International Journal of Scientific Research in Engineering and Management (IJSREM)
Volume: 10 Issue: 11 | Nov - 2026 SJIF Rating: 8.659 ISSN: 2582-3930

IOT-Based EV Battery Monitoring System with Cloud AI Health Prediction

Bhanuprakash K. R.[1], Chandan S. G.[1], Darshana K. A.[1], Mrs. Amrutavarshini S Hadimani[1]
[1]Department of Electronics and Communication Engineering Kalpataru Institute of Technology, Tiptur, Karnataka, India

***Abstract*—Electric vehicle batteries require continuous monitoring to support safe operation, reliable performance, and timely maintenance. This paper presents an Internet of Things (IoT)-based electric vehicle battery monitoring prototype implemented using an ESP32 microcontroller. The system monitors battery voltage, current, temperature, and flame-sensor information. A local TFT touchscreen displays system information, while an embedded HTTP server provides battery information through a local interface. The system also supports cloud data transmission through the ThingSpeak platform. A React-based web dashboard retrieves cloud data and presents selected battery parameters.**

Battery percentage, health indicators, maintenance status, power consumption, and nominal range are calculated using the implemented firmware logic. The health indicator is based on programmed threshold rules rather than a trained machine-learning model. Similarly, the range value is a nominal estimate derived from battery percentage and is not a measurement of actual vehicle range.

To extend the monitoring system towards intelligent battery analysis, a cloud-based AI health prediction framework is proposed for analysing historical battery measurements. The proposed framework can support future battery state-of-health estimation when suitable labelled battery ageing data, model training, and experimental validation are available. This paper describes the system architecture, hardware implementation, software design, monitoring workflow, proposed AI methodology, and functional evaluation of the prototype.

***Index Terms*—Electric vehicle, battery monitoring, ESP32, Internet of Things, ThingSpeak, cloud computing, battery health prediction, machine learning, flame detection, web dashboard.**

I. INTRODUCTION

A. Background

Electric vehicles (EVs) are increasingly important in modern transportation because they use electrical energy for propulsion and help reduce dependence on conventional fuels. The battery is a major component of an electric vehicle, and its operating condition influences vehicle reliability, energy availability, driving range, and safety. Monitoring parameters such as voltage, current, and temperature helps users understand battery operating conditions and identify abnormal readings.

Recent developments in the Internet of Things (IoT), embedded systems, cloud computing, and artificial intelligence have enabled battery measurements to be collected, stored, and analysed remotely. An IoT-based monitoring system can collect measurements through sensors, process them using an embedded controller, and make selected information available through local and remote interfaces. Cloud platforms provide facilities for storing historical battery data, which can be analysed using machine-learning techniques to support battery health prediction.

In this project, an ESP32 microcontroller provides battery monitoring through a TFT touchscreen, an embedded web interface, and the ThingSpeak cloud platform. The system collects important battery parameters and makes them available for remote visualization through a web-based dashboard. A cloud-based AI health prediction framework is also proposed to analyse historical battery measurements and support future battery health estimation. The implemented health indicator is currently rule-based; actual AI prediction requires a trained and experimentally validated machine-learning model.

B. Problem Statement

Battery parameters are not always easily accessible to users through a simple monitoring interface. Without continuous visibility of important measurements, it can be difficult to observe changes in battery operating conditions or receive timely indications of potentially abnormal situations.

The problem addressed in this project is the development of a low-cost monitoring prototype that collects battery voltage, current, temperature, and flame-sensor information and presents the resulting data through local and remote interfaces. The system also provides calculated battery indicators and status alerts using the implemented firmware logic.

In addition, there is a need to investigate how historical battery measurements can be analysed using cloud-based machine-learning techniques to support battery health prediction. Such prediction requires suitable battery ageing data, appropriate health labels, model training, and experimental validation.

