Smart Energy Analyzer for Load Forecasting and Adaptive Control

19EEE331 – Smart Grid & IoT

Report – Batch 7

Submitted by

CB.EN.U4ELC22004 Kalyan A

CB.EN.U4ELC22006 Javali B

CB.EN.U4ELC22027 Meshak A

CB.EN.U4ELC22065 Kiran V

DEPARTMENT OF ELECTRICAL AND ELECTRONICS ENGINEERING

AMRITA SCHOOL OF ENGINEERING,

AMRITA VISHWA VIDYAPEETHAM,

COIMBATORE - 641112

March 2025

AMRITA VISHWA VIDYAPEETHAM
DEEMED TO BE UNIVERSITY

BONAFIDE CERTIFICATE

This is to certify that the Smart Grid & IoT project report entitled **Smart Energy Analyzer for Load Forecasting and Adaptive Control,**

submitted by

CB.EN.U4ELC22004	**Kalyan A**
CB.EN.U4ELC22006	**Javali B**
CB.EN.U4ELC22027	**Meshak A**
CB.EN.U4ELC22065	**Kiran V**

is in partial fulfillment of the requirements for the award of the **Degree of Bachelor of Technology** in **"Electrical and Electronics Engineering"** is a bonafide record of the work carried out at Amrita School of Engineering, Coimbatore.

Internal Examiner

TABLE OF CONTENTS

INTRODUCTION

Modern power grids, especially at the residential and microgrid level, face growing challenges due to unpredictable load patterns, limited infrastructure adaptability, and rising energy demands. Traditional systems lack intelligent, decentralized decision-making capabilities—making them inefficient during peak loads or grid fluctuations. This project addresses these challenges by designing a Smart Grid Monitoring and Control System that integrates: Load Forecasting using machine learning (LSTM – Long Short-Term Memory in our case) to anticipate future power consumption, Adaptive Control via dynamic threshold-based load shedding—where the system can automatically turn off non-essential loads based on real-time and forecasted power data. Built on top of IoT(Internet of Things), edge computing, and machine learning, this system empowers users with: Real-time power visualization, Configurable control over loads (manual/automatic), Predictive insights to reduce wastage and prevent overloads and to aim for a future-ready, user-centric grid solution that’s smart, scalable.

LITERATURE REVIEW

With the rise of IoT and smart energy systems, a lot of recent work has focused on making energy management more efficient, responsive, and automated. Traditional energy systems mostly relied on manual monitoring or basic automation, but now with real-time data, wireless communication, and machine learning, things are moving towards smarter, self-adjusting grids.

One of the earlier works in this space was by Mishra et al. [1], where they built a simple IoT-based smart energy management system. Their setup mainly focused on collecting energy usage data and doing basic control, but it didn't include any kind of prediction or learning—something that's becoming essential in today's smart grids.

To bring in intelligence, researchers have started applying deep learning models. For example, Abdel-Basset et al. [2] proposed a system called Energy-Net for smart cities. They used deep learning to analyze energy patterns and improve how energy is distributed in real-time. Their work shows how AI can help reduce energy waste and balance the load across different zones.

When it comes to smaller setups like smart homes or microgrids, Khan et al. [3] came up with an AI-assisted hybrid approach. Their model combines deep learning with fuzzy logic, making it flexible and better at handling uncertain or fast-changing situations. This kind of hybrid logic is useful for real-time decisions, like turning off non-essential loads when total power crosses a threshold—which is exactly the kind of functionality our system aims to implement.

In terms of forecasting, which is a big part of our project, LSTM neural networks are quite popular. Dahlan et al. [5] used LSTM to predict building energy usage. Their results showed that these models can accurately forecast short-term consumption, which helps in better planning and automation—like deciding when to shed load or alert users before overload happens.

Another practical issue is dealing with missing or delayed IoT data. Xue et al. [10] tackled this by proposing a method to recover missing data in smart energy systems. That kind of solution is very relevant in MQTT or wireless setups where packets might get dropped or sensors might fail temporarily.

Overall, the literature supports the idea of combining IoT (for real-time data), AI (for predictions), and smart control (like relay switching) into a single system. Our project builds on these concepts, aiming to create a smart grid dashboard that can monitor power, make decisions based on thresholds, and use LSTM-based forecasting to improve reliability.

OBJECTIVE

The main objective of this project is to develop and deploy an intelligent real-time monitoring and control system for domestic or small industrial power grids. The system should ensure energy efficiency, load prioritization, and predictive control by integrating embedded communication, edge computing, and Artificial Intelligence(AI)-based forecasting.

Key goals include:

