

**19EEE331 Smart Grid & IOT**

**IoT driven Peak load shifting using battery system**

**A Project Report**

**Submitted by**

CB.EN.U4EEE22062 V SIVA DURGA SANAKR

CB.EN.U4EEE22064 M DHEERAJ KUMAR REDDY

CB.EN.U4EEE22065 S MEGANILA

CB.EN.U4EEE22066 UNAIS I

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

AMRITA SCHOOL OF ENGINEERING,

AMRITA VISHWA VIDYAPEETHAM,

COIMBATORE – 641112

MARCH 2025

**Table of Content:**

* Introduction
* Objectives
* Literature review
* Methodology
* System overview
* Tools & Systems
* Nodes and Network
* Results & Analysis
* Conclusion

**INTRODUCTION:**

The increasing need for electricity, fueled by fast industrialization and urbanization, poses a significant threat to power systems globally. The centralized generation-based traditional energy grid is not well-equipped to handle peak demand times efficiently, leading to higher running costs, grid instability, and inefficient use of energy. In order to address these issues, contemporary energy systems are transforming into smart grids—digital, automated power grids that host renewable energy resources, distributed energy resources (DERs), and smart control strategies.

Peak load shifting is one of the most promising solutions for balancing energy supply and demand in smart grids, a strategy which seeks to shift electrical load from peak to off-peak periods. This method not only helps in alleviating grid infrastructure stress but also fosters cost reduction and energy efficiency. The introduction of Internet of Things (IoT) technologies has also transformed the energy sector, allowing for real-time monitoring, remote management, and data-based decision-making.

This project, "IoT-Driven Peak Load Shifting using Pseudo Battery System," investigates a novel method for peak load management through a simulated energy storage system. Rather than employing a physical battery, the project uses a pseudo battery system—a resistive component that emulates battery charging and discharging behavior—governed by relays and ESP32 microcontrollers. The system makes smart decisions regarding when to charge or discharge based on the load patterns determined using a DBSCAN-based machine learning algorithm.

Real-time current sensing is done through ESP32 end nodes, and data is sent to an edge node (laptop) through socket protocol. The edge node does local processing, keeps a rolling time-series dataset in CSV, and sends updates of all data to the Firebase Realtime Database. Based on this data, the system computes optimal charging and discharging times and sends control signals to the ESP32 nodes for relay actuation. Moreover, it calculates statistics like cost without battery, cost with battery, savings, original load profile, and optimized load profile.

A web interface based on Streamlit is created to offer real-time visualization, live monitoring of energy usage, battery condition, and tariff analytics so that users can comprehend the advantages of peak load shifting.

This project presents an affordable, scalable, and smart energy management system that emulates smart battery behavior with basic hardware and software elements. It illustrates how IoT, edge computing, and machine learning can be integrated to maximize grid performance and consumer energy consumption, serving as the basis for future smart grid applications.

**OBJECTIVE:**

The objective of this project, titled "IoT-Driven Peak Load Shifting using Pseudo Battery System," is to design and implement a smart, data-driven energy management solution that addresses the challenges of peak load on electrical grids using IoT and machine learning technologies. The project focuses on simulating battery behavior through resistive components while leveraging real-time data acquisition, cloud integration, and intelligent control to achieve optimized load distribution and energy cost savings.

**The specific objectives of this project are as follows:**

**1. To simulate a smart grid load shifting system using a pseudo battery model**

* Replace physical batteries with rheostats to simulate charging and discharging operations, allowing low-cost experimentation and control logic testing.

**2. To implement a real-time IoT-based monitoring system**

* Use ESP32 microcontrollers integrated with current sensors to collect live electrical consumption data.
* Transmit sensed data to the edge node (laptop) using socket communication for further processing.

**3. To calculate and log power consumption data efficiently**

* Multiply real-time current readings with a fixed DC voltage to compute power.
* Store the computed power values in a CSV file with 24-hour format columns and 300-row storage limit.
* Implement an automatic row-shifting mechanism to remove the oldest entry when the dataset exceeds 300 rows, ensuring lightweight storage at the edge node.

**4. To store and synchronize data with the cloud**

* Continuously upload all sensed and calculated data to Firebase Realtime Database for permanent storage and remote accessibility.
* Ensure Firebase stores both historical and real-time data without deletion.

**5. To apply machine learning for intelligent decision-making**

* Use the DBSCAN clustering algorithm on the edge node to analyze load patterns and cluster similar usage periods.
* Extract four control parameters from clustering output:
* charging\_start\_time
* charging\_duration
* discharging\_start\_time
* discharging\_duration

**6. To automate relay operations based on machine learning decisions**

* Send the control parameters from the edge node to another ESP32 (relay control node) via socket protocol.
* Automate relay switching to simulate pseudo battery charging during off-peak hours and discharging during peak hours, with 4-hour cycles for each operation.

**7. To compute energy cost savings and consumption analysis**

* Use data from Firebase to calculate:
* cost\_without\_battery
* cost\_with\_battery
* total savings
* original load profile
* optimized load profile

**8. To develop a real-time web interface for monitoring and analytics**

**Build a Streamlit web UI to display:**

* Live monitoring of load consumption (original vs optimized)
* Real-time battery status with animations for charging/discharging
* Tariff breakdown and savings visualization

**9. To demonstrate a scalable and modular smart energy management system**

* Showcase how low-cost microcontrollers, real-time data processing, machine learning, and cloud computing can work together in a modular and scalable setup applicable to real-world smart grids, homes, or industries.

**Literature Review:**

**1.Coordinated Control and Load Shifting‐Based Demand Response**

* This paper discusses demand-side management (DSM) using coordinated control strategies.
* It highlights how load shifting and battery storage reduce peak demand and improve energy efficiency.
* Key Insight: A hybrid optimization approach with IoT improves grid stability and energy cost savings.

**2.Smart Grid Energy Optimization with IoT-Based Load Shifting**

* Focuses on IoT-driven energy optimization for peak load shifting.
* Uses machine learning-based demand forecasting to improve energy scheduling.
* Key Insight: Predictive load management enhances energy efficiency and reduces peak-hour dependency**.**

**3.Peak-Load Reduction by Coordinated Response of Photovoltaics, Battery Storage, and Household Loads**

* Investigates how solar energy, battery storage, and Household Loads can work together for peak load shifting.
* Uses coordinated control to balance supply and demand dynamically.
* Key Insight: Integration of renewables, IoT, and storage systems is crucial for modern energy management.

**4.Energy Management in Smart Cities Based on IoT**

* The study highlights the importance of Home Energy Management as a Service (HEMaaS) to reduce peak demand.
* IoT sensors and Q-learning algorithms are applied to optimize energy consumption.
* Smart grids and intelligent devices enable real-time demand-side management (DSM).
* Key Insight: The study proves that Neural Fitted Q-learning can significantly shift peak loads, reducing stress on the grid.

**5.IoT-Enabled Proposal for Adaptive Self-Powered Renewable Energy Management**

* Discusses renewable energy integration and battery storage to manage peak loads.
* Uses IoT for real-time monitoring of energy consumption.
* Adaptive energy management based on machine learning for load shifting.
* Key Insight: The study suggests that real-time analytics and automation improve energy efficiency.

**6. Design and Implementation of IoT-Based Smart Energy Management System**

* Proposes an IoT-driven smart energy system with automated demand response.
* The system integrates battery storage, renewable energy sources, and IoT sensors.
* Key Insight: The paper proves that dynamic load shifting reduces peak demand and optimizes grid stability.

**7.Peak Load Shifting and Energy Storage Optimization**

* Analyzes the impact of battery storage and IoT on peak load shifting.
* Examines case studies in residential and industrial settings.
* Key Insight: The study concludes that battery storage is crucial for reducing peak loads in smart grids.

**8.Energy Demand Reduction through IoT-Based Smart Grid**

* Uses real-time monitoring, demand-side response, and predictive analytics.
* Evaluates energy savings through IoT-enabled smart grids.
* Key Insight: Machine learning and IoT significantly enhance energy efficiency and peak load management.

**9.Smart Energy Management System with IoT Integration**

* This paper explores the role of IoT-based energy management systems (EMS) in reducing peak loads.
* It presents a real-time data acquisition system using sensors and cloud computing.
* Key Insight: IoT enhances demand-side management by dynamically adjusting power usage and improving efficiency in smart grids.

**10.IoT-Based Battery Storage for Load Balancing**

* Discusses battery energy storage for peak load shifting in industrial applications.
* Implements an IoT-driven monitoring system for load balancing.
* Key Insight: IoT-based automation improves battery utilization and reduces peak energy costs.

**11. Demand Side Management using Smart Battery Systems**

* Focuses on demand-side management (DSM) strategies to reduce peak demand.
* Proposes smart battery scheduling algorithms for efficient energy storage utilization.
* Key Insight: IoT-enabled smart battery management systems reduce dependency on the grid and enable better load distribution.

**12.IoT for Energy Storage and Peak Load Reduction**

* Explores renewable energy integration and energy storage for reducing peak load.
* Uses machine learning for predicting and optimizing energy consumption.
* Key Insight: AI and IoT enhance grid reliability by enabling real-time demand response and peak load shifting.

**13.Optimization of Load Shifting in Smart Homes**

* Proposes home energy management systems (HEMS) using IoT and battery storage.
* Focuses on user-centric load scheduling and optimization techniques.
* Key Insight: Smart scheduling reduces peak loads and enhances the efficiency of home energy systems**.**

**SYSTEM OVERVIEW/BLOCK DIAGRAM:**

This project consists of a distributed IoT architecture involving multiple nodes, each with distinct responsibilities, working in coordination to enable intelligent peak load shifting. The system comprises three main components: End Nodes, Edge Node, and Cloud Interface (Firebase + Streamlit UI).

**End node:**

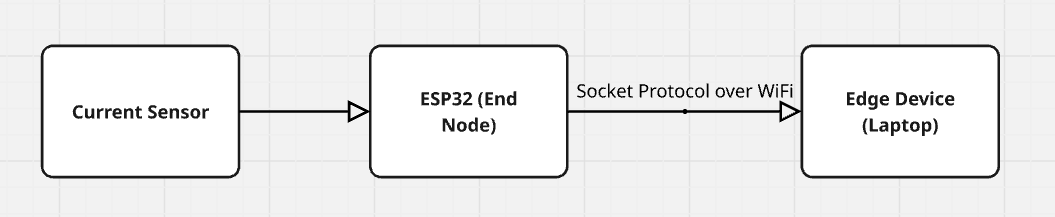
The End Node is a critical component of the IoT-based energy management system. It is responsible for sensing real-time electrical data and controlling the pseudo battery system through relays. The design is divided into two separate roles—Sensor Node and Relay Control Node—each implemented using ESP32 microcontrollers. These microcontrollers interact with the Edge Node (Laptop) via Socket protocol, allowing real-time data exchange and system coordination.

**Time Compression Logic:**

In this simulation, one real-world **30 seconds is equivalent to 1 hour** of virtual time. Thus, a **24-hour cycle is simulated in 12 minutes**. Based on this scaling:

* A 4-hour charging or discharging duration is simulated as 2 minutes (4 × 30 seconds).
* This time-scaling logic is embedded in both sensing and control loops of the End Nodes.

**(A) Sensor End Node Functionality**

****

**Purpose:**

To continuously monitor current consumption from the load and send real-time data to the Edge Node for analysis.

