

Submission Due Date: ……29 January 2025…….……

Project Title: ……solar -net : Leveraging Transformers for Enhanced Solar Power Prediction……

Word Count: ……917…… **Page Count:** ……10……..…….……

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: ……Akif Roshan Shaik……

Date: …… …29 January 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 Solar-Net: Leveraging Transformers for Enhanced Solar Power Prediction

Akif Roshan Shaik
Student ID: 23204231

1. Introduction

This manual provides a comprehensive guide to configuring the Solar-Net research framework for enhanced solar power prediction using machine learning (ML) and deep learning (DL) models, with a focus on leveraging Transformers. The steps outlined here include library imports, data preparation, preprocessing, model building, and evaluation. The models used in this research include Support Vector Regression (SVR), Random Forest Regression (RF), Gradient Boosting Machine (GBM), LSTM, and a Transformer-based model called Solar-Net.

2. System Specification

Before running the models, ensure your system meets the following requirements:

- Operating System: Windows, Linux, or macOS
- Python Version: 3.8 or higher
- RAM: Minimum of 8GB
- Processor: Intel i5 or equivalent (recommended for faster computation)
- Disk Space: At least 10GB of free disk space
- These specifications will ensure that the system can handle the computational demands of training and deploying the deep learning model, especially when using large datasets and complex neural network architectures.

3. Required Libraries and Dependencies:

To run the experiments, you must install the necessary Python libraries. You can install them using pip by running the following command in your terminal or command prompt:

- NumPy: For numerical computing.
- Pandas: For data manipulation.
- Scikit-Learn: For standard machine learning algorithms.
- TensorFlow/Keras : For LSTM and deep learning models.
- Matplotlib and Seaborn: For visualizing results.
- Scikit-Optimize: For hyperparameter tuning.
- Plotly: For Visualization results
- Scikeras: for integrating Keras with scikit-learn SciPy: for scientific calculations (e.g., z-scores)

4. Execution of the Code Implementation

Step 1: Import Libraries
Begin by importing the necessary libraries for data analysis, preprocessing, model training, and evaluation.

```
# Import necessary libraries
import numpy as np
import pandas as pd
import seaborn as sns
import plotly.express as px
import matplotlib.pyplot as plt
import plotly.graph_objects as go
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor, GradientBoostingRegressor
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.models import Model,Sequential
from tensorflow.keras.layers import Input, LSTM, Dense, Dropout, LayerNormalization, MultiHeadAttention,
from sklearn.preprocessing import MinMaxScaler
from keras_tuner import HyperModel, RandomSearch
from scikeras.wrappers import KerasRegressor
from scipy import stats

# Set random seed for reproducibility
np.random.seed(42)
tf.random.set_seed(42)

import warnings
warnings.filterwarnings('ignore')
```

Step 2: Load and Explore the Data
Download and load the Solar Power Generation Dataset from Kaggle. The dataset is available at:

https://www.kaggle.com/datasets/stucom/solar-energy-power-generation-dataset

```
# Load the dataset
data = pd.read_csv('solar_power_data.csv')
```

