



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.

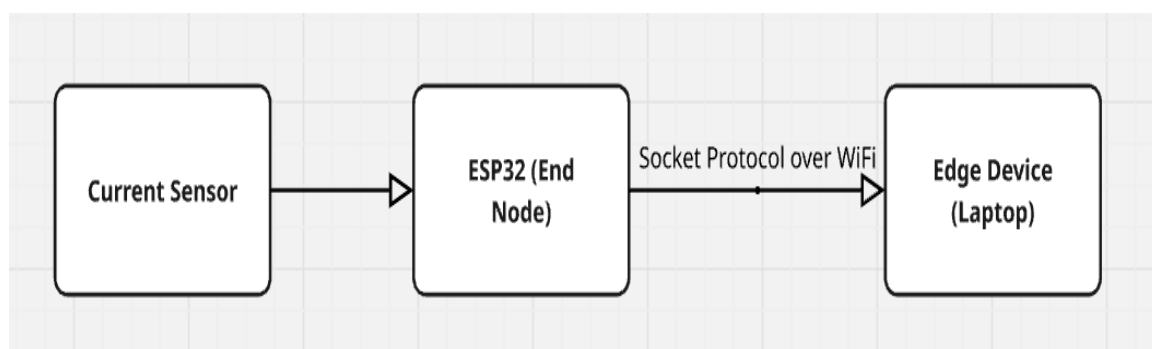
Time Compression Logic:

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.

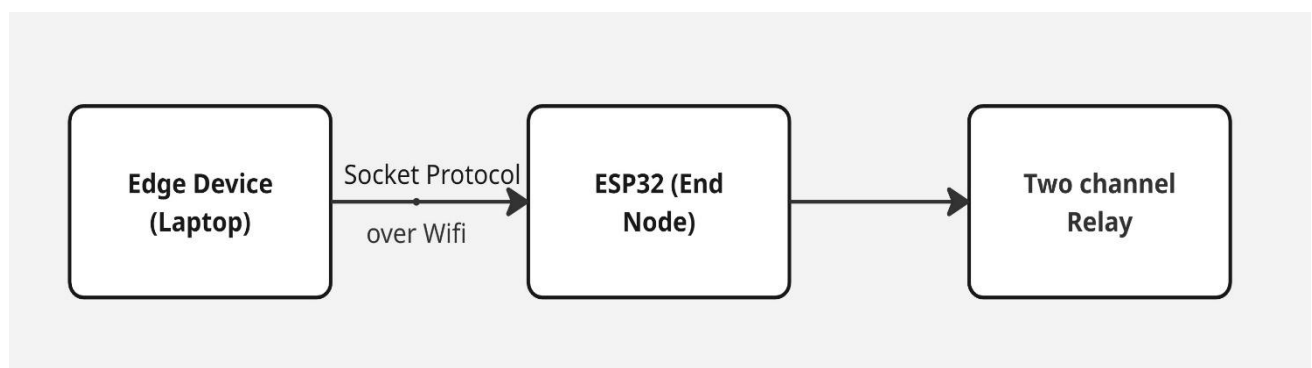
Sensor output is read by an ESP32, which:

- Samples current at fixed intervals.
- Assumes voltage as a constant (e.g., 12V).
- Calculates instantaneous power:
 - $\text{Power (W)} = \text{Current (A)} \times 12\text{V}$
- 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.

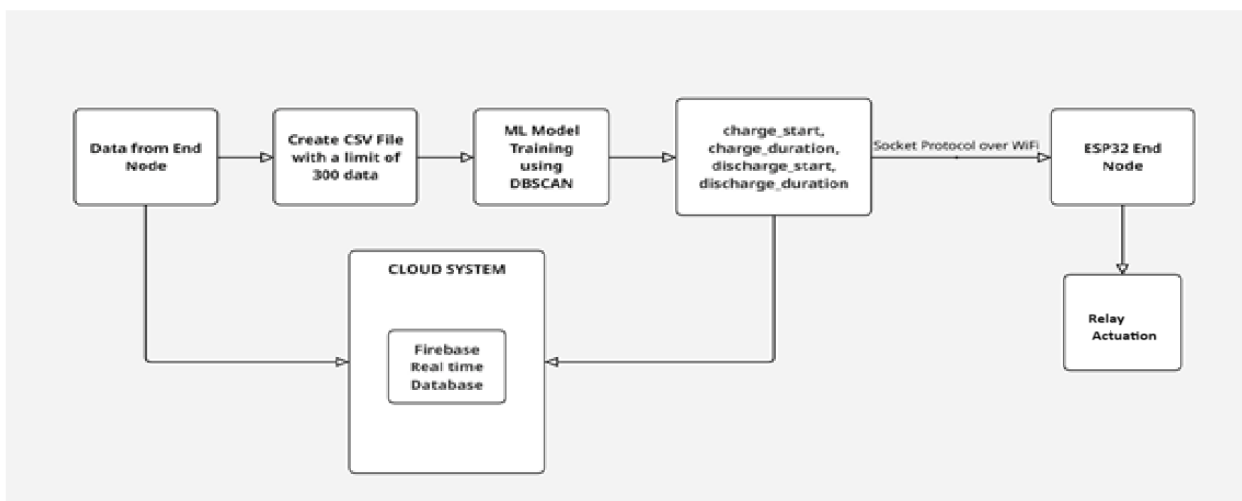
The Edge Node, after processing historical consumption data using the DBSCAN clustering algorithm, sends four key control parameters to this ESP32 node:

- 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.