**Components & Operation**

Current Sensor (e.g., ACS712) measures current flowing through the load.

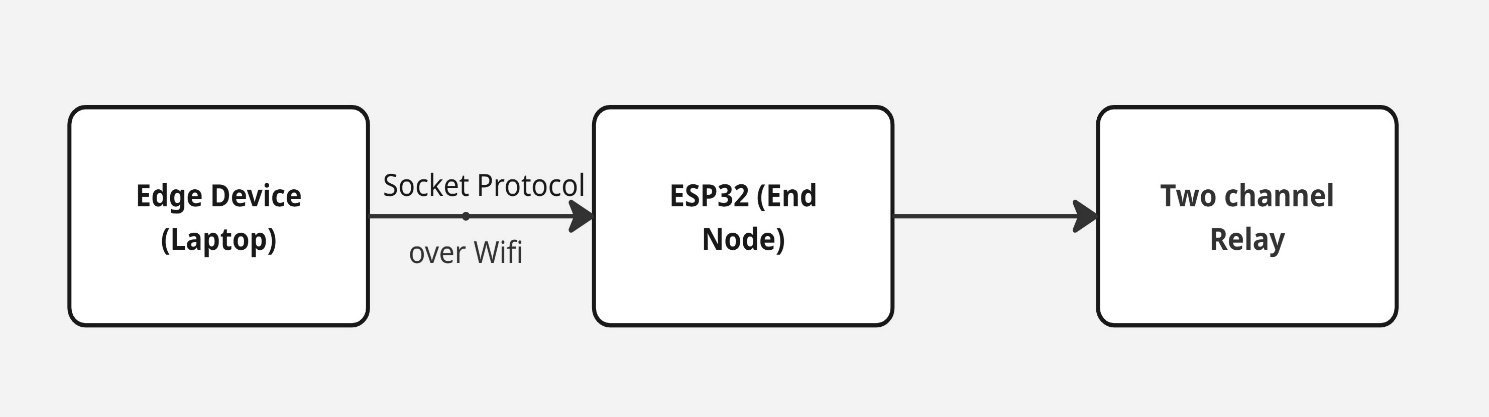
Sensor output is read by an ESP32, which:

* Samples current at fixed intervals.
* Assumes voltage as a constant (e.g., 12V).
* Calculates instantaneous power:
* Power (W) = Current (A) × 12V
* Sends data to the Edge Node (Laptop) over TCP Socket.

**Timing**

* Every 30 seconds (1 simulated hour), one power value is generated and stored.
* A complete dataset for one simulated day contains 24 readings, sent over 12 minutes.

**(B) Relay Control End Node Functionality**

****

This ESP32 node is responsible for controlling the charging and discharging operations of the pseudo battery system using relays:

* Relay 1 controls charging, connecting a resistive element (rheostat) to the DC supply.
* Relay 2 controls discharging, allowing another DC source to power the main load.

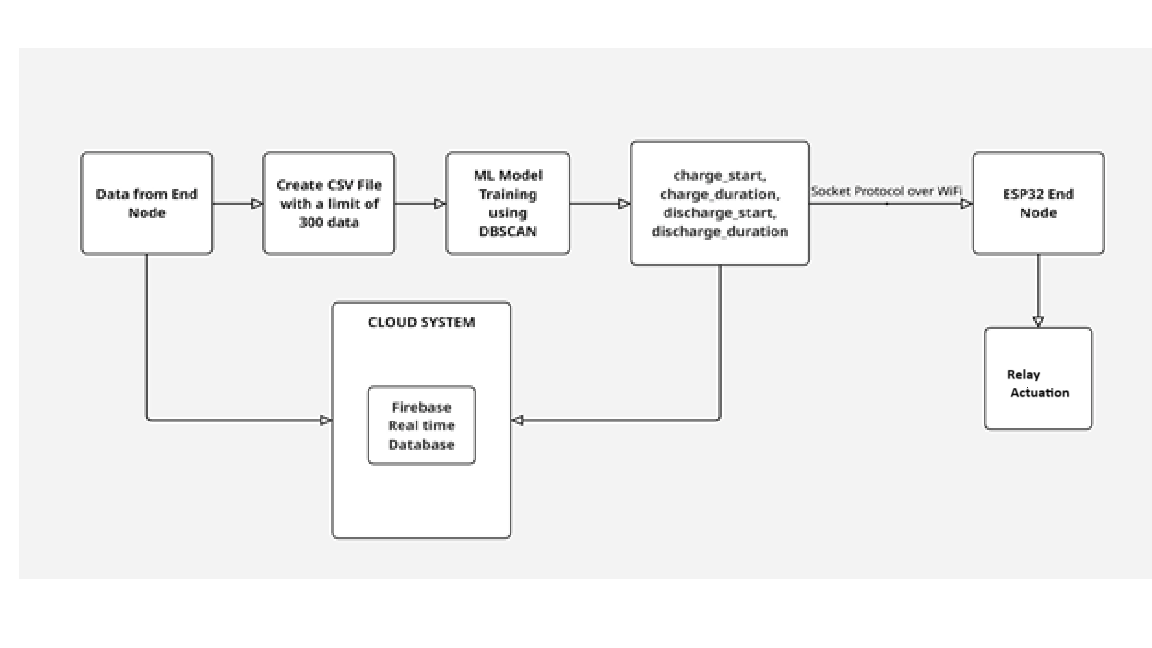
The Edge Node, after processing historical consumption data using the DBSCAN clustering algorithm, sends four key control parameters to this ESP32 node:

* charging\_start\_time
* charging\_duration
* discharging\_start\_time
* discharging\_duration

Upon receiving the timing instructions, the ESP32 actuates the relays accordingly to shift the load operation intelligently.

**Edge computing:**

The Edge Node in this project refers to the laptop or local processing system responsible for handling all real-time data processing, decision-making, and communication between the end nodes and the cloud platform. It plays a crucial role in enabling intelligent peak load shifting by performing sensing data collection, machine learning-based analysis, and issuing control instructions to the actuation system.

****

**Key Responsibilities of the Edge Node**

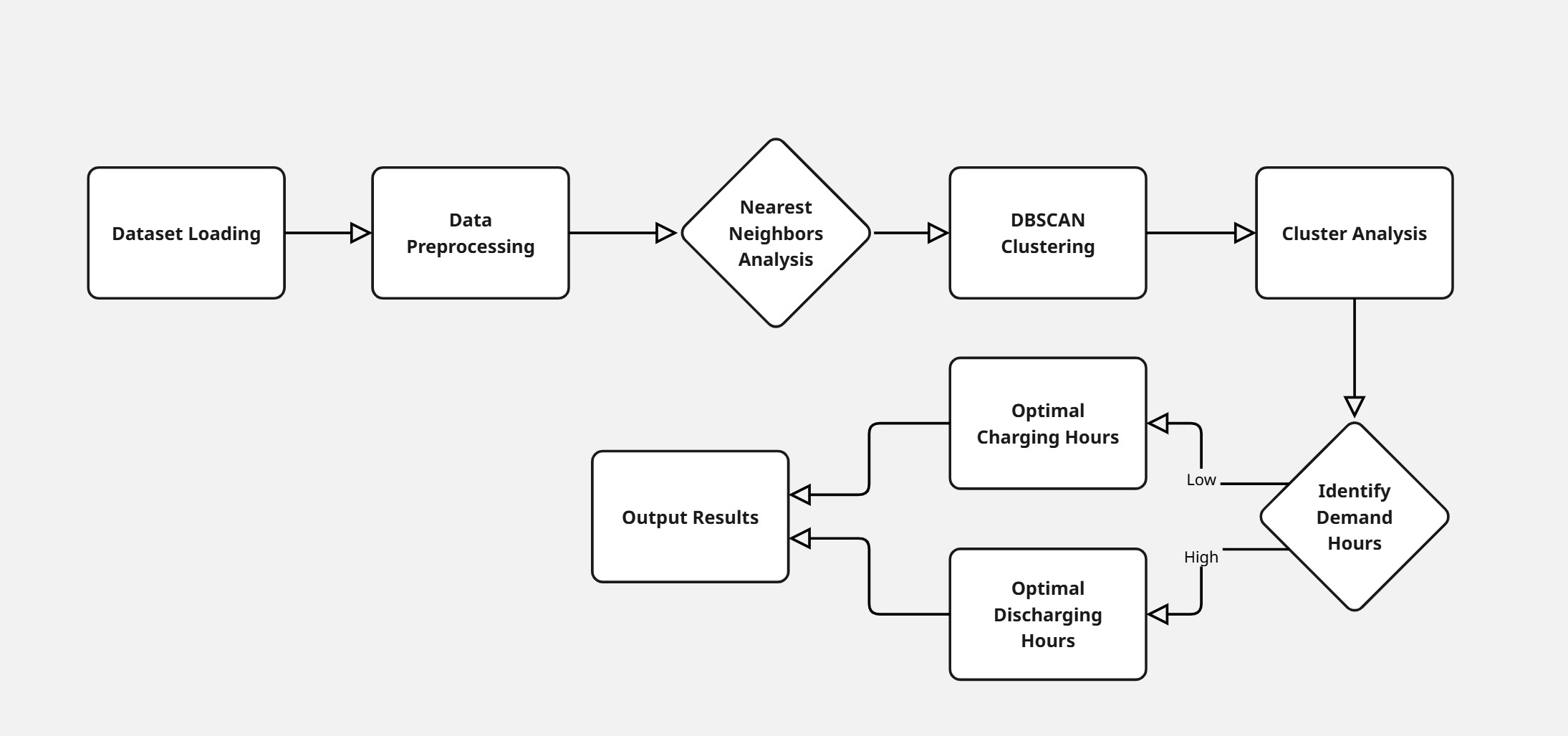
1. Receive real-time current data from the sensing ESP32 end node using Socket protocol.
2. Calculate power consumption using the constant voltage assumption (Power = Current × 12V).
3. Store data locally in a CSV file with a rolling window of 300 rows (days) and 24 columns (hours).
4. Upload all data to Firebase Realtime Database for cloud storage and visualization.
5. Train an ML model (DBSCAN) on the stored power data to detect patterns for load shifting.
6. Extract control values:

* charging\_start\_time
* charging\_duration
* discharging\_start\_time
* discharging\_duration

1. Send these four parameters to the actuation ESP32 end node through Socket protocol over Wi-Fi.
2. Send the same parameters to Firebase to enable cloud-based data analytics and UI visualization.

**ML model:**

The core of intelligent load management in this project lies in the machine learning (ML) model implemented at the edge node. The purpose of the model is to identify optimal time slots for charging and discharging a pseudo battery, based on load patterns observed over time. By leveraging unsupervised clustering using DBSCAN (Density-Based Spatial Clustering of Applications with Noise), the system is able to autonomously detect peak and off-peak load periods and act accordingly.



**ML Workflow Description**

**1. Dataset Loading**

* The edge node receives data from the sensing ESP32 in real-time.
* This data is saved in a rolling CSV file representing power consumption for 300 days, each day having 24 hourly values.

**2. Data Preprocessing**

* The dataset is normalized or scaled to bring all values to a similar range.
* Missing or noisy data points are filtered or interpolated.
* The data is structured in a format suitable for clustering (300 samples × 24 features).