1. **Real-Time Power Monitoring:**
   Consistently monitor power usage from many end nodes and aggregate the readings at a central edge node. Power readings will be sampled at regular intervals (every 5 seconds) to provide high-resolution monitoring for sound decision-making.
2. **Load Classification and Control:**
   Implement functionality to classify electrical loads as essential and non-essential. Depending on the overall power consumption and a user-specified threshold, the system will control relays for non-necessary loads automatically or manually to avoid overconsumption or grid overload.
3. **ESP-NOW for End-to-Edge Communication:**
   Use ESP-NOW, a low-latency and lightweight connectionless protocol, to enable effective one-way or two-way communication between ESP-based end nodes and the central ESP-based edge node. The protocol is chosen following extensive testing that validated its reliability, speed, and applicability for short-range, peer-to-peer communication in resource-limited environments.
4. **MQTT for Server Communication:**
   Employ the Message Queuing Telemetry Transport(MQTT) protocol to transmit aggregated data from the edge node to a central server or application layer. MQTT also facilitates two-way communication for remote relay control, threshold setup, and system monitoring. Topic-based publish-subscribe architecture ensures modularity and scalability of the communication system.
5. **Forecasting Power Consumption Using LSTM:**
   Long Short-Term Memory (LSTM) neural network to predict short-term future power demand using recent historical data. The system periodically executes on the server to predict the future load and invoke proactive load-shedding policies if forecasted power is beyond safe limits.
6. **Interactive User Interface(UI) Dashboard:**
   A user friendly PyQt-based dashboard to: Display real-time total and per-node power consumption plots, Visualize LSTM-based predicted values, Offer relay control for every node (manual and automatic), Display node essentiality status and enable user toggling, Dynamically adjust the load threshold via a slider-based interface, Send alerts upon power exceeding limits or upon load-shedding.

SYSTEM OVERVIEW

This smart grid system is a multi-layered architecture combining embedded sensing, edge computing, wireless communication, and intelligent control through a graphical user interface. The goal is to enable real-time power monitoring, intelligent load control, and energy prediction using IoT hardware and machine learning.

The system operates across five major functional layers:

1. Sensing: Each End Node (ESP32) is equipped with:

- **ZMPT101B** for AC voltage sensing
- **ACS712** for Current measurement

These sensors provide continuous, localized electrical parameter readings at the node level, enabling accurate and distributed monitoring of power consumption.

Figure 1: ACS712 Sensor

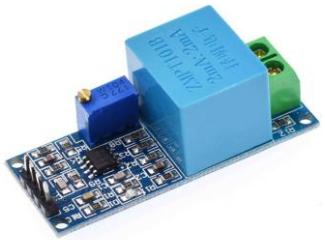

Figure 2: ZMPT101B Sensor

2. Computation

On each End Node:

- Power is calculated in real time as $P=V\times I$ using the sensed voltage and current values.
- The Edge Node further aggregates total power from all End Nodes for system-wide analysis and control and do some parsing of the data it received.

Additionally, the Server runs an LSTM model that uses historical data to forecast short-term total power usage, providing predictive insight into load trends.

3. Control(Actuation)

The system supports relay switching at each End Node, which can be triggered in two modes:

- **Manual Control** – Activated through the dashboard UI by the user.
- **Auto Control** – Triggered by the Server when the total power exceeds a user-defined threshold. In this mode, all non-essential nodes are automatically shut off to reduce load.

Relays are controlled through commands received over ESP-NOW, enabling quick response and localized actuation.

Figure 1 Courtesy: https://www.amazon.in/Range-ACS712-Module-Current-Sensor/dp/B0CBKFCLNY
Figure 2 Courtesy: http://robu.in/product/ac-voltage-sensor-module-zmpt101b-single-phase/

Figure 3: Single Channel Relay Module

Figure 4: Four Channel Relay Module

4. Communication

The architecture uses a dual-protocol approach:

- **ESP-NOW** for local, low-latency, peer-to-peer communication between End Nodes and the Edge Node. This is used for transmitting sensor data and receiving control commands.
- **MQTT** for server-side messaging. The Edge Node publishes data (*power/total*, *power/node*) and listens to relay commands (*relay/manual*, *relay/auto*) via a local MQTT broker.
- **HTTP** for server to Firebase the data will be sent to the database.

5. User Interface (UI)

A PyQt-based dashboard acts as the control center for the system. It provides:

- Real-time power visualization (total + per-node, with forecast overlay)
- Relay control buttons (ON/OFF per node)
- Essential / Non-Essential tagging for each node
- Threshold slider for dynamic control
- Visual alerts and logs for power limit violations and system events
- LSTM forecast plotting for predictive insight

All actions performed via the UI are reflected in real-time across the system, with MQTT ensuring that the server, edge, and end devices stay synchronized.

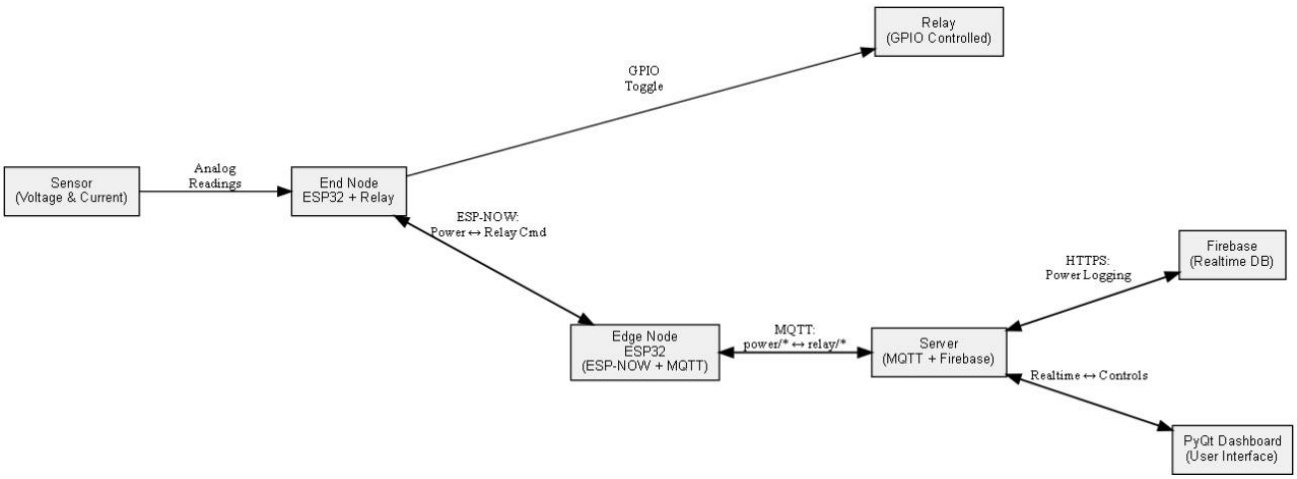

Figure 5: System Block Diagram

Figure 3 Courtesy: https://probots.co.in/1-channel-relay-5v.html
Figure 4 Courtesy: https://www.amazon.in/Channel-5V-Relay-Module-Optocoupler/dp/B08R23W6N1