C. Motivation

The motivation for this work is to make battery information easier to access and understand. A combination of embedded sensing, wireless communication, local display, and web-based visualization can provide users with a convenient way to observe battery operating conditions.

The integration of safety-related sensing and programmed status indicators also provides a foundation for further investigation into battery monitoring and maintenance. Cloud connectivity provides an opportunity to store historical measurements and explore data-driven battery health analysis.

The proposed cloud AI framework extends the monitoring concept towards intelligent battery analysis. It provides a

direction for future development in which historical voltage, current, temperature, and other relevant battery information can be used to investigate battery health estimation.

D. Objectives

The main objectives of the proposed system are:

- To measure battery voltage and current using suitable sensor modules.
- To monitor battery temperature using a DS18B20 temperature sensor.
- To incorporate flame-sensor input for safety-related monitoring.
- To process sensor measurements using an ESP32 microcontroller.
- To display battery measurements and system status on a TFT touchscreen.
- To provide battery information through an embedded HTTP interface.
- To support cloud data transmission using ThingSpeak.
- To visualize selected battery parameters through a React-based dashboard.
- To calculate battery indicators and nominal range estimates using the implemented firmware logic.
- To propose a cloud-based machine-learning framework for analysing historical battery measurements and supporting future battery health prediction.

E. Contributions of this Research

The main contributions of this work are summarized as follows:

- **Integrated sensing:** The prototype combines voltage, current, temperature, and flame-sensor inputs in an ESP32-based system.
- **Local monitoring:** A TFT touchscreen presents battery measurements and system status directly on the device.
- **Embedded web interface:** The ESP32 provides selected battery information through a local HTTP interface.
- **Cloud connectivity:** ThingSpeak is used to transmit selected data for remote monitoring.
- **Web-based visualization:** A React dashboard retrieves cloud measurements and presents them through graphical components.
- **Calculated indicators:** The firmware derives battery percentage, health-related indicators, maintenance information, and nominal range estimates using programmed logic.
- **Cloud AI framework:** A proposed cloud-based machine-learning methodology is described for analysing historical battery measurements and supporting future battery health prediction.

The calculated indicators are prototype estimates and have not been established as validated predictions of battery degradation or actual vehicle range. The proposed AI framework requires suitable training data, model development, and experimental validation before its prediction performance can be established.

II. Literature Review

The development of the Internet of Things (IoT), embedded systems, and cloud computing has enabled the development of battery monitoring systems that collect sensor measurements and provide remote access to battery information. Various studies have explored online battery monitoring, battery management, state-of-charge estimation, and battery health prediction. These approaches aim to improve battery visibility, support condition assessment, and provide useful information for maintenance. However, integrating sensor acquisition, embedded processing, cloud communication, safety monitoring, and accessible visualization remains an important consideration when designing a practical electric vehicle battery monitoring system.

Research has reviewed machine-learning techniques for electric vehicle battery capacity degradation and health estimation. Such techniques address the importance of estimating battery condition as batteries age and their capacity changes. Machine-learning approaches can support health estimation when suitable training data and validation methods are available. However, a monitoring prototype that calculates health indicators using programmed rules should distinguish these indicators from predictions produced by a validated machine-learning model.

Other research has reviewed machine-learning, deep-learning, and digital-twin approaches for battery health prediction in electric vehicles. These approaches demonstrate the broader research interest in using data-driven techniques to estimate battery condition. Their practical effectiveness depends on the quality of available data, the selected prediction method, and validation under relevant operating conditions. For an embedded monitoring system, computational requirements and the availability of reliable historical measurements must also be considered.

Research on online battery monitoring and management systems based on IoT technologies highlights the importance of collecting battery information and supporting remote observation. IoT-based monitoring provides a foundation for making measurements accessible beyond a local display. In a practical implementation, sensor calibration, communication reliability, and the presentation of measurements to users remain important considerations.