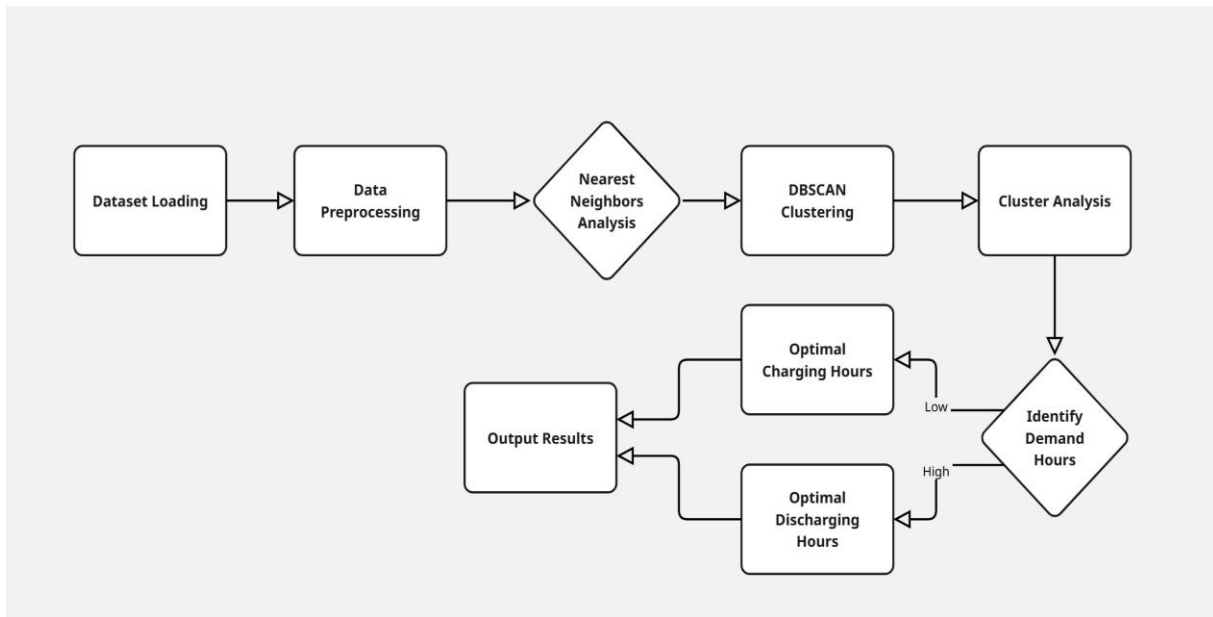


Key Responsibilities of the Edge Node

1. Receive real-time current data from the sensing ESP32 end node using Socket protocol.
2. Calculate power consumption using the constant voltage assumption ($\text{Power} = \text{Current} \times 12\text{V}$).
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
7. Send these four parameters to the actuation ESP32 end node through Socket protocol over Wi-Fi.
8. 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 \times 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:
 - $\text{Power (W)} = \text{Current (A)} \times 12\text{V}$
- 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.

2. Current Sensor (ACS712):

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

3. DC Power Supply:

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

4. Rheostat (Variable Resistor):

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

5. 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.

2. Python:

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

3. scikit-learn (Python Library):

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

4. Streamlit:

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

5. 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)

2. 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.

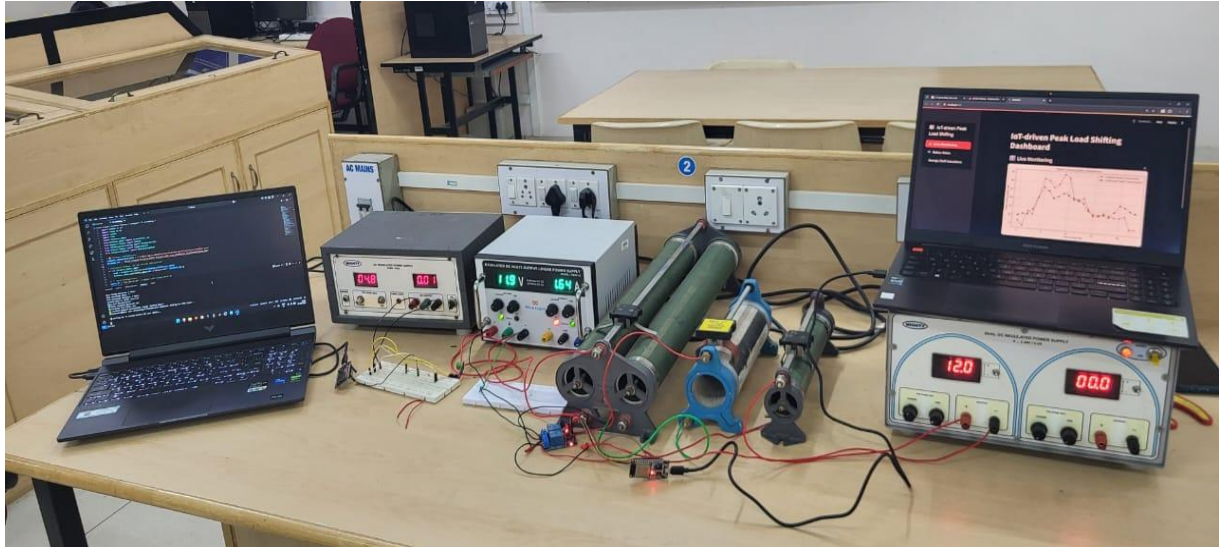
Results & Analysis:

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

1. Project Setup Overview
 - 1.1 Hardware Connections and Layout
2. End Node 1 – Sensing Unit
 - 2.1 ESP32 Client Code for Sensing and Transmission
 - 2.2 CSV File Snapshot – Power Data Logging
3. Edge Node – Data Processing Unit
 - 3.1 Server Code to Receive Sensor Data
 - 3.2 DBSCAN ML Model – Pattern Detection
4. End Node 2 – Actuation Unit
 - 4.1 ESP32 Server Code for Receiving Battery Scheduling
5. Firebase Cloud Integration
 - 5.1 Real-Time Database Snapshot
6. Streamlit Web UI Implementation
 - 6.1 Tariff Calculation Logic
 - 6.2 Main Dashboard UI Code
7. Web UI Output and Visualization
 - 7.1 Live Monitoring of Load Consumption
 - 7.2 Battery Status Visualization
 - 7.2.1 Charging State
 - 7.2.2 DisCharging State
 - 7.2.3 Idle
 - 7.3 Tariff Comparison and Savings Display

1. Project Setup Overview:

1.1 Hardware Connections and Layout:



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

2. End Node 1 – Sensing Unit:

2.1 ESP32 Client Code for Sensing and Transmission:

```
sketch_apr4a | Arduino IDE 2.3.5
File Edit Sketch Tools Help
ESP32 Dev Module

sketch_apr4a.ino
1 #include <WiFi.h>
2
3 // Wi-Fi Credentials
4 const char* ssid = "Galaxy F23 5G 047F";
5 const char* password = "xmkl7700";
6
7 // Server details
8 const char* serverIP = "192.168.37.213";
9 const int serverPort = 5000;
10
11 // ACS712 Sensor Configuration
12 #define ACS712_PIN 34
13 #define SENSITIVITY 0.185
14 #define VCC 3.3
15 #define ADC_RESOLUTION 4095.0
16 #define ZERO_CURRENT_VOLTAGE 1.65
17
18 unsigned long lastSendTime = 0; // Track time between sends
19 const unsigned long sendInterval = 30000; // 30 seconds
20
21 int hourCounter = 1; // Track current hour (1 to 24)

Output Serial Monitor X
Message (Enter to send message to 'ESP32 Dev Module' on 'COM4') New Line 115200 baud
Power Reading: 11.976 W
Power Reading: 10.010 W
Power Reading: 11.317 W
Power Reading: 10.846 W
Power Reading: 11.003 W
Power Reading: 10.010 W
Power Reading: 11.108 W
Power Reading: 8.546 W
Time to send data for Hour 2
Sent to server: 0.712
Power Reading: 12.571 W

Ln 61, Col 1 ESP32 Dev Module on COM4
ENG IN 14:20 06-04-2025
```


- 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.