METHODOLOGY

Communication & Data Flow Overview

The system adopts a hybrid communication model tailored for control and sensing in real-time. ESP32 End Nodes communicate with an Edge Node with ESP-NOW, a low-latency, connectionless protocol suitable for low-power smart grid deployments. The Edge Node, in turn, communicates to the Server with MQTT, utilizing a local Mosquitto broker over Wi-Fi. Server-side data logging and forecasting is done via Firebase REST API. Users access the system through a PyQt GUI for real-time monitoring, relay control, and threshold setting. Communication channels are bidirectional, which ensures in-time data dissemination and command processing.

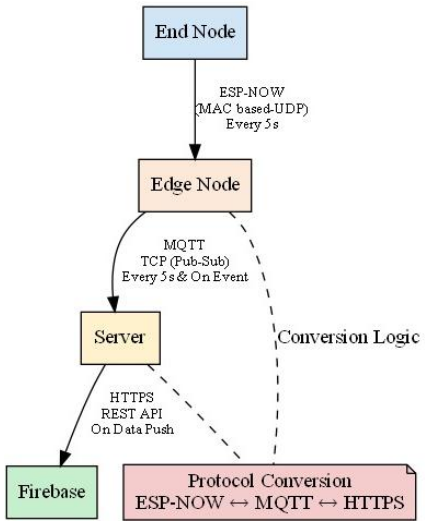

Figure 6: Communication in the System

ESP-NOW Communication (End Nodes ↔ Edge Node)

ESP-NOW offers a direct, low-latency peer-to-peer communication on MAC addressing. Every data transfer from End Node to Edge Node features a small 13-byte payload encapsulated in a packed C structure. The payload holds the node's individual ID, voltage, current, and calculated power. The structure evades compiler padding using the __attribute__(packed)) directive, squeezing in maximum efficiency under ESP-NOW's 250-byte threshold. ASCII-based relay control commands ("ON"/"OFF") are also sent to end nodes from the Edge Node, allowing real-time actuation.

Payload Structure:

```
typedef struct __attribute__((packed)) {
uint8_t senderID;    // 1 byte
float voltage;       // 4 bytes
float current;       // 4 bytes
float power;         // 4 bytes
} PowerData;
```

MQTT Protocol (Edge Node ↔ Server)

The Edge Node acts as an MQTT client, publishing aggregated and per-node power data while subscribing to control instructions. Specifically, total power is sent on the *power/total* topic every 5 seconds, while individual node data is transmitted via *power/nodes*. Whereas the Edge Node becomes a subscriber for Relay commands arrive through *relay/manual* and *relay/auto*, while *status/relay* and *relay/manual/ack* serve for acknowledgment and status verification.

Firebase Integration (Server ↔ Cloud)

The server sends power data to Firebase in the form of HTTP POST requests, in JSON format. Every entry contains the power value and a timestamp (in IST). This data is utilized for real-time visualization as well as input for the LSTM-based forecasting module.

For example:

```
{
  "value": 345.3,
  "timestamp": "2025-04-07 10:00:00"
}
```

End Node Functionality

Each End Node integrates voltage and current sensors (ZMPT101B and ACS712) connected to GPIOs 34 and 35, respectively. Power is calculated locally after applying a filter to remove minor current fluctuations (<5mA). The computed power data is periodically sent to the Edge Node via ESP-NOW. In parallel, the node listens for command packets and switches the relay (connected via GPIO 5) ON or OFF based on the received instructions.

Edge Node Functionality

The Edge Node aggregates incoming power data from up to four End Nodes. It computes total power consumption and publishes it to the *power/total* MQTT topic. Node-specific readings are published to *power/nodes*. Upon receiving relay commands via MQTT, it translates them to ESP-NOW commands for respective nodes. Internal arrays like relayState[] and manualOverride[] manage node-specific states and overrides, with utility functions like getNodeIndex() used for mapping logic.

Server Functionality (MQTT Listener & Firebase Logger)

The Server subscribes to power/total and power/nodes, logging the data to Firebase under */realtime/power_total* and appending node-wise data to node_data.csv. Each CSV entry includes a timestamp, node ID, power reading, and its classification (essential or non-essential). Timestamps are standardized to IST. The server runs a persistent MQTT client, ensuring reliable data capture and providing input to the forecasting system.

Tools and Systems

The system takes advantage of a well-balanced blend of low-cost hardware and high-performance software tools to attain real-time monitoring, control, and prediction.

Hardware support is provided through several ESP32 microcontrollers that are used in the form of 4 End Nodes and one central ESP32S2 Edge Node. One ZMPT101B sensor for sensing AC voltage and ACS712 sensor for current measurement is present in every End Node. Relay modules (Single channel, 4 channels) are directly actuated control signal-wise that can switch loaded devices smoothly.