Studies have also investigated IoT- and cloud-integrated deep-learning approaches for state-of-charge estimation in battery-powered electric vehicles. Cloud integration can support remote access to measurements, while estimation methods can provide information beyond raw sensor readings. The accuracy of a state-of-charge estimate, however, depends on the method and the experimental conditions under which it is evaluated.

Further research has examined battery management using IoT and machine learning to predict the remaining useful life of lithium-ion batteries. Remaining useful life estimation is an important research direction because it aims to provide information about future battery degradation. Such prediction

requires appropriate data and a validated estimation method. A rule-based remaining-life indicator, by comparison, should be treated as an indicative output until its relationship with actual battery ageing has been experimentally established.

The reviewed research highlights several important aspects of EV battery monitoring, including measurement of electrical and thermal parameters, remote data access, battery condition estimation, and support for maintenance decisions. Cloud platforms can help make measurements accessible, while local displays provide immediate access to system status. Safety-related sensing can complement these functions, although its effectiveness depends on sensor characteristics, placement, and the response logic implemented in the controller.

The proposed IoT-Based EV Battery Monitoring System integrates an ESP32 microcontroller with voltage and current sensing, a DS18B20 temperature sensor, a flame sensor, and a TFT touchscreen. An embedded HTTP interface provides local access to selected battery information, while ThingSpeak supports cloud data transmission. A React-based dashboard retrieves the cloud measurements for remote visualization. The firmware calculates battery indicators and a nominal range estimate using programmed logic. A cloud AI health prediction methodology is also proposed as a future extension of the monitoring architecture.

A. Comparative Analysis of Existing Studies

The studies reviewed above address different aspects of battery monitoring and condition estimation. IoT-based monitoring focuses on collecting and communicating battery measurements, while machine-learning research investigates battery health, state of charge, degradation, and remaining useful life. These approaches are related but serve different purposes.

The proposed prototype emphasizes the integration of sensing, local display, embedded web access, and cloud connectivity. Unlike a study focused primarily on a data-driven prediction method, this implementation combines hardware measurements with local and remote interfaces. The proposed AI framework extends this architecture towards historical data analysis and battery health prediction.

A quantitative comparison of accuracy or performance would require consistent experimental conditions and verified results from the respective studies.

B. Research Gap

Based on the reviewed studies and the requirements of the proposed prototype, the following design considerations are identified:

- Integrating battery voltage, current, temperature, and flame-sensor information into a single embedded monitoring system.
- Providing local visualization through a TFT touchscreen and remote access through a web dashboard.
- Supporting cloud data transmission to enable remote observation of selected battery parameters.
- Clearly distinguishing programmed battery indicators from validated state-of-health, state-of-charge, and remaining-useful-life predictions.
- Investigating cloud-based machine-learning methods for analysing historical battery measurements.
- Improving the reliability of battery percentage and range estimates through calibration and experimental validation.
- Evaluating sensor accuracy, communication reliability, and safety-alert behaviour under controlled test conditions.

The proposed prototype addresses the integration of sensing, embedded processing, local visualization, web monitoring, and cloud data transmission. Further work is required to validate the calculated battery indicators and to develop and evaluate data-driven prediction methods using suitable battery datasets.

III. Proposed System and Research Methodology

A. Overview of the Proposed System

The proposed IoT-Based EV Battery Monitoring System integrates embedded sensing, local visualization, wireless communication, and cloud connectivity to monitor important battery operating parameters. The prototype uses an ESP32 microcontroller to collect and process measurements from voltage, current, temperature, and flame sensors. The processed information is presented on a local TFT touchscreen and made available through an embedded HTTP interface and the ThingSpeak cloud platform.

A React-based web dashboard retrieves the latest cloud measurements and displays selected battery parameters. The system is designed to provide users with convenient access to battery measurements and safety-related status information. The firmware calculates battery percentage and other indicators using its implemented logic. It also provides a nominal range estimate based on battery percentage. These calculated values are intended for prototype monitoring and should not be interpreted as experimentally validated battery degradation predictions or actual vehicle range.

