SOLAR CHEAT SCAN: DETECTION OF TAMPERING OF SOLAR FORECASTS USING ML

A Capstone Project report submitted
in partial fulfilment of requirement for the award of degree

BACHELOR OF TECHNOLOGY

in

SCHOOL OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

by

RANGI VANI PRIYA	**(2203A51441)**
MOOLA HARSHITHA	**(2203A51430)**
DEVIREDDY VINISHA	**(2203A51200)**
ANNAM VARSHINI	**(2203A51830)**
MOHAMMAD SAMREEN	**(2203A51829)**

Under the guidance of

Dr V. Venkataramana
Professor, CS&AI.

SR UNIVERSITY

SR University, Ananthasagar, Warangal, Telagnana-506371 sru SR UNIVERSITY

CERTIFICATE

This is to certify that this project entitled **“SOLAR CHEAT SCAN: AI BASED DETECTION OF TAMPERING OF SOLAR FORECASTS”** is the bonafied work carried out by **RANGI VANI PRIYA, MOOLA HARSHITHA, DEVIREDDY VINISHA REDDY, ANNAM VARSHINI , SAMREEN** as a Capstone Project for the partial fulfilment to award the degree BACHELOR OF TECHNOLOGY in School of Computer Science and Artificial Intelligence during the academic year 2024-2025 under our guidance and Supervision.

Dr. V. Venkataramana
Professor
School of CS&AI,
SR University
Ananthasagar, Warangal

Dr. M. Sheshikala
Professor & Head,
School of CS&AI,
SR University
Ananthasagar,Warangal

Reviewer-1
Name:
Designation: Assistant professor
Signature:

Reviewer-2
Name:
Designation: Assistant professor
Signature:

ACKNOWLEDGEMENT

We owe an enormous debt of gratitude to our Capstone project guide **Dr.V.Venkatramana sir, Asst.Prof** as well as Head of the School of CS&AI , **Dr. M.Sheshikala, Professor** for guiding us from the beginning through the end of the Capstone Project with their intellectual advices and insightful suggestions. We truly value their consistent feedback on our progress, which was always constructive and encouraging and ultimately drove us to the right direction.

We express our thanks to project co-ordinators **Mr. Sallauddin Md, Asst. Prof.** for their encouragement and support.

Finally, we express our thanks to all the teaching and non-teaching staff of the department for their suggestions and timely support.

ABSTRACT

Accurate solar power generation forecasting is now crucial for system balance and effective power management due to the growing dependence on solar energy. However, forecast statistics alteration—whether intentional or unintended—can pose critical operational, financial, and reliability issues for sun farms. In this extensive project, "**Solar Cheat Scan: Detection of Tampering of Solar Forecasts Using ML**" is a processs that employs gadget studying and Machine Learning strategies to perceive anomalies or manipulations in sun forecast facts is offered. Predictive fashions like Random Forest Regressor, Gradient Boost Regressor, XGBoost Regressor are skilled the usage of historical weather and solar electricity data to estimate the expected AC electricity output. To find any strange deviations that might point to facts inconsistencies or tampering, these predicted values are then compared with the actual measured electricity. Metrics like MAE,RMSE,R2 score for ac power forecasting and Accuracy,Recall,Precision,F1-Score for detecting tampering are used to evaluate model performance with the intention to assure accuracy and dependability. The outcomes of the experiments show that the Proposed model is capable of efficaciously detecting viable forecast tampering, with model appearing the pleasant. A extra stable and sustainable renewable power destiny is made viable through this mission, which sooner or later will increase transparency, facilitates actual-time tracking, and boosts the self belief and dependability of sun power structures.

Keywords: Forecast Tampering Detection, Data Integrity, Weather Data Analysis, Random Forest,Stacking Model,Real-Time Monitoring

ABOUT THE ORGANIZATION OR COMPANY

This project was carried out at SR University, Ananthasagar, Warangal, Telangana, under the School of Computer Science and Artificial Intelligence. SR University is known for its strong focus on innovation, research, and technology-driven education. The institution provides excellent facilities and guidance for students to develop practical skills and work on real-world projects.

The supportive learning environment and mentorship from faculty members helped our team successfully design and develop the Solar Cheat Scan Guidance and Recommendation System as part of the academic curriculum.

TABLE OF CONTENTS

INTRODUCTION

Solar electricity is a main renewable supply because of its sustainability and cost-effectiveness. Accurate sun energy forecasting is vital for grid balance, strength management, and green operation of solar farms. Forecasts help expect power era based on climate records, enabling better storage and distribution making plans. The integration of renewable energy sources, especially solar power, into modern energy systems is a key step toward achieving global sustainability goals. Predicting solar power generation accurately is essential for effective grid management, balancing demand and supply, and optimizing costs. However, the increasing reliance on data-driven forecasting systems creates a new risk: data tampering or manipulation. Tampered or fake solar generation data can mislead decision-making, disrupt power management strategies, and lead to financial and operational inefficiencies across the energy value chain.

However, any tampering or errors in forecast statistics can be a reason to operational disasters, monetary losses, and decreased agree with in sun systems. The project "Solar Cheat Scan:Detection of Tampering of Solar Forecasts Using ML" addresses this venture using Data and ML to discover anomalies or manipulations in solar forecast records. Models like Random Forest and Stacking model which combines **Random Forest, Gradient Boosting,** and **XGBoost** as base learners, with a **Random Forest Regressor** serving as the meta-learner for final prediction are used to evaluate anticipated and real power outputs. This machine guarantees facts integrity, transparency, and reliability, ultimately assisting steady and sustainable sun electricity management.

In addition to prediction accuracy, the system focuses on explainability and transparency using explainable Evaluation Metrices. This allows users and energy analysts to understand the reasoning behind each prediction. Such clarity is essential for building trust in AI systems used in critical infrastructure, like energy forecasting and monitoring. The main goal of this project is to create a reliable, understandable, and automated detection framework that protects the integrity of solar power data. The findings from this study could strengthen operational resilience, promote data authenticity, and support the secure digital transformation of the renewable energy sector.

RELATED WORK

Research in solar power forecast tampering detection focuses on ensuring data integrity, accuracy, and reliability in energy forecasting systems. Early studies mainly improved forecast precision using statistical and ML models like ANN and Random Forest, without addressing data manipulation risks. Recent approaches employ deep learning and anomaly detection models such as CNN-LSTM and AE-LSTM to identify irregularities or cyberattacks in solar data. However, most existing methods lack real-time scalability and transparency.

Tang, Mao, and Nelms (2023) explored the vulnerability of solar power forecasting models to adversarial tampering. Their work focused on understanding how small, intentionally crafted perturbations could manipulate the output of machine learning–based forecasting systems. Using Deep Neural Networks (DNN) and regression techniques, they demonstrated that forecasting models are highly susceptible to adversarial noise, which can significantly distort prediction accuracy. This study emphasized the importance of developing secure and tamper-resistant forecasting systems. However, it primarily examined adversarial attacks in controlled environments and did not address real-time operational tampering detection or practical deployment scenarios.

Sun, He, and Zhang (2022) introduced a deep learning–based anomaly detection framework designed to identify cyberattack-induced irregularities in solar forecasting data. Their proposed approach used a hybrid CNN–LSTM model combined with probabilistic forecasting to detect unusual trends and deviations caused by malicious interference. The study demonstrated high detection accuracy and improved robustness in maintaining data integrity during cyber threats. Despite its strengths, the framework required heavy computational resources, making it less suitable for real-time applications or deployment across large solar farms where scalability is critical.

Ibrahim, Alsheikh, and Awaysheh (2022) conducted a comprehensive comparison of various machine learning algorithms for anomaly detection in solar photovoltaic (PV) time-series data. Techniques such as Autoencoder-LSTM (AE-LSTM), Isolation Forest, and Prophet were evaluated based on their performance in identifying operational anomalies. The results showed that the AE-LSTM model achieved superior accuracy and contributed to enhancing the reliability of solar PV systems. However, the study focused solely on general anomaly detection and did not consider intentional tampering or direct verification between predicted and actual power values, leaving a gap in addressing forecast manipulation.

PROBLEM STATEMENT

As renewable energy becomes more vital to our future, solar power has taken center stage as one of the most reliable and sustainable energy sources. The success of solar systems depends heavily on the accuracy and integrity of data collected and used for forecasting power generation. These forecasts help in planning grid operations, balancing supply and demand, scheduling maintenance, and making financial decisions.However, in real-world applications, solar generation data is not always trustworthy. It can be affected by sensor faults, communication errors, or even deliberate tampering. When this data is manipulated, it can lead to wrong forecasts, operational inefficiencies, and financial losses. More importantly, it can erode trust in automated systems that depend on this data.

Detecting such tampered or abnormal data is not easy. Traditional rule-based or statistical methods often fail because they cannot analyze the complex, nonlinear patterns present in solar power generation data. Moreover, many existing models act like black boxes offering predictions without clear explanations of how those results were reached. To tackle these issues, this research introduces SolarCheatScan, an intelligent machine learning framework designed to identify and explain tampered data in solar power forecasts. The system uses a stacking ensemble model that combines the strengths of multiple algorithms—Random Forest, Gradient Boosting, and XGBoost—with a Random Forest meta-model to improve accuracy and robustness. This approach makes solar energy monitoring more transparent, trustworthy, and effective in maintaining the integrity of renewable energy data.

OBJECTIVES

The objectives of the Solar Cheat Scan: Detection of Tampering of Solar Forecasts Using ML are as follows:

1. To collect and carefully preprocess real and forecasted AC power data so that the machine learning model receive clean and meaningful inputs.
2. To split the dataset into training and testing phases for accurate model evaluation.
3. To design a machine learning–based forecasting model using Random Forest and expert analytical techniques.
4. To predict AC power generation for selected time intervals using trained ML models.
5. To compare forecasted AC power values with real AC power measurements to identify deviations.
6. To detect tampered or anomalous data points by observing unusual difference between predicted and actual values.
7. To apply evaluation metrics such as MAE, RMSE, and R^2 to measure the accuracy and reliability of the forecasting model.

8. To visualize actual vs. predicted AC power using graphs for better interpretation and detection of anomalies.
9. To filter and display the dataset at 15-minute intervals to support clear anomaly detection and analysis.
10. To identify potential tampering events by analyzing abnormal variations in real vs. forecasted AC power values.
11. To improve the forecasting accuracy by comparing multiple machine learning models in the future
12. To develop a system that can be extended for real-time dashboard visualization and automated tampering alerts.

SCOPE OF THE PROJECT

The scope of this research includes the development, evaluation, and interpretation of a machine learning–based framework for detecting tampered solar power generation data. The project covers the following major areas of work:

1. **Data Collection and Understanding:**
   Working with forecasted and actual solar generation datasets to identify key features such as *AC Power* and relevant environmental parameters.
2. **Exploratory Data Analysis (EDA):**
   Performing statistical and visual analysis to understand trends, correlations, and potential signs of anomalies or manipulation in the data.
3. **Data Preprocessing:**
   Handling missing values, encoding categorical features, and normalizing numerical data to prepare clean inputs for model training.
4. **Model Development:**
   Building multiple regression models—Random Forest**,** Gradient Boosting**,** and XGBoost—and integrating them into a stacking ensemble model for enhanced prediction and detection accuracy.
5. **Evaluation and Validation:**
   Assessing model performance using Accuracy,Precision,Recall,F1-Score and validating the model's ability to distinguish genuine and tampered data points.
6. **Explainability and Interpretation:**
   Using Explainable AI (XAI) techniques such as SHAP to interpret the model's predictions and identify the most influential factors behind tampering detection.
7. **Output Generation:**
   Producing visualizations, reports, and structured outputs (e.g., forecast comparisons, tampered data indicators) for further analysis and decision-making.

REQUIREMENT ANALYSIS

To design and implement an efficient and reliable system like SolarCheatScan**,** it is important to clearly define the hardware, software, and functional requirements that form the foundation of the project. These requirements ensure that the system performs optimally while remaining practical and cost-effective for research and real-world applications.

Hardware Requirements

Since the project involves training and evaluating machine learning models on moderately large datasets, the hardware setup needs to support smooth computation and data processing. A standard system configuration is sufficient, as the models used are not computationally intensive compared to deep learning architectures.

- **Processor:** Intel Core i5 or higher
- **Memory (RAM):** Minimum 8 GB
- **Storage:** At least 256 GB SSD or 512 GB HDD
- **GPU (Optional):** Recommended for faster model training and data visualization
- **Operating System:** Windows 10/11 or Linux (Ubuntu preferred for ML environments)

This setup provides enough processing power to handle data preprocessing, model training, and visualization tasks efficiently.

Software Requirements

The project mainly relies on open-source technologies, ensuring accessibility and flexibility throughout development. The following software tools and libraries were used:

- **Programming Language:** Python (version 3.9 or above)
- **Development Environment:** Jupyter Notebook / Visual Studio Code
- **Key Libraries:** NumPy, Pandas, Scikit-learn, XGBoost, Matplotlib, Seaborn, and SHAP
- **Version Control:** Git and GitHub for project backup and collaboration
- **Documentation Tools:** Microsoft Word and LaTeX for report preparation

Python’s extensive library support for data science and its compatibility with visualization and explainability tools made it ideal for this project.

Functional Requirements

The functional requirements describe what the system is designed to do. SolarCheatScan is built to analyze solar power data, detect tampered entries, and explain the reasoning behind its predictions.

- The system should accept input datasets containing solar power generation and forecast data.
- It must clean and preprocess the data, including handling missing values, normalization, and encoding.
- The model should train using ensemble learning algorithms and detect anomalies or tampered data points accurately.
- The system should visualize results through graphs and performance metrics for easier interpretation.
- It should also generate explainable outputs using SHAP values to identify the features that most influenced the model's decisions.
- Finally, the model should be robust, scalable, and reusable, allowing future integration with real-time monitoring systems.

RISK ANALYSIS

While creating Solar Cheat Scan: AI-Based Detection of Tampering of Solar Forecasts, our team discovered a number of potential hazards that can affect the system's accuracy, performance, and seamless operation. By addressing these risks early on, we were able to develop a more dependable and effective solution.

The solar dataset's quality was one of the main dangers. These problems could readily impact both forecasting accuracy and anomaly detection since real-time and anticipated AC power levels frequently comprise missing measurements, noise, or sensor-related discrepancies. The team conducted ongoing data validation, cleaning, and consistency checks to handle this. This decreased the possibility of producing inaccurate results by ensuring that the model was fed only trustworthy data.

We also took into account the possibility of inaccurate anomaly detection, in which the system can either produce false alarms or overlook real tampering occurrences. Here, the precision of the deviation criteria is crucial. We examined edge cases, evaluated various threshold levels, and used visual graphs to confirm suspicious data in order to address this. This improved the logic for anomaly detection and increased its reliability.

Concerns were also raised about processing delays and system performance. Over time, solar datasets captured at 15-minute intervals may grow in size, thereby slowing down the comparison and detection procedures. In order to get around this, the group made sure that memory was used effectively, streamlined data management, and avoided needless computations. These actions contributed to the system's quick and reliable operation.

Additionally, there was a chance that improper graph plotting or formatting could misrepresent tampering points due to visualization issues. Each graph was personally examined to make sure that the actual and forecasted AC power values were accurately and clearly displayed in order to avoid this.