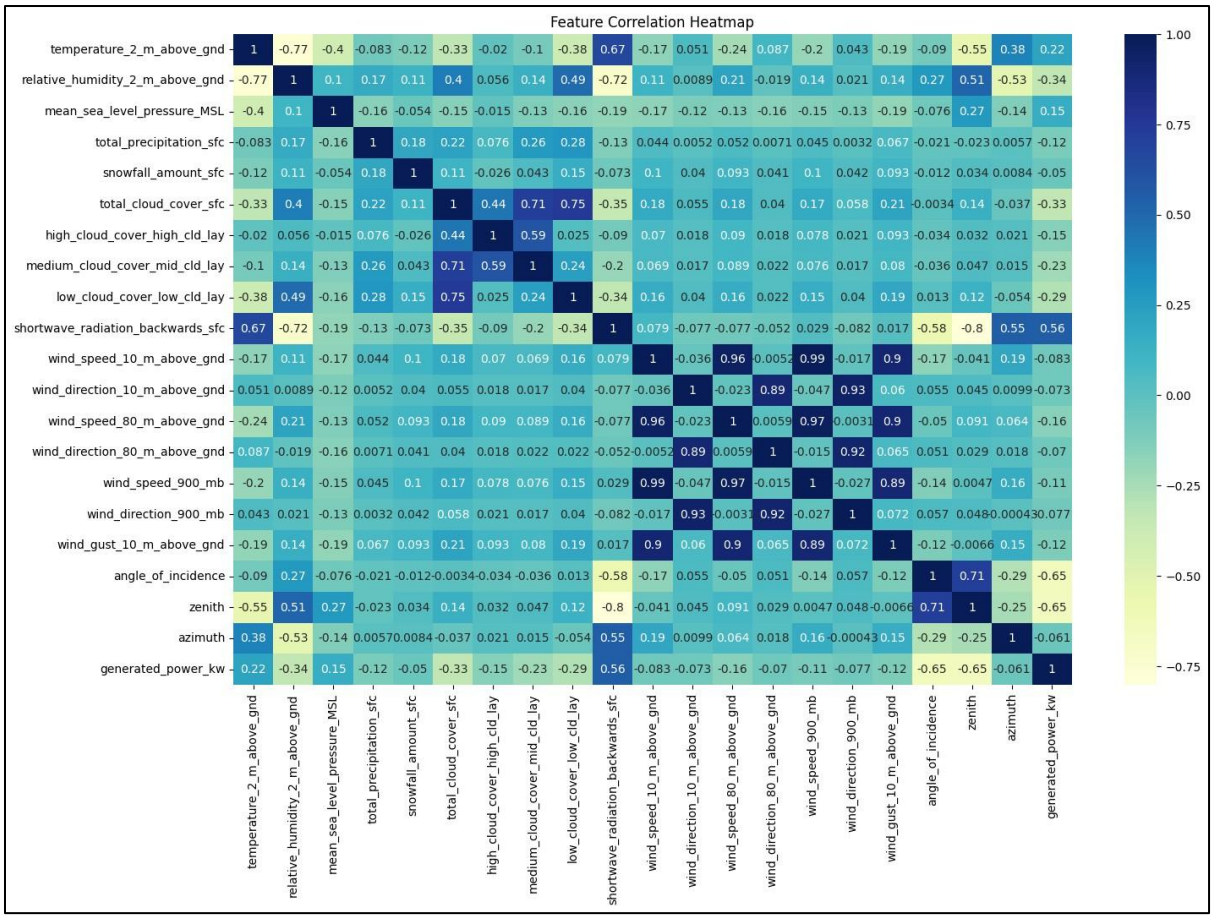
Python

```
# Display the First Few Attributes
data.head(10)
```

Step 3: Data Preprocessing & Exploratory Data Analysis (EDA)
Remove outliers using the Z-score method.

- Visualize relationships between the features and target variable (generated power).
- Perform correlation analysis and generate heatmaps to understand feature dependencies.

```python
# Check for missing values
if data.isnull().sum().any():
    data = data.fillna(data.mean())  # Simple imputation for missing values
```




```python
# Detect and remove outliers using Z-score
z_scores = np.abs(stats.zscore(data.select_dtypes(include=[np.number])))
filtered_entries = (z_scores < 3).all(axis=1)
data = data[filtered_entries]
```

Step 4: Feature Engineering

- Feature Scaling: Normalize the features using StandardScaler to prepare the data for machine learning algorithms.
- Train-Test Split: Divide the dataset into training and testing sets for model evaluation.

```
# Standardize the data
scaler=StandardScaler()
for col in data.select_dtypes(include=np.number).columns:
    data[col]=scaler.fit_transform(data[[col]])
```

```
# Separate features and target
X = data.drop('generated_power_kw', axis=1)
y = data['generated_power_kw']
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Step 5: Machine Learning Model Training

5.1 Support Vector Regression (SVR)

- Use grid search to tune the hyperparameters of the Support Vector Regression model.
- Train and evaluate the SVR model using cross-validation.

```
# SVR model with hyperparameter tuning
svr = SVR(max_iter=280)
param_grid_svr = {'kernel': ['linear', 'rbf', 'poly'], 'C': [0.1, 1, 10], 'epsilon': [0.01, 0.1, 1]}
grid_search_svr = GridSearchCV(svr, param_grid_svr, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_svr.fit(X_train, y_train)

# Best SVR model
best_svr = grid_search_svr.best_estimator_
y_pred_svr = best_svr.predict(X_test)

# Create a DataFrame to store the metrics
metrics_df = pd.DataFrame(columns=["Model", "R2 Score (%)", "Mean Squared Error", "Mean Absolute Error"])

#Evalute the Support Vector Machine Model
r2_score_svr = round(r2_score(y_test,y_pred_svr) * 100, 2)
mean_sq_svr = mean_squared_error(y_test,y_pred_svr)
mean_ab_svr = mean_absolute_error(y_test,y_pred_svr)
metrics_df.loc[len(metrics_df)] = ["Support Vector Machine", r2_score_svr, mean_sq_svr, mean_ab_svr]
print("R2 Score for Support Vector Machine Model: ",r2_score_svr,"%")
print("Mean Square Error of Support Vector Machine Model: ",mean_sq_svr)
print("Mean Absolute Error of Support Vector Machine Model: ",mean_ab_svr)
```

5.2 Random Forest Regression

- Similar to SVR, use grid search to tune the Random Forest model and evaluate it.