2.2 CSV File Snapshot – Power Data Logging:

FileHomeInsertPage LayoutFormulasDataReviewViewHelp

CutCopyFormat Painter

Clipboard

Calibri11A⁺

B I U

Merge & Center

General

Conditional FormattingFormat as TableCell Styles

InsertDelete Format

AutosumFillSort & Find & Filter > SelectAdd-ins

CommentsShare

AI

Day

Hour_0Hour_1Hour_2Hour_3Hour_4Hour_5Hour_6Hour_7Hour_8Hour_9Hour_10Hour_11Hour_12Hour_13Hour_14Hour_15Hour_16Hour_17Hour_18Hour_19Hour_20Hour_21Hour_22Hour_23

Day_1Day_2Day_3Day_4Day_5Day_6Day_7Day_8Day_9Day_10Day_11Day_12Day_13Day_14Day_15Day_16Day_17Day_18Day_19Day_20Day_21Day_22Day_23Day_24Day_25Day_26Day_27Day_28Day_29Day_30Day_31Day_32Day_33Day_34Day_35Day_36Day_37Day_38Day_39

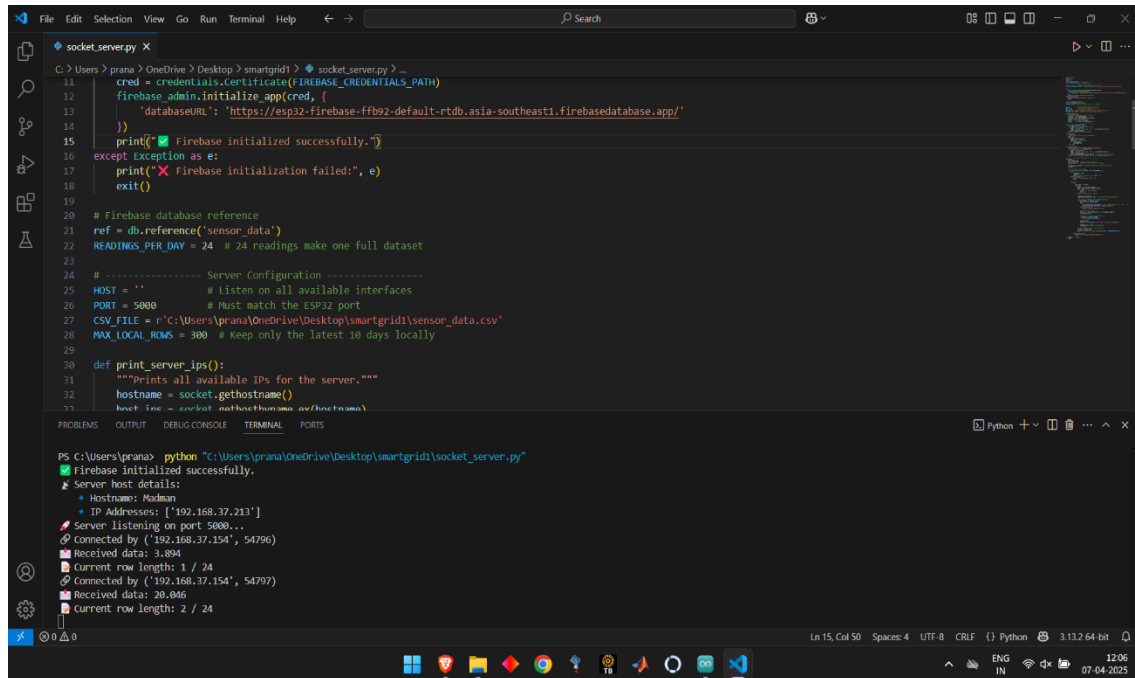
Day_1	5	9	5	13	24	32	25	33	32	39	35	24	28	16	22	16	16	29	12	17	13	6	2	8
Day_2	6	10	6	11	19	32	32	32	31	32	38	29	26	16	18	18	12	12	16	12	9	3	6	7
Day_3	3	6	10	9	29	31	31	37	34	37	31	20	22	16	20	15	16	15	16	12	16	11	5	3
Day_4	9	4	8	11	26	32	26	37	36	39	34	21	15	22	14	12	14	18	9	10	13	2	2	
Day_5	6	3	8	11	28	31	33	27	35	38	37	27	27	16	20	21	20	11	9	17	9	6	4	3
Day_6	4	3	7	19	27	29	27	33	32	33	34	22	25	22	23	16	12	18	11	12	15	3	8	4
Day_7	7	13	9	15	28	30	30	32	31	40	39	29	26	17	15	18	14	15	16	11	16	13	8	11
Day_8	11	12	15	14	27	32	30	39	38	40	29	22	21	19	15	14	15	15	14	13	12	10	7	9
Day_9	10	11	7	19	24	32	35	35	33	31	35	23	26	19	19	14	11	11	16	12	6	10	4	
Day_10	2	4	13	14	28	30	32	35	32	41	34	21	18	17	20	12	17	19	8	10	16	7	12	2
Day_11	4	9	13	14	27	32	34	36	34	41	37	21	24	20	14	20	15	17	9	8	10	11	10	9
Day_12	11	8	11	14	20	25	33	37	34	40	35	24	28	21	17	15	21	13	17	12	10	6	3	1
Day_13	9	3	8	10	21	30	25	31	34	40	38	29	18	15	22	21	12	16	13	14	8	6	3	6
Day_14	1	6	15	10	24	26	29	32	33	32	32	20	25	25	22	13	17	18	10	11	16	7	2	9
Day_15	9	8	10	16	20	30	26	35	35	41	41	28	28	17	13	22	17	9	17	11	13	12	9	
Day_16	4	4	8	17	29	28	27	27	29	40	33	22	27	15	14	16	14	16	11	12	12	4	8	8
Day_17	3	7	10	19	19	24	27	29	31	36	35	19	27	25	13	14	22	18	9	15	15	10	12	10
Day_18	4	9	14	15	27	32	28	29	39	31	35	26	23	24	23	14	12	12	11	18	7	12	7	3
Day_19	4	5	10	19	25	29	30	28	36	33	33	28	23	25	17	15	12	13	16	12	9	11	9	9
Day_20	6	6	5	10	28	24	27	33	32	35	37	20	22	17	18	18	12	11	12	13	13	13	11	11
Day_21	8	9	15	11	25	27	30	34	35	35	37	27	25	21	13	20	22	19	18	16	9	8	11	10
Day_22	6	7	8	15	26	31	31	30	31	37	32	27	20	25	16	17	15	12	12	8	13	6	10	10
Day_23	4	3	10	19	25	31	31	30	31	37	32	28	18	24	15	14	13	11	16	16	13	10	9	7
Day_24	6	10	18	17	24	26	29	32	33	41	32	27	24	24	13	16	21	16	12	8	10	3	2	6
Day_25	9	6	12	19	24	26	25	29	38	32	40	26	19	19	22	17	14	14	12	10	14	4	3	7
Day_26	8	8	14	16	19	32	33	30	36	33	36	21	19	12	19	12	21	15	18	13	8	13	10	6
Day_27	5	10	5	18	23	30	29	31	39	41	32	24	18	19	18	13	22	15	17	16	16	10	4	2
Day_28	8	8	7	19	20	25	27	30	38	38	33	20	25	16	20	20	14	20	9	11	15	4	6	6
Day_29	5	13	6	16	24	31	25	37	34	40	41	24	21	19	17	12	21	11	14	8	17	7	2	7
Day_30	8	12	11	17	29	26	33	35	33	40	31	27	21	20	18	13	15	11	14	15	12	7	7	3
Day_31	9	8	10	15	24	31	32	35	32	32	33	20	26	19	17	17	20	10	16	11	12	10	2	8
Day_32	7	3	6	14	20	25	30	37	32	33	41	19	23	22	20	17	13	13	9	11	14	11	3	2
Day_33	6	3	13	19	25	28	32	27	38	34	39	23	27	21	22	22	16	16	16	18	11	3	3	5
Day_34	9	6	16	19	28	33	30	31	35	38	36	25	18	24	17	22	19	15	8	13	15	3	10	9
Day_35	4	4	6	15	24	30	29	35	34	33	40	27	22	18	14	14	18	13	12	11	13	5	6	11
Day_36	7	13	13	11	24	24	27	37	33	33	40	19	25	19	19	21	22	11	11	17	14	9	4	10
Day_37	10	7	10	11	21	28	31	34	36	41	40	27	22	22	19	18	16	18	14	13	6	4	10	7
Day_38	8	8	14	10	25	31	28	37	39	35	33	21	24	21	19	18	16	13	16	16	13	4	8	7
Day_39	10	10	11	13	21	29	29	28	36	36	35	24	23	25	16	12	12	13	16	13	17	7	11	7

power_consumption_300_days_fina

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.