On the software front, the PyQt5 library is employed to create a user-friendly and responsive desktop GUI for real-time visualization, relay control, and threshold setting. Data is pushed to Firebase, which serves as the cloud backend for logging total power and storing forecast outputs. For load prediction, a local LSTM model implemented using TensorFlow/Keras operates on recent power data to make near-future consumption trend estimates. Mosquitto MQTT is used as the local message broker for efficient communication between the Server and the Edge Node.

The system employs a two-protocol communication scheme: ESP-NOW, a light MAC-based, peer-to-peer protocol for low-latency data transfer between End Nodes and the Edge Node, and MQTT, a PUB/SUB protocol that is used between the Edge Node and the Server. The layered communication approach provides minimal data capture and control signal delivery delay and keeps the system modularity and scalability intact across all layers of the system.

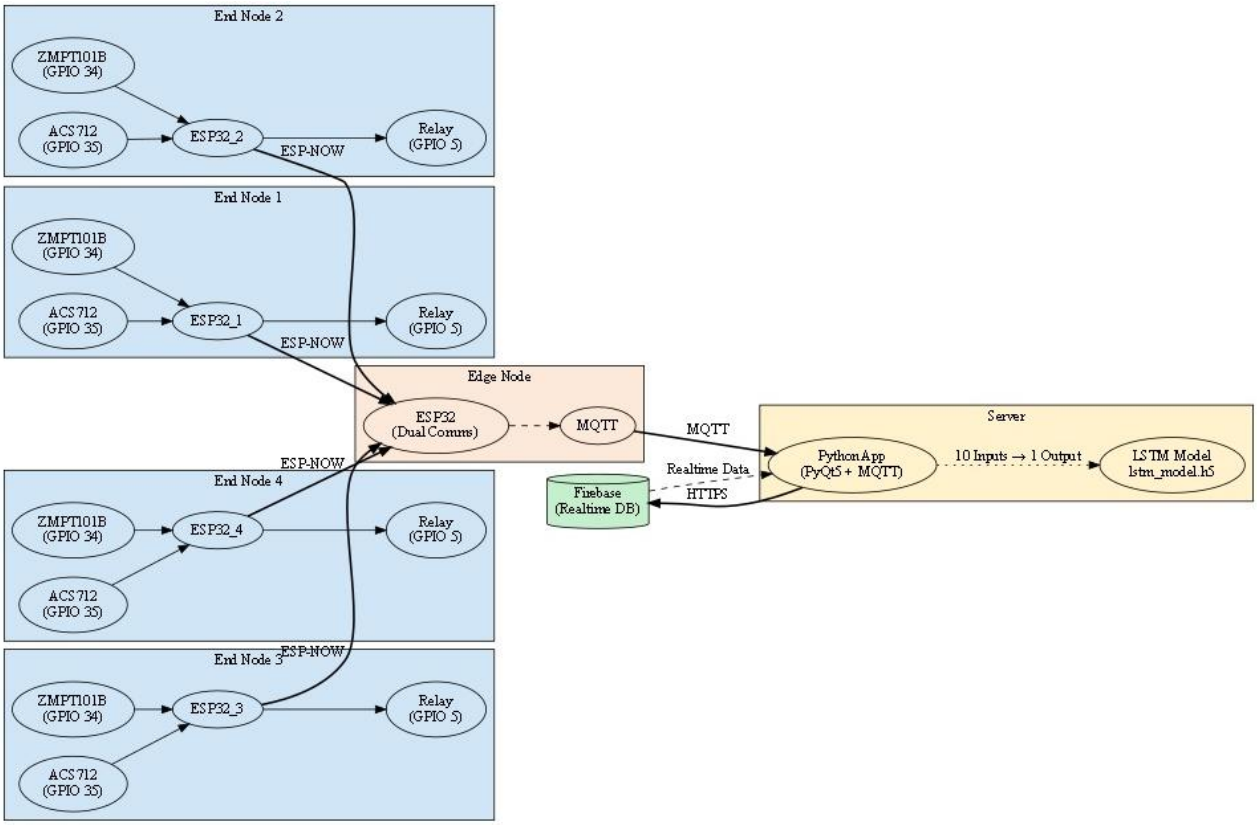

Figure 7: System Components and Connections

Nodes

End Node — *"Sense, Compute, Act"*

Every End Node has a built-in ESP32 microcontroller and has the task of sensing and controlling a single electrical load. These nodes are self-sustaining for sensing and actuation purposes but have a bidirectional connection with the Edge Node through ESP-NOW. Hardware consists of a ZMPT101B sensor for AC voltage sensing, an ACS712 sensor for measuring current, and a relay module for turning the load ON/OFF.