A cloud AI health prediction framework is proposed to extend the monitoring architecture. The framework describes how historical battery measurements can be collected, processed, and analysed using a machine-learning model after suitable training data and reference health labels become available.

The overall system consists of five functional components: the sensing layer, embedded processing layer, local display and safety layer, web monitoring layer, and cloud communication layer. These components work together to collect measurements, process sensor data, display information, and support remote observation.

B. System Architecture

The proposed system follows a modular architecture in which sensors provide measurements to the ESP32 microcontroller. The microcontroller processes these measurements and updates the local TFT touchscreen. It also provides selected battery information through a local HTTP interface.

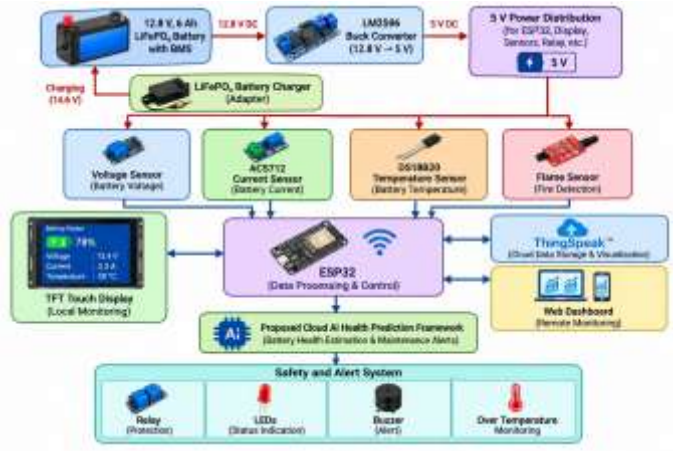


Fig. 1. Overall system architecture of the proposed EV battery monitoring system.

Separately, the ESP32 transmits selected measurements to ThingSpeak. The React dashboard retrieves the latest uploaded cloud data for remote visualization.

The cloud AI framework is designed as an additional data-analysis stage. Historical measurements can be used as input features for a trained prediction model when suitable reference battery health labels are available.

Fig. 1 illustrates the overall architecture of the proposed EV battery monitoring prototype.

1) Sensor Layer: The sensing layer collects the parameters required for battery monitoring. A voltage sensor module measures the battery voltage, while the ACS712 current sensor measures current. A DS18B20 sensor monitors temperature, and a flame sensor provides an input for fire-related safety monitoring. The measurements are connected to the ESP32 for further processing. Measurement accuracy depends on sensor calibration, electrical connections, and operating conditions.

2) Embedded Processing Layer: The ESP32 acts as the central controller. It reads sensor measurements, performs the programmed calculations, updates system status, and coordinates the local display and communication functions. The firmware averages multiple analog readings for voltage and current measurements. It calculates battery percentage and related indicators using its implemented logic. It also evaluates the flame-sensor input and updates the configured safety outputs.

3) Local Display and Safety Layer: The ILI9341 TFT touchscreen presents battery measurements and system information locally. The display provides an interface for viewing the operating status without requiring access to the web dashboard. The firmware also uses a buzzer and indicator LEDs for safety-related alerts. Relay operation is controlled by the programmed firmware logic. The actual effectiveness of the alerts and relay response requires testing under controlled conditions.

4) Web Monitoring Layer: The ESP32 hosts an embedded HTTP server that provides a local web page and selected battery information in JSON format through the `/api/battery` endpoint. This interface supports local access to the device data.

The separate React-based dashboard retrieves the latest feed from the ThingSpeak cloud service. It presents selected battery parameters using dashboard components and graphical representations. Thus, the local ESP32 HTTP interface and the React dashboard's cloud-data connection are separate communication paths in the implemented prototype.

5) Cloud Communication Layer: The prototype uses ThingSpeak to transmit selected measurements to a cloud service. The firmware uploads voltage, current, temperature, battery percentage, local health indicator, estimated remaining life, flame status, and load status through configured channel fields.