3. Edge Node – Data Processing Unit:

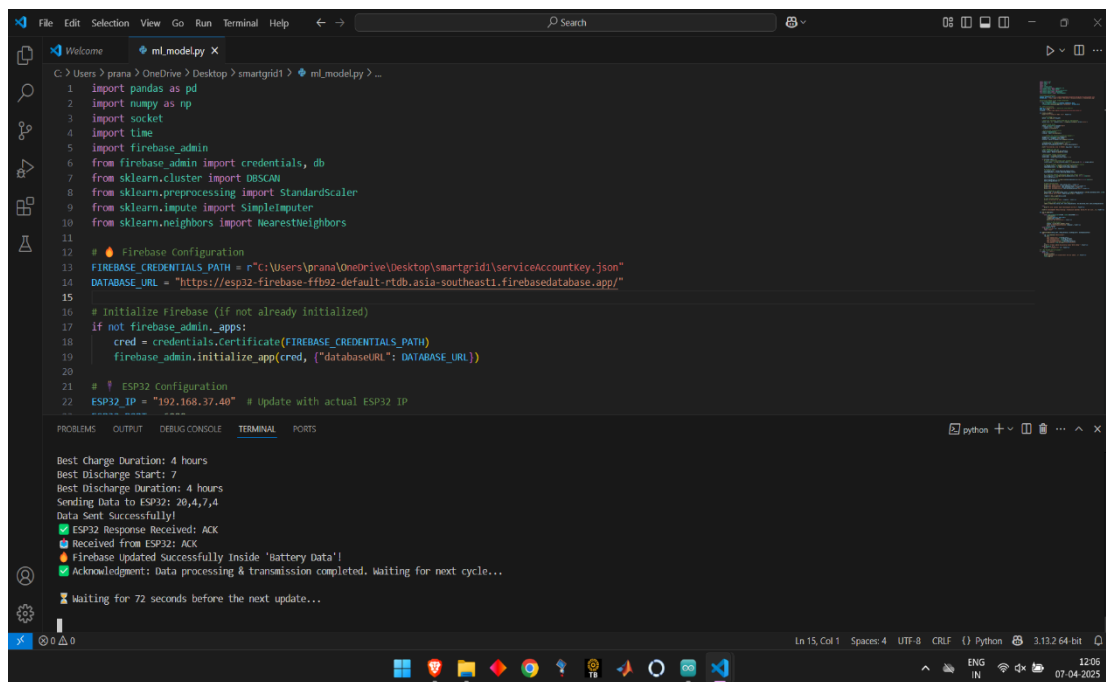
3.1 Server Code to Receive Sensor Data:



```
socket_server.py
11 cred = credentials.Certificate(FIREBASE_CREDENTIALS_PATH)
12 firebase_admin.initialize_app(cred, {
13     'databaseURL': 'https://esp32-firebase-ffb02-default-rtdb.asia-southeast1.firebaseio.com/'
14 })
15 print("✅ Firebase initialized successfully.")
16 except Exception as e:
17     print("❌ Firebase initialization failed:", e)
18     exit()
19
20 # Firebase database reference
21 ref = db.reference('sensor_data')
22 READINGS_PER_DAY = 24 # 24 readings make one full dataset
23
24 # ----- Server Configuration -----
25 HOST = '*' # Listen on all available interfaces
26 PORT = 5000 # Must match the ESP32 port
27 CSV_FILE = r'C:\Users\prana\OneDrive\Desktop\smartgrid1\sensor_data.csv'
28 MAX_LOCAL_ROWS = 300 # Keep only the latest 10 days locally
29
30 def print_server_ips():
31     """Prints all available IPs for the server."""
32     hostname = socket.gethostname()
33     host_ip = socket.gethostbyname(hostname)
34
35 PS C:\Users\prana> python "C:\Users\prana\OneDrive\Desktop\smartgrid1\socket_server.py"
✅ Firebase initialized successfully.
Server host details:
  Hostname: Rodman
  IP Addresses: ['192.168.37.213']
Server listening on port 5000...
Connected by ('192.168.37.154', 54796)
Received data: 3.894
Current row length: 1 / 24
Connected by ('192.168.37.154', 54797)
Received data: 20.046
Current row length: 2 / 24
```

- 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.

3.2 DBSCAN ML Model – Pattern Detection:



The screenshot shows a Visual Studio Code editor window with a file named `ml_model.py` open. The script is a Python program that interacts with a Firebase database and an ESP32 microcontroller. It imports libraries like `pandas`, `numpy`, `socket`, `time`, `firebase_admin`, `sklearn.cluster`, `sklearn.preprocessing`, `sklearn.impute`, and `sklearn.neighbors`. It configures Firebase with a service account key and a database URL, initializes it, and then sets up an ESP32 IP address. The terminal output shows the script's execution, including logging the best charge duration (4 hours), best discharge start (7), and best discharge duration (4 hours). It also shows data being sent to the ESP32, successful responses, and Firebase updates.