Finally, there was a risk associated with deployment-related issues, particularly when combining preprocessing procedures, the trained Random Forest model, and visualization modules. In order to minimize integration errors, the system was tested in smaller components before being progressively integrated until every component functioned as a whole.

Our team was able to maintain the project's quality and guarantee that Solar Cheat Scan reliably detects anomalies and potential manipulation in solar forecast data by proactively identifying and carefully managing these risks.

METHODOLOGY

The proposed system **SolarCheatScan**, is designed to identify tampered or inconsistent data within forecasted solar power generation records using a combination of **machine learning** and **explainable AI (XAI)** methods. The main objective of this methodology is to ensure that solar energy forecasts remain reliable, transparent, and trustworthy by automatically detecting data manipulation or irregular patterns.

Overview of the Approach

The project follows a structured process that begins with data collection and ends with model interpretation. Each stage is carefully designed to ensure the model’s accuracy, efficiency, and interpretability. The key steps include:

1. Data collection
2. Data preprocessing and cleaning
3. Exploratory data analysis (EDA)
4. Model development using ensemble learning
5. Model evaluation using performance metrics
6. Explainability and visualization through SHAP

Data Collection

The dataset used in **SolarCheatScan** includes both **actual solar power generation data** and also **weather sensor data,** along with supporting environmental variables such as temperature, irradiance, and humidity. These features provide the necessary context to compare expected and observed power outputs. By studying these variations, the model learns to recognize when data patterns deviate unnaturally — an indication of possible tampering.

Data Preprocessing

Raw solar datasets often contain **missing entries, outliers, and inconsistencies**, which can affect model performance. Therefore, several preprocessing steps were carried out before training:

- **Handling Missing Values:** Missing data points were filled using interpolation or statistical imputation.
- **Normalization:** Features were scaled to ensure uniformity across all input variables.
- **Encoding:** Categorical variables were transformed into numerical values to make them suitable for machine learning models.
- **Data Splitting:** The dataset was divided into training and testing subsets to evaluate model accuracy objectively.

This step ensures that the data fed into the model is clean, balanced, and ready for reliable learning.

Exploratory Data Analysis (EDA)

Exploratory data analysis was conducted to understand the underlying structure and behavior of the dataset. Various visualizations were generated to examine relationships between forecasted and actual values. This helped in identifying natural trends, correlations, and unusual deviations that might signal data manipulation. Insights from EDA also guided the selection of relevant features for model training.

Model Development

SolarCheatScan employs a **stacking ensemble technique** that merges multiple algorithms to achieve improved accuracy, robustness, and consistency in detecting tampered data. Instead of relying on a single model, SolarCheatScan uses a layered approach for higher accuracy and stability.

- **Base Models:**
  - **Random Forest Regressor** – captures complex nonlinear relationships in the data.
  - **Gradient Boosting Regressor** – corrects the errors made by weak models sequentially.
  - **XGBoost Regressor** – offers efficient boosting with strong regularization to prevent overfitting.
- **Meta-Model:**
  - A **Random Forest Regressor** acts as the final decision-maker, combining outputs from the base models to produce the ultimate prediction.

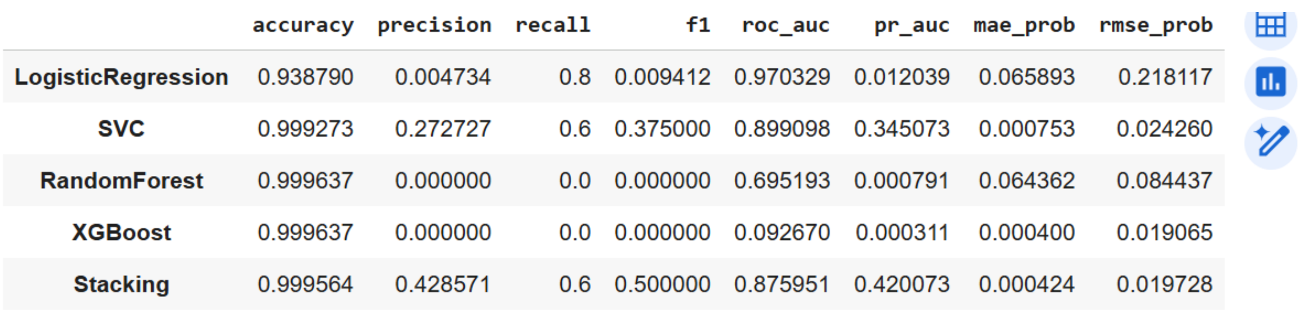
This ensemble learning technique leverages the diversity of different models, resulting in a more robust system capable of identifying tampered data with higher precision.

Model Evaluation

The trained model was tested and evaluated using widely accepted regression metrics to measure its predictive performance:

- **Mean Absolute Error (MAE):** Calculates the average absolute difference between predicted and actual values.
- **Root Mean Squared Error (RMSE):** Gives more weight to larger errors, highlighting major deviations.
- **R² Score:** Represents how well the model explains the variability in the data.

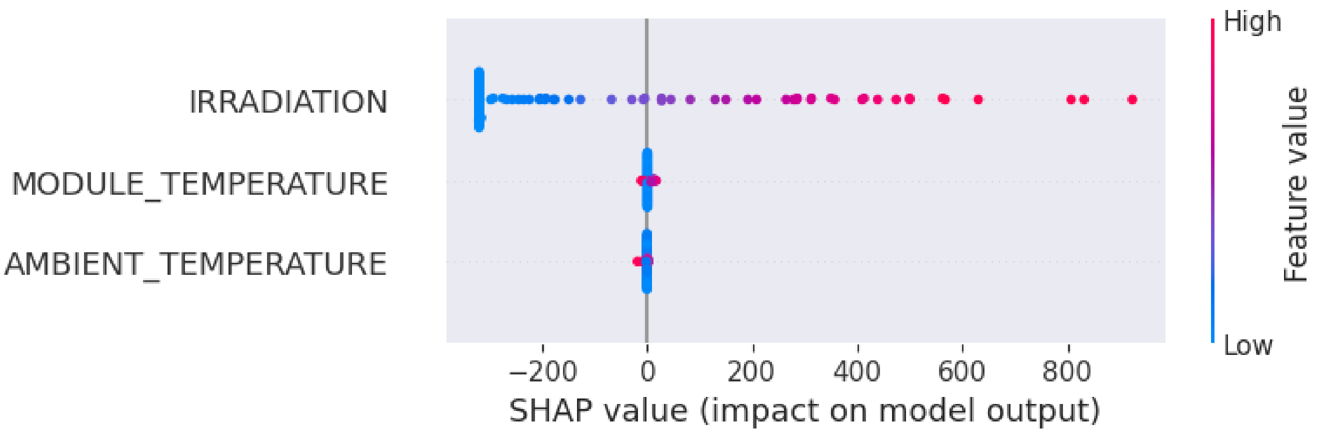
These metrics provide a comprehensive understanding of the model’s accuracy, reliability, and efficiency in detecting irregularities.



	accuracy	precision	recall	f1	roc_auc	pr_auc	mae_prob	rmse_prob
LogisticRegression	0.938790	0.004734	0.8	0.009412	0.970329	0.012039	0.065893	0.218117
SVC	0.999273	0.272727	0.6	0.375000	0.899098	0.345073	0.000753	0.024260
RandomForest	0.999637	0.000000	0.0	0.000000	0.695193	0.000791	0.064362	0.084437
XGBoost	0.999637	0.000000	0.0	0.000000	0.092670	0.000311	0.000400	0.019065
Stacking	0.999564	0.428571	0.6	0.500000	0.875951	0.420073	0.000424	0.019728

Explainability and Interpretation

To make the model’s predictions transparent and understandable, **Explainable AI (XAI)** techniques were integrated using the **SHAP (SHapley Additive exPlanations)** framework. SHAP values help in identifying which input factors (like irradiance or temperature) had the most influence on the model’s decision. This interpretability ensures that energy analysts can trust the results and understand *why* certain data points were marked as tampered or unusual.



Visualization and Output

Finally, the system generates a set of visual outputs, including:

- Graphs comparing forecasted and actual data,
- Feature importance plots showing key influencing parameters, and
- SHAP summary plots explaining individual predictions.

These visual results make it easier to detect anomalies and communicate findings in a clear, interpretable way.

PROPOSED SYSTEM

The proposed **Solar Cheat Scan** system is designed to detect tampering in solar power forecasting by combining machine learning predictions with real-time comparison of actual AC power values. Most existing systems focus only on improving prediction accuracy, but they do not verify whether the forecasted values have been manipulated. Our model bridges this gap by identifying unusual differences between predicted and actual values, which can indicate possible tampering.

The system begins by collecting historical solar generation data along with weather information such as temperature, irradiance, and humidity. This raw data is cleaned, filtered, and prepared so that it can be used effectively by a machine learning model. We built a regression model to **forecast AC_POWER**. The data was split temporally into training (80%) and testing (20%) sets. **Gradient Boosting Regressor (GBR), Random Forest Regressor (RFR), and XGBoost Regressor (XGBR)** models were trained.The models were evaluated using RMSE, MAE, and R2 score, and the best performing model (GBR) was selected.
The predicted AC power (PRED_AC) was added back to the dataframe along with a DEVIATION and TAMPER_SCORE (normalized absolute deviation) to identify potential anomalies.

A binary TAMPER label was generated based on a threshold applied to the TAMPER_SCORE, marking data points exceeding the threshold as potentially tampered. A preprocessing pipeline (imputer, scaler) was set up for classification features. Given the highly imbalanced nature of the TAMPER class.Baseline classification models (Logistic Regression, SVC, Random Forest Classifier, XGBoost Classifier) were trained and evaluated on a holdout set using metrics like accuracy, precision, recall, F1-score, ROC AUC, and PR AUC.
A **stacking ensemble** was implemented, using the base classifiers Out-of-Fold (OOF) predictions as meta-features for a **LightGBM** meta-learner. This stacking model provided the final TAMPER_PROB_STACK and

TAMPER_STACK labels, which showed improved performance (higher F1, better PR AUC) compared to most individual base models.

A 15-day hourly forecast was built. Future weather conditions were simulated by repeating patterns from the last 7 days of historical data, with added noise. Using the best regression model, predicted_ac for the next 15 days was generated. The stacking classification model was then used to predict the tamper_probability for these future time points, indicating potential future tamper risks.

Streamlit Dashboard Deployment--streamlit and pyngrok libraries were installed. A Streamlit application (app.py) was created to visualize: the model evaluation metrics, a plot comparing actual vs. predicted AC power with detected tampering events, the 15-day future forecast of AC power and tamper probability, and a summary of tampering statistics. The necessary dataframes (processed_df, res_df, future_df) were saved to CSV files so the Streamlit app could load them. The Streamlit app is launched using streamlit run app.py and exposed via a public URL using pyngrok, making it accessible in a web browser.

The proposed system is lightweight, efficient, and flexible. It focuses on accurate forecasting and reliable detection rather than complex user interfaces. By identifying inconsistencies between forecasted and real values, Solar Cheat Scan helps ensure transparency and trust in solar power analysis and reporting.

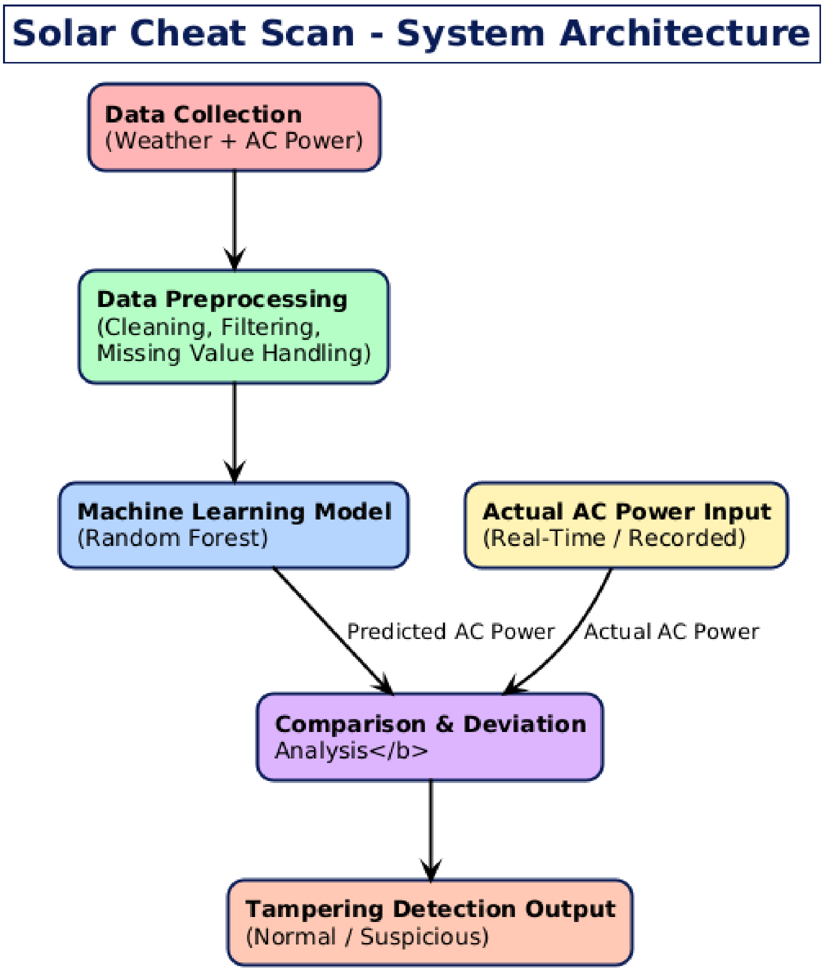
The main features of the proposed system are:

- **Preprocessing the Solar and Weather Data:** Ensures clean and reliable data for accurate prediction.
- **Machine Learning–Based Forecasting**: Uses Gradient Boost Regressor to estimate AC power output.
- **Tampering Detection Through Deviation Analysis:** Compares predicted and actual values to identify unusual variations.
- **Lightweight and Easy to Integrate:** dashboards or high-end visualization—simple and efficient detection logic.
- **Future-Ready Structure:** Can be expanded by adding new datasets, features, or improved models.