Cloud transmission supports remote observation of the uploaded data. Data availability depends on network access and the configured cloud communication parameters.

C. Proposed Cloud AI Health Prediction Framework

The proposed cloud AI framework is intended to analyse historical battery measurements and investigate battery health prediction. It extends the existing IoT monitoring architecture by introducing a data-analysis stage between cloud data collection and the presentation of prediction results.

The framework is proposed as a future development of the prototype. A trained and experimentally validated AI model has not yet been integrated into the current firmware or dashboard.

1) Data Collection: The ESP32 collects battery voltage, current, temperature, battery percentage, and other available monitoring information. Selected measurements are uploaded to the ThingSpeak cloud platform.

Historical measurements can be collected from the cloud platform and organized into a dataset for further analysis. The quality and duration of the collected data are important for developing a reliable prediction model.

2) Data Preprocessing: Before machine-learning analysis, the collected data should be checked for missing values, invalid measurements, duplicate records, and inconsistent sensor readings.

The preprocessing stage can include:

- Removing duplicate records.
- Handling missing sensor measurements.
- Identifying invalid or abnormal readings.
- Converting measurements into suitable numerical formats.
- Organizing historical measurements into training and testing datasets.

This stage helps prepare the collected information for machine-learning analysis.

3) Feature Selection: The proposed model can investigate relevant battery parameters as input features. These may include:

- Battery voltage.
- Battery current.

- Battery temperature.
- Battery percentage or estimated state of charge.
- Operating time and charge-discharge history.
- Historical battery operating conditions.

The selection of input features depends on the availability and reliability of the collected data.

For genuine state-of-health prediction, suitable reference labels are required. One possible reference is measured battery capacity:

$$SOH(\%) = \frac{C_{measured}}{C_{rated}} \times 100 \quad (1)$$

where $C_{measured}$ is the measured battery capacity and C_{rated} is the rated battery capacity.

A rule-based health indicator cannot replace measured capacity as a reference label for validating genuine battery state of health.

4) Proposed Machine-Learning Model: A regression-based machine-learning model, such as Random Forest Regression, can be investigated for battery health estimation. The model can learn relationships between battery operating parameters and reference state-of-health values when a suitable labelled dataset is available.

The general prediction process can be represented as

$$\acute{SOH} = f_{\vartheta}(V, I, T, H_d) \quad (2)$$

where:

- $\acute{SOH}$ represents the predicted battery state of health.
- V represents battery voltage.
- I represents battery current.
- T represents battery temperature.
- H_d represents relevant historical battery data.
- f_{ϑ} represents the trained prediction function.

The model parameters are determined during training using appropriate reference data. The selected model must be evaluated using data that was not used during training.

5) Model Training and Validation: The collected and labelled dataset can be divided into training and testing subsets. The training data is used to develop the prediction model, while the testing data is used to evaluate its performance.

Possible evaluation metrics include mean absolute error (MAE), root mean square error (RMSE), and coefficient of determination (R^2).

Mean absolute error is expressed as

$$MAE = \frac{1}{n}\sum_{i=1}^{n} |y_i - \hat{y}_i| \quad (3)$$

Root mean square error is expressed as

$$RMSE = \sqrt{\frac{1}{n}\sum^{n} (y_i - \hat{y}_i)^2} \quad (4)$$

where y_i represents the reference value, $\hat{y}_i$ represents the predicted value, and n represents the number of test samples.

These metrics can be calculated after model training and testing. No AI accuracy or prediction-performance result is claimed for the current prototype.

6) Cloud-Based Prediction and Visualization: After a suitable model has been trained and validated, it can be deployed in a cloud-based processing environment. Historical or newly uploaded battery measurements can then be supplied to the model to obtain estimated battery health information.