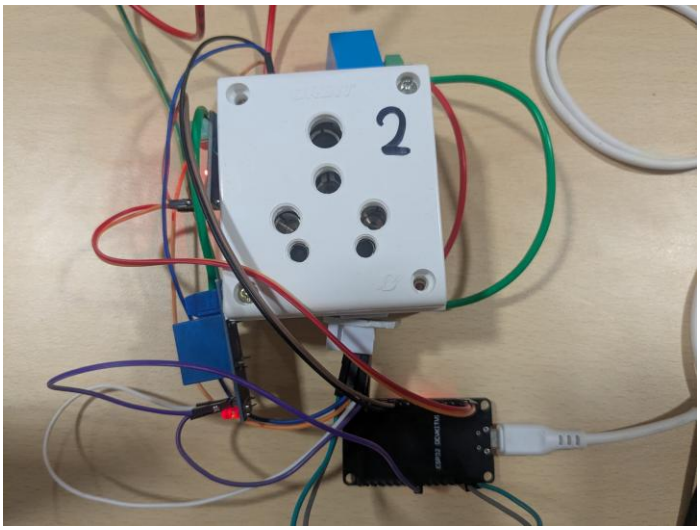

Figure 8: End Node 2 (Smart Plug)

The workflow of the node starts by measuring voltage and current using its analog inputs (GPIO 34 and 35) and then calculating power in real-time as P=V×I. To avoid minor variations, the node filters currents less than 5mA. After processing, the data is packaged into a small PowerData struct and transmitted to the Edge Node every 5 seconds via ESP-NOW. Concurrently, the node is listening for "ON"/"OFF" commands from the Edge Node based on ASCII and engages the relay tied to GPIO 5 in an appropriate manner. The system assures ultra-fast low-power communication, Wi-Fi independency, thus making the End Node smart, standalone, and scalable.

Edge Node — *"Aggregate, Decide, Dispatch"*

The Edge Node, being ESP32S2-based, acts as an on-site aggregator and protocol converter from the ESP-NOW-operated End Nodes to the MQTT-based server or dashboard. It aggregates power readings from a maximum of four End Nodes, calculates the overall power by adding up individual readings, and sends this data to an on-site MQTT broker.

It translates ESP-NOW packets into MQTT messages, publishing per-node data to topics such as *power/node* and totals to power/total. It also subscribes to *relay/manual* for manual ON/OFF control and *relay/auto* for auto triggered relay control. Relay control is done locally, through either user inputs from the PyQt dashboard or automatic threshold checks received through MQTT. The commands are then sent back to the End Nodes through ESP-NOW.

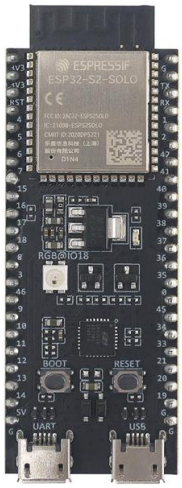

Figure 9: Edge Node ESP32S2

Internally, the Edge Node keeps arrays like relayState[] and manualOverride[] to monitor the state of every load and override status. This two-protocol support enables it to act as the bridge between low-level sensing and high-level control logic, making quick, local decisions to minimize latency and offload the server.

Server/Dashboard — *"Log, Visualize, Predict"*

The Server is a PyQt GUI program that serves as the system's control center, providing real-time monitoring, predictive analysis, and user-controlled management. It subscribes to MQTT topics such as *power/total*, *power/node*, and *status/relay*, and publishes control commands to *relay/manual* and *relay/auto*. It also logs real-time data to Firebase via REST APIs, including total power and relay states, and adds per-node readings to a CSV file for historical review.

For predictive control, the server loads a local LSTM model (lstm_model.h5) that predicts the next expected value based on the last 10 total power readings. Predicted results are superimposed on real-time power plots, giving a visual insight into future consumption patterns.

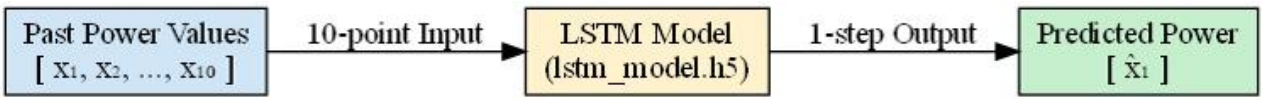

Figure 10: LSTM Model Working

The user can define a threshold that is controlled through a slider in the GUI. Once the cumulative power crosses this limit, the system will automatically cut off all non-essential loads by issuing control commands via the Edge Node. The dashboard supports real-time visualizations of per-node and total power, relay states, classification toggles (Essential/Non-Essential), and alert indicators for power overuse or forced shutdowns. Using Multithreading, The Dashboard UI and the LSTM model runs parallelly and gives the output to the use

PyQT Documentation : https://pypi.org/project/PyQt5/
ESP32S2 Datasheet: https://www.espressif.com/sites/default/files/documentation/esp32-s2_datasheet_en.pdf

Network Architecture and Protocols

Communication Overview

The proposed smart grid architecture employs a tiered hybrid communication model, wherein each layer is optimized for low latency, lightweight messaging, or secure cloud synchronization depending on the system tier. The communication flow between system components is illustrated as follows:

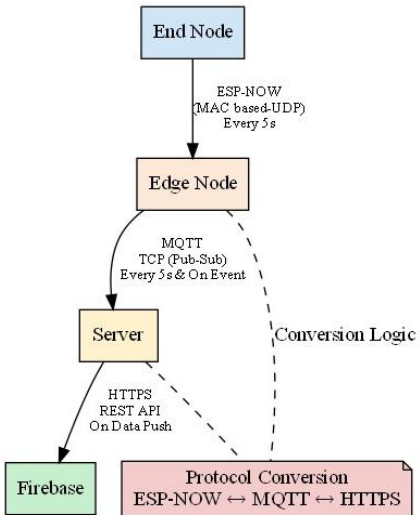

Figure 11: Network Architecture

- **Sensor to End Node:** Sensors provide analog voltage and current signals, which are digitized to digital values through on-board Analog-to-Digital Converters (ADC). There is no network communication at this point.
- **Edge Node to End Node:** Communication is done through ESP-NOW, a MAC-layer wireless protocol that supports direct device-to-device messaging without IP-based routing.
- **Edge Node to Server:** Power readings and control signals are transferred over the local network using the MQTT protocol running over TCP/IP.
- **Server to Firebase:** The server communicates with Firebase Realtime Database through HTTPS REST API, recording power data and forecasts for remote retrieval.
- **Server to Dashboard:** A local PyQt-based dashboard displays real-time data through event-driven UI updates. Although no direct networking is employed, the system depends on multithreaded socket-like behavior for asynchronous operation.