**3. Nearest Neighbors Analysis**

* To determine a suitable value for the epsilon parameter in DBSCAN, k-distance graph analysis is used.
* This helps in selecting a threshold distance for forming clusters based on data density.

**4. DBSCAN Clustering**

* The DBSCAN algorithm is applied to group days with similar power usage patterns.
* It detects natural groupings without requiring the number of clusters to be predefined.
* Outliers or abnormal load days are automatically treated as noise.

**5. Cluster Analysis**

* Each cluster is analyzed to understand usage behavior.
* The time intervals with consistent high power demand are identified as peak hours.
* Intervals with consistently low demand are labeled as off-peak hours.

**6. Identify Demand Hours**

* The cluster statistics are used to generate:
* Optimal Charging Hours: during off-peak times (to reduce grid stress).
* Optimal Discharging Hours: during peak hours (to support the grid and reduce cost).

**7. Output Results**

The system computes and outputs the following four key values:

* charging\_start\_time
* charging\_duration
* discharging\_start\_time
* discharging\_duration

These values are sent to:

* The actuator ESP32 node for controlling the relays.
* Firebase Realtime Database for cloud access and web UI integration.

**Why DBSCAN?**

Unlike k-means, DBSCAN:

* Does not require the number of clusters to be predefined.
* Can handle noise and outliers, which is common in real-world energy usage data.
* Groups based on density, which helps capture varying usage behavior across different days.

**Methodology:**

The proposed system aims to achieve peak load shifting in a smart grid environment by leveraging IoT sensing, real-time data collection, machine learning-based demand analysis, and cloud integration. The methodology follows a systematic, modular approach involving hardware components, communication protocols, machine learning models, and cloud infrastructure.

**1. Hardware Setup and Load Simulation**

* A DC power supply is used to simulate the grid.
* A rheostat acts as the variable main load to represent consumption behavior.
* A pseudo battery system is simulated using a secondary rheostat (for charging) and a separate DC source (for discharging).
* Two relays are connected to control charging and discharging paths.
* The ESP32 microcontroller reads the current drawn using a current sensor (e.g., ACS712), and the voltage is assumed to be constant at 12V.

**2. Data Acquisition and Transmission (End Node 1)**

* The ESP32 reads real-time current values.
* Power is calculated as:
* Power (W)=Current (A)×12V
* These values are sent over Socket protocol (TCP) to the edge node (laptop) every 30 seconds.
* Each 30 seconds simulates 1 hour in real-world time, hence 1 day = 12 minutes.

**3. Edge Node Data Logging and Processing**

* The laptop receives the power data and stores it in a CSV file.
* The CSV file is structured with 24 columns (hour1–hour24) and 300 rows (day1–day300).
* Once the 301st row is recorded, the first row is deleted, maintaining a rolling buffer of 300 days.
* Simultaneously, all data is pushed to the Firebase Realtime Database for permanent cloud storage.

**4. Machine Learning – DBSCAN for Demand Profiling**

* When 300 days of data are available, the model trains using DBSCAN (Density-Based Spatial Clustering of Applications with Noise).
* Steps involved:
* Preprocessing: Normalizing values and removing noise.
* Nearest Neighbor Distance Calculation: Used to determine optimal epsilon value.
* Clustering: DBSCAN groups similar daily patterns based on load behavior.
* Cluster Analysis:
* Identifies off-peak hours (low usage).
* Identifies peak hours (high usage).
* Based on cluster insights, the following are extracted:
* charging\_start\_time
* charging\_duration
* discharging\_start\_time
* discharging\_duration
* Each duration is fixed to 4 hours (2 minutes real-time).

**5. Relay Control (End Node 2)**

The 4 control parameters are sent from the edge node to ESP32 (Relay Control Node) via socket protocol over Wi-Fi.

The ESP32:

* Closes the charging relay during off-peak hours.
* Closes the discharging relay during peak hours.
* Relay logic is handled using real-time clock simulation aligned with the 30-sec per hour scheme.

**6. Firebase Realtime Database and Cloud Operations**

Firebase acts as the central cloud system for:

* Storing real-time load and battery usage data.
* Storing ML-generated parameters.
* Supporting the web interface.

From these values, the following metrics are computed:

* Cost Without Battery
* Cost With Battery
* Savings
* Original Consumption Data
* Optimized Consumption Data

**7. Streamlit Web UI**

A user-friendly web interface is developed using Streamlit, with the following features:

**Live Monitoring:**

* Displays real-time power consumption graph with both original and optimized load curves.

**Battery Status:**

* Shows animation during charging/discharging based on relay status from Firebase.

**Tariff Comparison:**

* Displays cost metrics and savings for transparent visualization.

**Tools & systems:**

To successfully design, develop, and deploy the IoT-based peak load shifting system, a combination of hardware components, software tools, communication protocols, and cloud platforms has been used. Each of these tools and systems plays a vital role in ensuring efficient data acquisition, processing, analysis, control, and visualization.

**Hardware Tools:**

1. **ESP32 Microcontroller:**

* Acts as the end node for both sensing and actuation.
* Equipped with Wi-Fi capability for wireless communication.
* Used in two nodes:
* Node 1: For sensing current using a current sensor.
* Node 2: For relay actuation based on charging/discharging control.

1. **Current Sensor (ACS712):**

* Measures real-time current flowing through the load.
* Analog or digital output is read by ESP32.

1. **DC Power Supply:**

* Simulates the grid supply.
* Provides a stable 12V source to the entire system.

1. **Rheostat (Variable Resistor):**

* Used as the main load to mimic variable power consumption.
* Another rheostat is used for simulating the pseudo battery during charging.

1. **Relays (SPDT or DPDT):**

* Used to switch between charging and discharging modes.
* Controlled by ESP32 GPIO pins.

**Software Tools:**

1. **Arduino IDE / PlatformIO:**

* Used to program the ESP32 boards.
* Enables development and debugging of firmware for sensing and relay actuation.

1. **Python:**

* Handles socket programming between ESP32 and the edge node.
* Used for CSV file creation, data preprocessing, and ML model development.

1. **scikit-learn (Python Library):**

* Provides the implementation for DBSCAN clustering algorithm.
* Used to extract optimal charging/discharging hours from historical data.

1. **Streamlit:**

* Used to build the web-based dashboard.
* Provides real-time data visualization, battery animation, and tariff calculations.

1. **Firebase Realtime Database:**

* Acts as the cloud backend for storing real-time power data, control values, and ML outputs.
* Supports two-way communication between the edge node and web UI.

**Protocols and Communication:**

1. **Socket Programming (TCP over Wi-Fi):**

* Ensures real-time communication between:
* ESP32 (sensor node) ↔ Edge Node (Laptop)
* Edge Node (Laptop) ↔ ESP32 (relay node)

1. **Firebase SDK (Python):**

* Used for pushing and retrieving data from the Firebase Realtime Database.

**Machine Learning Components:**

**DBSCAN (Density-Based Spatial Clustering of Applications with Noise):**

* A clustering algorithm that groups similar load patterns without needing predefined number of clusters.
* Helps in identifying peak and off-peak hours for optimized load shifting.

**Cloud Infrastructure:**

**Firebase Realtime Database (Google Cloud):**

* Provides a NoSQL cloud-hosted database.
* Stores:
* Real-time consumption data
* Charging/discharging timings
* Cost calculations
* Battery status

**Data Management:**

**CSV File Handling (Edge Node):**

* Data is stored in a CSV file with a fixed size of 300 rows and 24 columns (hourly consumption).
* Used for training the ML model.
* Oldest entries are deleted to maintain edge node efficiency.

**Nodes and Network:**

In an IoT-based Smart Grid application, nodes are devices that either sense, process, or control electrical parameters, and the network refers to the communication framework that connects them. This system includes three primary types of nodes, all interconnected via Wi-Fi, along with a cloud-based layer for storage and visualization.

1. Node Classification
2. Network Architecture
3. Cloud Integration & Role

**1.Node Classification**

**A. End Node 1 – Sensing Node (ESP32 + Current Sensor)**

Function: Acts as the source of real-time power consumption data.

Components:

* ESP32 Microcontroller – Reads current data from the sensor.
* Current Sensor (e.g., ACS712/INA219) – Measures the real-time current drawn by the load.

Operation:

* Current values are read and multiplied with a constant voltage (12V) to compute power.
* Sends the power data to the edge node every 30 seconds (representing one hour in simulation).

Data is transmitted via TCP socket communication over Wi-Fi.

**B. Edge Node – Processing Unit (Laptop)**

Function: Serves as the system’s computational brain and coordination hub.

Responsibilities:

* Receives real-time power data from the sensing node.
* Stores data in a CSV file with a maximum of 300 rows (equivalent to 12 minutes simulating 25 days).
* Implements logic to append new data and remove oldest entries to maintain storage limits.
* Pushes all incoming and stored data to Firebase Realtime Database.
* Trains a DBSCAN clustering model to detect optimal charging and discharging hours.
* Sends the computed charging start time, charging duration, discharging start time, and discharging duration to the actuation node via socket protocol.
* Sends all parameters to Firebase for cloud storage and dashboard visualization.

**C. End Node 2 – Actuation Node (ESP32 + Relay Module)**

Function: Controls the pseudo battery (rheostat) charging and discharging circuits.

Components:

* ESP32 microcontroller
* Relay Module (SPDT or DPDT)

Operation:

* Receives control parameters (charging and discharging schedule) from the edge node.

Controls two separate relays:

* One relay connects the pseudo battery for charging during off-peak hours.
* Another relay connects the pseudo battery for discharging during peak hours.
* Relay actuation simulates intelligent battery behavior for load shifting.

**2. Network Architecture**

The nodes are interconnected through a Wi-Fi-based local area network, which provides a lightweight and low-latency communication environment.

**Communication Flow:**

* Sensing Node → Edge Node:
* Protocol: TCP socket over Wi-Fi
* Frequency: Every 30 seconds
* Data Sent: Instantaneous current (and inferred power) values.
* Edge Node → Firebase:
* Method: Firebase Python SDK
* Frequency: Real-time update for every new reading
* Data Stored: Power data, ML model outputs, battery schedule.
* Edge Node → Actuation Node:
* Protocol: TCP socket over Wi-Fi
* Data Sent: Charging start time, charging duration, discharging start time, discharging duration.
* Firebase → Streamlit Web UI:
* Purpose: Visualization of:
* Live power consumption
* Battery status and behavior
* Tariff comparison (cost with and without battery)

**3. Cloud Integration & Role**

The system uses Firebase Realtime Database as a central storage hub and communication bridge between nodes and the front-end interface.

* Enables persistent storage of all consumption data beyond the local CSV limit.
* Facilitates remote access and visualization through the Streamlit dashboard.
* Stores ML model results and computed metrics.