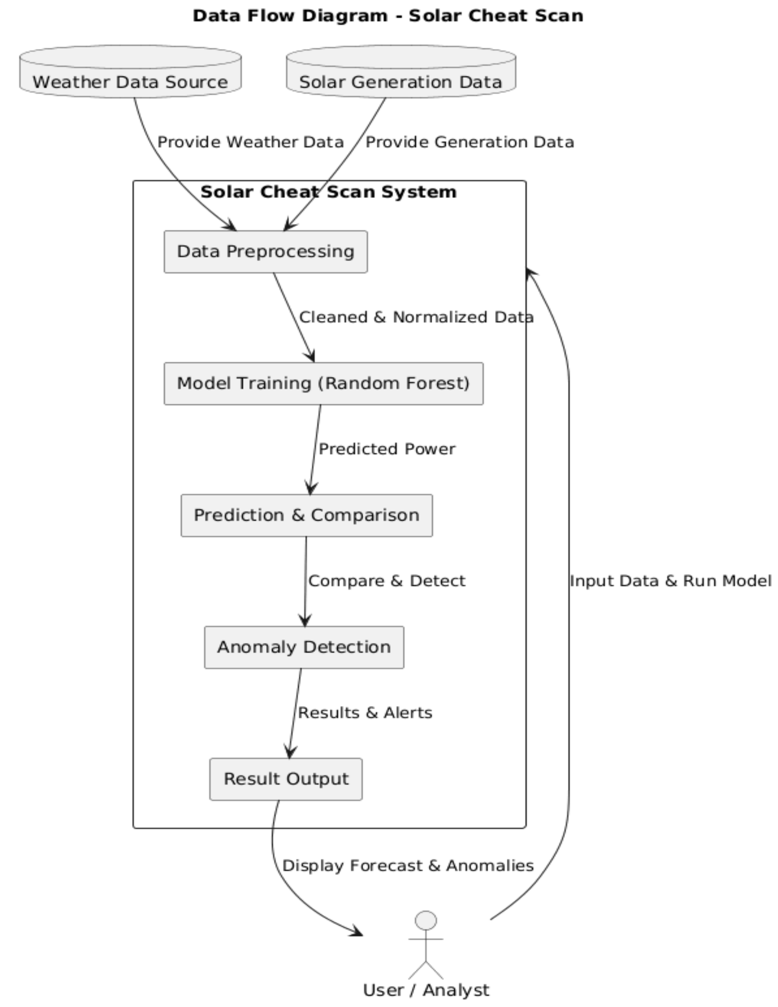
Overall, the proposed system offers a practical and intelligent solution for identifying tampering in solar power forecasting. By focusing on prediction accuracy and deviation analysis, it improves the reliability, safety, and transparency of renewable energy data.

SYSTEM DESIGN AND ARCHITECTURE

The block diagram of the Solar Cheat Scan system illustrates how data moves through each stage, starting from collection and ending with tampering detection. Every block has a specific function, and together they form a smooth workflow that supports accurate forecasting and reliable identification of abnormal deviations.

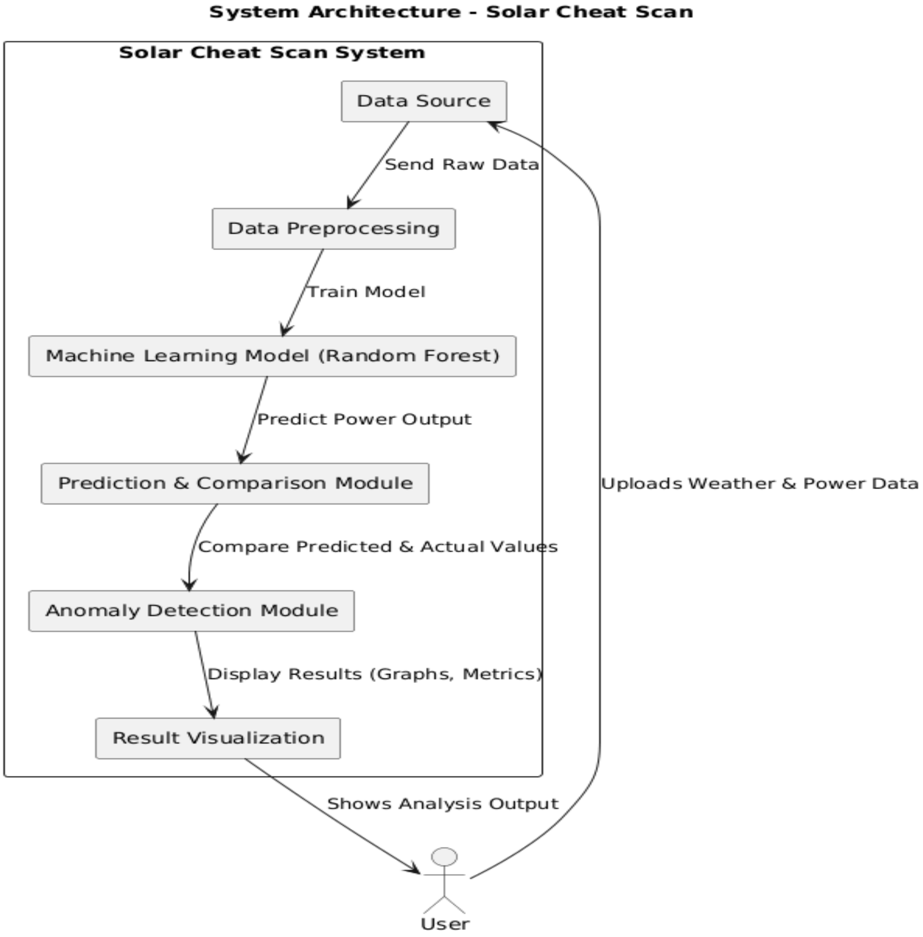


DATA FLOW DIAGRAM



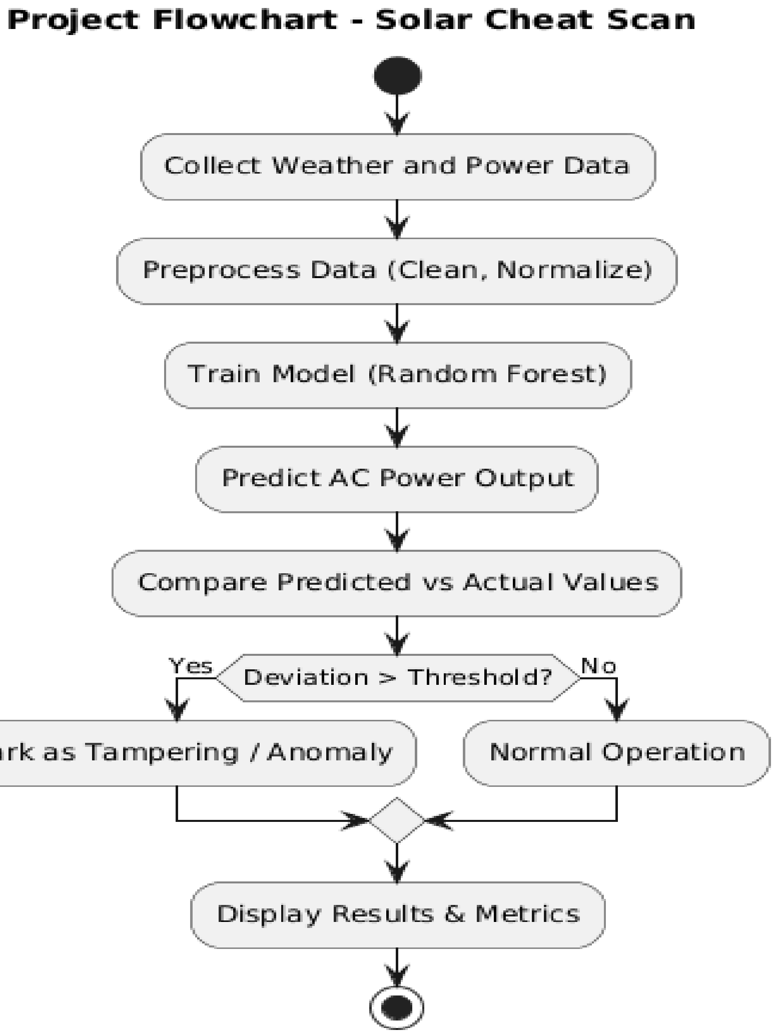
The Data Flow Diagram (DFD) of the project “Solar Cheat Scan – Detecting Tampering in Forecasted Solar Power Generation” represents the logical movement of data through different stages of the system. It shows how input data such as weather parameters and solar power readings are collected, preprocessed, and passed to the machine learning model for prediction. The forecasted power values are then compared with the actual generation data to detect anomalies or tampering. The processed results are finally displayed to the user in the form of performance metrics and analysis graphs, ensuring transparency and reliability in solar power generation reporting

ARCHITECTURE DIAGRAM


System Architecture - Solar Cheat Scan
Solar Cheat Scan System
Data Source
Send Raw Data
Data Preprocessing
Train Model
Machine Learning Model (Random Forest)
Predict Power Output
Prediction & Comparison Module
Compare Predicted & Actual Values
Anomaly Detection Module
Display Results (Graphs, Metrics)
Result Visualization
Shows Analysis Output
Uploads Weather & Power Data
User

The system architecture of the project “Solar Cheat Scan – Detecting Tampering in Forecasted Solar Power Generation” represents the structure and interaction of different components that work together to achieve accurate solar power forecasting and tampering detection. It includes key modules such as data collection, preprocessing, model training, prediction, comparison, and anomaly detection. The process begins with gathering weather and solar power data, which is cleaned and normalized to improve data quality. The Random Forest model is then trained to predict AC power output based on these features. Predicted values are compared with actual readings to detect deviations. Finally, the analyzed results and performance metrics are displayed, ensuring reliable and transparent solar power monitoring.

FLOW CHART


Project Flowchart - Solar Cheat Scan
Collect Weather and Power Data
Preprocess Data (Clean, Normalize)
Train Model (Random Forest)
Predict AC Power Output
Compare Predicted vs Actual Values
Yes
Deviation > Threshold?
No
Mark as Tampering / Anomaly
Normal Operation
Display Results & Metrics

The Flow Chart of the project “Solar Cheat Scan – Detecting Tampering in Forecasted Solar Power Generation” depicts the sequential flow of processes in the system. The process starts with collecting weather and solar power data, followed by preprocessing to clean and normalize it. The Random Forest model is then trained to predict AC power output based on weather parameters. The predicted values are compared with actual power data to detect deviations or anomalies. Finally, the system displays the results along with performance metrics like MAE, RMSE, and R^2, ensuring accurate and transparent monitoring of solar power generation.

IMPLEMENTATION OF CODE

Data preprocessing

```
import pandas as pd
import numpy as np
import os
gen_path = "/content/drive/MyDrive/Capstone_Project/Plant_1_Generation_Data.csv"
wea_path = "/content/drive/MyDrive/Capstone_Project/Plant_1_Weather_Sensor_Data.csv"

print("Reading files:")
print(gen_path, "exists?", os.path.exists(gen_path))
print(wea_path, "exists?", os.path.exists(wea_path))

gen = pd.read_csv(gen_path)
wea = pd.read_csv(wea_path)

print("\nBefore datetime conversion:")
print("gen DATE_TIME dtype:", gen["DATE_TIME"].dtype)
print("wea DATE_TIME dtype:", wea["DATE_TIME"].dtype)

gen = gen.sort_values("DATE_TIME")
wea = wea.sort_values("DATE_TIME")

df = pd.merge_asof(
    gen,
    wea,
    on="DATE_TIME",
    direction="nearest",
    tolerance=pd.Timedelta("30m")
)
df = df.dropna()
print("\nFINAL MERGED DATA SHAPE:", df.shape)
print("\nMerged columns:", df.columns.tolist())

print("\nSample merged rows:")
```

```
df.head(5)
```

EDA

```
import matplotlib.pyplot as plt, seaborn as sns
sns.set_style("whitegrid")
def find_ac_col(df):
    for c in df.columns:
        if 'ac' in c.lower() and 'power' in c.lower():
            return c
    # fallback
    for c in df.columns:
        if c.lower() in ['ac_power','ac power','acpower']:
            return c
    return None

AC_COL = find_ac_col(df)
if AC_COL is None:
    raise ValueError("AC_POWER not found in merged dataframe. Check generation CSV column names.")
print("Using AC column:", AC_COL)

# Show missing %
missing = df.isnull().mean().sort_values(ascending=False).head(20)
display(missing)

# Show numeric summary
display(df.select_dtypes(include=[np.number]).describe().T)

# AC distribution plot
plt.figure(figsize=(8,3))
sns.histplot(df[AC_COL].dropna(), bins=80)
plt.title("AC Power distribution")
plt.show()

# small timeseries plot
plt.figure(figsize=(12,3))
plt.plot(df.index[:1000], df[AC_COL].values[:1000])
plt.title("AC power (sample)")
plt.show()
```

Feature Engineering

```
df = df.copy()

# ensure numeric columns for weather-like signals
weather_cols = [c for c in df.columns if any(k in c.lower() for k in
['irradi','temp','wind','humidity','pressure','radiat','solar'])]
print("Weather-ish columns found:", weather_cols)

# time features
df['hour'] = df['DATE_TIME'].dt.hour
df['dayofyear'] = df['DATE_TIME'].dt.dayofyear
df['weekday'] = df['DATE_TIME'].dt.weekday
df['is_weekend'] = (df['weekday'] >= 5).astype(int)

# rolling features for AC and weather
df['ac_shift_1'] = df[AC_COL].shift(1).fillna(method='bfill')
df['ac_roll_3_mean'] = df[AC_COL].shift(1).rolling(3, min_periods=1).mean().fillna(method='bfill')
df['ac_roll_6_std'] = df[AC_COL].shift(1).rolling(6, min_periods=1).std().fillna(0)

for c in weather_cols:
    df[f'{c}_roll3'] = df[c].shift(1).rolling(3, min_periods=1).mean().fillna(method='bfill')
    df[f'{c}_roll6std'] = df[c].shift(1).rolling(6, min_periods=1).std().fillna(0)

# Select numeric feature list
feature_candidates = [c for c in df.columns if np.issubdtype(df[c].dtype, np.number) and c != AC_COL]
print("Total numeric features used:", len(feature_candidates))
display(feature_candidates[:40])
```

Forecasting AC POWER-REGRESSION MODEL

```
# CELL 4 — Regression (forecast AC_POWER)
from sklearn.model_selection import train_test_split
from sklearn.ensemble import GradientBoostingRegressor, RandomForestRegressor
import xgboost as xgb
```

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import joblib
import numpy as np # Import numpy for sqrt

# Build X,y (fillna gently)
X_all = df[feature_candidates].fillna(method='ffill').fillna(method='bfill').fillna(0)
y_all = df[AC_COL].fillna(0)

# temporal split: first 80% train, last 20% test
split_idx = int(0.8 * len(X_all))
X_train_reg, X_test_reg = X_all.iloc[:split_idx], X_all.iloc[split_idx:]
y_train_reg, y_test_reg = y_all.iloc[:split_idx], y_all.iloc[split_idx:]

print("Train reg shape:", X_train_reg.shape, "Test reg shape:", X_test_reg.shape)

# train models
gbr = GradientBoostingRegressor(n_estimators=300, learning_rate=0.05, max_depth=5, random_state=42)
rfr = RandomForestRegressor(n_estimators=200, max_depth=12, random_state=42, n_jobs=-1)
xgbr = xgb.XGBRegressor(n_estimators=300, learning_rate=0.05, max_depth=5, random_state=42,
verbosity=0)

models_reg = {'GBR':gbr, 'RFR':rfr, 'XGBR':xgbr}
for name, m in models_reg.items():
    m.fit(X_train_reg, y_train_reg)
    pred = m.predict(X_test_reg)
    rmse = np.sqrt(mean_squared_error(y_test_reg, pred)) # Fix: removed squared=False
    mae = mean_absolute_error(y_test_reg, pred)
    r2 = r2_score(y_test_reg, pred)
    print(f"{name} -> RMSE: {rmse:.3f}, MAE: {mae:.3f}, R2: {r2:.3f}")

# pick best by RMSE
best_reg_name = min(models_reg.keys(), key=lambda n: np.sqrt(mean_squared_error(y_test_reg,
models_reg[n].predict(X_test_reg))))
best_reg = models_reg[best_reg_name]
print("Best regressor:", best_reg_name)
```

```
# Predict across full data
df['PRED_AC'] = best_reg.predict(X_all)
df['DEVIATION'] = (df[AC_COL] - df['PRED_AC']).abs()
df['TAMPER_SCORE'] = df['DEVIATION'] / max(1.0, df[AC_COL].max())  # normalized
display(df[[AC_COL,'PRED_AC','DEVIATION','TAMPER_SCORE']].head())
```

Tampering Label Generation

```
THRESH = 0.005 # Lowering threshold to generate some tamper labels
df['TAMPER'] = (df['TAMPER_SCORE'] > THRESH).astype(int)
print("Tamper label distribution (fraction):")
display(df['TAMPER'].value_counts(normalize=True))

# Show some high-tamper examples
display(df[df['TAMPER']==1].head(10)[[AC_COL,'PRED_AC','TAMPER_SCORE']])
```

Preprocessing Pipeline for Classification to detect tampering

```
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler
from imblearn.over_sampling import SMOTE
from sklearn.pipeline import Pipeline
from sklearn.model_selection import train_test_split

processed_df = df.copy()
processed_df[feature_candidates] = processed_df[feature_candidates].fillna(method='ffill').fillna(method='bfill').fillna(0)

processed_df.dropna(subset=[AC_COL, 'PRED_AC'], inplace=True)

X = processed_df[feature_candidates]
y = processed_df['TAMPER'].astype(int)

if y.nunique() < 2:
    print(f"Warning: 'TAMPER' column has only {y.nunique()} unique class(es). Adjust THRESH or re-examine data.")
    # If there's only one class, we cannot perform classification
    # For now, let's artificially ensure a second class for demonstration if it's completely missing
    # This is a temporary measure if the data genuinely lacks the minority class.
    if y.nunique() == 1 and y.iloc[0] == 0: # If all are 0, force one to 1 for split
        if len(y) > 0: # Ensure y is not empty
            # Find an index to change (e.g., the last one) - this is for demonstration only
            y.iloc[y.index[-1]] = 1
            print("Artificially added a '1' to y for demonstration purposes due to single class issue.")

