

IoT-Based Low-Cost Renewable Energy Prediction System Using Machine Learning

Pranjal Singla¹, Rakshit Anand¹, Yash Gusain¹, Neha Gupta²

¹ Student of School of Engineering and Technology, Sushant University, Gurgaon, Haryana, India.

² Associate Professor of School of Engineering and Technology, Sushant University, Gurgaon, Haryana, India.

ABSTRACT

So here's the thing—renewable energy is everywhere now. Especially solar. But there's a catch. It's not really reliable. You never know how much you're going to get because it all depends on the weather—sunlight, temperature, clouds, you name it. That unpredictability is a headache if you're trying to actually use solar power seriously.

In this project, we basically tried to build something cheap and simple that could predict solar energy output in real time, without needing expensive lab equipment. We used an ESP32 microcontroller, a DHT11 sensor for temperature and humidity, and a simple LDR (light sensor) to estimate sunlight. The whole thing costs less than ten dollars. That's it.

Then we trained three machine learning models—Linear Regression, Random Forest, and Gradient Boosting—on a small solar dataset. And funny enough, the simplest one, Linear Regression, worked best for us. Not what we expected, but that's how it turned out.

We also made a backend using FastAPI so you can get predictions, see past data, and even look at a 24-hour forecast using ARIMA. Plus, we added a feature that shows how much CO₂ you're saving. All of this is shown on a basic web dashboard.

Honestly, the main takeaway is that you don't need fancy hardware to build something useful. Even basic stuff can give you decent predictions.

Keywords: Internet of Things; renewable energy prediction; machine learning; solar energy forecasting; ESP32 microcontroller; FastAPI

1. Introduction

The way we get electricity is changing. Earlier, most of it came from burning coal or other non-renewable sources. But now, solar power is becoming a big deal. In fact, in many parts of the country, you see solar panels on rooftops, farms, even streetlights.

But there's a problem that nobody seems to talk about enough. Solar energy is unpredictable. Like, really unpredictable. If a cloud passes by, your power output drops. If it's too hot, panels become less efficient. Because of this, planning becomes a nightmare—grid operators don't know how much power will be available, and regular users can't rely on it fully.

This is where machine learning comes in. Instead of guessing, you can use past data to predict future output. If you know how temperature and sunlight affected output yesterday, you can make a smart guess for tomorrow. A lot of researchers have already done work on this, but most of their systems either use expensive sensors or don't work with live data.

So we thought—why not build something simple? We took an ESP32 (a tiny, cheap chip with WiFi), added a few basic sensors, and connected everything to a machine learning model. The goal was to make something that anyone could replicate without burning a hole in their pocket.

2. Literature Review

A lot of smart people have already tried to solve this problem. But if you really dig into the papers, most of them focus only on the math—the algorithms—and forget about the hardware side.

For example, Yadav and Sharan [1] used Artificial Neural Networks and found that temperature and sunlight are the biggest factors affecting solar output. That’s exactly why we picked those two as our main inputs.

Sharma et al. [2] compared different models like SVR, ANN, and decision trees. They found that SVR works well when data is noisy. But their setup used expensive instruments like pyranometers, which cost hundreds of dollars. That’s just not practical for a hobbyist or a small setup.

Eze et al. [3] worked on forecasting using ARIMA models. They showed that ARIMA is actually pretty decent for short-term predictions—like a day or two ahead. So we borrowed that idea and added it to our system.

Kapoor and Bedi [4] talked about IoT systems and suggested that cheap devices like ESP32 are good for collecting data, but the heavy processing should be done on a server. We followed that advice exactly.

Then there’s Kalogirou [5], who made a really interesting point. He said that sometimes, simple models like Linear Regression can beat complex ones. And honestly? That’s exactly what happened with us. We were surprised too.

Hossain et al. [6] showed that Gradient Boosting also gives good results, which is why we included it in our comparison even though it didn’t perform best for us.

Some other papers talked about blockchain for secure energy systems [9][11]. Not directly related to our work, but useful ideas for the future.

So yeah, most research either focuses on models or hardware, but rarely both. That’s what we tried to do—bring everything together.

3. Methodology

We divided the whole system into four parts: hardware, machine learning, backend, and frontend. Let me explain each one in plain terms.

3.1 Hardware Implementation

We wanted it cheap. Really cheap. So we picked an ESP32 DevKit V1 as the brain. It costs about 4 dollars, has built-in WiFi, and can read sensors easily. For temperature and humidity, we used the DHT11. It’s not super accurate—temperature error is about $\pm 2^{\circ}\text{C}$ and humidity error is $\pm 5\%$ —but it’s good enough for a rough estimate. And it costs just 1 dollar.