**Results & Analysis:**

**The following topics include image attachments as part of the results and analysis section:**

1. Project Setup Overview

* Hardware Connections and Layout

1. End Node 1 – Sensing Unit

* ESP32 Client Code for Sensing and Transmission
* CSV File Snapshot – Power Data Logging

1. Edge Node – Data Processing Unit

* Server Code to Receive Sensor Data
* DBSCAN ML Model – Pattern Detection
* Client Code to Send ML Output to Actuation Node

1. End Node 2 – Actuation Unit

* ESP32 Server Code for Receiving Battery Scheduling
* Relay Control Logic for Charging and Discharging

1. Firebase Cloud Integration

* Real-Time Database Snapshot

1. Streamlit Web UI Implementation

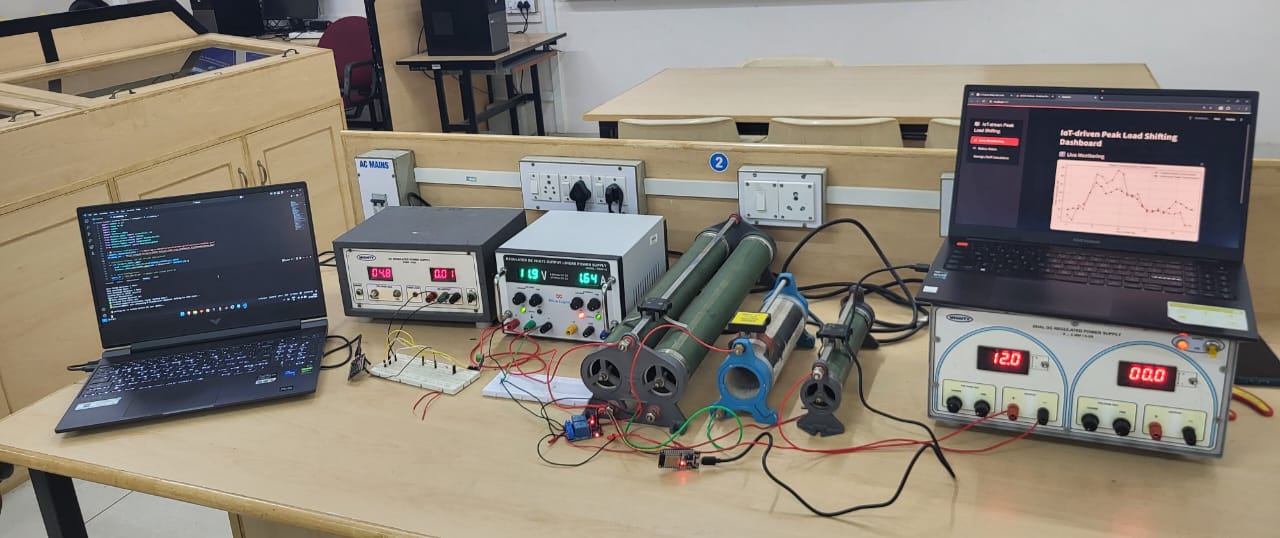
* Tariff Calculation Logic
* Main Dashboard UI Code

1. Web UI Output and Visualization

* Live Monitoring of Load Consumption
* Battery Status Visualization
* Charging State
* DisCharging State
* Idle
* Tariff Comparison and Savings Display

1. **Project Setup Overview**

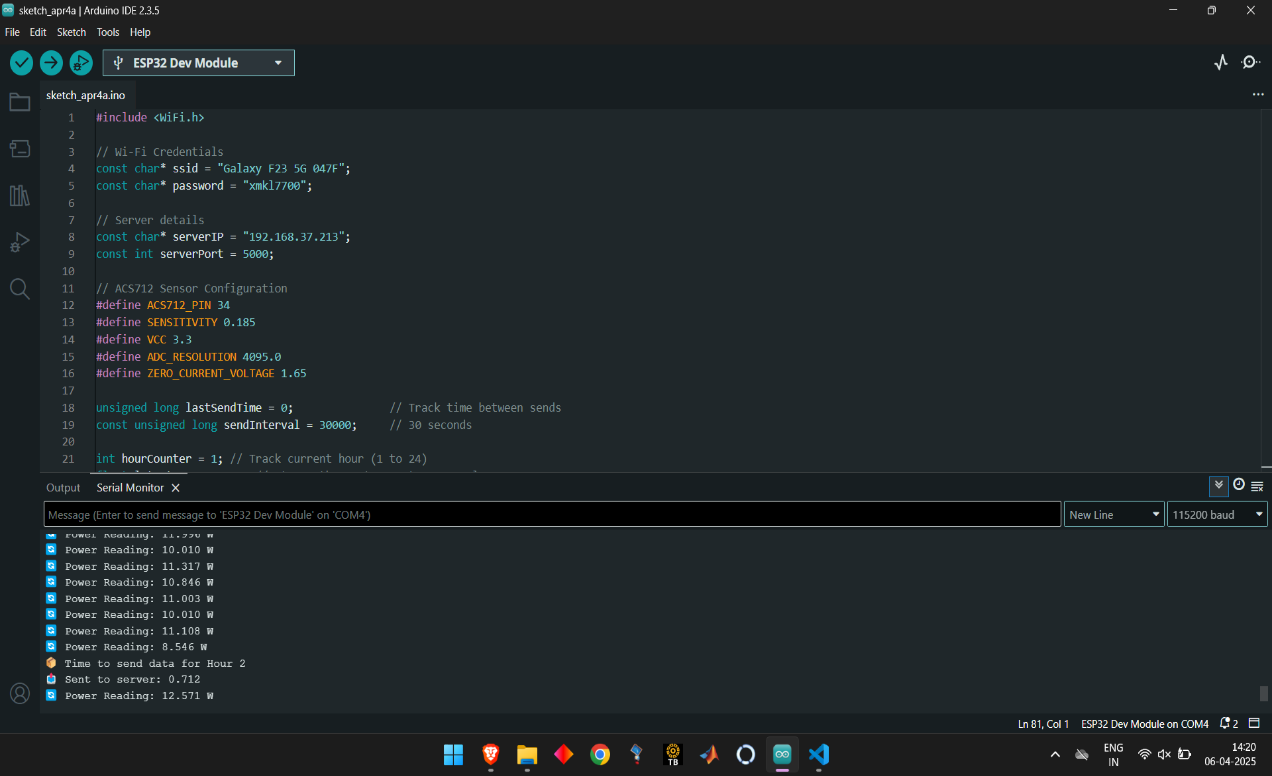
* **Hardware Connections and Layout:**



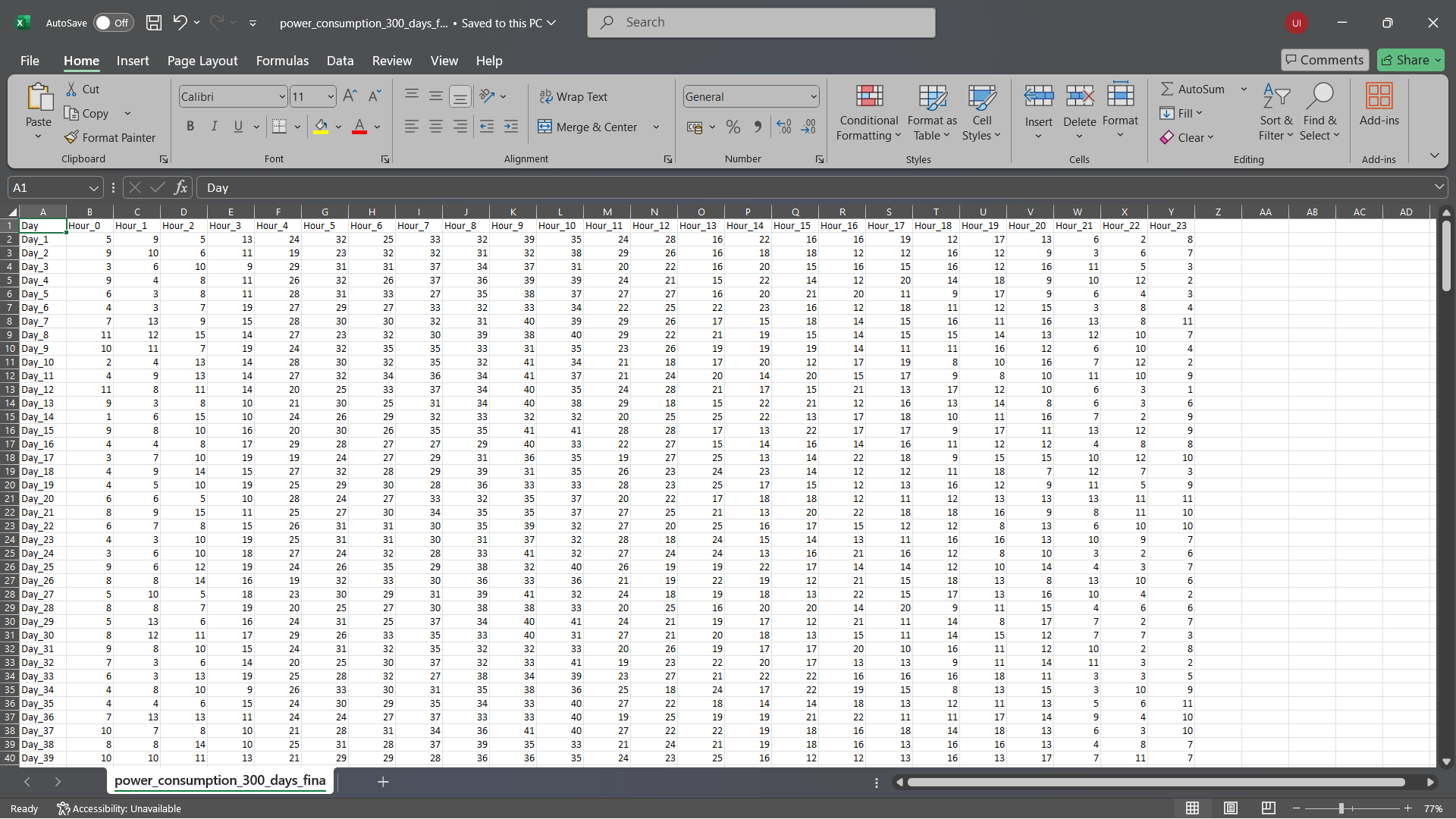
Complete hardware setup showing connections between DC source, rheostat (load), ESP32 nodes, relays, and sensors.