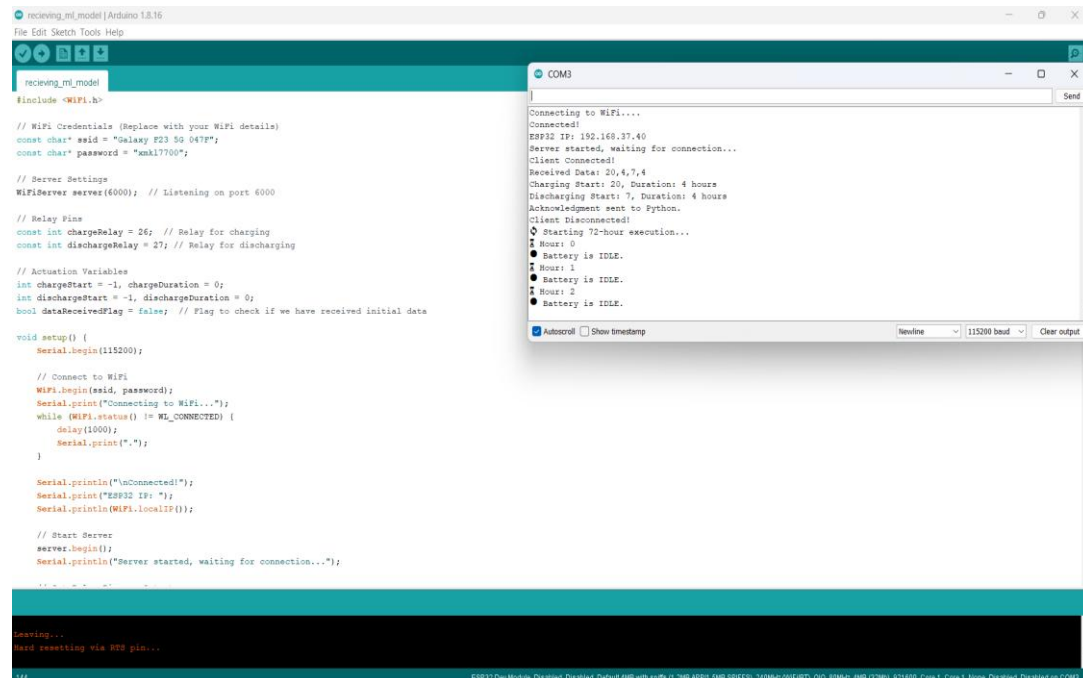
```
1 import pandas as pd
2 import numpy as np
3 import socket
4 import time
5 import firebase_admin
6 from firebase_admin import credentials, db
7 from sklearn.cluster import DBSCAN
8 from sklearn.preprocessing import StandardScaler
9 from sklearn.impute import SimpleImputer
10 from sklearn.neighbors import NearestNeighbors
11
12 # Firebase Configuration
13 FIREBASE_CREDENTIALS_PATH = r"C:\Users\prana\OneDrive\Desktop\smartgrid1\serviceAccountKey.json"
14 DATABASE_URL = "https://esp32-firebase-ffb92-default-rtdb.asia-southeast1.firebaseio.com/"
15
16 # Initialize Firebase (if not already initialized)
17 if not firebase_admin._apps:
18     cred = credentials.Certificate(FIREBASE_CREDENTIALS_PATH)
19     firebase_admin.initialize_app(cred, {"databaseURL": DATABASE_URL})
20
21 # ESP32 Configuration
22 ESP32_IP = "192.168.37.40" # Update with actual ESP32 IP
```

Best Charge Duration: 4 hours
Best Discharge Start: 7
Best Discharge Duration: 4 hours
Sending Data to ESP32: 20,4,7,4
Data Sent Successfully!
ESP32 Response Received: ACK
Received from ESP32: ACK
Firebase Updated Successfully Inside 'Battery Data'
Acknowledgment: Data processing & transmission completed. Waiting for next cycle...
Waiting for 72 seconds before the next update...

- 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.

4. End Node 2 – Actuation Unit:

4.1 Server code for getting Battery from ML model:



The screenshot displays the Arduino IDE interface. The main window shows the sketch 'receiving_ml_model' with the following code:

```
#include <WiFi.h>

// WiFi Credentials (Replace with your WiFi details)
const char* ssid = "Galaxy P23 5G 0478";
const char* password = "xmk17700";

// Server Settings
WiFiServer server(6000); // Listening on port 6000

// Relay Pins
const int chargeRelay = 26; // Relay for charging
const int dischargeRelay = 27; // Relay for discharging

// Actuation Variables
int chargeStart = -1, chargeDuration = 0;
int dischargeStart = -1, dischargeDuration = 0;
bool dataReceivedFlag = false; // Flag to check if we have received initial data

void setup() {
  Serial.begin(115200);

  // Connect to WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.print(".");
  }

  Serial.println("\nConnected!");
  Serial.print("ESP32 IP: ");
  Serial.println(WiFi.localIP());

  // Start Server
  server.begin();
  Serial.println("Server started, waiting for connection...");
}
```

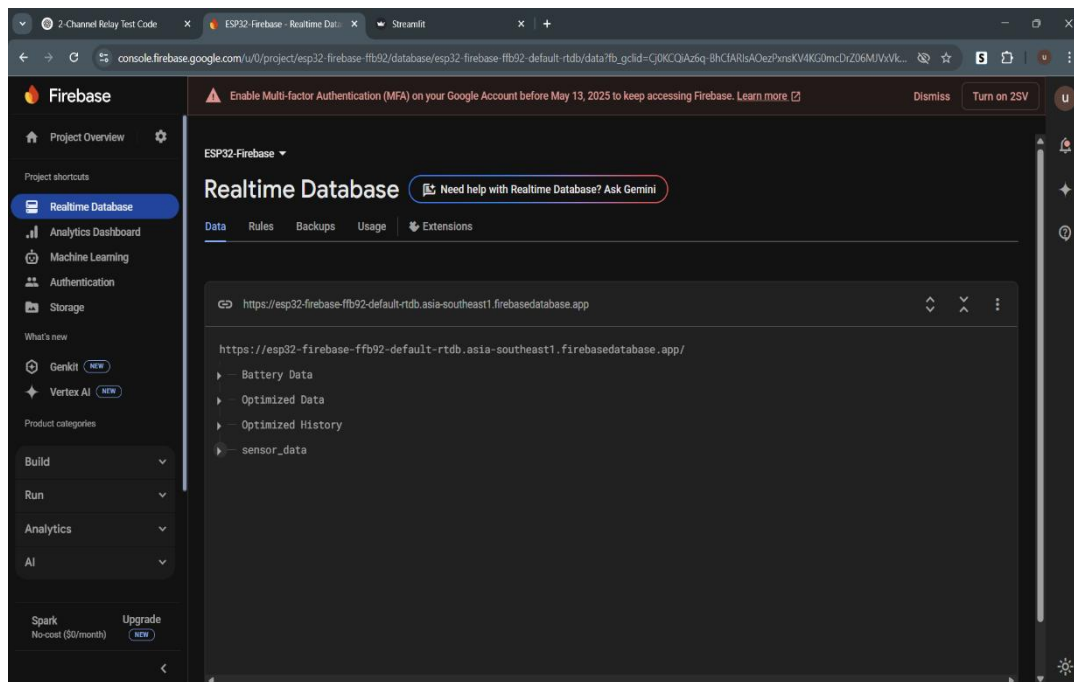
The serial monitor (COM3) shows the following output:

```
Connecting to WiFi....
Connected!
ESP32 IP: 192.168.37.40
Server started, waiting for connection...
Client Connected!
Received Data: 20,4,7,4
Charging Start: 20, Duration: 4 hours
Discharging Start: 7, Duration: 4 hours
Acknowledgment sent to Python.
Client disconnected!
Starting 72-hour execution...
Hour: 0
Battery is IDLE.
Hour: 1
Battery is IDLE.
Hour: 2
Battery is IDLE.
```

- 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.