For sunlight, we used a simple LDR (Light Dependent Resistor) with a $10\text{k}\Omega$ resistor in a voltage divider setup. The LDR gives an analog voltage that changes with light intensity. We connected it to the ESP32’s ADC pin, which converts the voltage (0–3.3V) to a number between 0 and 4095. Then we used a rough formula to convert that to irradiance in W/m^2 : $\text{irradiance} = \text{ldr_voltage} \times 500$. Is it perfect? No. But it gives a relative measure, and that’s enough for the ML model to learn patterns.

Here’s the total cost breakdown:

Table 1: Hardware Components and Estimated Cost

<i>Component</i>	<i>Purpose</i>	<i>Qty</i>	<i>Est. Cost (USD)</i>
------------------	----------------	------------	------------------------

<i>ESP32 DevKit</i>	Microcontroller + WiFi	1	4.00
<i>DHT11 Sensor</i>	Temperature & Humidity	1	1.00
<i>LDR (GL5516)</i>	Light sensor	1	0.20
<i>10kΩ Resistor</i>	For LDR circuit	1	0.05
<i>Breadboard + wires</i>	For connections	1	1.50
<i>USB Cable</i>	Power and data	1	1.00

We wrote the code for ESP32 using the Arduino IDE (C++). It just keeps reading the sensors and sends the data over USB serial at 115200 baud. On the computer side, a Python script reads this serial stream using the pyserial library and stores the latest values. A background thread handles this so the main server doesn't freeze.

Here's a simple diagram of how everything connects:

SYSTEM ARCHITECTURE — IoT + ML Renewable Energy Prediction

DHT11 Sensor (Temp + Humidity) →
LDR Sensor (Irradiance) →
ESP32 WiFi MCU
▼ Serial / USB (115200 baud) ▼
FastAPI REST Backend ↔ ML Models (LR / RF / GB) ↔ SQLite DB
▼ HTTP / JSON REST API ▼
HTML Dashboard ↔ Chart.js (Forecast Chart) ↔ OpenWeather (Live Weather)

Figure 1: System Architecture Block Diagram

3.2 Software and Algorithm

We had three main software parts: training the ML models, building the API, and making the dashboard.

Training Data

We used a small CSV file (*solar_dataset.csv*) with 10 records. I know, that's tiny. But for a prototype, it works. The fields were temperature (°C), humidity (%), irradiance (W/m²), and power_output (W). Here's the data:

Table 2: Training Dataset Sample (*solar_dataset.csv*)

<i>Temperature (°C)</i>	<i>Humidity (%)</i>	<i>Irradiance (W/m²)</i>	<i>Power Output (W)</i>
30	60	800	230
32	55	820	240
28	65	750	210
35	50	900	270
29	70	780	220
31	58	810	235

33	53	850	255
27	68	720	200
34	52	880	265
26	72	700	190

We used temperature, humidity, and irradiance as inputs (X), and power_output as the target (y). Split was 80% train, 20% test, with a fixed random seed (42) so results are repeatable.

We trained three models using scikit-learn:

- **Linear Regression** – basic, no tuning
- **Random Forest** – 100 trees
- **Gradient Boosting** – default settings (100 stages, learning rate 0.1)

Then we saved all three models into one file (models.pkl) using joblib. We also calculated RMSE, MAE, and R^2 for each model and saved them to metrics.json.

24-Hour Forecast

For forecasting, we used ARIMA with order (2,1,2). That means 2 autoregressive terms, 1 differencing, and 2 moving average terms. We picked this based on Eze et al. [3]. Every time someone hits the /forecast endpoint, the model re-fits and spits out 24 hourly predictions.

CO₂ Savings

We used India's grid emission factor—0.82 kg CO₂ per kWh—from the Central Electricity Authority [7]. So for every kWh predicted, we multiply by 0.82 to show how much CO₂ is saved.

Here's the overall workflow:

MACHINE LEARNING WORKFLOW

START: Raw Sensor Data (Temperature, Humidity, LDR Voltage)



Data Preprocessing — Normalize & Clean



Feature Engineering — Irradiance Estimation from LDR



Split Dataset (80% Train / 20% Test)



Train Three Models: Linear Regression | Random Forest | Gradient Boosting



Evaluate: RMSE, MAE, R^2 Score



Select Best Model — Linear Regression ($R^2 = 0.971$)



Serialize Models → models.pkl



FastAPI /predict Endpoint — Real-time Inference



Store Prediction → SQLite | Return Energy (kWh) + CO₂ Saved (kg)



END: Dashboard Visualization + 24-Hour ARIMA Forecast

Figure 2: Machine Learning Workflow Flowchart

3.3 Backend API Architecture

We built the backend using FastAPI. It's fast, easy to use, and automatically generates documentation. We enabled CORS so the web dashboard (running on a different port) can talk to the API. SQLite was chosen for storage because it's lightweight and doesn't need a separate database server.