1. **End Node 1 – Sensing Unit**

* **ESP32 Client Code for Sensing and Transmission**

****

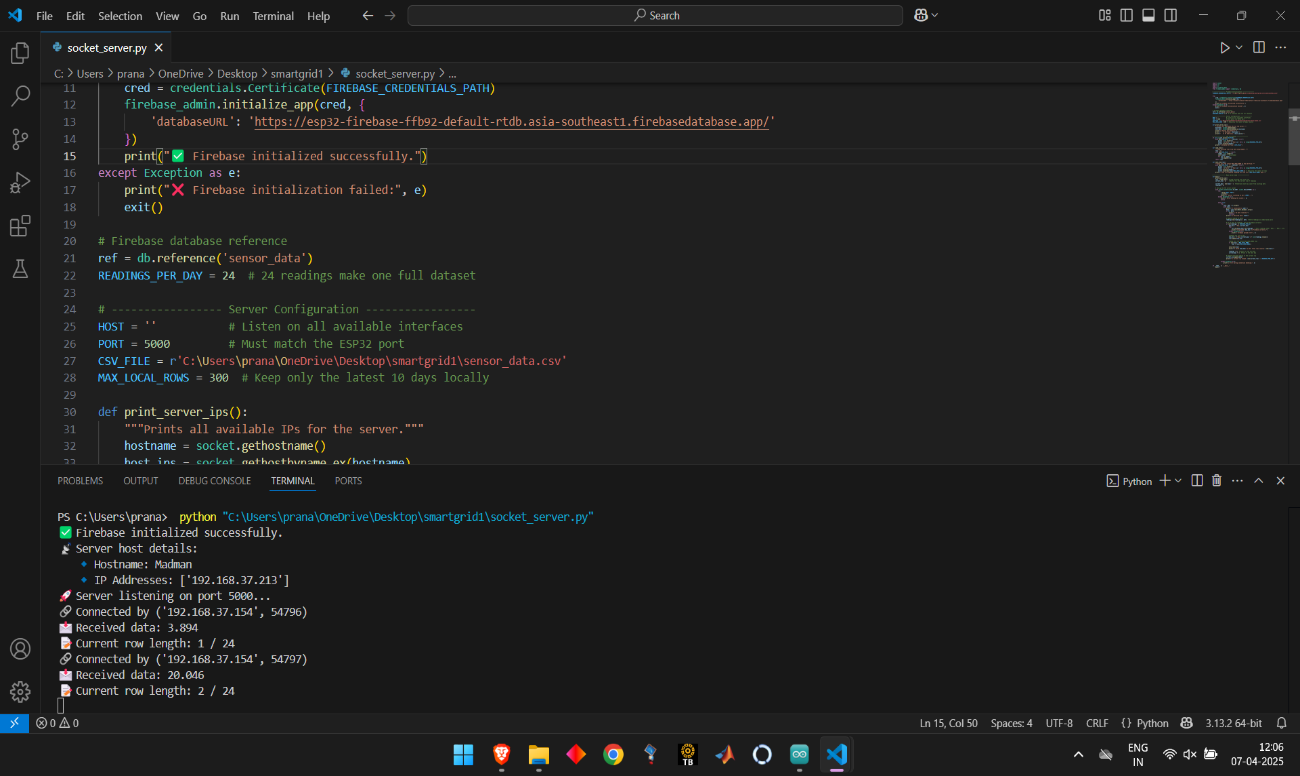
* Wi-Fi Connection: Links ESP32 to a given Wi-Fi network and shows the allocated IP address.
* Reads current from ACS712 sensor (connected to pin 34) and estimates power assuming a 12V system.
* It converts ADC readings to voltage and then to current by applying the sensor sensitivity, and lastly to power (P = IV).
* Transmits power data to a remote server over TCP every 30 seconds via WiFiClient.
* Has an hour counter (1–24), which increments with every data send to mimic hourly logging.
* Always shows real-time power measurements on the Serial Monitor every second.
* **CSV File Snapshot – Power Data Logging**

****

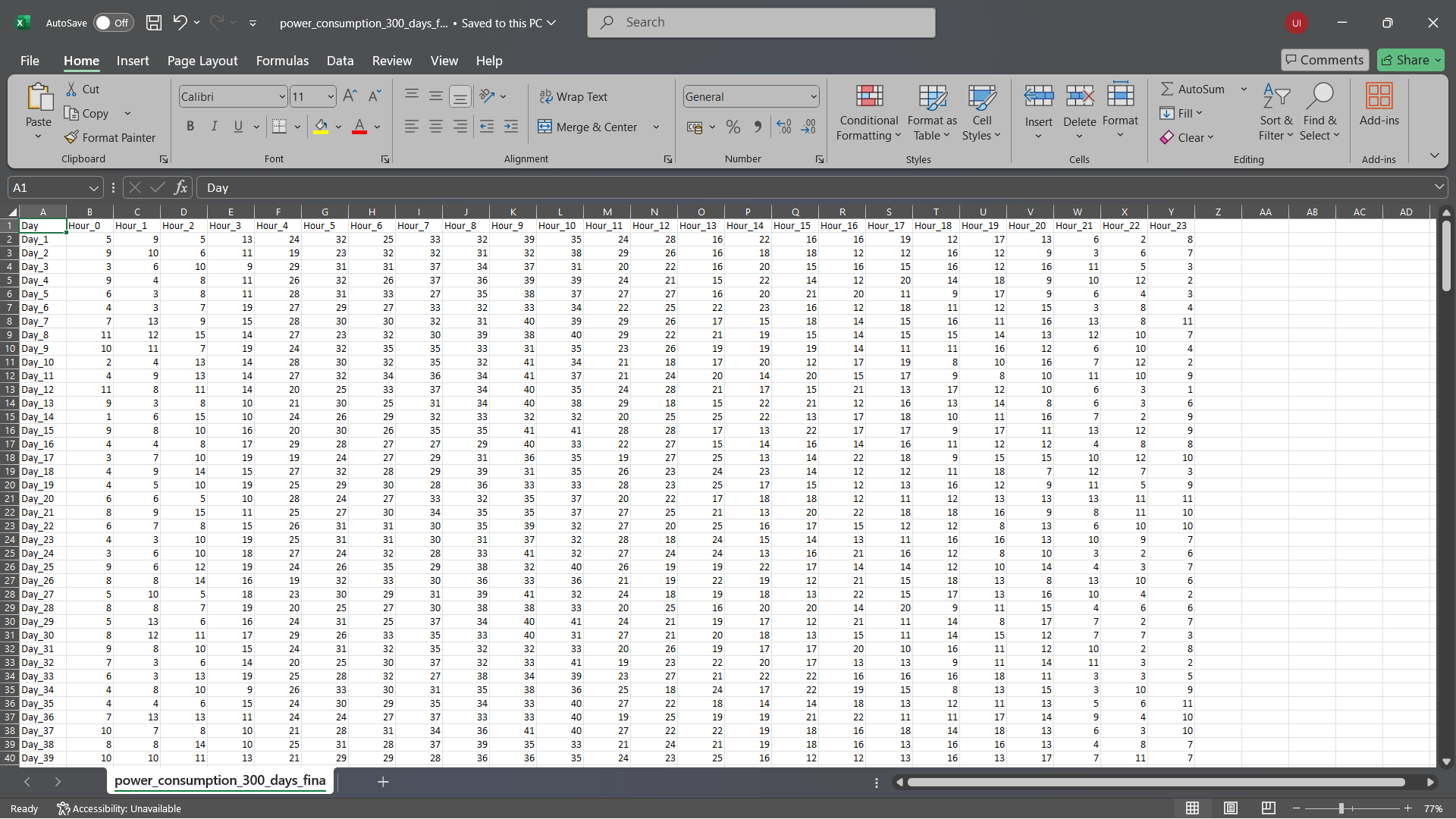
Snapshot of the CSV file storing real-time power consumption data. Each row represents a day, and each column corresponds to an hour of the day, capturing 24-hour power usage patterns for ML analysis.

1. **Edge Node – Data Processing Unit**

* **Server Code to Receive Sensor Data:**

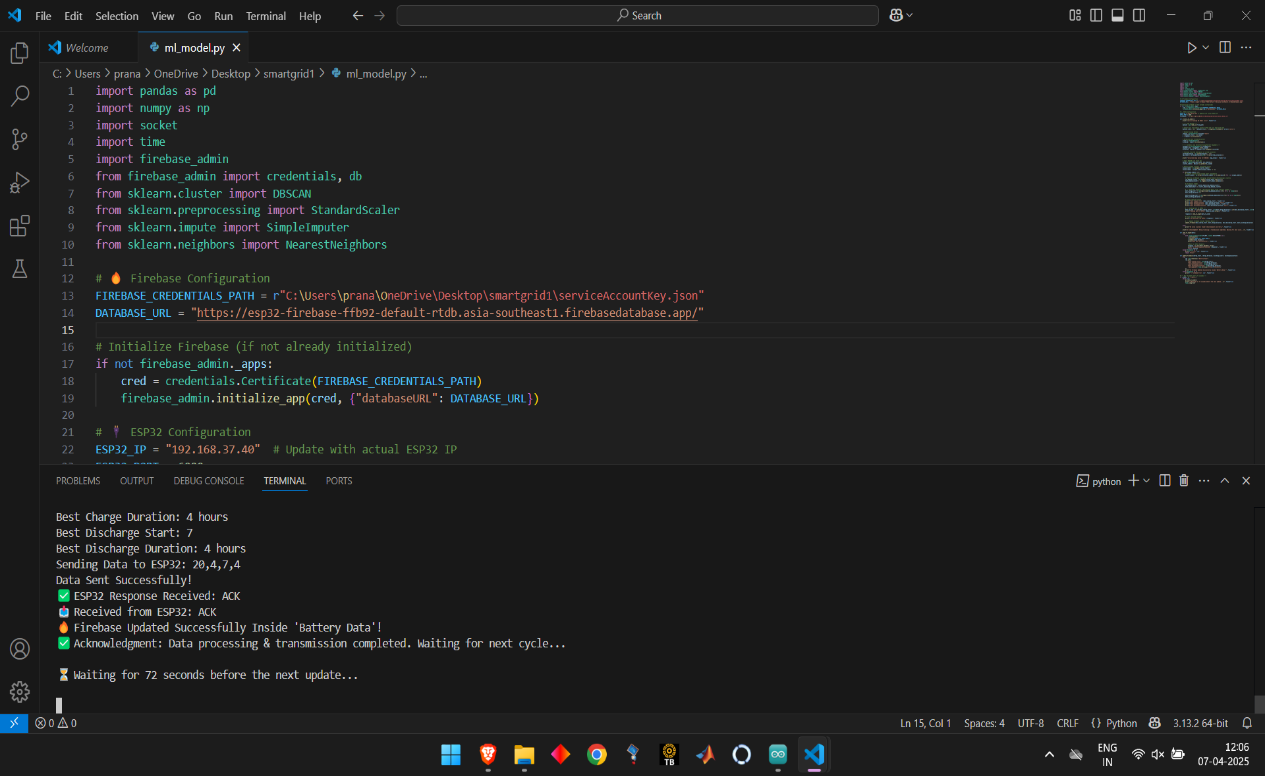
****

* Initiates Firebase Realtime Database from a service account key and stores daily readings under path structures such as Day\_1, Day\_2, etc.
* Has a local CSV file (sensor\_data.csv) with 24 hourly readings per day, retaining only the most recent 300 days (configurable) to enable light storage.
* Listeners for incoming connections on port 5000, accepts power data transmitted by the ESP32 every 30 seconds.
* Groups every 24 readings into one "day" of data. Upon completion, saves that day's data to both Firebase and the CSV file.
* Trims the CSV file automatically to keep only the last 300 days to avoid uncontrolled growth over time.
* Keeps printing the IP addresses of the server, connection information, received data, progress so far, and upload status to the console continuously.