```
# Random Forest model with hyperparameter tuning
rf = RandomForestRegressor(max_depth = 5,random_state=42)
param_grid_rf = {'n_estimators': [50, 100, 200], 'min_samples_split': [2, 5, 10]}
grid_search_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_rf.fit(X_train, y_train)

# Best Random Forest model
best_rf = grid_search_rf.best_estimator_
y_pred_rf = best_rf.predict(X_test)
```

5.3 Gradient Boosting Machine (GBM)

- Use grid search to tune the Gradient Boosting model and evaluate it.

```
# Gradient Boosting model with hyperparameter tuning
gbm = GradientBoostingRegressor(n_estimators = 8,random_state=42)
param_grid_gbm = {'learning_rate': [0.01, 0.1, 0.2], 'max_depth': [3, 5, 7]}
grid_search_gbm = GridSearchCV(gbm, param_grid_gbm, cv=5, scoring='neg_mean_squared_error', n_jobs=-1)
grid_search_gbm.fit(X_train, y_train)

# Best GBM model
best_gbm = grid_search_gbm.best_estimator_
y_pred_gbm = best_gbm.predict(X_test)
```

Step 6: Deep Learning Models

6.1 Long Short-Term Memory (LSTM)

- Reshape data for time series modeling.
- Build an LSTM model using Sequential API from Keras.

```
# Build the LSTM model
model = Sequential()  # Initialize a sequential model
# Add the first LSTM layer with 100 units and return sequences for the next LSTM layer
model.add(LSTM(units=50, return_sequences=True, input_shape=(x_train_lstm.shape[1], x_train_lstm.shape[2])))
# Add the second LSTM layer without returning sequences
model.add(LSTM(units=50))
# Add a Dropout layer to reduce overfitting
model.add(Dropout(rate=0.2))
# Add a Dense layer with a single output unit for regression
model.add(Dense(units=1))

# Compile the model with Adam optimizer and mean squared error loss function
model.compile(optimizer='adam', loss='mean_squared_error')
```

Python

```
# Train the model on the training data
model.fit(x_train_lstm, y_train_scaled, epochs=50, batch_size=128)
```

Python

6.2 Transformer Model (Solar-Net)

- The Transformer model uses attention mechanisms to improve the prediction of solar power generation.
- Implement the Solar-Net model using Keras Tuner for hyperparameter optimization.

```
# Reshape data for transformer
timesteps = 1
x_train_trans = X_train.values.reshape((X_train.shape[0], timesteps, X_train.shape[1]))
x_test_trans = X_test.values.reshape((X_test.shape[0], timesteps, X_test.shape[1]))

# Scale the target variable
scaler_y = MinMaxScaler()
y_train_scaled = scaler_y.fit_transform(y_train.values.reshape(-1, 1))
y_test_scaled = scaler_y.transform(y_test.values.reshape(-1, 1))
```

```python
# Define the Transformer Model (Solar-Net) HyperModel
class TransformerHyperModel(HyperModel):
    def build(self, hp):
        inputs = Input(shape=(x_train_trans.shape[1], x_train_trans.shape[2]))

        # Hyperparameters
        num_heads = hp.Int('num_heads', min_value=2, max_value=8, step=2)
        ff_dim = hp.Int('ff_dim', min_value=16, max_value=128, step=16)
        dropout_rate = hp.Float('dropout_rate', 0.1, 0.5, step=0.1)

        # Build transformer block
        x = self.transformer_block(inputs, num_heads, ff_dim, dropout_rate)
        x = GlobalAveragePooling1D()(x)
        x = Dropout(0.3)(x)
        outputs = Dense(1)(x)

        model = Model(inputs, outputs)
        model.compile(optimizer='adam', loss='mean_squared_error', metrics=['mae'])
        return model

    def transformer_block(self, inputs, num_heads, ff_dim, dropout):
        # Multi-head attention layer
        attention_output = MultiHeadAttention(num_heads=num_heads, key_dim=inputs.shape[-1])(inputs, inpu
        attention_output = Dropout(dropout)(attention_output)
        out1 = LayerNormalization(epsilon=1e-6)(inputs + attention_output)

        # Feed-forward network
        ffn_output = Dense(ff_dim, activation='relu')(out1)
        ffn_output = Dropout(dropout)(ffn_output)

        # Ensure dimensions match for addition
        if out1.shape[-1] != ffn_output.shape[-1]:
            ffn_output = Dense(out1.shape[-1])(ffn_output)

        return LayerNormalization(epsilon=1e-6)(out1 + ffn_output)
```

```python
# Start the tuning process
tuner.search(x_train_split, y_train_split, epochs=100, validation_data=(x_val_split, y_val_split))

# Get the best model and hyperparameters
best_model = tuner.get_best_models(num_models=1)[0]
best_hyperparameters = tuner.get_best_hyperparameters(num_trials=1)[0]