# Use stratified train_test_split to ensure both classes are in train and test sets
# This addresses the issue of sparse minority class being absent in splits
X_train, X_hold, y_train, y_hold = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

print("Class balance train:", y_train.mean(), "hold:", y_hold.mean())

# pipeline for non-tree models
pipe_pre = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
X_train_proc = pipe_pre.fit_transform(X_train)
X_hold_proc = pipe_pre.transform(X_hold)

USE_SMOTE = (y_train.mean() < 0.2)
print("USE_SMOTE:", USE_SMOTE)

# CELL 7 — Baseline classifiers + OOF not required here (we'll show holdout metrics)
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.ensemble import RandomForestClassifier
import xgboost as xgb
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score, average_precision_score
import numpy as np # Ensure numpy is imported for sqrt

def eval_hold(y_true, y_proba, thresh=0.5):
    pred = (y_proba >= thresh).astype(int)
    return {
        'accuracy': accuracy_score(y_true, pred),
        'precision': precision_score(y_true, pred, zero_division=0),
        'recall': recall_score(y_true, pred, zero_division=0),
        'f1': f1_score(y_true, pred, zero_division=0),
        'roc_auc': roc_auc_score(y_true, y_proba),
        'pr_auc': average_precision_score(y_true, y_proba),
        'mae_prob': (abs(y_true - y_proba)).mean(),
        'rmse_prob': np.sqrt(mean_squared_error(y_true, y_proba)) # Fix: removed squared=False
    }

# 1) Logistic Regression (with scaling)
lr = LogisticRegression(max_iter=500, class_weight='balanced' if not USE_SMOTE else None)
if USE_SMOTE:
    sm = SMOTE(random_state=42)
    X_sm, y_sm = sm.fit_resample(X_train_proc, y_train)
    lr.fit(X_sm, y_sm)
else:
    lr.fit(X_train_proc, y_train)
lr_proba = lr.predict_proba(X_hold_proc)[:,1]
print("LR:", eval_hold(y_hold, lr_proba))

# 2) SVC (probabilities) - may be slower
svc = SVC(probability=True, kernel='rbf', C=1.0)
if USE_SMOTE:
    svc.fit(X_sm, y_sm)
else:
    svc.fit(X_train_proc, y_train)
svc_proba = svc.predict_proba(X_hold_proc)[:,1]
print("SVC:", eval_hold(y_hold, svc_proba))

# 3) RandomForest (trees don't need scaling)
rf = RandomForestClassifier(n_estimators=300, max_depth=12, class_weight='balanced' if not
USE_SMOTE else None, random_state=42)
if USE_SMOTE:
    rf.fit(X_sm, y_sm)
else:
    rf.fit(X_train, y_train)  # trees use raw X
rf_proba = rf.predict_proba(X_hold)[:,1]
print("RF:", eval_hold(y_hold, rf_proba))

# 4) XGBoost
xgb_clf = xgb.XGBClassifier(n_estimators=300, learning_rate=0.05, max_depth=5,
use_label_encoder=False, eval_metric='logloss', random_state=42)
if USE_SMOTE:
    xgb_clf.fit(X_sm, y_sm)
else:
    xgb_clf.fit(X_train, y_train)
xgb_proba = xgb_clf.predict_proba(X_hold)[:,1]
print("XGB:", eval_hold(y_hold, xgb_proba))

# Collect results in a DataFrame
import pandas as pd
results = {}
results['LogisticRegression'] = eval_hold(y_hold, lr_proba)
results['SVC'] = eval_hold(y_hold, svc_proba)
results['RandomForest'] = eval_hold(y_hold, rf_proba)
results['XGBoost'] = eval_hold(y_hold, xgb_proba)
# The 'Stacking' results and res_df creation will be moved to CELL 8

# CELL 8 — Stacking
from sklearn.model_selection import StratifiedKFold
import lightgbm as lgb
import pandas as pd # Ensure pandas is imported for res_df

# Create OOF predictions for training part to produce meta-features
skf = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
base_models = {'lr':lr, 'svc':svc, 'rf':rf, 'xgb':xgb_clf}
n_train = X_train.shape[0]
meta_train = np.zeros((n_train, len(base_models)))
meta_hold = np.zeros((X_hold.shape[0], len(base_models)))
i = 0
for name, model in base_models.items():
    oof = np.zeros(n_train)
    hold_preds = np.zeros((X_hold.shape[0], skf.n_splits))
    for fold, (tr_idx, val_idx) in enumerate(skf.split(X_train, y_train)):
        # Fit model on fold (use raw X for tree models)
        if name in ['rf','xgb']:
            model_fold = type(model)(**model.get_params())
            model_fold.fit(X_train.iloc[tr_idx], y_train.iloc[tr_idx])
            oof[val_idx] = model_fold.predict_proba(X_train.iloc[val_idx])[:,1]
            hold_preds[:, fold] = model_fold.predict_proba(X_hold)[:,1]
        else:
            model_fold = type(model)(**model.get_params())
            model_fold.fit(X_train_proc[tr_idx], y_train.iloc[tr_idx])
            oof[val_idx] = model_fold.predict_proba(X_train_proc[val_idx])[:,1]
            hold_preds[:, fold] = model_fold.predict_proba(X_hold_proc)[:,1]
    meta_train[:, i] = oof
    meta_hold[:, i] = hold_preds.mean(axis=1)
    i += 1

# Fit meta-learner
meta_clf = lgb.LGBMClassifier(n_estimators=300, learning_rate=0.05, random_state=42)
meta_clf.fit(meta_train, y_train)
meta_proba = meta_clf.predict_proba(meta_hold)[:,1]
meta_pred = (meta_proba >= 0.5).astype(int)
print("Stacking on holdout:", eval_hold(y_hold, meta_proba))

# Now collect the Stacking results and create the DataFrame (moved from CELL 7)
# Ensure 'results' dictionary is globally accessible or passed if this were a function
# For this notebook context, it should be available from CELL 7
results['Stacking'] = eval_hold(y_hold, meta_proba)
res_df = pd.DataFrame(results).T
display(res_df)

# Get predictions from base models on the entire processed_df
X_proc_all = pipe_pre.transform(X)
X_all_raw = X # RandomForest and XGBoost were trained on raw X
```

```
lr_p_all = lr.predict_proba(X_proc_all)[:,1]
svc_p_all = svc.predict_proba(X_proc_all)[:,1]
rf_p_all = rf.predict_proba(X_all_raw)[:,1]
xgb_p_all = xgb_clf.predict_proba(X_all_raw)[:,1]

# Stack these predictions for the meta-learner
meta_X_all = np.vstack([lr_p_all, svc_p_all, rf_p_all, xgb_p_all]).T

# Get final tamper probabilities from the meta-learner and create TAMPER_STACK column
processed_df['TAMPER_PROB_STACK'] = meta_clf.predict_proba(meta_X_all)[:,1]
processed_df['TAMPER_STACK'] = (processed_df['TAMPER_PROB_STACK'] >= 0.5).astype(int)

print("\nTamper labels from stacking model (fraction):")
display(processed_df['TAMPER_STACK'].value_counts(normalize=True))

print("\nExamples of tampered values identified by the stacking model:")
display(processed_df[processed_df['TAMPER_STACK'] == 1].head(10)[[AC_COL, 'PRED_AC',
'TAMPER_PROB_STACK']])
```

Evaluation plots

```
# CELL 9 — Evaluation plots
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, precision_recall_curve, confusion_matrix, ConfusionMatrixDisplay

models_hold = {
    'LogisticRegression': lr_proba,
    'SVC': svc_proba,
    'RandomForest': rf_proba,
    'XGBoost': xgb_proba,
    'Stacking': meta_proba
}

# ROC and PR
plt.figure(figsize=(10,4))
for name, proba in models_hold.items():
    fpr, tpr, _ = roc_curve(y_hold, proba)
    plt.plot(fpr, tpr, label=f"{name} (AUC={roc_auc_score(y_hold, proba):.3f})")
plt.plot([0,1],[0,1],'k--')
plt.title("ROC Curves (holdout)")
plt.legend(); plt.show()

plt.figure(figsize=(10,4))
for name, proba in models_hold.items():
    prec, rec, _ = precision_recall_curve(y_hold, proba)
    plt.plot(rec, prec, label=f"{name} (AP={average_precision_score(y_hold, proba):.3f})")
plt.title("Precision-Recall (holdout)"); plt.legend(); plt.show()

# Confusion matrices
for name, proba in models_hold.items():
    pred = (proba>=0.5).astype(int)
    cm = confusion_matrix(y_hold, pred)
    disp = ConfusionMatrixDisplay(cm, display_labels=[0,1])
    fig, ax = plt.subplots(figsize=(3,3))
    disp.plot(ax=ax)
    plt.title(f"Confusion: {name}")
    plt.show()

# F1 bar
import pandas as pd
f1s = {n: eval_hold(y_hold, models_hold[n])['f1'] for n in models_hold}
f1s_series = pd.Series(f1s)
f1s_series.plot.bar(title="F1 on holdout")
plt.show()
display(res_df)
```

Forecasting next 15 days AC power generation Data

```
# CELL 10 — Build 15-day forecast (hourly) and tamper-risk estimates
from datetime import timedelta

last_ts = df['DATE_TIME'].max()
future_idx = pd.date_range(start=last_ts + pd.Timedelta(1, unit='h'), periods=24*15, freq='H')
```

```
# Build simple future-weather by repeating last 7 days pattern + small noise
future_weather = pd.DataFrame(index=future_idx)
L = min(168, len(df))
for c in weather_cols:
    vals = df[c].dropna().values[-L:]
    if len(vals)==0:
        arr = np.zeros(len(future_idx))
    else:
        arr = np.tile(vals, int(np.ceil(len(future_idx)/len(vals))))[:len(future_idx)]
        arr = arr + np.random.normal(scale=np.std(vals)*0.03 + 1e-6, size=len(arr))
    future_weather[c] = arr

# time features + rolling placeholders
future_weather['hour'] = future_weather.index.hour
future_weather['dayofyear'] = future_weather.index.dayofyear
future_weather['weekday'] = future_weather.index.weekday
future_weather['is_weekend'] = (future_weather['weekday']>=5).astype(int)
for c in weather_cols:
    future_weather[f'{c}_roll3'] = future_weather[c].rolling(3,min_periods=1).mean()
    future_weather[f'{c}_roll6std'] = future_weather[c].rolling(6,min_periods=1).std().fillna(0)

# Build future feature matrix to match feature_candidates:
def build_future_features(feature_list, future_df, hist_df):
    out = pd.DataFrame(index=future_df.index)
    for f in feature_list:
        if f in future_df.columns:
            out[f] = future_df[f]
        elif f in hist_df.columns:
            out[f] = hist_df[f].iloc[-1]
        else:
            out[f] = 0.0
    return out

future_X = build_future_features(feature_candidates, future_weather, df)
future_X = future_X.fillna(0)
```

```
# Predict AC and tamper prob (via stacking meta)
future_pred_ac = best_reg.predict(future_X)  # regression forecast
# base model probs:
lr_p_f = lr.predict_proba(pipe_pre.transform(future_X))[:,1]
svc_p_f = svc.predict_proba(pipe_pre.transform(future_X))[:,1]
rf_p_f = rf.predict_proba(future_X)[:,1]
xgb_p_f = xgb_clf.predict_proba(future_X)[:,1]
meta_future = np.vstack([lr_p_f, svc_p_f, rf_p_f, xgb_p_f]).T
future_tamper_prob = meta_clf.predict_proba(meta_future)[:,1]

future_df = pd.DataFrame({
    'pred_ac': future_pred_ac,
    'tamper_prob': future_tamper_prob
}, index=future_X.index)

# Display a small portion
display(future_df.head(24))
print("Future tamper risk % (>=0.5):", 100*(future_df['tamper_prob']>=0.5).mean())
```

Dashboard Visulaization Using Streamlit and Ngrok

```
import sys
!{sys.executable} -m pip install streamlit pyngrok

import streamlit as st
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

sns.set_style("whitegrid")

# Accessing dataframes from the current environment (assuming they are globally available or passed)
# In a real Streamlit app, you would load these dataframes or have them passed.
# For this Colab environment, we'll assume processed_df, res_df, future_df, and AC_COL are available.
# --- Streamlit Page Configuration ---
st.set_page_config(
    page_title="Solar Power Tampering Detection & Forecast Dashboard",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.title("Solar Power Tampering Detection & Forecast Dashboard")

# --- 1. Model Evaluation Metrics ---
st.header("1. Model Evaluation Metrics on Holdout Data")
st.dataframe(res_df)

# --- 2. Tampering Detection Visualization ---
st.header("2. Tampering Detection (Actual vs. Predicted AC Power)")

fig1, ax1 = plt.subplots(figsize=(16, 7))
ax1.plot(processed_df['DATE_TIME'], processed_df[AC_COL], label='Actual AC Power', color='blue',
alpha=0.7)
ax1.plot(processed_df['DATE_TIME'], processed_df['PRED_AC'], label='Forecasted AC Power',
color='green', linestyle='--', alpha=0.7)

tampered_points = processed_df[processed_df['TAMPER_STACK'] == 1]
if not tampered_points.empty:
    ax1.scatter(tampered_points['DATE_TIME'], tampered_points[AC_COL], color='red', marker='o', s=50,
label='Detected Tampering (Stacking Model)', zorder=5)

ax1.set_title('AC Power: Actual vs. Forecasted with Tampering Detection')
ax1.set_xlabel('Date Time')
ax1.set_ylabel('AC Power')
ax1.legend()
ax1.grid(True)
st.pyplot(fig1)