Here are all the API endpoints:

Table 3: FastAPI REST Endpoint Summary

Method	Endpoint	Input	Output
POST	/predict	Sensor input + model name	Predicted kWh + CO ₂ saved
GET	/metrics	No input	Returns RMSE, MAE, R ² for all models
GET	/forecast	No input	24-step ARIMA forecast list
GET	/history	No input	Last 20 predictions from DB
GET	/weather/{city}	City name	Live temp, humidity, irradiance
GET	/forecast/{city}	City name	24-hour weather forecast
GET	/sensor	No input	Live ESP32 sensor readings

For the /weather/{city} endpoint, we call OpenWeatherMap API. Cloud cover is converted to irradiance using: $\text{irradiance} = 1000 \times (1 - \text{cloudiness}/100)$. Again, not perfect, but good enough for a demo.

The /sensor endpoint returns the latest reading from the ESP32, so the dashboard can auto-fill inputs with one click.

3.4 Frontend Dashboard

The dashboard is a single HTML page with a sidebar and four sections: Prediction, Model Metrics, Forecast, and History. The prediction panel has three buttons: Predict (manual entry), Use Live Weather (auto-fills from OpenWeatherMap), and Use Sensor Data (auto-fills from ESP32). The metrics section shows the performance table. The forecast section draws a line chart using Chart.js. The history section shows past predictions.

We kept the design simple—dark sidebar, clean cards, plain CSS. No heavy frameworks.

4. Results

4.1 Model Performance Metrics

We tested all three models on the 20% test data. Here's what we got:

Table 4: Regression Model Performance Metrics on Test Set

Model	RMSE	MAE	R ² Score	Rank
Linear Regression	2.12	2.10	0.971	#1
Random Forest	4.12	3.83	0.891	#2

Gradient Boosting | 5.75 5.63 0.789 #3

Linear Regression scored an R^2 of 0.971—meaning it explains 97% of the variation in power output. Its errors are small, around 2 watts on average. Random Forest was second, and Gradient Boosting came last.

Honestly, we expected the ensemble models to do better. But Kalogirou [5] was right—simple models can win when the relationship is nearly linear. And in solar power, irradiance and output are almost linear.

The ARIMA(2,1,2) model gave us a smooth 24-hour forecast. It showed output dropping at night and rising in the morning, which makes physical sense. We didn’t formally evaluate forecast accuracy because of the small dataset, but that’s something we want to do later.

4.2 Comparison with Related Works

Here’s how our system compares with the literature:

Table 5: Feature Comparison — Proposed System vs. Related Works

<i>Feature</i>	<i>Related Works [1–6]</i>	<i>Proposed System</i>
<i>Hardware Cost</i>	High (dedicated hardware)	Low (~\$8)
<i>ML Models Used</i>	Single model (LSTM/SVM)	Three models compared (LR, RF, GB)
<i>Real-time IoT Feed</i>	Mostly simulated	Live sensor feed via serial
<i>ARIMA Forecasting</i>	Not integrated	24-hour forecast built in
<i>Weather API</i>	Not integrated	OpenWeatherMap live auto-fill
<i>CO₂ Estimation</i>	Not present	Included per prediction
<i>Dashboard</i>	Standalone scripts	Full interactive web dashboard
<i>Deployment</i>	Cloud-only	Local + cloud ready
<i>Best R^2</i>	~0.85–0.92	0.971 (Linear Regression)
<i>Open Source</i>	Partially	Fully open, modular code

5. Discussion

So here’s something interesting. Linear Regression beat Random Forest and Gradient Boosting. Why? I think it’s because the relationship between irradiance, temperature, and power output is almost linear in nature. If you double the sunlight, you roughly double the power. Ensemble methods are great for complex, messy relationships, but for simple linear patterns, they can actually overfit, especially with a tiny dataset like ours (just 10 records).

The LDR-based irradiance estimation is definitely the weakest link in the hardware. LDRs respond to visible light, not specifically the wavelengths that solar panels use. Also, their response is not perfectly linear. But since we trained the ML model on the same LDR values, it learned to compensate. For absolute accuracy, you’d need a real pyranometer, but that defeats the low-cost goal.

The OpenWeatherMap integration is actually pretty useful. If someone doesn't have the hardware, they can just type a city name and get an estimate. The conversion from cloudiness to irradiance is crude— 1000 W/m^2 times $(1 - \text{cloudiness}/100)$ —but again, it's good enough for a demonstration.

ARIMA worked fine, but we hardcoded the order (2,1,2) based on Eze et al. [3]. In a real system, we'd use auto-ARIMA to pick the best order dynamically. Also, refitting on every request is slow. In production, a daily batch job would be better.

The CO₂ savings feature is simple but effective. It's just multiplication, but seeing "You saved X kg of CO₂" makes the dashboard feel more meaningful, especially for non-technical users.