****

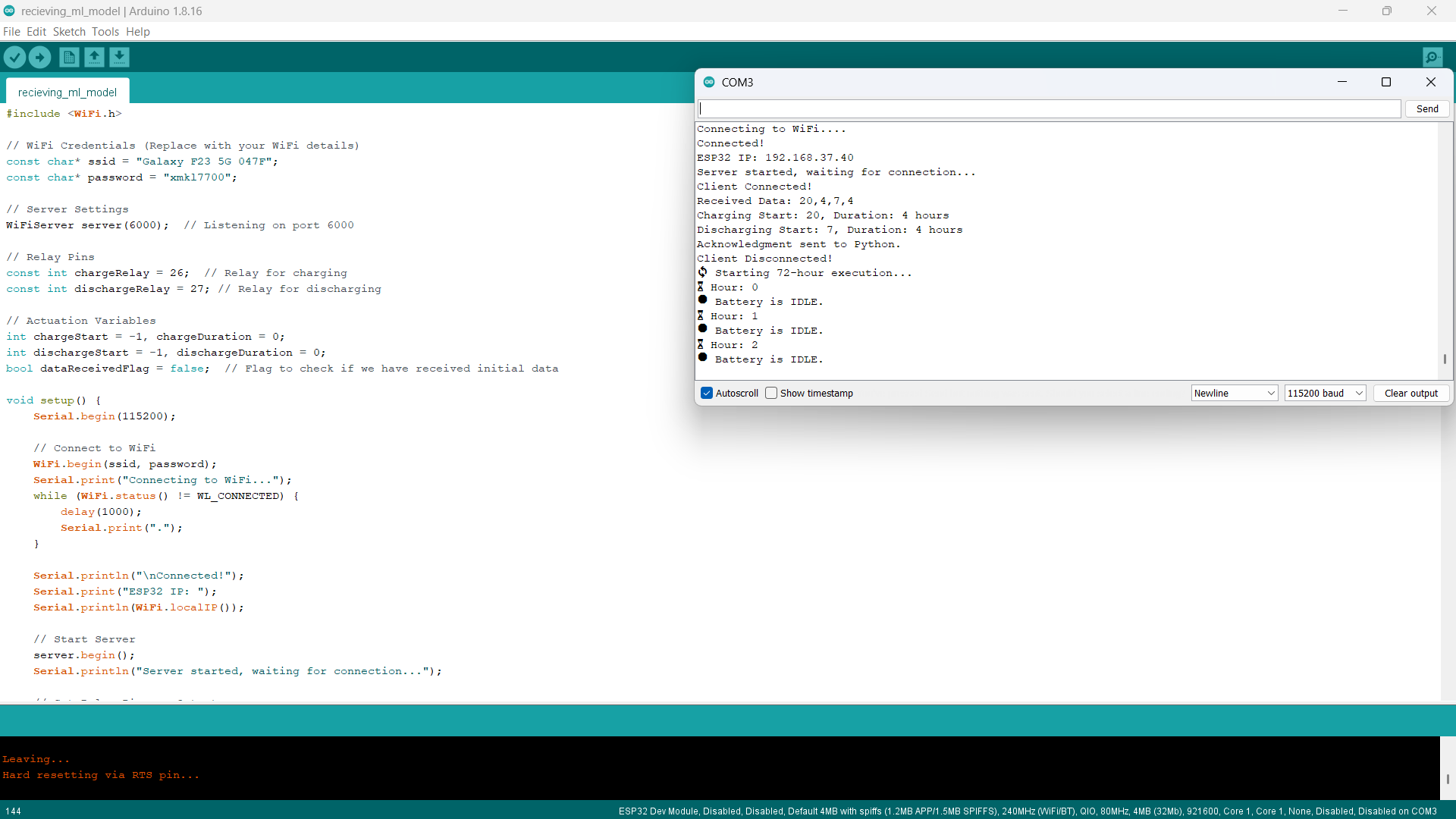
1. **Code for creating ML model and Sensing it cloud and End Node:**

* Processes CSV data by reading and pre-processing with imputation and standardization.
* Selects eps automatically with k-nearest neighbors for DBSCAN.
* Identifies low and high demand clusters with DBSCAN clustering.
* Determines optimal 4-hour charging and discharging windows.
* Sends computed timings to ESP32 via TCP socket.
* Updates Firebase with scheduling information and executes every 12 minutes.

****

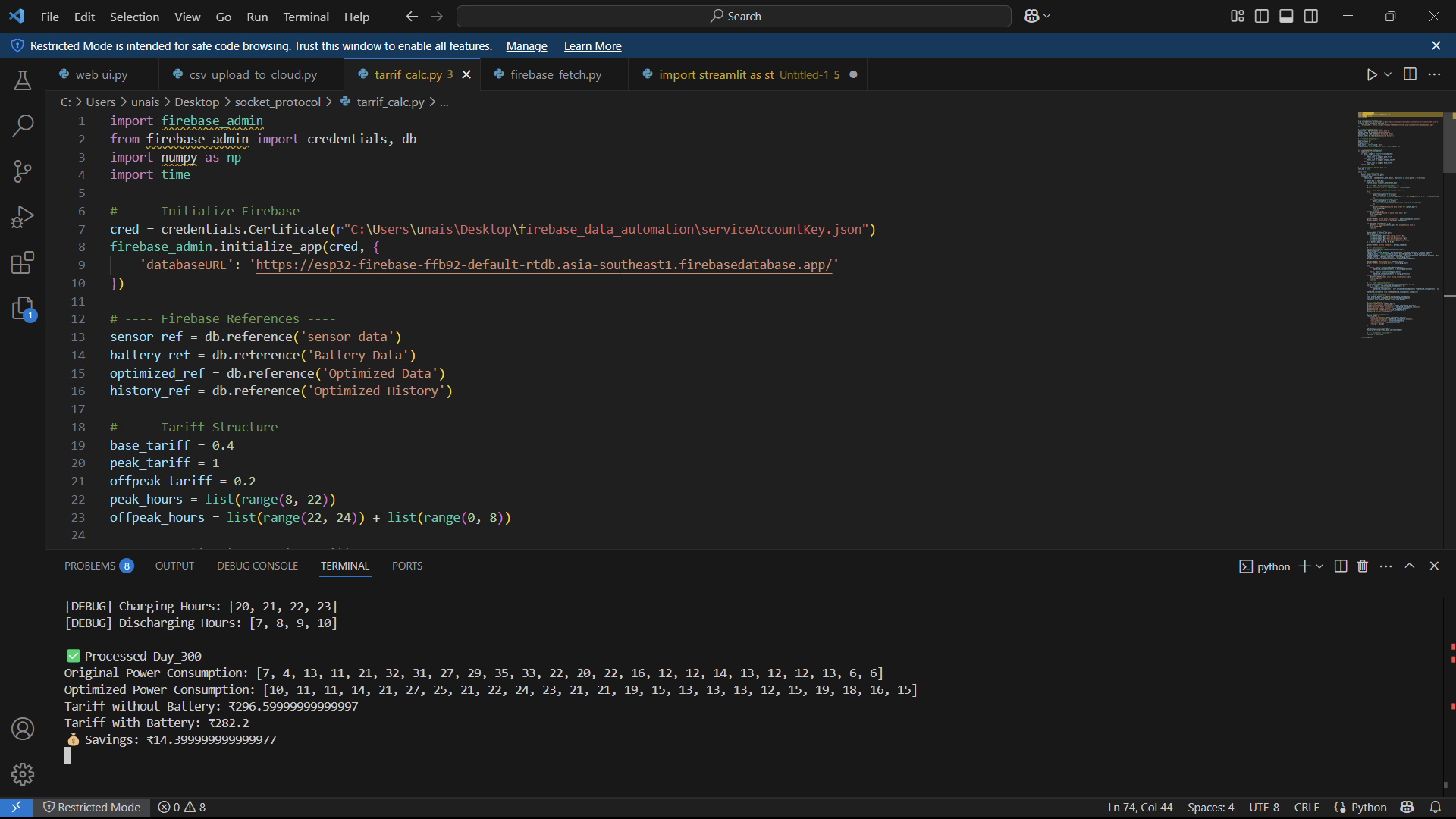
1. **Server code for getting Battery from ML model:**

* Attaches ESP32 to WiFi and begins a TCP server on port 6000.
* Awaits data from the Python server in the format: chargeStart,chargeDuration,dischargeStart,dischargeDuration.
* Interprets the received data and responds back if correct.
* employs two relay pins (GPIO 26 and 27) for charging and discharging management.
* Simulates a 72-hour cycle where each hour is symbolized by a brief delay.
* Manages the relays according to received schedule and resets after every complete cycle.

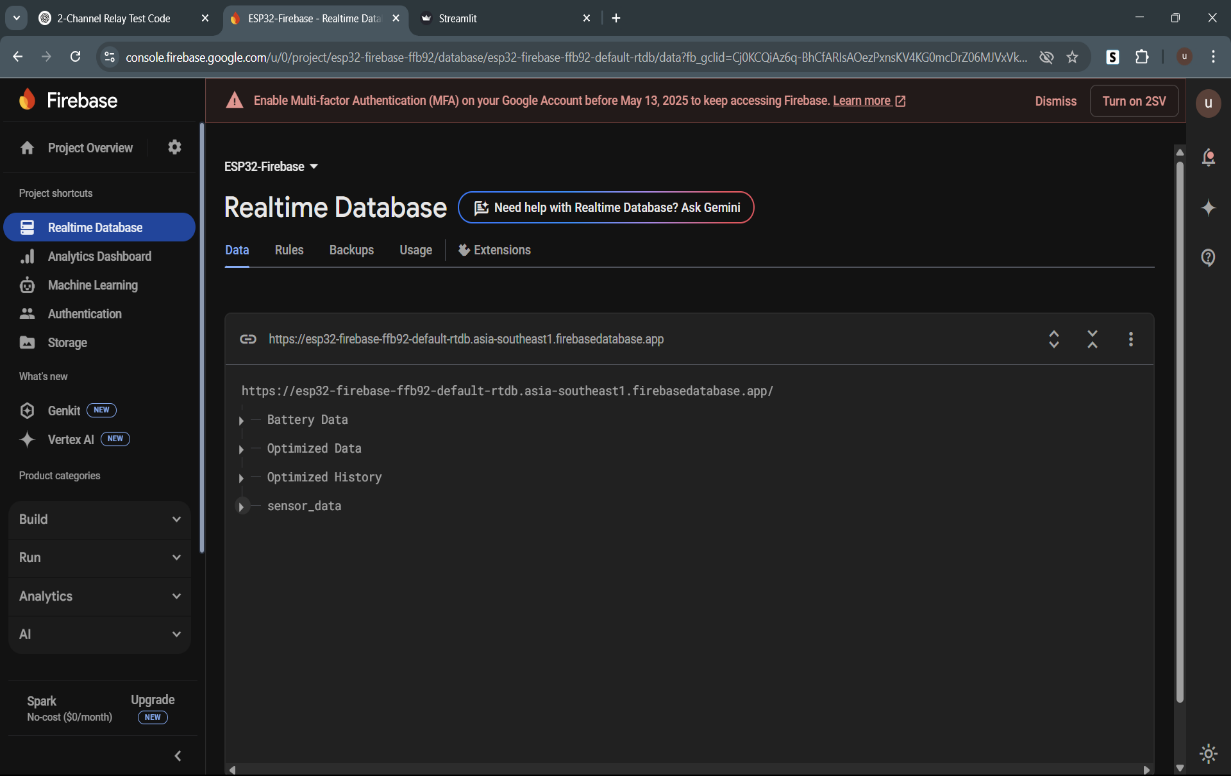
****

1. **Tarrif Calculation:**

* Connects to Firebase and retrieves the most recent day's power consumption data (24 hourly values).
* Retrieves battery schedule (charge\_start, duration, etc.) from Firebase.
* Simulates battery charging/discharging by modifying the power consumption accordingly.
* Clamps values and smooths to maintain optimized consumption realistic.
* Computes electricity costs before and after optimization based on tariff structure.
* Stores the optimized data and cost savings back into Firebase and Optimized History.