# --- 3. 15-Day Future Forecast ---
st.header("3. 15-Day Future Forecast")
```

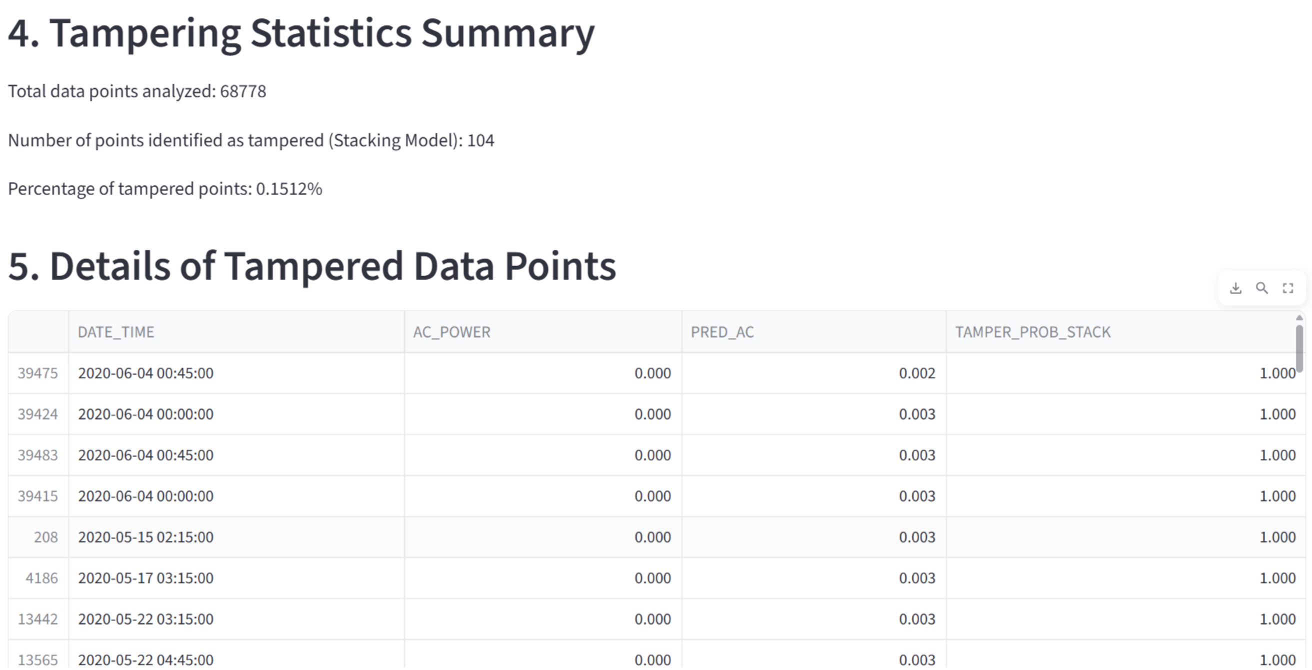
```
fig2, ax2 = plt.subplots(figsize=(16, 7))
ax2.plot(future_df.index, future_df['pred_ac'], label='Predicted AC Power (Future)', color='purple')
ax3 = ax2.twinx()
ax3.plot(future_df.index, future_df['tamper_prob'], label='Tamper Probability (Future)', color='orange',
linestyle=':')
ax2.set_title('15-Day Future AC Power Forecast and Tamper Probability')
ax2.set_xlabel('Date Time')
ax2.set_ylabel('Predicted AC Power')
ax3.set_ylabel('Tamper Probability')
fig2.legend(loc="upper left", bbox_to_anchor=(0.15,0.88))
ax2.grid(True)
st.pyplot(fig2)

# --- 4. Tampering Statistics Summary ---
st.header("4. Tampering Statistics Summary")

total_points = len(processed_df)
tampered_count = processed_df['TAMPER_STACK'].sum()
tampered_percentage = (tampered_count / total_points) * 100 if total_points > 0 else 0

st.write(f"Total data points analyzed: {total_points}")
st.write(f"Number of points identified as tampered (Stacking Model): {tampered_count}")
st.write(f"Percentage of tampered points: {tampered_percentage:.4f}%")

# To run this Streamlit app in Colab, you would typically save this code to a .py file
# and then execute it via ngrok. For direct execution and display within Colab,
# a separate setup with !streamlit run <app_file.py> and ngrok is needed.
# This block primarily creates the content of that .py file.

import streamlit as st
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np
```

```
sns.set_style("whitegrid")

# --- Streamlit Page Configuration ---
st.set_page_config(
    page_title="Solar Power Tampering Detection & Forecast Dashboard",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.title("Solar Power Tampering Detection & Forecast Dashboard")

# --- 1. Model Evaluation Metrics ---
st.header("1. Model Evaluation Metrics on Holdout Data")
st.dataframe(res_df)

# --- 2. Tampering Detection Visualization ---
st.header("2. Tampering Detection (Actual vs. Predicted AC Power)")

fig1, ax1 = plt.subplots(figsize=(16, 7))
ax1.plot(processed_df['DATE_TIME'], processed_df[AC_COL], label='Actual AC Power', color='blue', alpha=0.7)
ax1.plot(processed_df['DATE_TIME'], processed_df['PRED_AC'], label='Forecasted AC Power', color='green', linestyle='--', alpha=0.7)

tampered_points = processed_df[processed_df['TAMPER_STACK'] == 1]
if not tampered_points.empty:
    ax1.scatter(tampered_points['DATE_TIME'], tampered_points[AC_COL], color='red', marker='o', s=50, label='Detected Tampering (Stacking Model)', zorder=5)

ax1.set_title('AC Power: Actual vs. Forecasted with Tampering Detection')
ax1.set_xlabel('Date Time')
ax1.set_ylabel('AC Power')
ax1.legend()
ax1.grid(True)
st.pyplot(fig1)
```

```
# --- 3. 15-Day Future Forecast ---
st.header("3. 15-Day Future Forecast")

fig2, ax2 = plt.subplots(figsize=(16, 7))
ax2.plot(future_df.index, future_df['pred_ac'], label='Predicted AC Power (Future)', color='purple')
ax3 = ax2.twinx()
ax3.plot(future_df.index, future_df['tamper_prob'], label='Tamper Probability (Future)', color='orange',
linestyle=':')
ax2.set_title('15-Day Future AC Power Forecast and Tamper Probability')
ax2.set_xlabel('Date Time')
ax2.set_ylabel('Predicted AC Power')
ax3.set_ylabel('Tamper Probability')
fig2.legend(loc="upper left", bbox_to_anchor=(0.15,0.88))
ax2.grid(True)
st.pyplot(fig2)

# --- 4. Tampering Statistics Summary ---
st.header("4. Tampering Statistics Summary")

total_points = len(processed_df)
tampered_count = processed_df['TAMPER_STACK'].sum()
tampered_percentage = (tampered_count / total_points) * 100 if total_points > 0 else 0

st.write(f"Total data points analyzed: {total_points}")
st.write(f"Number of points identified as tampered (Stacking Model): {tampered_count}")
st.write(f"Percentage of tampered points: {tampered_percentage:.4f}%")

# To run this Streamlit app in Colab, you would typically save this code to a .py file
# and then execute it via ngrok. For direct execution and display within Colab,
# a separate setup with !streamlit run <app_file.py> and ngrok is needed.
# This block primarily creates the content of that .py file.

# Save the Streamlit app code to a file
with open('app.py', 'w') as f:
    f.write('''
import streamlit as st
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import numpy as np

sns.set_style("whitegrid")

# Accessing dataframes from the current environment (assuming they are globally available or passed)
# In a real Streamlit app, you would load these dataframes or have them passed.
# For this Colab environment, we'll assume processed_df, res_df, future_df, and AC_COL are available.

# --- Streamlit Page Configuration ---
st.set_page_config(
    page_title="Solar Power Tampering Detection & Forecast Dashboard",
    layout="wide",
    initial_sidebar_state="expanded"
)

st.title("Solar Power Tampering Detection & Forecast Dashboard")

# --- 1. Model Evaluation Metrics ---
st.header("1. Model Evaluation Metrics on Holdout Data")
st.dataframe(res_df)

# --- 2. Tampering Detection Visualization ---
st.header("2. Tampering Detection (Actual vs. Predicted AC Power)")

fig1, ax1 = plt.subplots(figsize=(16, 7))
ax1.plot(processed_df['DATE_TIME'], processed_df[AC_COL], label='Actual AC Power', color='blue',
alpha=0.7)
ax1.plot(processed_df['DATE_TIME'], processed_df['PRED_AC'], label='Forecasted AC Power',
color='green', linestyle='--', alpha=0.7)

tampered_points = processed_df[processed_df['TAMPER_STACK'] == 1]
if not tampered_points.empty:
    ax1.scatter(tampered_points['DATE_TIME'], tampered_points[AC_COL], color='red', marker='o', s=50, label='Detected Tampering (Stacking Model)', zorder=5)

ax1.set_title('AC Power: Actual vs. Forecasted with Tampering Detection')
ax1.set_xlabel('Date Time')
ax1.set_ylabel('AC Power')
ax1.legend()
ax1.grid(True)
st.pyplot(fig1)

# --- 3. 15-Day Future Forecast ---
st.header("3. 15-Day Future Forecast")

fig2, ax2 = plt.subplots(figsize=(16, 7))
ax2.plot(future_df.index, future_df['pred_ac'], label='Predicted AC Power (Future)', color='purple')
ax3 = ax2.twinx()
ax3.plot(future_df.index, future_df['tamper_prob'], label='Tamper Probability (Future)', color='orange', linestyle=':')
ax2.set_title('15-Day Future AC Power Forecast and Tamper Probability')
ax2.set_xlabel('Date Time')
ax2.set_ylabel('Predicted AC Power')
ax3.set_ylabel('Tamper Probability')
fig2.legend(loc="upper left", bbox_to_anchor=(0.15,0.88))
ax2.grid(True)
st.pyplot(fig2)

# --- 4. Tampering Statistics Summary ---
st.header("4. Tampering Statistics Summary")

total_points = len(processed_df)
tampered_count = processed_df['TAMPER_STACK'].sum()
tampered_percentage = (tampered_count / total_points) * 100 if total_points > 0 else 0

st.write(f"Total data points analyzed: {total_points}")
st.write(f"Number of points identified as tampered (Stacking Model): {tampered_count}")
st.write(f"Percentage of tampered points: {tampered_percentage:.4f}%")
```

```
'''
)
print("Streamlit app code saved to app.py")



from pyngrok import ngrok
import os

# Terminate any previous ngrok tunnels
ngrok.kill()

# Get ngrok authentication token from user input
NGROK_AUTH_TOKEN = input("Please enter your ngrok authtoken (find it at
https://dashboard.ngrok.com/get-started/your-authtoken): ")