5. Firebase Cloud Integration:

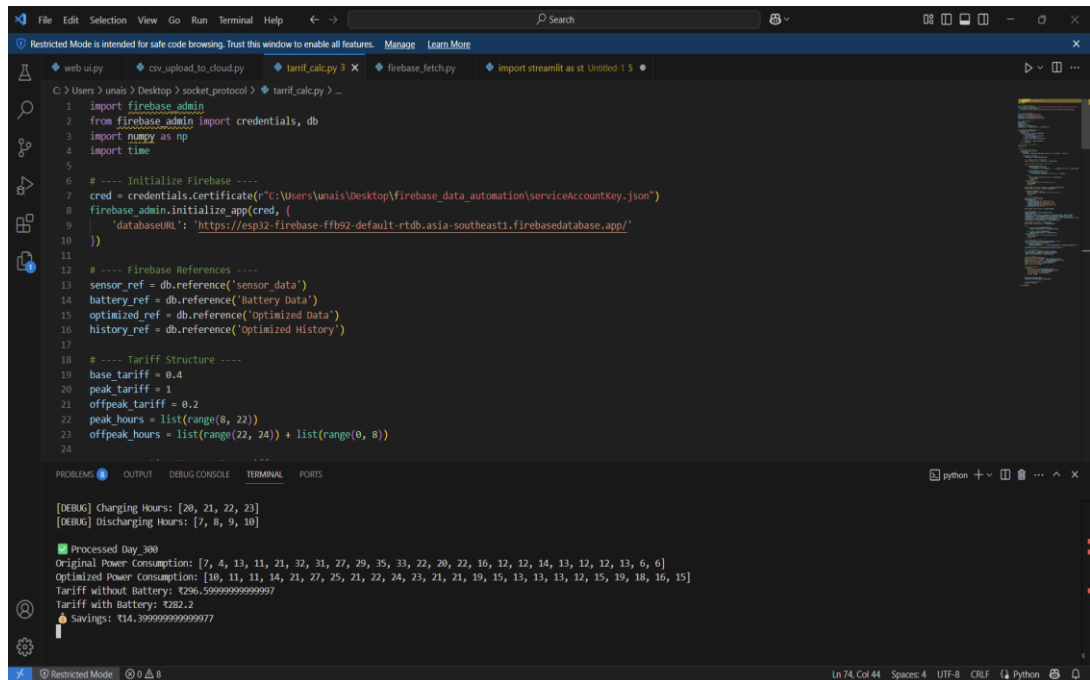
5.1 Real-Time Database Snapshot:



- **Structured Data Storage:** The Firebase Realtime Database is used to manage critical project data such as Battery Data, Optimized Data, Optimized History, and sensor_data, enabling seamless real-time updates and efficient backend communication.
- **IoT Integration Backbone:** This database supports the ESP32-based IoT system by storing and syncing energy consumption and optimization values, ensuring effective load shifting and monitoring.
- **Scalable & Cloud-Based:** Being hosted on Google Firebase, the system benefits from cloud scalability, real-time syncing, and accessibility—making it ideal for remote energy management applications.

6. Streamlit Web UI Implementation:

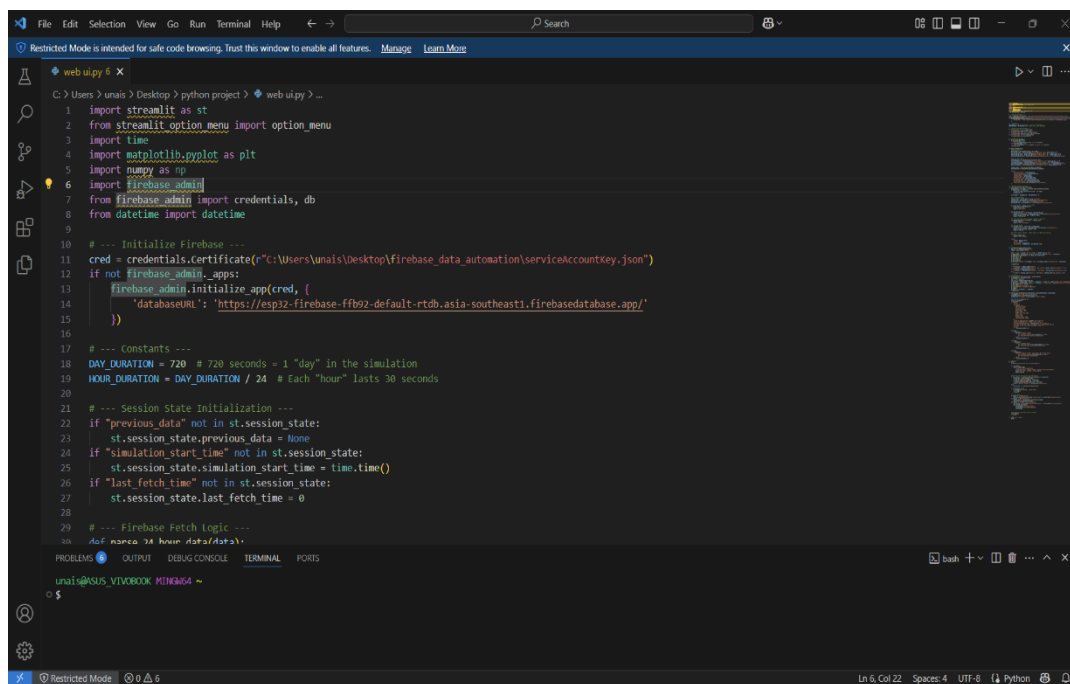
6.1 Tariff Calculation Logic:



```
1 import firebase_admin
2 from firebase_admin import credentials, db
3 import numpy as np
4 import time
5
6 # ---- Initialize Firebase ----
7 cred = credentials.Certificate("C:\\Users\\unais\\Desktop\\firebase_data_automation\\serviceAccountKey.json")
8 firebase_admin.initialize_app(cred, {
9     'databaseURL': 'https://esp32-firebase-ffb92-default-rtdb.asia-southeast1.firebaseio.com/'
10 })
11
12 # ---- Firebase References ----
13 sensor_ref = db.reference('sensor_data')
14 battery_ref = db.reference('Battery Data')
15 optimized_ref = db.reference('Optimized Data')
16 history_ref = db.reference('Optimized History')
17
18 # ---- Tariff Structure ----
19 base_tariff = 0.4
20 peak_tariff = 1
21 offpeak_tariff = 0.2
22 peak_hours = list(range(8, 22))
23 offpeak_hours = list(range(22, 24)) + list(range(0, 8))
24
25
26 [DEBUG] Charging Hours: [20, 21, 22, 23]
27 [DEBUG] Discharging Hours: [7, 8, 9, 10]
28
29 Processed Day_300
30 Original Power Consumption: [7, 4, 13, 11, 21, 32, 31, 27, 29, 35, 33, 22, 20, 22, 16, 12, 12, 14, 13, 12, 12, 13, 6, 6]
31 Optimized Power Consumption: [10, 11, 11, 14, 21, 27, 25, 21, 22, 24, 23, 21, 21, 19, 15, 13, 13, 13, 12, 15, 19, 18, 16, 15]
32 Tariff without Battery: $296.59999999999997
33 Tariff with Battery: $282.2
34 Savings: $14.399999999999977
```

- 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.