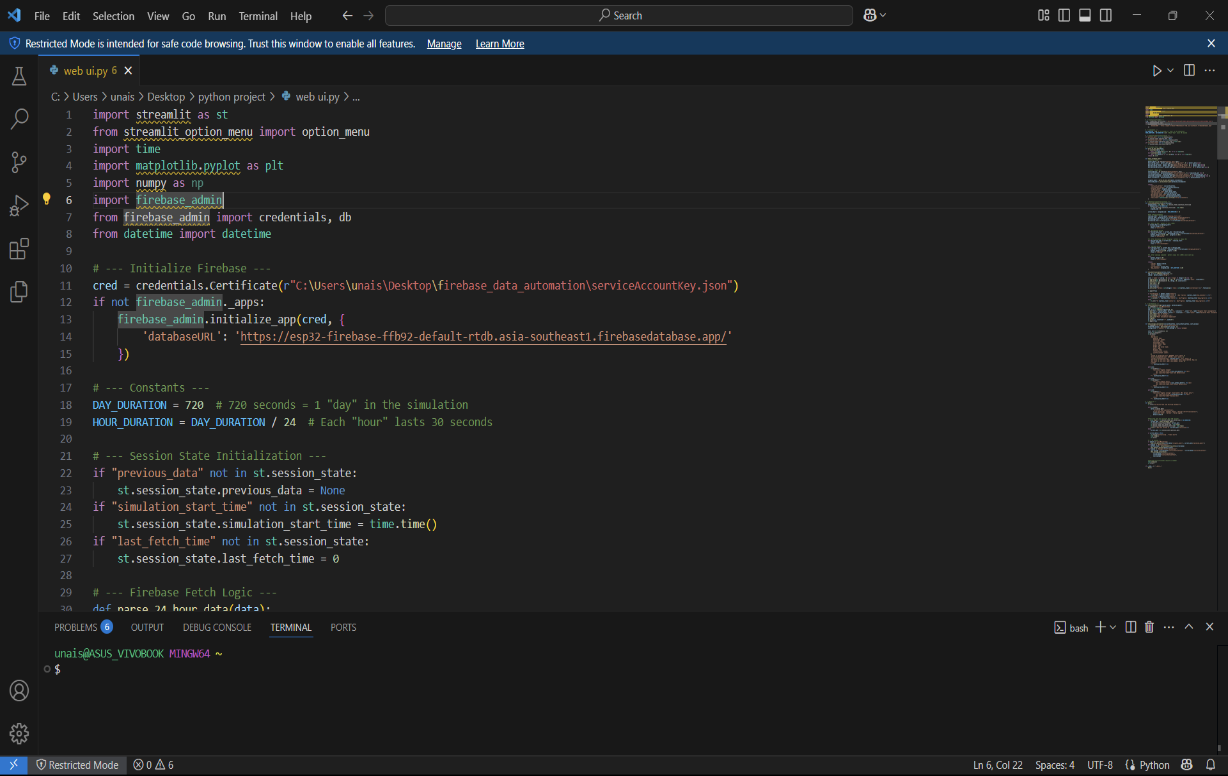
****

1. **Firebase cloud:**

****

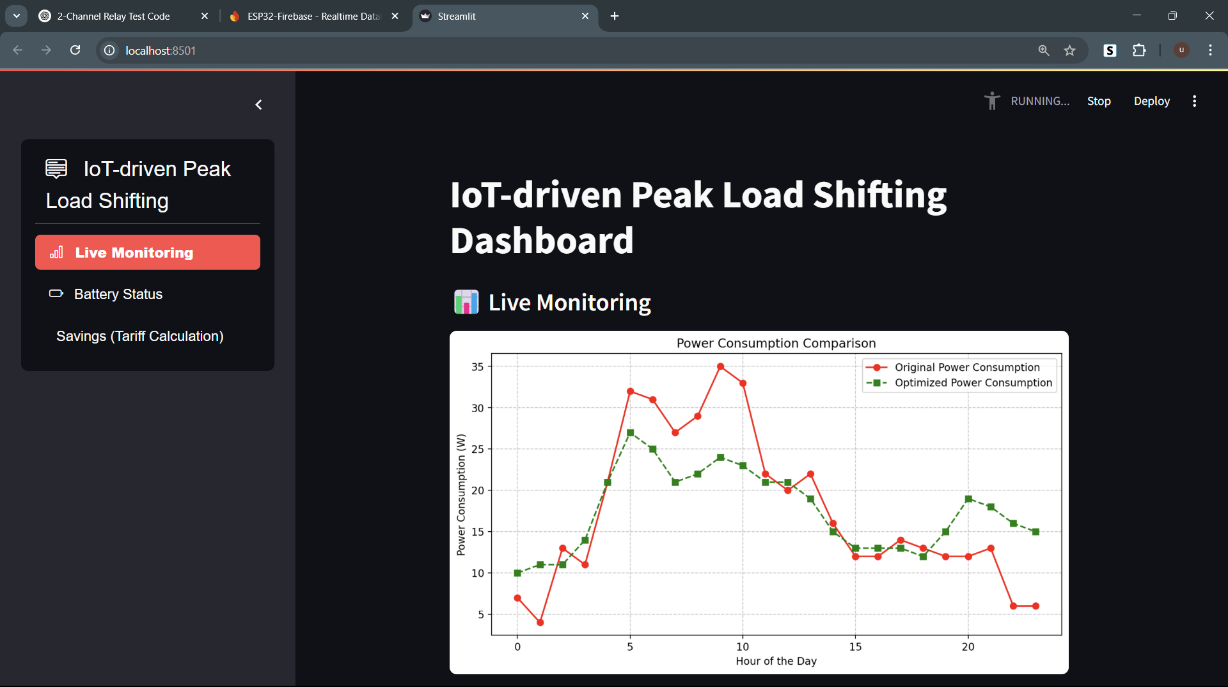
1. **Web UI:**

* Firebase Integration: Integrate with Firebase Realtime Database to retrieve latest battery timing and optimized power consumption information.
* Compares original and optimized power consumption on a 24-hour line graph for real-time display.
* Simulates battery charge status according to charging/discharging schedule, updating visually with progress and status.
* Calculates and displays cost with and without battery, displaying savings in a styled, color-coded format.
* Updates information every simulated "day" (720 seconds) and refreshes the visuals every 10 seconds.
* IHas a sidebar menu of navigation with options for "Live Monitoring", "Battery Status", and "Savings".