# Set ngrok authtoken
ngrok.set_auth_token(NGROK_AUTH_TOKEN)

# Run Streamlit in background
!nohup streamlit run app.py --server.port 8501 &> streamlit.log &

# Create a public URL with ngrok
public_url = ngrok.connect(addr='8501')
print(f"Streamlit App URL: {public_url}")

# Optionally, print streamlit logs for debugging
# !cat streamlit.log
```

OUTPUT



4. Tampering Statistics Summary

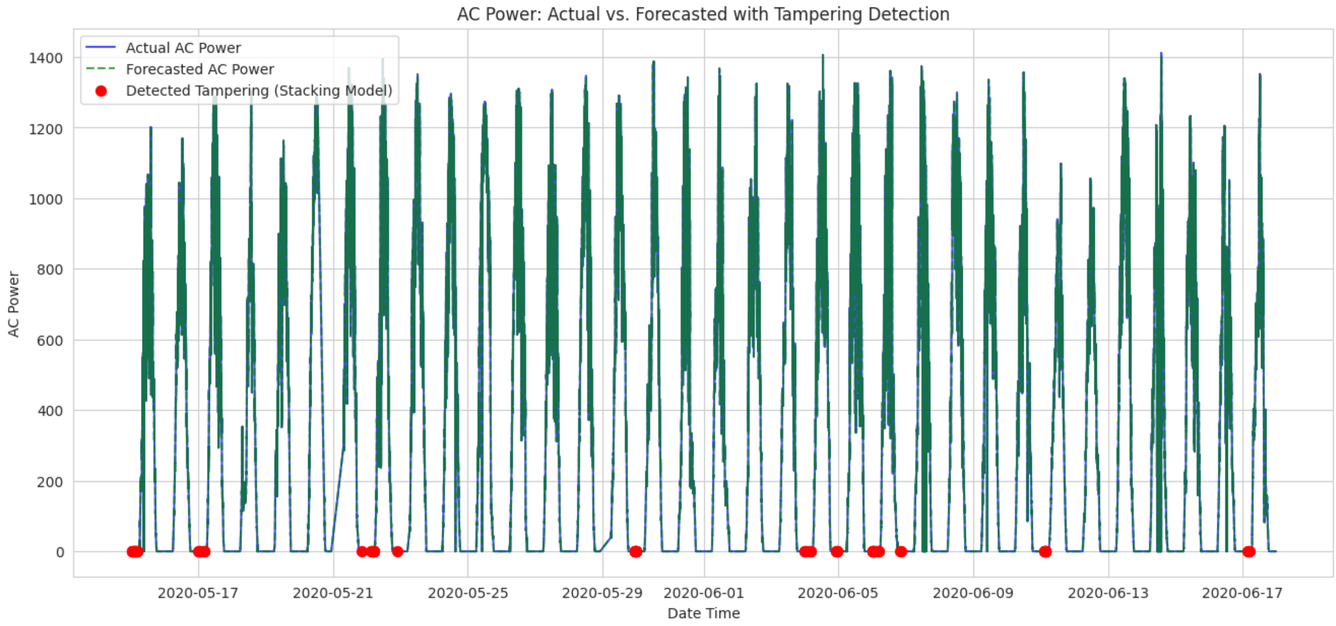
Total data points analyzed: 68778

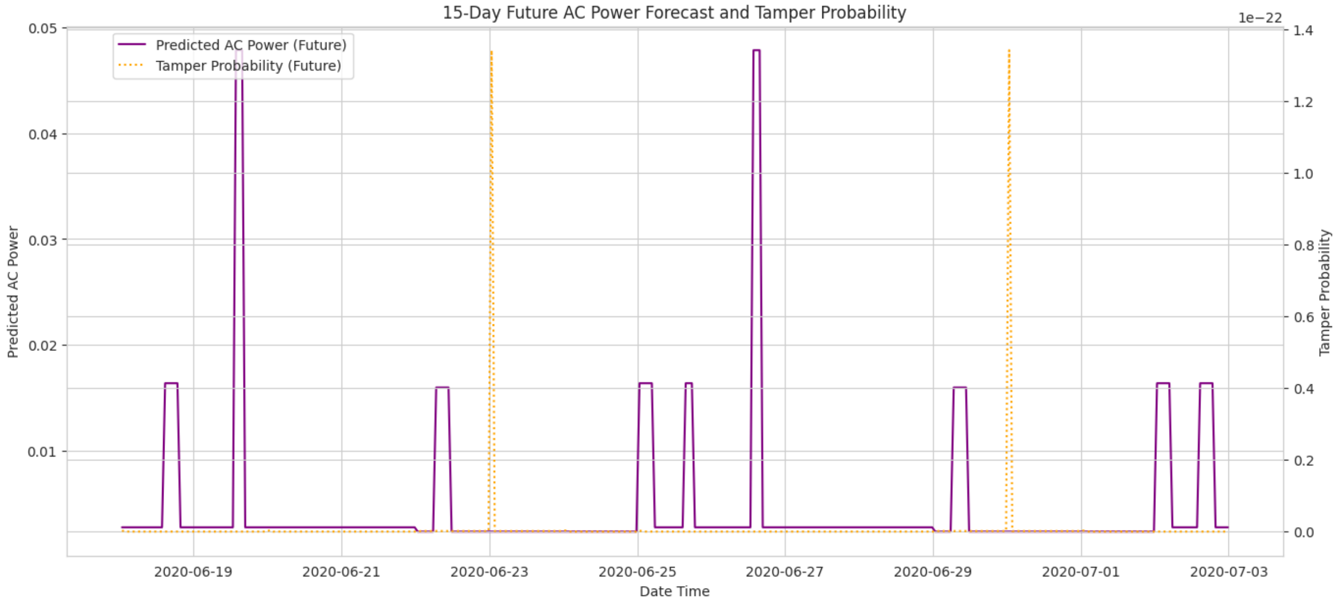
Number of points identified as tampered (Stacking Model): 104

Percentage of tampered points: 0.1512%

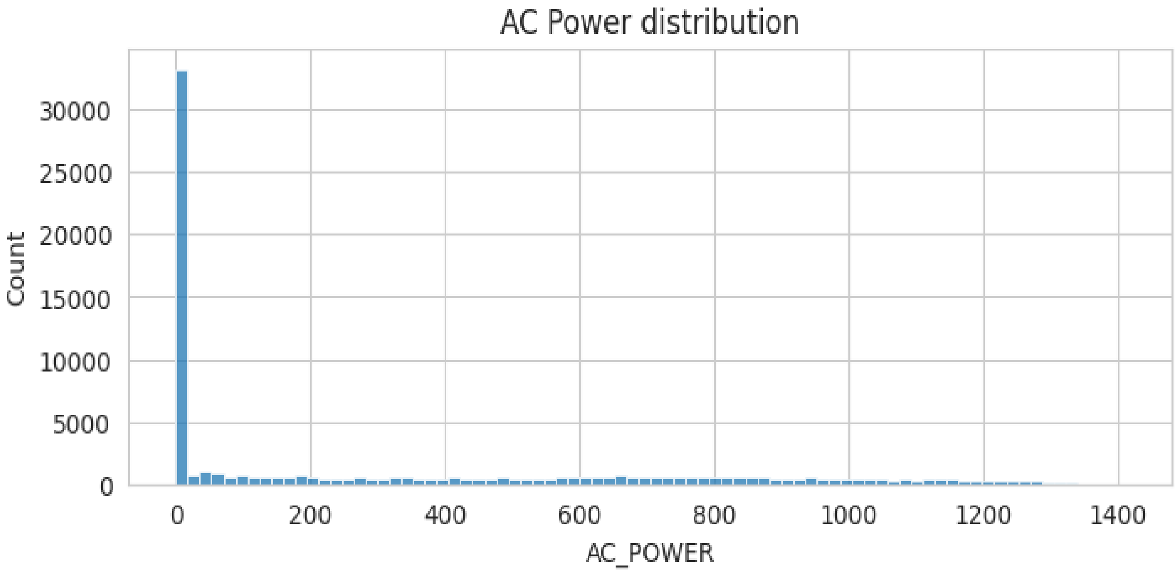
5. Details of Tampered Data Points

	DATE_TIME	AC_POWER	PRED_AC	TAMPER_PROB_STACK
39475	2020-06-04 00:45:00	0.000	0.002	1.000
39424	2020-06-04 00:00:00	0.000	0.003	1.000
39483	2020-06-04 00:45:00	0.000	0.003	1.000
39415	2020-06-04 00:00:00	0.000	0.003	1.000
208	2020-05-15 02:15:00	0.000	0.003	1.000
4186	2020-05-17 03:15:00	0.000	0.003	1.000
13442	2020-05-22 03:15:00	0.000	0.003	1.000
13565	2020-05-22 04:45:00	0.000	0.003	1.000

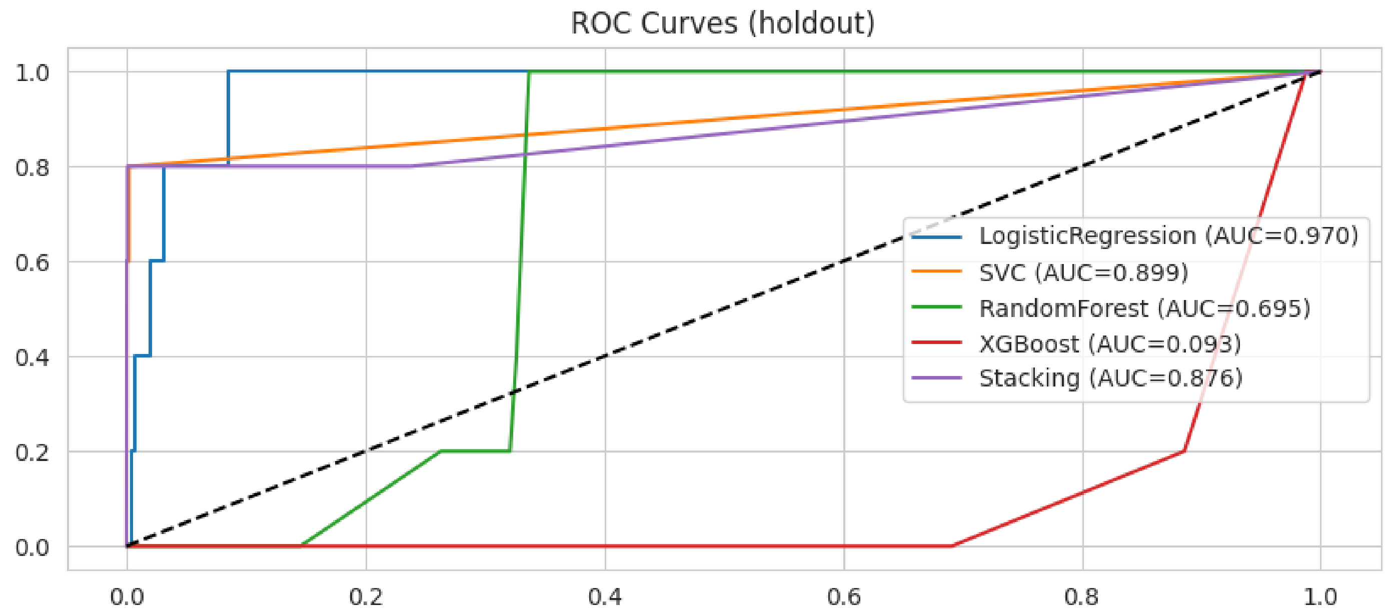


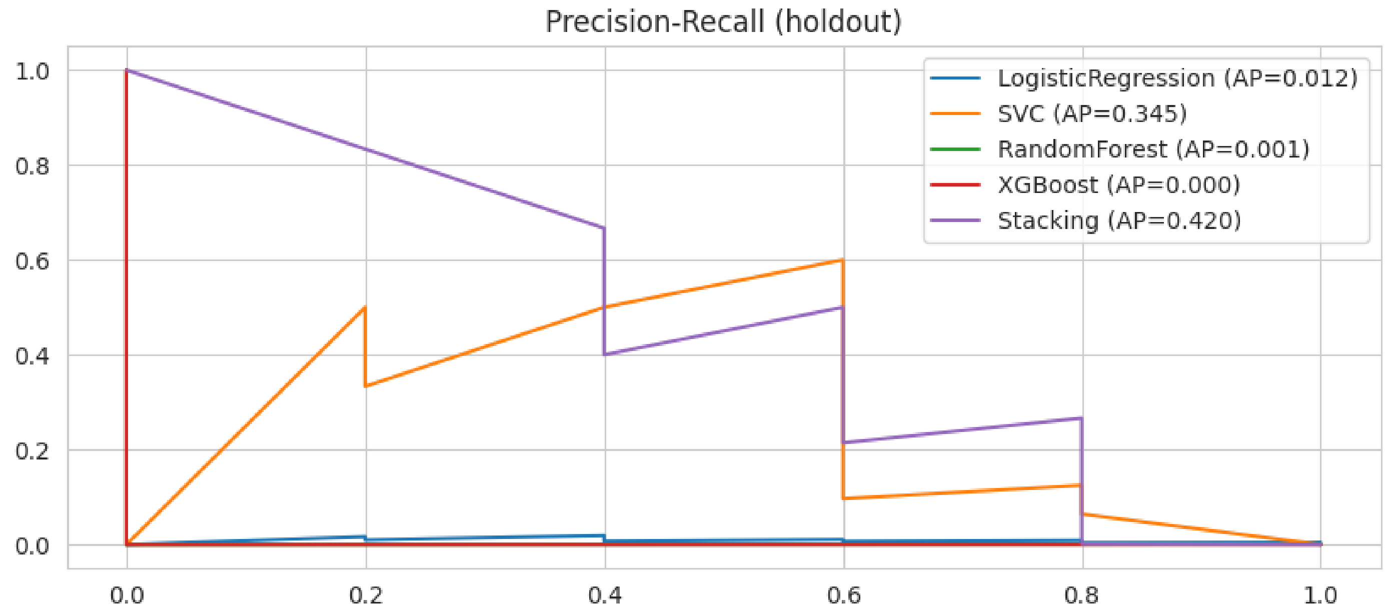


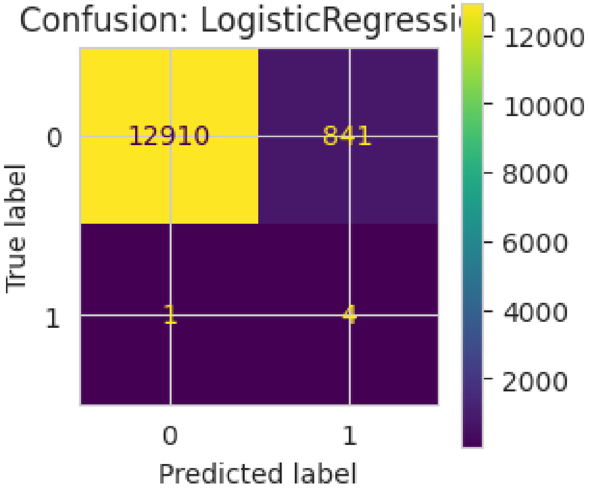
15-Day Future AC Power Forecast and Tamper Probability
Predicted AC Power (Future)
Tamper Probability (Future)
Predicted AC Power
0.05
0.04
0.03
0.02
0.01
1e−22
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Tamper Probability
2020-06-19
2020-06-21
2020-06-23
2020-06-25
2020-06-27
2020-06-29
2020-07-01
2020-07-03
Date Time

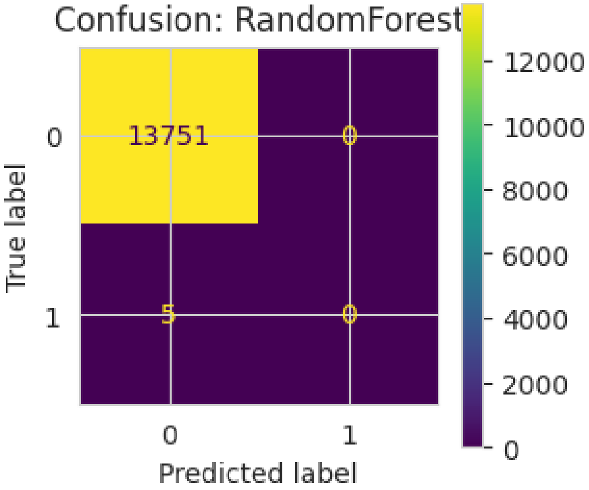


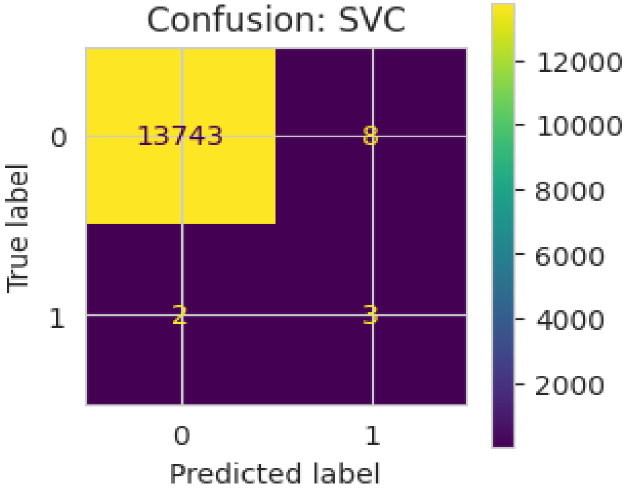
AC Power distribution
Count
30000
25000
20000
15000
10000
5000
0
0
200
400
600
800
1000
1200
1400
AC_POWER

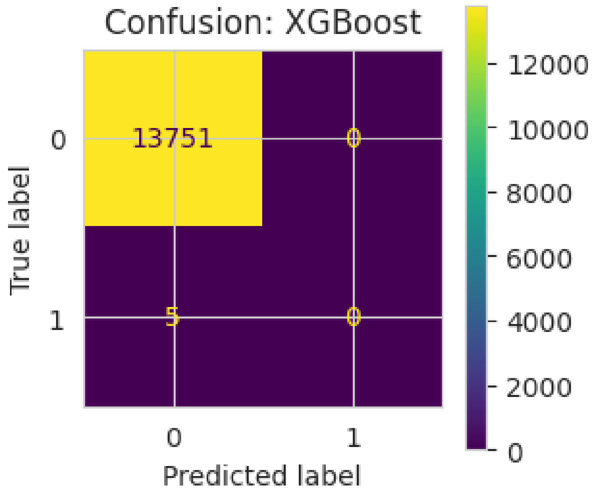

ROC Curves (holdout)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
LogisticRegression (AUC=0.970)
SVC (AUC=0.899)
RandomForest (AUC=0.695)
XGBoost (AUC=0.093)
Stacking (AUC=0.876)

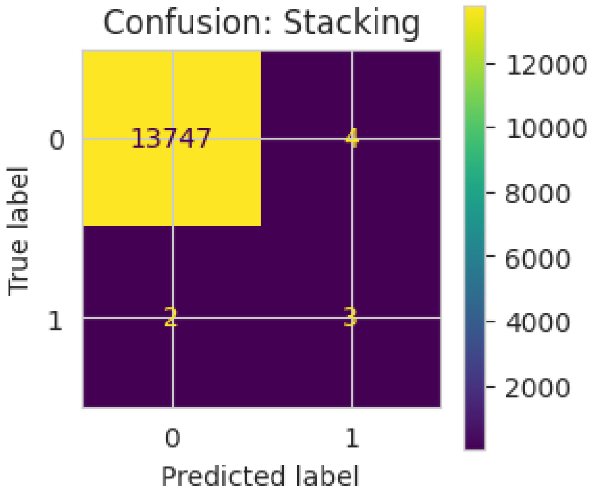

Precision-Recall (holdout)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
LogisticRegression (AP=0.012)
SVC (AP=0.345)
RandomForest (AP=0.001)
XGBoost (AP=0.000)
Stacking (AP=0.420)

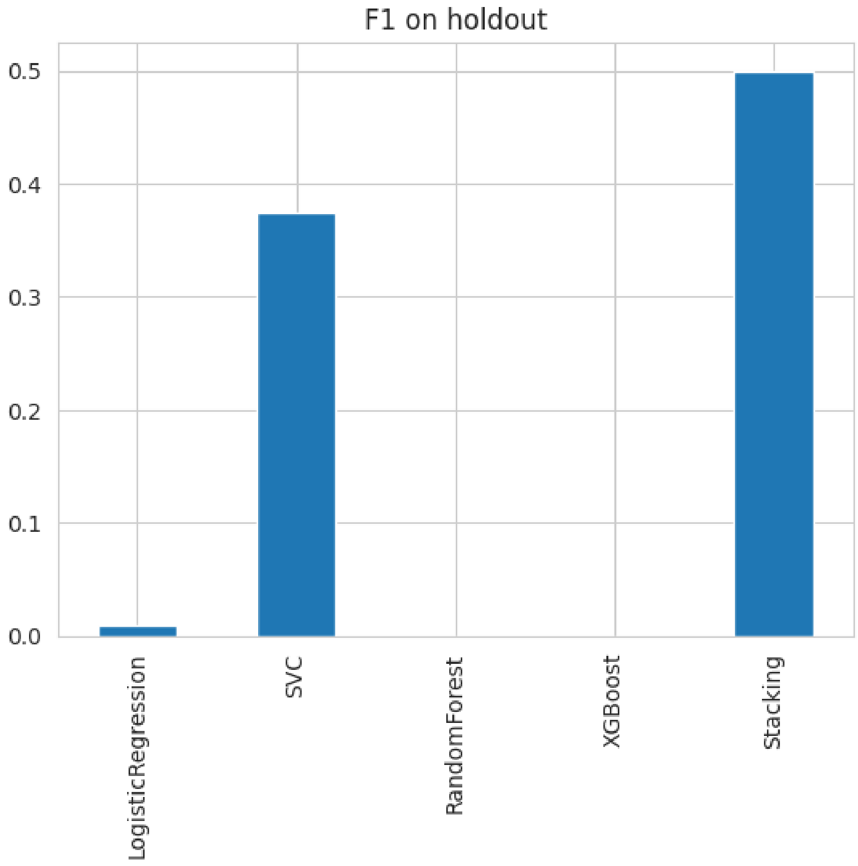

Confusion: LogisticRegression
True label
0
1
12910
841
1
4
0
1
Predicted label
12000
10000
8000
6000
4000
2000

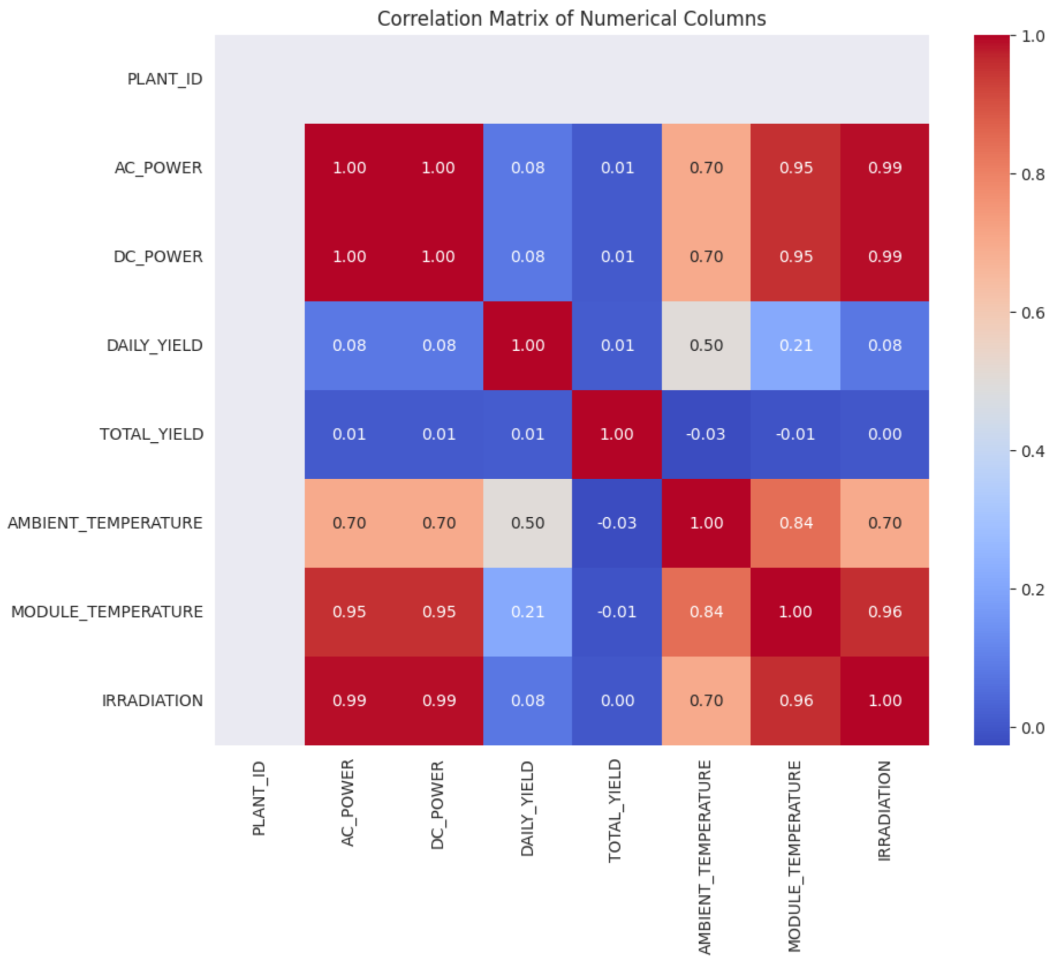

Confusion: RandomForest
True label
0
1
13751
0
5
0
0
1
Predicted label
12000
10000
8000
6000
4000
2000
0

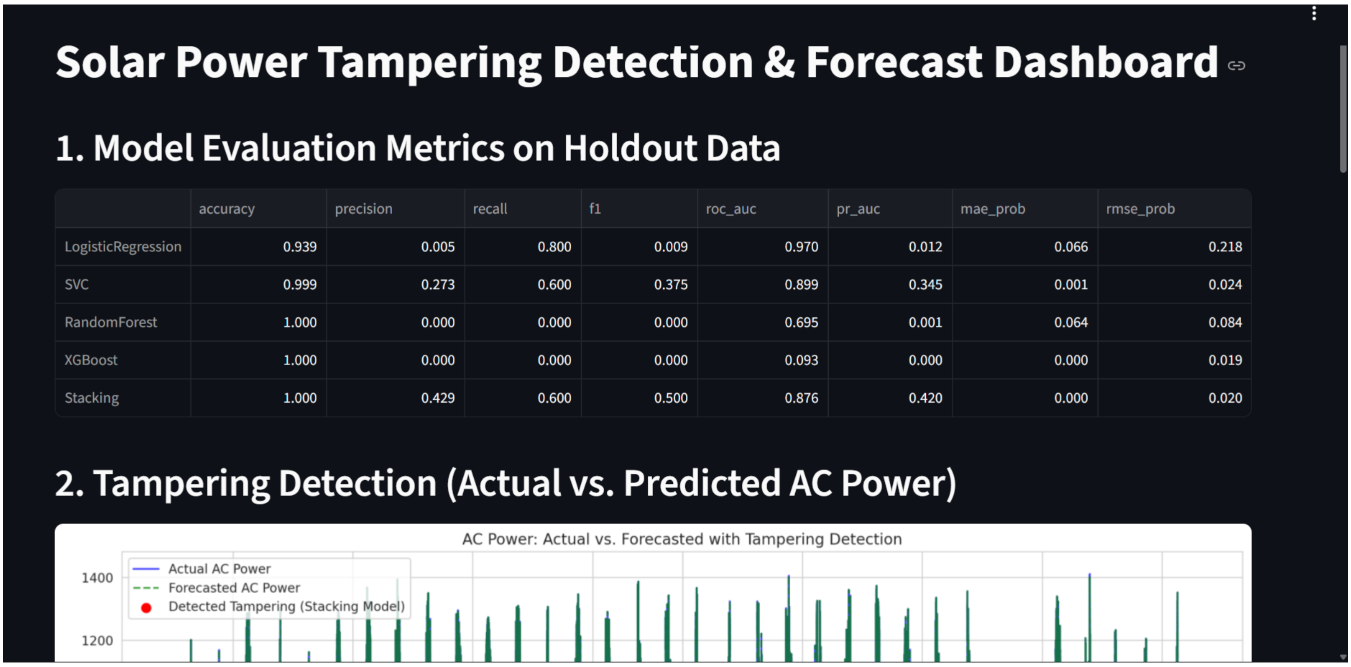

Confusion: SVC
True label
0
1
13743
8
2
3
0
1
Predicted label
12000
10000
8000
6000
4000
2000


Confusion: XGBoost
True label
0
1
13751
0
5
0
0
1
Predicted label
12000
10000
8000
6000
4000
2000
0


Confusion: Stacking
True label
0
1
13747
4
2
3
0
1
Predicted label
12000
10000
8000
6000
4000
2000


F1 on holdout
0.5
0.4
0.3
0.2
0.1
0.0
LogisticRegression
SVC
RandomForest
XGBoost
Stacking





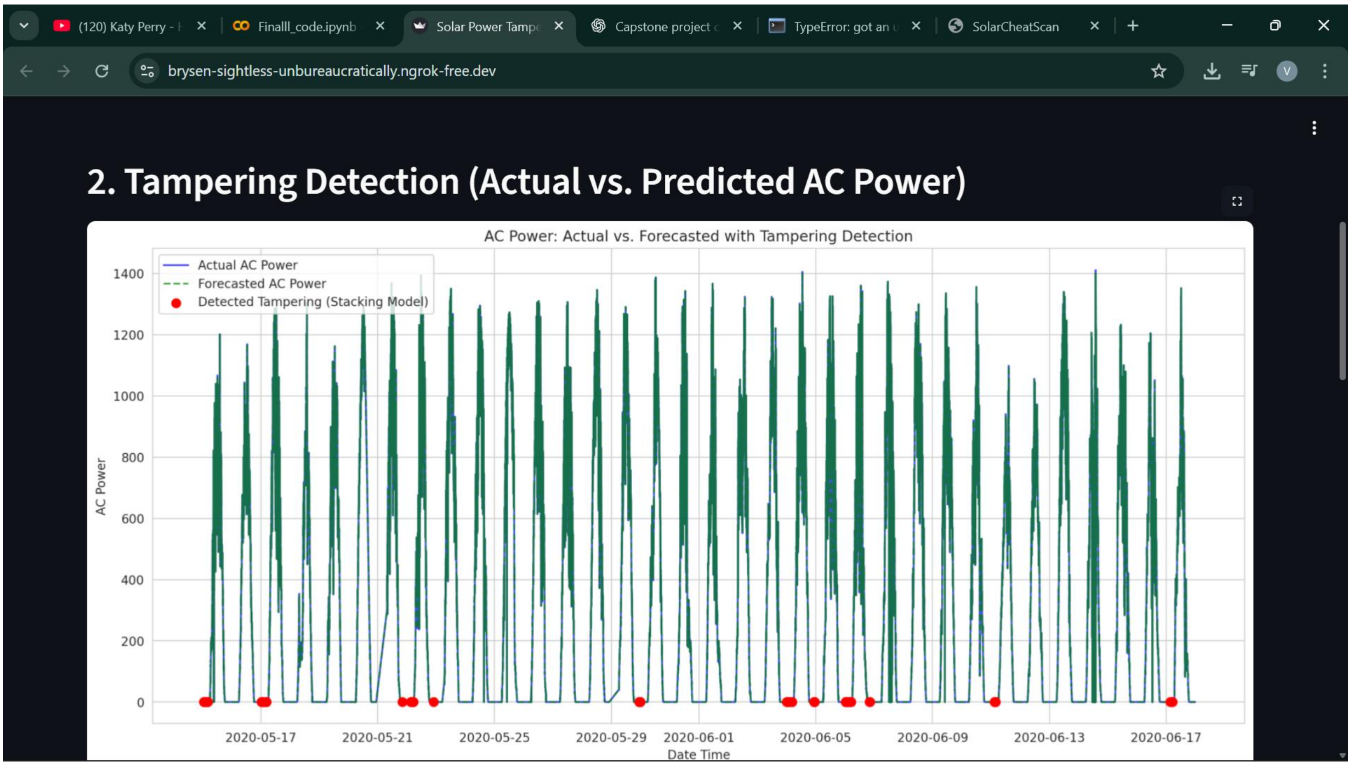
Solar Power Tampering Detection & Forecast Dashboard

1. Model Evaluation Metrics on Holdout Data

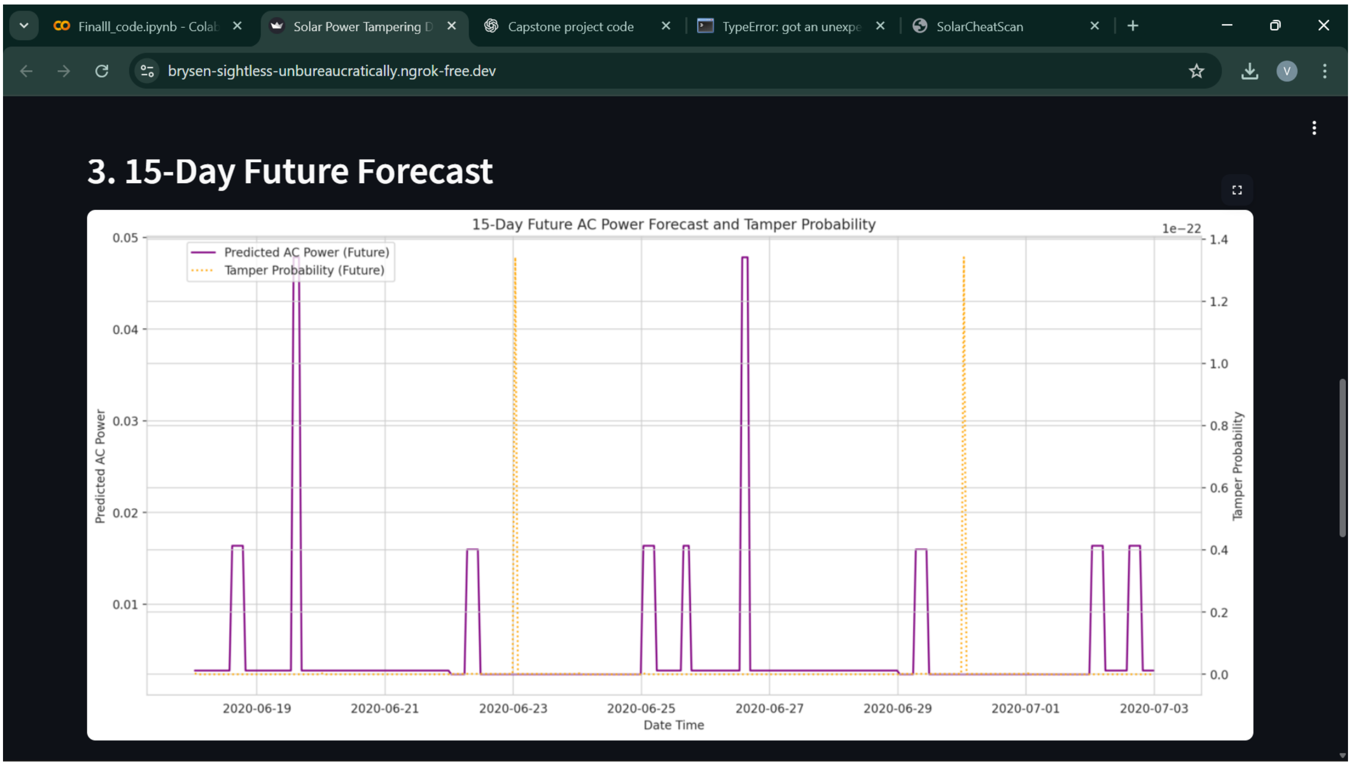
	accuracy	precision	recall	f1	roc_auc	pr_auc	mae_prob	rmse_prob
LogisticRegression	0.939	0.005	0.800	0.009	0.970	0.012	0.066	0.218
SVC	0.999	0.273	0.600	0.375	0.899	0.345	0.001	0.024
RandomForest	1.000	0.000	0.000	0.000	0.695	0.001	0.064	0.084
XGBoost	1.000	0.000	0.000	0.000	0.093	0.000	0.000	0.019
Stacking	1.000	0.429	0.600	0.500	0.876	0.420	0.000	0.020

2. Tampering Detection (Actual vs. Predicted AC Power)

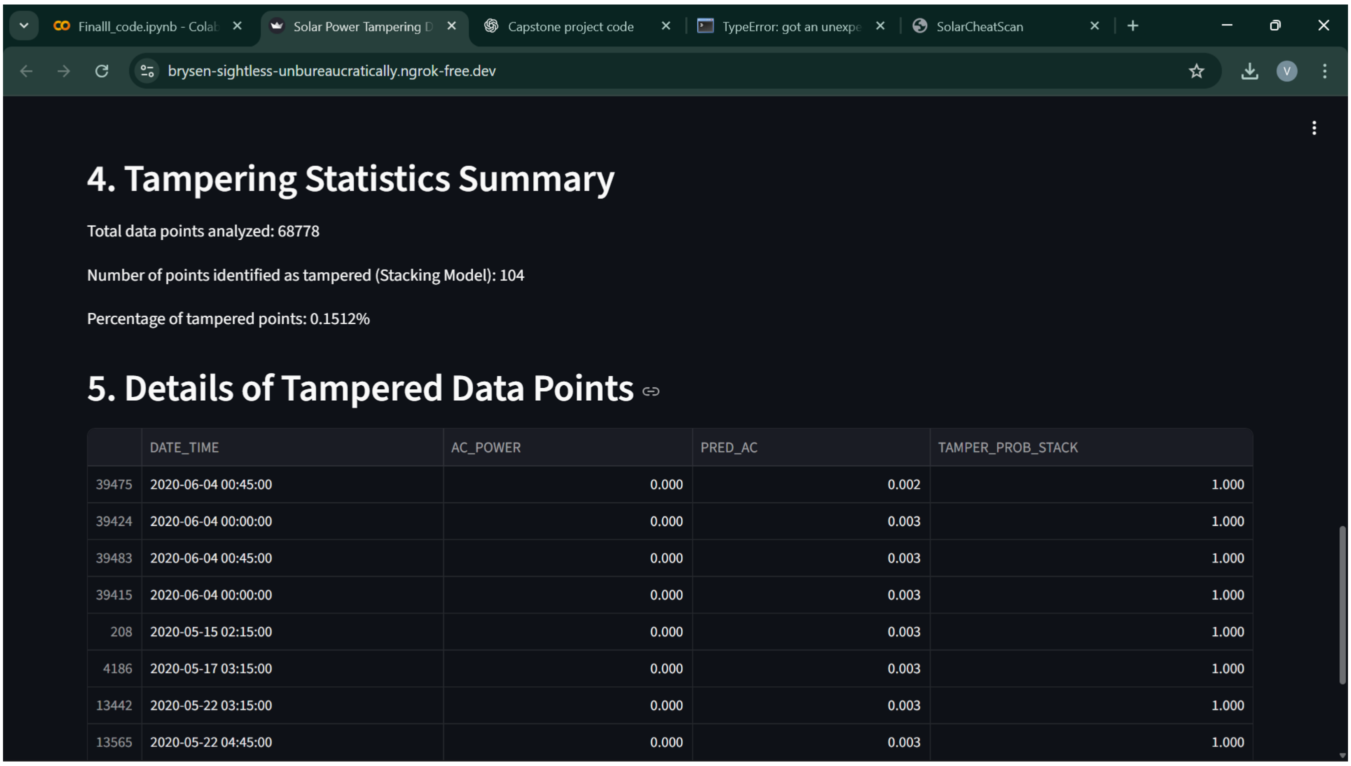
AC Power: Actual vs. Forecasted with Tampering Detection
Actual AC Power
Forecasted AC Power
Detected Tampering (Stacking Model)
1400
1200



2. Tampering Detection (Actual vs. Predicted AC Power)



3. 15-Day Future Forecast



4. Tampering Statistics Summary

Total data points analyzed: 68778

Number of points identified as tampered (Stacking Model): 104

Percentage of tampered points: 0.1512%

5. Details of Tampered Data Points

	DATE_TIME	AC_POWER	PRED_AC	TAMPER_PROB_STACK
39475	2020-06-04 00:45:00	0.000	0.002	1.000
39424	2020-06-04 00:00:00	0.000	0.003	1.000
39483	2020-06-04 00:45:00	0.000	0.003	1.000
39415	2020-06-04 00:00:00	0.000	0.003	1.000
208	2020-05-15 02:15:00	0.000	0.003	1.000
4186	2020-05-17 03:15:00	0.000	0.003	1.000
13442	2020-05-22 03:15:00	0.000	0.003	1.000
13565	2020-05-22 04:45:00	0.000	0.003	1.000

Visualization Dashboard of Tampering Detection

RESULTS

The table below summarizes the performance metrics for the machine learning models used for both the regression and classification models. The data was split temporally into training (80%) and testing (20%) sets. We have trained regression models like Gradient Boost regressor, Random Forest Regressor and XGBoost Regressor to forecast the AC power generation. The models were evaluated using RMSE, MAE, and R2 score, and the best performing model (GBR) was selected.