6.2 Main Dashboard UI Code:

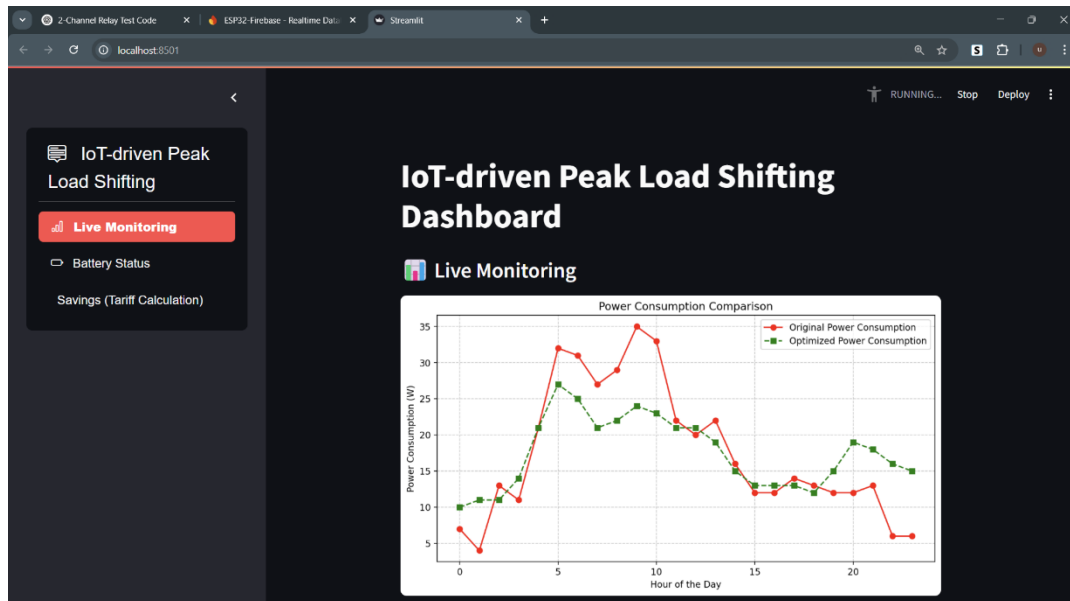


```
1 import streamlit as st
2 from streamlit_option_menu import option_menu
3 import time
4 import matplotlib.pyplot as plt
5 import numpy as np
6 import firebase_admin
7 from firebase_admin import credentials, db
8 from datetime import datetime
9
10 # --- Initialize Firebase ---
11 cred = credentials.Certificate(r"C:\Users\unais\Desktop\firebase_data_automation\serviceaccountkey.json")
12 if not firebase_admin._apps:
13     firebase_admin.initialize_app(cred, {
14         'databaseURL': 'https://esp32-firebase-ffb02-default-rtdb.asia-southeast1.firebaseio.com/'
15     })
16
17 # --- Constants ---
18 DAY_DURATION = 720 # 720 seconds = 1 "day" in the simulation
19 HOUR_DURATION = DAY_DURATION / 24 # Each "hour" lasts 30 seconds
20
21 # --- Session State Initialization ---
22 if "previous_data" not in st.session_state:
23     st.session_state.previous_data = None
24 if "simulation_start_time" not in st.session_state:
25     st.session_state.simulation_start_time = time.time()
26 if "last_fetch_time" not in st.session_state:
27     st.session_state.last_fetch_time = 0
28
29 # --- Firebase Fetch Logic ---
30 def fetch_data():
31     # Fetch data from Firebase Realtime Database
32     # ... (code for fetching data) ...
33     return data
34
35 # Main function to run the simulation
36 def main():
37     # Sidebar menu
38     option_menu(
39         title="Dashboard",
40         menu_items=[
41             ("Live Monitoring", "#", "📡"),
42             ("Battery Status", "#", "🔋"),
43             ("Savings", "#", "💰"),
44         ],
45         default_index=0,
46     )
47
48     # Main content area
49     # ... (code for displaying data and charts) ...
50
51 # Run the main function
52 if __name__ == "__main__":
53     main()
```

- **Firebase Integration:** Integrate with Firebase Realtime Database to retrieve the 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.
- It has a sidebar menu of navigation with options for "Live Monitoring", "Battery Status", and "Savings".

7. Web UI Output and Visualization:

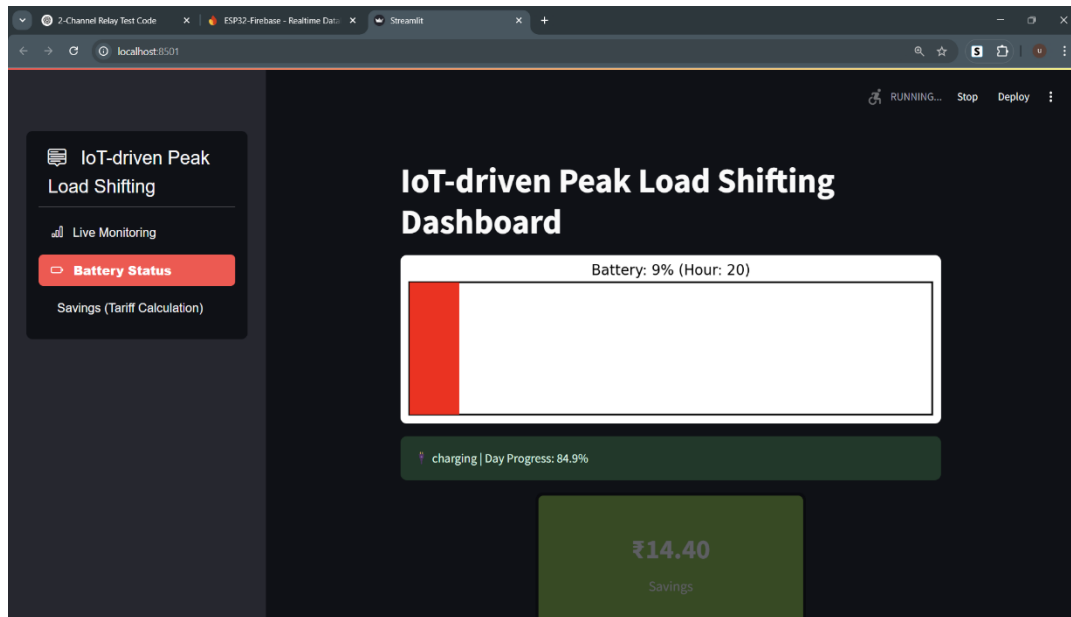
7.1 Live Monitoring of Load Consumption:



- **Real-Time Monitoring Interface:** The dashboard provides a live comparison of original vs. optimized power consumption, enabling real-time tracking of load shifting effectiveness.
- **Effective Peak Load Reduction:** The graph shows a clear reduction in power usage during peak hours (in red) compared to optimized values (in green), highlighting the efficiency of the IoT-based control strategy.
- **User-Friendly Design:** The sidebar includes quick access to other modules like Battery Status and Tariff Calculation, making the system intuitive and suitable for energy management applications.