Protocol Layers and Functionality

Layer	Protocol	Transport Layer	Primary Role	Direction
End–Edge	ESP-NOW	MAC (UDP-like)	Low-latency, connectionless telemetry	Bidirectional
Edge–Server	MQTT	TCP/IP	Light weight publish-subscribe messaging	Bidirectional
Server–Cloud	HTTPS (REST API)	TLS over HTTP	Secure data logging and synchronization	Unidirectional (Server to Firebase)

Table 1 : Protocol Layers Tables

ESP-NOW: MAC-Level Wireless Communication

ESP-NOW is a proprietary communication protocol developed by Espressif, designed for rapid, connectionless, low-overhead transmission between ESP-based devices.

- Transport Layer: Operates directly over the IEEE 802.11 MAC layer, bypassing the IP stack.
- Communication Model: Supports unicast and broadcast using MAC addresses.
- Latency: Typically, <3 ms, making it highly suitable for real-time embedded communication.
- Association: Does not require traditional Wi-Fi connection or access point.
- End Nodes periodically send PowerData structs to the Edge Node.
- The Edge Node may respond with ASCII-based "ON"/ "OFF" commands for load control.

MQTT: Edge Node to Server Messaging

MQTT is a publish-subscribe messaging protocol designed for lightweight, bandwidth-efficient communication in constrained environments.

- Broker: Mosquitto, hosted locally within the LAN.
- QoS Level: QoS 0 (fire-and-forget, best-effort delivery).
- Transport Layer: TCP/IP over Wi-Fi.

Topics Utilized:

Topic	Direction	Purpose
power/total	Edge → Server	Aggregated power data
power/nodes	Edge → Server	Per-node power readings
relay/manual	Server → Edge	Manual relay control
relay/auto	Server → Edge	Threshold-based control
status/relay	Edge → Server	Relay state acknowledgements

Table 2: MQTT Topics Table

HTTPS REST API: Server to Firebase Cloud

The server synchronizes system data with Firebase Realtime Database via secure HTTPS RESTful communication.

- Transport: HTTPS (TLS 1.2+) over standard HTTP protocols.
- Authentication: Managed via Firebase Auth tokens or static API keys.
- Tools: Python requests library or firebase_admin SDK.

Endpoints Used:

- /realtime/power_total: Logs total system power at 5-second intervals.
- /forecast/power: Stores predicted power values generated by the local LSTM model.

Protocol Conversion and Bridging Logic

The system architecture necessitates seamless translation between protocols at various stages:

Source Protocol	Target Protocol	Interface Component	Bridging Logic
ESP-NOW	MQTT	Edge Node	Parses incoming PowerData, aggregates total power, publishes structured JSON
MQTT	HTTPS (REST)	Server	Logs power data to Firebase upon message arrival
MQTT	PyQt Events	Server	Emits signals to PyQt threads to update UI in real-time

Table 3: Protocol Bridging Table

MQTT Open-Source Documentation: https://mosquitto.org/
ESP-NOW Documentation: https://github.com/espressif/esp-now
Firebase Database Documentation: https://firebase.google.com/docs/database

RESULTS

Real-time Power Monitoring: The PyQt-based dashboard effectively visualizes both total and per-node power consumption. The top graph displays *Total Power (W)*, updated at 5-second intervals. A secondary graph shows the power usage of an individual node, such as *NODE2*, selectable via the interface.

Relay Control Mechanism: The user interface features manual relay control buttons labeled *Relay X ON/OFF* for each node. Additionally, checkboxes allow classification of each node as either *Essential* or *Non-Essential*. Green and red indicators dynamically reflect each relay's operational status in real time.