MODEL	RMSE	MAE	R^2 SCORE
Gradient Boost Regressor	0.661	0.209	1.000
Random Forest Regressor	0.705	0.133	1.000
XGBoost Regressor	2.732	0.780	1.000

Table-1 shows the results of the regression model

To detect tampeing by actual AC power VS forecasted AC power we used classification models. Baseline classification models (Logistic Regression, SVC, Random Forest Classifier, XGBoost Classifier) were trained and evaluated on a holdout set using metrics like accuracy, precision, recall, F1-score, ROC AUC, and PR AUC. A **stacking ensemble** was implemented, using the base classifiers' Out-of-Fold (OOF) predictions as meta-features for a LightGBM meta-learner. This stacking model provided the final TAMPER_PROB_STACK and TAMPER_STACK labels, which showed improved performance (higher F1, better PR AUC) compared to most individual base models.

MODEL	ACCURACY	PRECISION	RECALL	F1-SCORE
Logistic Regression	0.938790	0.004734	0.8	0.009412
SVC	0.999273	0.272727	0.6	0.375000
Random Forest	0.999637	0.00000	0.0	0.00000
XGBoost	0.999637	0.00000	0.0	0.00000
Stacking Ensemble	0.999564	0.428571	0.6	0500000

Table-2 shows the result of the classification model that detects tampering in the forecasted data

Here in the classification table, The F1-score can be 0.0 when a classification model, particularly in the context of highly imbalanced datasets, fails to correctly identify any positive instances. In our Project context,the Random Forest, XGBoost, and Stacking models, their F1-scores were 0.0. This indicates that, at the default classification threshold of 0.5, these models did not predict any instances as belonging to the positive class (tampering) in the holdout dataset. In other words, they had 0 True Positives, which led to 0 Precision and 0 Recall, and consequently a 0 F1-score.

This is a common issue with highly imbalanced datasets where the positive class is very rare. Models might become overly conservative and predict only the majority class (no tampering) to maximize overall accuracy, especially if not explicitly tuned to handle imbalance.

LEARNING OUTCOME

Our team gained strong technical, analytical, and practical skills through the development of the project **“**Solar Cheat Scan – Detecting Tampering in Forecasted Solar Power Generation Using ML.” Working on this system gave us a deeper understanding of how weather parameters influence solar plant performance and how data-driven techniques can improve transparency and reliability in power generation reporting.

We learned how to collect, preprocess, and organize large datasets containing weather attributes and AC power output. This experience helped us understand the importance of feature selection, train–test splitting, data cleaning, and normalization in building accurate predictive models. Using the Gradient Boost algorithm, we developed a forecasting model capable of predicting AC power output based on historical data and weather conditions. This improved both our conceptual knowledge and our hands-on skills in designing, training, and fine-tuning machine learning models.

We also gained experience in evaluating model performance using metrics such as MAE (Mean Absolute Error), RMSE (Root Mean Squared Error), and R² (Coefficient of Determination),Accuracy,Precision,Recall and F1-Score. These evaluation methods helped us measure the model's accuracy and understand how well it generalizes to new data. By comparing predicted values with actual AC power readings, we learned how to detect unusual deviations that may indicate tampering. This demonstrated how anomaly detection can support transparency and operational safety in solar energy systems .

As we worked together on research, implementation, testing, and documentation, we improved not only our technical skills, but also our teamwork, communication, problem-solving, and time-management abilities. Through this work, we established a solid basis in data analytics, machine learning, anomaly detection, and renewable energy analysis. The knowledge and expertise obtained from establishing the Solar Cheat Scan system will help us with future academic research and offer up new options in domains like artificial intelligence, smart infrastructure management, and sustainable energy technologies.

FUTURE SCOPE

The Solar Cheat Scan system can be further improved and extended in several meaningful ways to enhance its accuracy, reliability, and real-world applicability:

1. **Real-Time Monitoring Integration:** Connecting the system with real-time data streams from solar plants to detect tampering instantly rather than through offline comparison.
2. **Advanced Forecasting Models:** Incorporating deep learning techniques such as LSTM, GRU, or CNN–LSTM hybrids to improve prediction accuracy during rapidly changing weather conditions.
3. **Dashboard for Visualization:** Adding a web-based dashboard to visualize predicted power, actual power, deviations, and alert notifications in an intuitive format.