The resulting prediction can be made available to a web dashboard for visualization. The dashboard can present the measured battery parameters together with the model-generated health estimate and related status information.

This cloud-based prediction stage is a proposed extension and requires model development, integration, and testing before it can be considered an implemented feature.

D. System Design

The proposed EV battery monitoring system follows a modular architecture that integrates sensor hardware, embedded processing, local display, safety-related outputs, and cloud-based monitoring. The ESP32 microcontroller acts as the central processing unit and coordinates the different components.

The system consists of the following functional modules:

- Battery sensing module.
- Embedded processing and control module.
- Local TFT display module.
- Safety monitoring module.
- Embedded HTTP communication module.
- Cloud communication and web dashboard module.
- Proposed cloud AI analysis module.

The hardware and software components communicate through sensor interfaces, GPIO connections, SPI communication, Wi-Fi, HTTP, and cloud services. This modular design separates measurement acquisition, processing, visualization, and remote monitoring.

E. Hardware Architecture

The hardware subsystem provides the embedded interface for measuring battery parameters and presenting the resulting information. It integrates the ESP32 controller, voltage and current sensors, temperature probe, flame sensor, TFT touchscreen, relay module, buzzer, and indicator LEDs.

Fig. 2 shows the physical hardware implementation of the proposed EV battery monitoring system.

1) ESP32 Microcontroller: The ESP32 serves as the main embedded controller. It is responsible for acquiring sensor measurements, processing the input values, updating the display, controlling the configured output devices, and communicating with the cloud platform through Wi-Fi.

Its main responsibilities include:

- Reading voltage and current sensor signals.
- Acquiring temperature and flame-sensor information.
- Calculating battery percentage and health-related indicators.
- Updating the TFT touchscreen with the latest information.

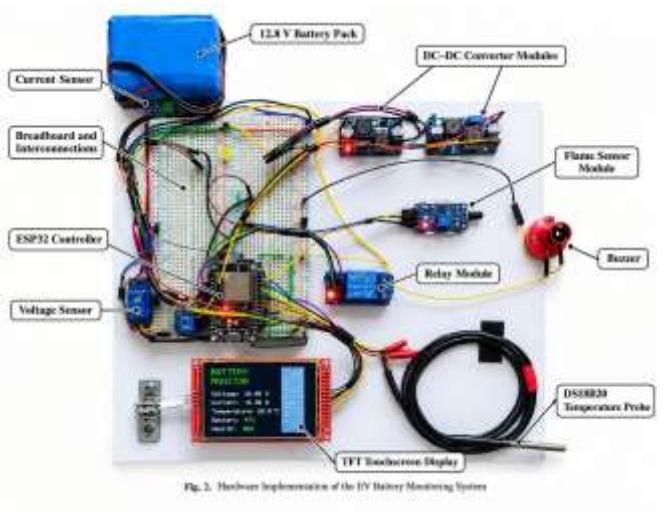


Fig. 2. Hardware implementation of the proposed EV battery monitoring system.

- Managing buzzer, LED, and relay output signals.
- Providing battery information through the local HTTP interface.
- Transmitting selected measurements to ThingSpeak.

2) Voltage Sensor: The voltage sensor module is used to measure the battery voltage. Its output is connected to an analog input of the ESP32. The controller reads the analog signal and converts it into a voltage value using the configured conversion and calibration parameters. The measured voltage is used in battery-percentage calculation and monitoring.

3) ACS712 Current Sensor: The ACS712 current sensor measures the current flowing through the monitored battery circuit. Its analog output is connected to an analog input of the ESP32. The firmware determines the current using the sensor output and a zero-current offset calibrated during initialization.

The current measurement helps identify charging or discharging behaviour and is also used in the calculation of instantaneous power. The current direction depends on the sensor wiring and calibration convention.

4) DS18B20 Temperature Sensor: The DS18B20 digital temperature sensor is used to monitor battery temperature. It communicates with the ESP32 through the One-Wire interface. The measured temperature is used for displaying thermal information and evaluating the programmed temperature-related safety conditions.