Now, the elephant in the room: our dataset is tiny. Only 10 records. That's fine for a prototype, but for real deployment, you'd need thousands of hourly readings. We're aware of this limitation. Future work should focus on continuous logging and expanding the dataset.

Blockchain integration, as discussed by Dargan et al. [11] and Dewan et al. [9], could be useful for making prediction records tamper-proof. And Gupta's work on smart grids [15][16] suggests ways to connect systems like ours to grid management platforms.

6. Conclusion

We built a complete, low-cost IoT system for predicting solar energy output. It uses an ESP32, DHT11, and LDR—total cost under ten dollars. The backend is FastAPI, with three ML models trained and compared. Linear Regression performed best with $R^2 = 0.971$. We also added a 24-hour ARIMA forecast, live weather auto-fill, CO₂ savings estimation, and a web dashboard.

The system is not perfect. The LDR is crude, the dataset is small, and the ARIMA order is hardcoded. But it works. And more importantly, it shows that you don't need expensive gear to get into renewable energy prediction. This is especially useful for students, hobbyists, and rural areas where cost is a barrier.

In the future, we want to expand the dataset, add more features (wind speed, UV index), use auto-ARIMA, and experiment with LSTM models. Cloud deployment and a mobile-friendly dashboard would also be great. And maybe blockchain for data integrity.

Overall, this was a fun, hands-on project that proved a simple point: cheap hardware + basic ML can still be surprisingly effective.

References

- [1] Yadav, A. K., & Sharan, H. (2014). Comparative study of different models for solar radiation prediction. *Renewable and Sustainable Energy Reviews*, 38, 472–481.
- [2] Sharma, N., Bhargava, R., & Purohit, G. (2019). Comparison of SVR, ANN, and decision tree for solar power prediction using rooftop PV data. *Procedia Computer Science*, 167, 2096–2104.
- [3] Eze, I. C., & Maduka, J. (2021). ARIMA and hybrid ARIMA-ML models for short-horizon solar energy forecasting. *Energy Reports*, 7(S6), 1214–1225.
- [4] Kapoor, S., & Bedi, J. (2020). IoT architectures for renewable energy monitoring: A survey of low-cost embedded systems. *IEEE Internet of Things Journal*, 7(10), 9741–9752.
- [5] Kalogirou, S. A. (2001). Artificial intelligence for the modeling and control of combustion processes: A review. *Progress in Energy and Combustion Science*, 29(6), 515–566.

- [6] Hossain, M., Mekhilef, S., Danesh, M., Olatomiwa, L., & Shamshirband, S. (2017). Application of extreme learning machine for short term output power forecasting of three grid-connected PV systems. *Journal of Cleaner Production*, 167, 395–405.
- [7] Central Electricity Authority, Government of India. (2022). *CO₂ Baseline Database for the Indian Power Sector — Version 17.0*. Ministry of Power, New Delhi.
- [8] Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. *International Conference on Learning Representations (ICLR 2015)*. arXiv:1412.6980.
- [9] Dewan, S., Duhan, L., & Gupta, N. (2021). Secure electronic voting system based on mobile app and blockchain. *Recent Advances in Computer Science and Communications*, 14(9).
- [10] Dargan, J., & Gupta, N. (2024). Integration of blockchain for IoT communication. In *Adaptive power quality for power management using smart technologies* (pp. 319–343). CRC Press, Taylor & Francis.
- [11] Dargan, J., Gupta, N., & Singh, L. (2021). Blockchain based energy management system: A proposed model. In *2021 International Conference on Technological Advancements and Innovations (ICTAI)* (pp. 510–514).
- [12] Mahla, S. K., Gupta, N., Papreja, A., & Mahla, S. K. (2017). Experimental studies of utilization of biogas with biodiesel in diesel engine generator for electricity generation. *International Journal of Engineering Research & Science*, 3(2), 24–29.
- [13] Mahla, S. K., Dhir, A., & Gupta, N. (2017). Emission characteristics of diesel engine fuelled with biogas and n-propanol–biodiesel–diesel blend under dual fuel mode. *International Journal of ChemTech Research*, 10(13), 1–8.
- [14] Gupta, N., & Pahuja, S. (2017). Vibration sensor for health monitoring of electrical machines in power station. *International Journal of Engineering Science and Technology*.
- [15] Gupta, N. (2024). An overview of e-mobility-based threats to the power grids: Introduction to smart grids and mobility ecosystem. In *E-mobility in electrical energy systems for sustainability* (pp. 142–156). IGI Global.
- [16] Gupta, N. (2024). Electrifying transportation and pioneering the future of sustainable electricity mobility. In *E-mobility in electrical energy systems for sustainability* (pp. 80–89). IGI Global.