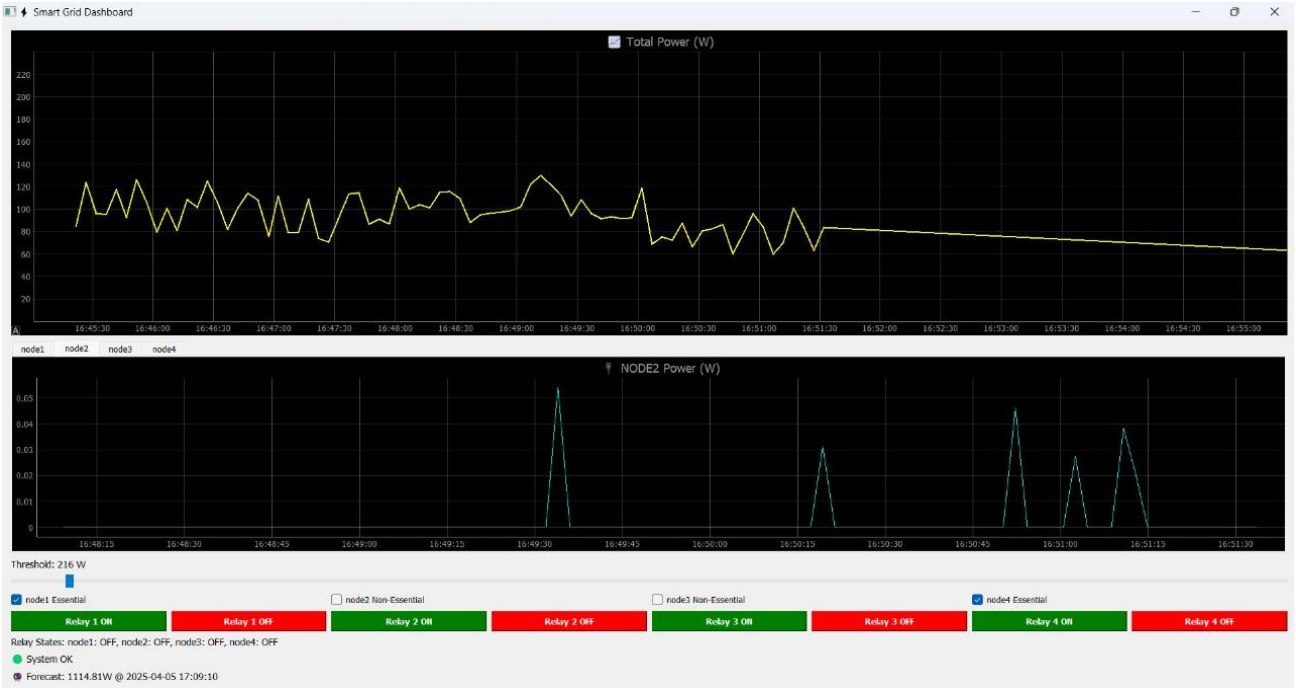

Figure 12: UI Dashboard

Threshold-Based Load Shedding: A threshold slider enables users to set the maximum allowable total power. When real-time power exceeds this threshold, the system automatically deactivates relays associated with non-essential nodes. For example, with a threshold set at *216W*, the system successfully turned off corresponding non-essential nodes.

LSTM Forecasting Integration: Forecasted power values are presented beneath the UI, e.g., *Forecast: 1114.81W @ 2025-04-05 17:09:10*. These values are generated once per minute by an LSTM model (*lstm_model.h5*) using the ten most recent power readings. Forecast data is simultaneously plotted alongside real-time power data for comparative visualization.

Firebase Logging: The system logs all total power values and relay states to Firebase via HTTPS every 5 seconds. This provides a reliable mechanism for remote monitoring and supports longitudinal analysis of power trends.

Performance Metrics:

Feature	Justification / Basis (Metrics)
End-to-End Latency	Total latency across Sensor → Edge → Server → Firebase path is <1s under normal conditions.
Forecast Update Rate	LSTM thread computes forecast every 60 seconds as per PyQt threading logic.
Power Sampling Rate	End nodes transmit power data every 5 seconds via ESP-NOW.
Forecast Accuracy	Expected to be ~90% (MAPE < 10%) with clean time-series input and proper training. But the forecast is 3 times more than the actual values.
Auto Load Shedding	Total detection and relay action time is <2 seconds including dashboard logic and MQTT messaging.
Network Resilience	ESP-NOW includes retry logic and ACK handling; resilient to mild interference.

Table 4: Performance Metrics

CONCLUSION

This project is successfully proving the design and development of an effective and intelligent Smart Grid Monitoring and Control System, especially for microcontroller-based edge computing environments. Through the combination of lightweight wireless communication protocols, real-time cloud logging, and AI-based forecasting, the system provides an integrated approach towards controlling power flow and load distribution in a dynamic and efficient way.

The end nodes are fitted with ZMPT101B and ACS712 sensors to measure voltage and current accurately. The measurements are processed through the onboard ADC of ESP32 and sent over ESP-NOW, a low-power and connectionless communication protocol that is particularly well-suited for embedded and low-latency applications.

The central part of the system is the edge node, which collects data from several end nodes via ESP-NOW. It employs threshold-based control logic to control load shedding and then publishes the sum and per-node power readings to the server through MQTT. This architecture supports rapid, localized decision-making with centralized monitoring.

The server is an MQTT client and plays several important roles. It stores incoming power data to Firebase via HTTPS REST APIs and also updates the PyQt dashboard in real time via Qt signals. In addition, an LSTM-based prediction model forecasts future power consumption patterns based on 10-point historical data, allowing the system to react to demand changes proactively. Forecasts are updated every minute, keeping the highly responsive and smart control loop.

A light protocol stack guarantees proper data flow around the system. The switch from ESP-NOW (edge-to-node communication) to MQTT (server-to-edge) and then to HTTPS and Qt signals (cloud/UI-to-server) illustrates a clean and scalable design. Each protocol was selected due to the special advantage it brings to the system to ensure the system stays lightweight and strong.

The dashboard based on PyQt5 features a rich interface with functionalities like real-time power plotting, relay control, marking of essential/non-essential load, dynamic threshold slider, and forecast overlay. There is instant visual feedback from a system status indicator—displaying "CRITICAL" when there is excess power and non-essential nodes are still running and "System OK" otherwise.

In summary, this project provides an in-depth and real-life example of how edge computing, IoT communication protocols, and AI-based prediction can be integrated into a smart grid solution that is efficient, resilient, and user-friendly.

CHALLENGES FACED

Tried, Tested, and Failed:

CoAP Observe Protocol:

Tried to implement CoAP with Observe for pushing sensor data from End Node to Edge.