5) Flame Sensor Module: The flame sensor provides a digital input for fire-related monitoring. It is connected to a digital input of the ESP32 and is configured according to the firmware input logic. When the configured detection condition is identified, the controller updates the fire status and applies the programmed safety response.

6) TFT Touchscreen Display: The ILI9341 TFT touchscreen provides the local visual interface of the system. It displays important battery measurements, including voltage, current, temperature, battery percentage, and health-related information. The display communicates with the ESP32 through the SPI interface. An XPT2046 touch controller is used for touchscreen input.

TABLE I
HARDWARE INTERFACE CONFIGURATION

Component	Interface
Voltage sensor	Analog input
ACS712 current sensor	Analog input
DS18B20	One-Wire
Flame sensor	Digital input
Relay module	Digital output
Buzzer	Digital output
Red LED	Digital output
Yellow LED	Digital output
Green LED	Digital output
ILI9341 TFT	SPI
XPT2046 touch controller	SPI

7) Relay, Buzzer, and Indicator LEDs: The prototype includes a relay module, a buzzer, and three indicator LEDs. These components provide programmed visual, audible, and switching outputs.

The buzzer and LEDs provide local indications of selected system conditions. The relay is controlled by the firmware logic. The actual operating behaviour of the relay must be verified against the final firmware and physical test results before claiming that the system automatically disconnects a battery load during every fault condition.

F. Hardware Pin Configuration

The principal hardware interfaces used in the prototype are summarized in Table I. The final GPIO assignments must correspond to the actual wiring and the final ESP32 firmware used during testing.

G. Software Architecture

The software architecture consists of embedded firmware, a local HTTP interface, the ThingSpeak cloud platform, and a React-based web dashboard. A cloud AI prediction framework is proposed as an additional processing stage.

1) ESP32 Embedded Firmware: The embedded firmware is developed for the ESP32 microcontroller. It is responsible for reading sensor values, performing calculations, updating the TFT display, managing the configured output devices, and communicating with ThingSpeak.

The firmware performs the following operations:

- Initializing the sensors and display interfaces.
- Reading and averaging analog voltage and current measurements.
- Acquiring temperature and flame-sensor values.
- Calculating battery percentage and health-related indicators.
- Evaluating configured safety conditions.
- Updating the local display and output devices.
- Providing selected measurements through an HTTP JSON endpoint.
- Uploading selected data to ThingSpeak.

2) Local HTTP Interface: The ESP32 hosts a local HTTP server. The server provides a web page for displaying selected battery information and exposes a JSON endpoint named `/api/battery`. The endpoint makes the latest measurements and calculated indicators available to devices that can access the ESP32 over the local network.

The local HTTP interface provides access to device information independently of the ThingSpeak cloud dashboard.

3) ThingSpeak Cloud Platform: ThingSpeak is used as the cloud communication platform. The ESP32 sends selected measurements to the configured ThingSpeak channel. The uploaded parameters are organized into the following fields:

- Field 1: Battery voltage.
- Field 2: Battery current.
- Field 3: Battery temperature.
- Field 4: Battery percentage.
- Field 5: Battery health indicator.
- Field 6: Estimated remaining life.
- Field 7: Flame detection status.
- Field 8: Load status.

The firmware is configured to transmit cloud updates at an interval of approximately 20 seconds. The availability of cloud information depends on successful Wi-Fi and ThingSpeak communication.

4) React-Based Web Dashboard: The web dashboard is developed using React and CSS. It retrieves the latest ThingSpeak feed and displays selected battery measurements through a graphical interface. The dashboard includes sections for battery monitoring, battery health, charging status, safety, maintenance, range estimation, and energy-related information.

The dashboard periodically requests the latest ThingSpeak feed. Its current polling interval is approximately five seconds, while the embedded firmware uploads new cloud data approximately every 20 seconds. Therefore, some dashboard refreshes may display the same latest cloud measurement until a new upload becomes available.