4. **Multi-Plant Support:** Expanding the system to monitor multiple solar plants simultaneously and provide centralized analytics for large-scale installations.
5. **Blockchain Integration for Security:** Using blockchain to create tamper-proof logs of predicted and actual values, increasing trust and data integrity.
6. **Fault Detection Expansion:** Extending the system to detect issues such as equipment failures, shading problems, or abnormal system behavior in addition to tampering.

7. **Automated Alert and Reporting System:** Enabling SMS, email, or app notifications when suspicious deviations are detected, along with automated weekly and monthly reports.
8. **API Integration for Smart Grids:** Allowing the system to communicate with smart grid infrastructures, energy auditors,and government monitoring systems.

By implementing these improvements, Solar Cheat Scan can evolve into a powerful and intelligent monitoring platform that enhances transparency, strengthens security, and supports efficient decision-making in the renewable energy sector.

CONCLUSION

The project "Solar Cheat Scan – Detecting Tampering in Forecasted Solar Power Generation Using ML" successfully demonstrated the application of machine learning-based forecasting to enhance transparency and reliability in solar power generation. By predicting AC power output using weather parameters and comparing it with real-time generation data, the system effectively detected unusual deviations that could indicate tampering or inefficiencies.

The assessment metrics—MAE, RMSE, and R² for regression model and Accuracy,Precision,Recall,F1-score for classification model—helped gauge the accuracy and overall performance of the Gradient Boosting Regressor which was trained to forecast the AC power Generation and also Stacking Ensemble model which was implemented using the base classifiers Out-of-Fold (OOF) predictions as meta-features for a LightGBM meta-learner, which was used to detect tampering in Actual AC power VS Forecasted AC power and also predicted the next 15 days AC power generation using these models along with tampering probability. The created solution enhances the operational integrity of solar farms, supports data-driven decision-making, and improves monitoring efficiency.

CHALLENGES FACED

During the course of the project, the team encountered several challenges which provided valuable learning experiences:

1. Data Quality and Availability: One of the first challenges was obtaining comprehensive and high-quality datasets. Careful preprocessing was necessary to guarantee dependable model training because of missing values, inconsistent formats, and noisy input.

2.**Model Optimization**: It took several trials and iterations to choose the right features and adjust the model's parameters to improve forecasting accuracy.

3.Handling Weather Variability: Since solar power generation depends heavily on dynamic weather conditions, sudden and unpredictable changes posed challenges in maintaining stable prediction accuracy.

4.Detection Threshold Selection: Determining what qualifies as "tampering" versus normal operational variation required detailed analysis and testing to avoid false alarms or missed detections.

5.Visualization and Real-Time Integration: Additional work was needed in data presentation and output validation to convey results and anomaly detections in an understandable fashion.

REFERENCE

1."Machine Learning Schemes for Anomaly Detection in Solar Power Plants" by M. Ibrahim et al. Vol. 15, No. 3, Energies, 2022.

Available at: https://www.mdpi.com/1996-1073/15/3/1082

2.Srivastava, R., Tiwari, A. N., and Giri, V. K. Heliyon, "Solar radiation forecasting using MARS, CART, M5, and Random Forest model: A case study for India" 5(10): e02692, 2019.
Accessible at: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6838948/

3."Machine Learning Models for Forecasting Solar Power Generation Using Single and Multiple Weather/Storage Features" Sustainability, Volume 16, Issue 14, 2024.

Available at: https://www.mdpi.com/2071-1050/16/14/6087

4."Solar Power Prediction Using Machine Learning." arXiv preprint, arXiv:2303.07875, 2023. Subramanian, E., Mithun Karthik, M., Prem Krishna, G., Vaisnav Prasath, D., & Sukesh Kumar,

Available at: https://arxiv.org/abs/2303.07875

5."Predictive Modeling and Anomaly Detection in Solar PV Inverters Using Machine Learning" by Francisti, J., Fodor, K., Balogh, Z., and Magdin, M. SSRN (2025).

Available at: https://ssrn.com/abstract=5490889

6.Abdelsattar, M., AbdelMoety, A., & Emad-Eldeen, A. “Advanced Machine Learning Techniques for Predicting Power Generation and Fault Detection in Solar Photovoltaic Systems.” Neural Computing & Applications, 37, 8825–8844, 2025.

Available at: https://link.springer.com/article/10.1007/s00521-025-11035-6