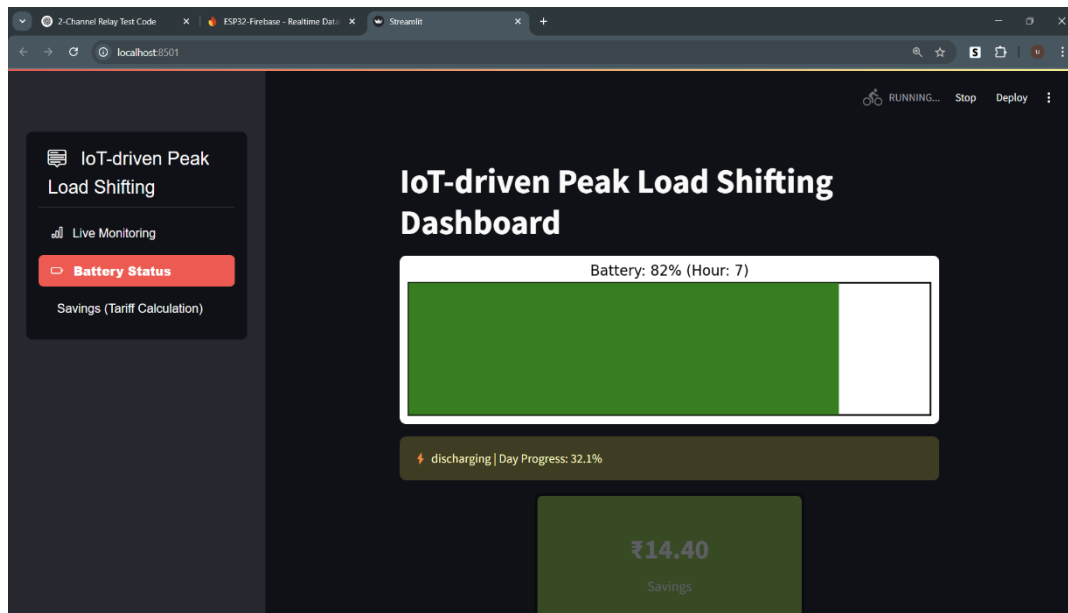
7.2 Battery Status Visualization:

7.2.1 Charging State:



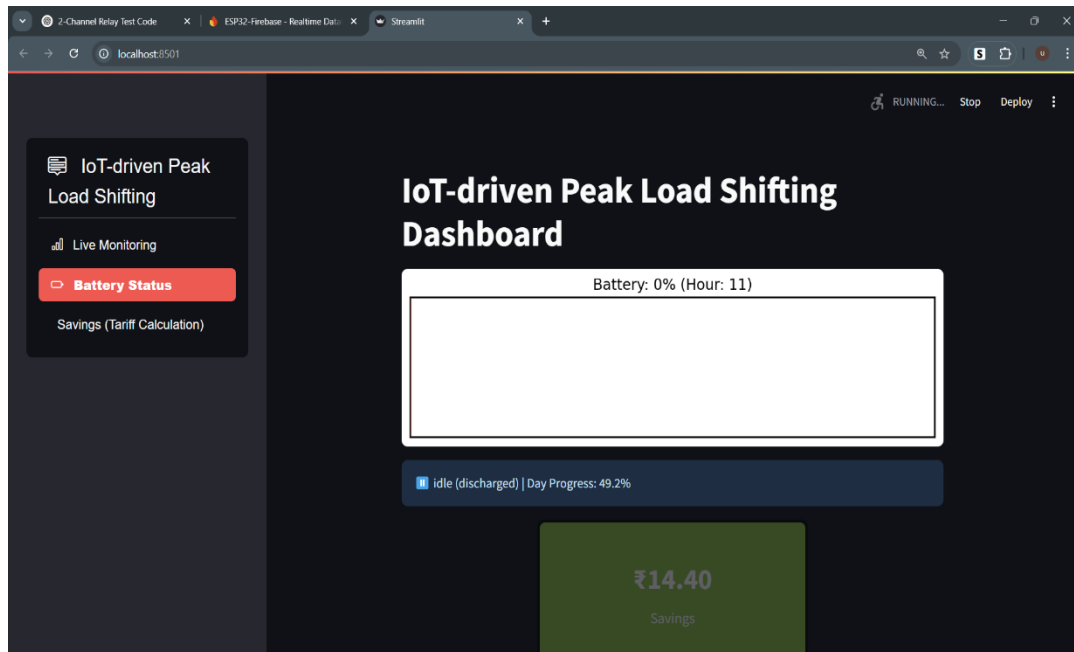
In the charging state, the battery begins charging at the 20th hour and continues for a duration of 4 hours during the off-peak period.

7.2.2 Discharging State:



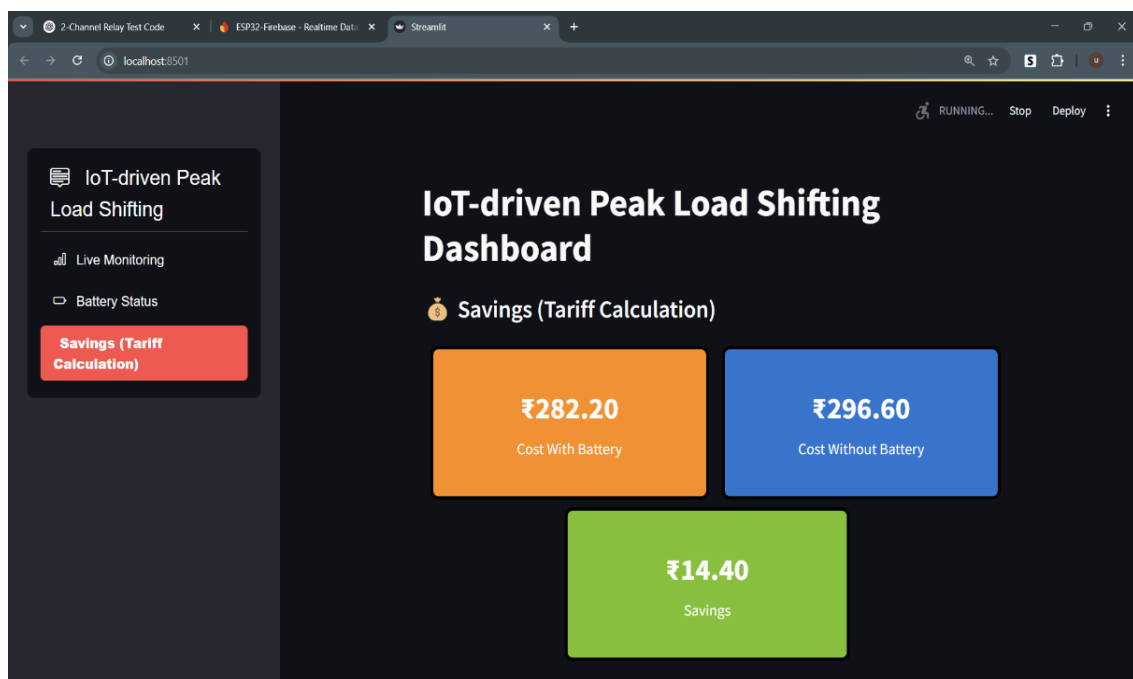
In the discharging state, the battery begins discharging at the 7th hour and continues for a duration of 4 hours during the high-peak period.

7.2.3 Idle:



In the idle state, the battery remains inactive, meaning it is neither charging nor discharging.

7.3 Tariff Comparison and Savings Display:



- **Cost Comparison Insight:** The cost of energy consumption with battery usage is ₹282.20, which is ₹14.40 less than the ₹296.60 cost without battery, indicating a noticeable reduction in energy expense.
- **Effective Peak Load Shifting:** The ₹14.40 savings highlights that peak load shifting using a battery system successfully reduces reliance on grid power during high-tariff periods.
- **Dashboard Transparency:** The interface provides a clear visual comparison of costs and savings, making it easier for users to evaluate the economic benefit of the battery-assisted IoT load shifting system.

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. Zuolkernan, 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>