****

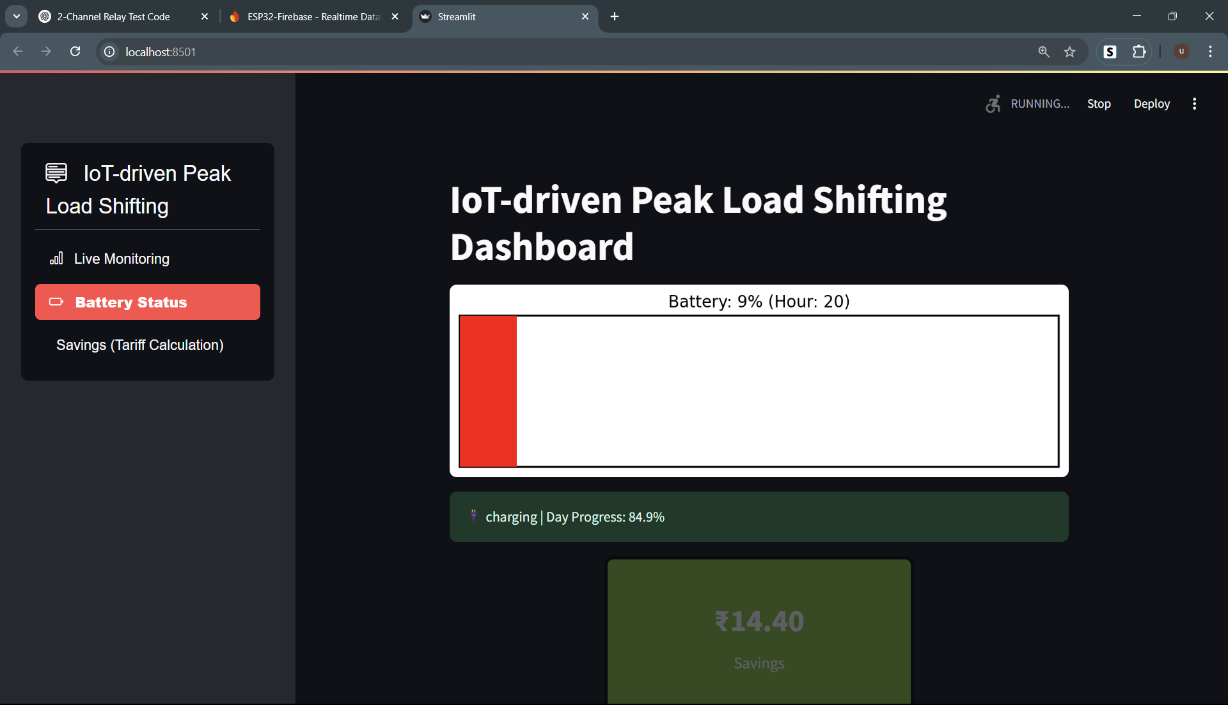
**Web UI results:**

1. **Live Monitoring:**

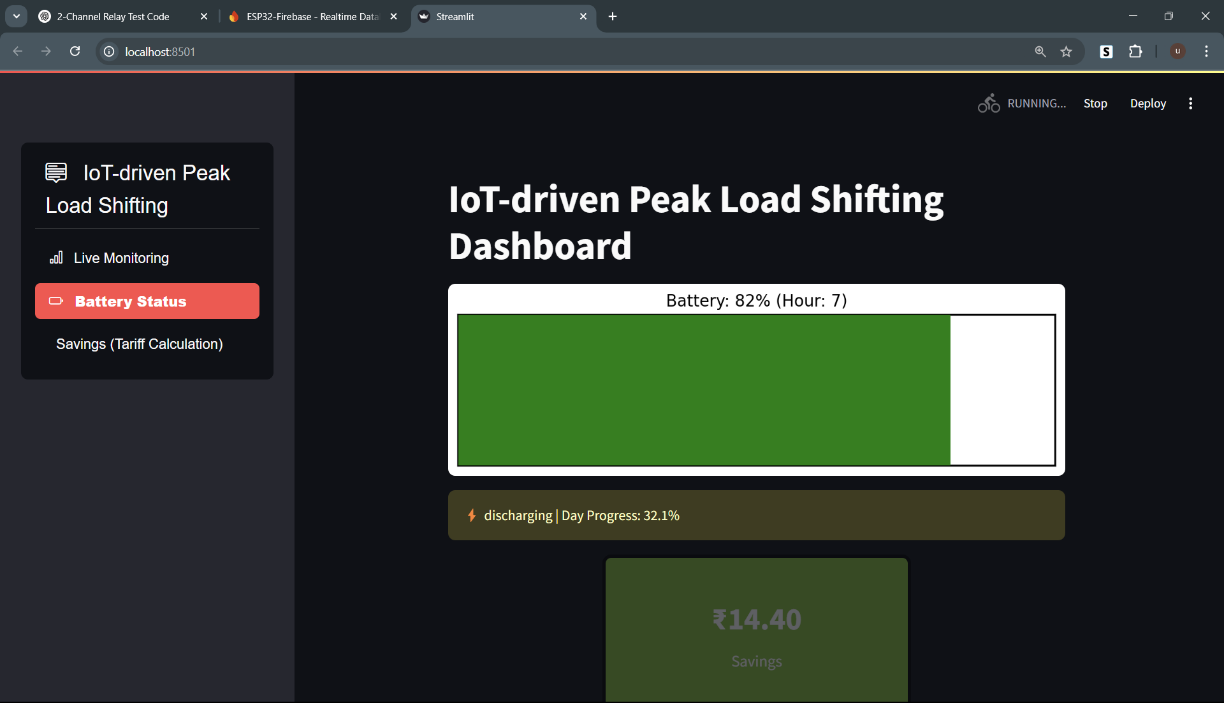
****

1. **Battery Status:**

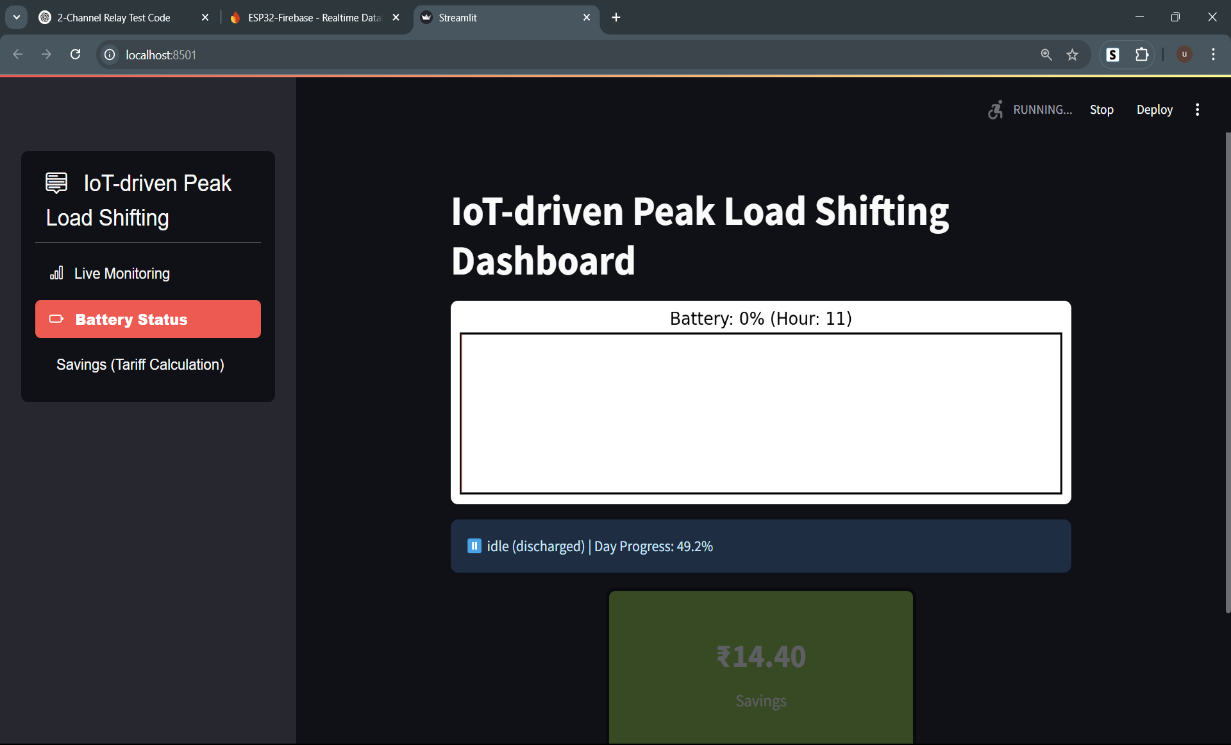
* **Battery Charging:**

****

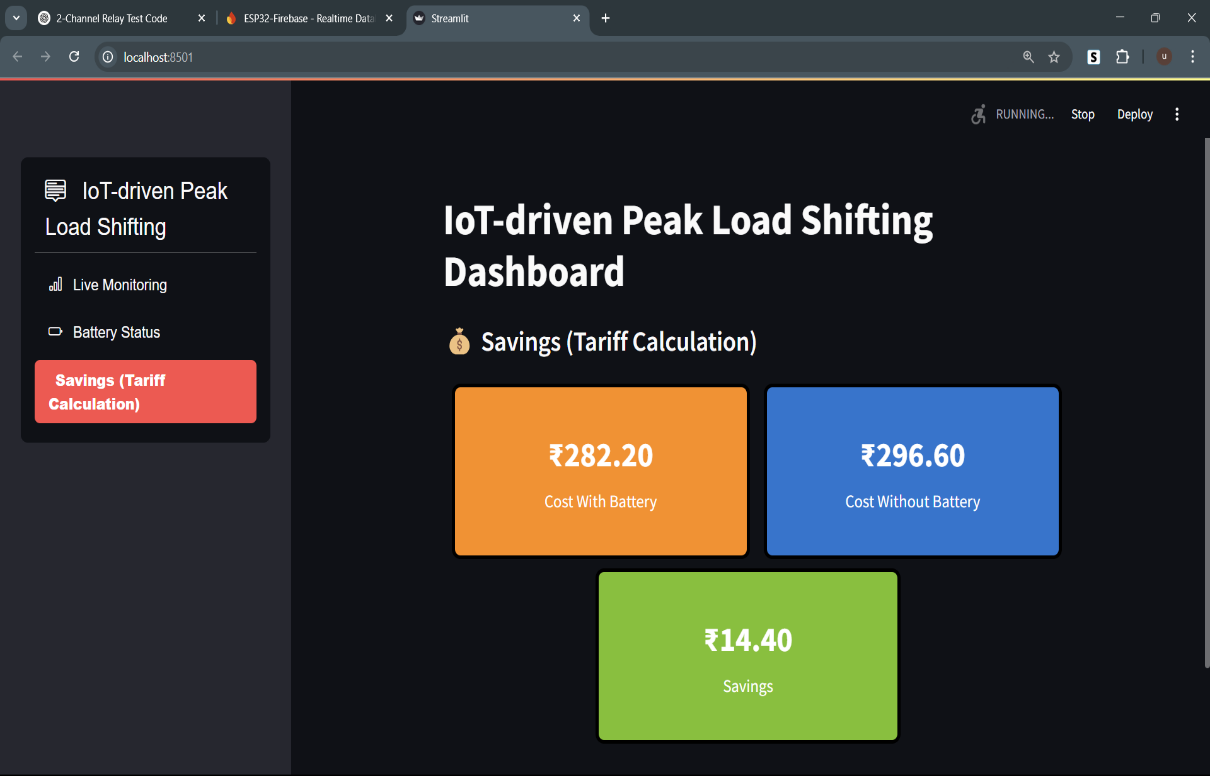
* **Battery Discharging:**

****

* **Ideal(no charging or discharging):**

****

1. **Savings(Tarrif calculation):**

****

**Conclusion:**

This project successfully demonstrates an intelligent and cost-effective approach to peak load management using an IoT-enabled pseudo battery system. By integrating ESP32-based sensing and actuation nodes with a centralized edge node for machine learning-based decision-making, the system was able to efficiently identify off-peak and peak hours and perform corresponding charging and discharging actions. The use of DBSCAN clustering on real-time power data enabled dynamic and unsupervised scheduling of battery operations, eliminating the need for manual intervention.

The implementation of Firebase cloud ensured real-time data synchronization, storage, and visualization. Furthermore, the Streamlit-based web dashboard provided an intuitive interface to monitor load consumption, battery status, and tariff-based savings—giving users valuable insights into their energy usage behavior.

Through this project, we have not only addressed the issue of peak load shifting but also showcased the potential of combining IoT, edge computing, and machine learning in a smart grid environment. The pseudo battery system simulated here can be scaled with real battery storage in future work, contributing towards energy efficiency and demand-side management in real-world applications.

**Reference:**

1. A. P. Al Mamun, N. Amin, T. T. Lie, and H. R. Pota, “A comprehensive review of demand side management for residential consumers,” *Sustainable Energy, Grids and Networks*, vol. 36, p. 101089, 2023, doi: 10.1016/j.segan.2023.101089.
2. I. R. F. Jasim, A. A. Al-Khafaji, A. M. Hameed, and A. S. Jasim, “Coordinated control and load shifting‐based demand response strategy for residential energy management system,” *International Transactions on Electrical Energy Systems*, vol. 33, no. 5, May 2023, doi: 10.1002/2050-7038.13437.
3. N. Javaid *et al.*, “Design and implementation of an IoT-based smart energy management system,” *IEEE Access*, vol. 6, pp. 31575–31585, 2018, doi: 10.1109/ACCESS.2018.2837144.
4. S. Hussain, M. Jamil, M. A. Khan, and A. Nayyar, “IoT-enabled proposal for adaptive self-powered renewable energy management in home systems,” in *2020 3rd International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, Sukkur, Pakistan, 2020, pp. 1–6, doi: 10.1109/iCoMET48670.2020.9073872.
5. A. S. Mohamed *et al.*, “Energy management and control for grid-connected photovoltaic systems with storage: A review,” *Energy Reports*, vol. 10, pp. 377–392, 2024, doi: 10.1016/j.egyr.2024.01.005.
6. K. Mahmud, M. J. Hossain, and G. E. Town, “Peak-load reduction by coordinated response of photovoltaics, battery storage, and electric vehicles,” *IEEE Access*, vol. 6, pp. 29353–29365, 2018, doi: 10.1109/ACCESS.2018.2837144.
7. G. Y. Lee *et al.*, “A self-powered energy harvesting wireless sensor node using piezoelectric energy harvesting for structural health monitoring,” *Sensors*, vol. 17, no. 2812, pp. 1–15, 2017, doi: [10.3390/s17122812](https://doi.org/10.3390/s17122812).
8. K. K. Mandal, S. N. Singh, and I. A. Kar, “Smart energy systems: Design, implementation, and deployment,” *Designs*, vol. 8, no. 1, p. 11, 2024, doi: [10.3390/designs8010011](https://doi.org/10.3390/designs8010011).
9. M. F. M. Elias *et al.*, “Smart home energy management system using Internet of Things and renewable energy sources,” *Sensors*, vol. 20, no. 3155, pp. 1–23, 2020, doi: [10.3390/s20113155](https://doi.org/10.3390/s20113155).
10. J. A. Khan *et al.*, “Design and implementation of a smart home energy management system with photovoltaic and battery storage integration,” *Energies*, vol. 16, no. 4957, pp. 1–19, 2023, doi: [10.3390/en16134957](https://doi.org/10.3390/en16134957).
11. A. Al-Ali, I. Zualkernan, R. Rashid, M. Al-Ayyoub, and H. Aloul, “IoT-based energy management system for smart buildings,” Energies, vol. 16, no. 12, p. 4835, Jun. 2023. [Online]. Available: <https://doi.org/10.3390/en16124835>
12. S. Rani, B. Sharma, and M. Kumar, “Smart energy management system for residential buildings using IoT and machine learning,” Discover Internet of Things, vol. 3, no. 1, Mar. 2025. [Online]. Available: <https://doi.org/10.1007/s43067-025-00198-w>
13. A. Yadav, S. P. Singh, and S. K. Singh, “Energy management in smart homes using IoT and hybrid optimization algorithm,” e & i Elektrotechnik und Informationstechnik, Mar. 2024. [Online]. Available: <https://doi.org/10.1007/s00502-024-02473-x>