- **Why did it fail?**
  - No proper libraries with CoAP, QoS or Observe support for ESP32
  - Handling subscriptions and acknowledgments became unreliable
  - Sparse documentation and limited community support

WebSocket-Based End-to-Edge Communication:

Tested edge updates using WebSockets directly from End Node.

- **Why did it fail?**
  - ESP32 couldn't maintain long-running WebSocket connections
  - Browser CORS policies and lack of secure WebSocket (wss) support
  - Heavy code overhead for minimal performance gain

Relay Hardware Compatibility:

Initially used low-level 1-channel relays with ESP32.

- **Why did it fail?**
  - Trigger logic mismatch (active-low vs active-high)
  - Inconsistent switching behavior
  - Couldn't handle rapid relay toggles due to debounce/glitching issues

Web Development Stack Mismatch:

Tried integrating web dashboard using Flask + JavaScript + Firebase.

- **Why did it fail?**
  - Complex syncing between real-time DB, WebSocket, and manual relay toggles
  - Final stack got bloated and fragile
  - PyQt offered faster development for UI and tighter control

Other Key Roadblocks:

- **ADC Calibration Pains:**
  ZMPT101B and ACS712 required noise filtering, gain correction, and offset tuning before values became useful.
- **ESP-NOW Packet Reliability:**
  No default ACK — had to implement custom retry logic on Edge Node for power data reliability.
- **Protocol Conversion Complexity:**
  ESP-NOW sends raw structs → MQTT works on JSON strings → Firebase expects REST-formatted dictionaries. Parsing and converting between these formats added processing complexity.
- **Local Time Management:**
  MQTT payloads lack timestamps. Server injects IST-based local timestamps for accurate logging and plotting.
- **Firebase API Rate Limits:**
  Hit write quotas during load testing. Solution: batched writes, conditional updates, and smarter logging intervals.
- **Threaded Forecasting in PyQt:**
  LSTM prediction caused UI freezes before being offloaded to a QThread for async processing.
- **Relay Control Arbitration:**
  Juggling manual, threshold-triggered, and forecast-driven relay commands needed a priority-based control system to avoid conflicts.
- **Debugging ESP-NOW:**
  No serial output during Wi-Fi transmission = debugging blind. Workaround: added LEDs, local logs, and timeouts to catch failures.

REFERENCES

[1] J. K. Mishra, S. Goyal, V. A. Tikkalwal, A. Kumar, "An IoT-Based Smart Energy Management System," in *Proc. 4th Int. Conf. Comput. Commun. Autom. (ICCCA*), Greater Noida, India, 2018, pp. 1–3.

[2] M. Abdel-Basset, H. Hawash, R. K. Chakrabortty, M. Ryan, "Energy-Net: A Deep Learning Approach for Smart Energy Management in IoT-Based Smart Cities," *IEEE Internet Things J.*, vol. 8, no. 15, pp. 12422–12435, Aug. 1, 2021.

[3] N. Khan, S. U. Khan, F. U. M. Ullah, M. Y. Lee, S. W. Baik, "AI-Assisted Hybrid Approach for Energy Management in IoT-Based Smart Microgrid," *IEEE Internet Things J.*, vol. 10, no. 21, pp. 18661–18675, Nov. 1, 2023.

[4] W. Li, T. Logenthiran, V.-T. Phan, W. L. Woo, "A Novel Smart Energy Theft System (SETS) for IoT-Based Smart Home," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 5531–5539, Jun. 2019.

[5] J. A. Dahlan, D. Ariatje, F. Hamami, Heryanto, "The Implementation of Building Intelligent Smart Energy Using LSTM Neural Network," in *Proc. Int. Conf. Artif. Intell. Mechatronics Syst. (AIMS)*, Bandung, Indonesia, 2021, pp. 1–5.

[6] Y.-J. Lin, H.-S. Tang, T.-Y. Shen, C.-H. Hsia, "A Smart Home Energy Management System Utilizing Neurocomputing-Based Time-Series Load Modeling and Forecasting Facilitated by Energy Decomposition for Smart Home Automation," *IEEE Access*, vol. 10, pp. 116747–116765, 2022.

[7] S. Goudarzi, M. H. Anisi, S. A. Soleymani, M. Ayob, S. Zeadally, "An IoT-Based Prediction Technique for Efficient Energy Consumption in Buildings," *IEEE Trans. Green Commun. Netw.*, vol. 5, no. 4, pp. 2076–2085, Dec. 2021.

[8] M. Jaysankhara, P. Shah, A. Sharma, P. Chand, K. S. Singh, "A Novel Approach for Short-Term Energy Forecasting in Smart Buildings," *IEEE Sens. J.*, vol. 23, no. 5, pp. 5307–5314, Mar. 1, 2023.

[9] U. Ali, M. U. Ramzan, M. Ali, M. E. Rana, A. Qayyum, "IoT-Driven Smart Energy Monitoring: Real-Time Insights and AI-Based Trend Predictions," in *Proc. IEEE Int. Students Conf. Res. Develop. (SCOReD)*, Kuala Lumpur, Malaysia, 2023, pp. 672–677.

[10] S. Xue, H. Huang, J. Liu, Q. Yang, L. Zhao, H. Wu, "An Effective Scheme to Solve Critical Data Missing Problems for IoT-Based Smart Energy Management," *IEEE Internet Things J.*, vol. 12, no. 4, pp. 4466–4474, Feb. 15, 2025.