# Train the best model on the full training data
best_model.fit(x_train_trans, y_train_scaled, epochs=100, batch_size=32)
```

Step 7: Comparison of Machine Learning Models

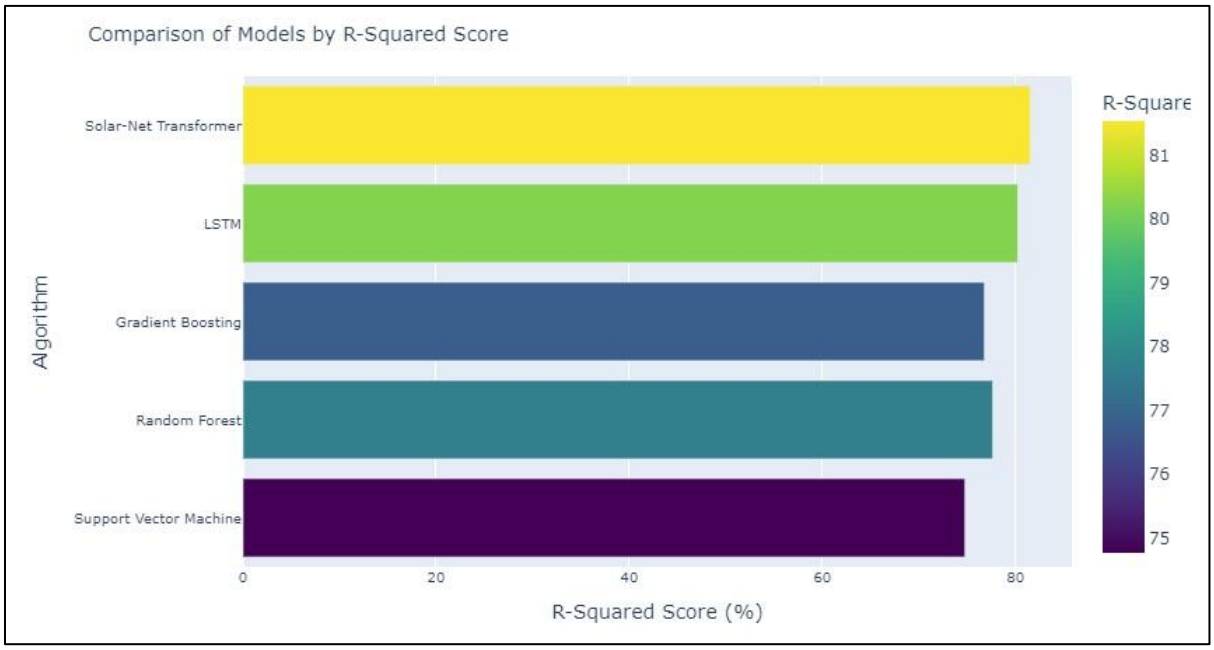
```python
# Display the metrics DataFrame
metrics_df
```

	Model	R2 Score (%)	Mean Squared Error	Mean Absolute Error
0	Support Vector Machine	74.77	0.260527	0.406757
1	Random Forest	77.65	0.230744	0.326031
2	Gradient Boosting	76.79	0.239631	0.364436
3	LSTM	80.24	0.204028	0.312378
4	Solar-Net Transformer	81.54	0.190613	0.289888

```python
# Create a horizontal bar chart
fig = px.bar(metrics_df,
             x='R2 Score (%)',
             y='Model',
             orientation='h',
             title='Comparison of Models by R-Squared Score',
             labels={'R2 Score (%)': 'R-Squared Score (%)', 'Model': 'Algorithm'},
             color='R2 Score (%)',  # Color bars by R2 Score
             color_continuous_scale='Viridis')  # Use a nice color scale

# Update layout for better aesthetics
fig.update_layout(
    xaxis_title='R-Squared Score (%)',
    yaxis_title='Algorithm',
    title_font_size=14,
    xaxis_tickfont_size=10,
    yaxis_tickfont_size=10,
    margin=dict(l=50, r=50, t=50, b=50)  # Adjust margins
)

# Show the plot
fig.show()
```



Running the Experiments

- Data Preparation: Preprocess the dataset, handle missing values, normalize the data, and split it into training and testing sets.
- Model Training: Train each model (SVM, Random Forest, Gradient Boosting, LSTM, Solar-Net Transformer) using the specified configurations.
- Model Evaluation: After training, evaluate each model's performance using the metrics mentioned above.
- Comparative Analysis: Analyze the results (R^2, MSE, MAE) across different dataset split ratios (70:30, 75:25, 80:20).

This manual has outlined the necessary steps to configure and run the machine learning & deep learning models for solar energy prediction. By following this guide, users can replicate the research and experiment with different configurations to further optimize the models.

References

Python: https://www.python.org
Dataset Source: https://www.kaggle.com/datasets/stucom/solar-energy-power-generationdataset