H. Battery Parameter Estimation

The monitoring process uses sensor measurements and programmed functions to calculate selected battery indicators.

Let V_b represent the measured battery voltage, I_b the measured current, and T_b the measured battery temperature. The measurement vector is defined as

$$\mathbf{x}(t) = \begin{bmatrix} V_b(t) & I_b(t) & T_b(t) & F(t) \end{bmatrix}^T \tag{5}$$

where $F(t)$ represents the flame-sensor input at time t.

1) Battery Percentage Estimation: The firmware maps the measured voltage to a battery-percentage indicator. The implemented mapping uses a lower voltage threshold of 9.0 V and an upper voltage threshold of 12.0 V. The percentage is calculated as

$$B(V_b) = \begin{cases} 0, & V_b \le 9.0 \\ 100 \left(\dfrac{V_b - 9.0}{12.0 - 9.0} \right), & 9.0 < V_b < 12.0 \\ 100, & V_b \ge 12.0 \end{cases} \tag{6}$$

where $B(V_b)$ represents the estimated battery percentage. This is a voltage-based prototype indicator and should not be treated as a calibrated state-of-charge measurement under all battery load and temperature conditions.

2) Battery Health Indicator: The firmware calculates a health-related indicator by starting from 100 and applying fixed penalties according to configured voltage, current, and temperature thresholds. A simplified representation is

$$H = \text{clip}(100 - P_T - P_V - P_I, 0, 100) \tag{7}$$

where H is the calculated health indicator and the penalty terms are defined by the implemented rules.

The temperature penalty P_T is 10 when the temperature exceeds 45 C, with an additional penalty of 20 when it exceeds 55°C. The voltage penalty P_V is 10 when the voltage falls below 11.5 V, with an additional penalty of 10 when it falls below 10.5 V. The current penalty P_I is 5 when the absolute current exceeds 3 A. The resulting value is limited to the range from 0 to 100.

This calculation is a rule-based health indicator. It is not a measurement of electrochemical state of health and is not generated by a trained machine-learning model.

3) Estimated Remaining Life: The firmware calculates an indicative remaining-life value using the health indicator:

$$L_{\text{est}} = 5.0 \left(\frac{H}{100} \right) \tag{8}$$

where L_{est} is expressed in years and H is the calculated health indicator. This is a programmed proportional estimate and has not been validated against long-term battery ageing tests.

4) Nominal Range Estimation: The nominal range estimate is calculated using a configured full-range value and the battery-percentage indicator:

$$R_{\text{est}} = R_{\text{full}} \frac{B}{100} \tag{9}$$

where $R_{\text{full}} = 50\,\text{km}$ is the configured nominal full-range constant and B is the calculated battery percentage. This proportional estimate does not account for vehicle efficiency, driving conditions, road gradient, speed, or other factors that influence actual range.

5) Instantaneous Power Calculation: The instantaneous power estimate is calculated from the measured voltage and current:

$$P(t) = |V_b(t) I_b(t)| \tag{10}$$

where $P(t)$ is expressed in watts. This calculation represents instantaneous power rather than accumulated energy consumption in watt-hours.

I. Safety Monitoring and Processing

The safety monitoring module evaluates the sensor information using the conditions programmed in the ESP32 firmware. The flame sensor is configured according to its digital input logic. The firmware also evaluates temperature and voltage thresholds to identify selected abnormal conditions.

The configured safety conditions include:

- Flame detection or a latched fire event.
- Overtemperature condition at or above 60°C.
- High-temperature warning at or above 45°C.
- Low-voltage condition below 5 V.

The safety status is updated according to the implemented decision logic. The buzzer and indicator LEDs provide local alert outputs. The relay is also a firmware-controlled output. Its actual response should be documented according to the verified hardware test rather than assumed from the presence of the relay module.

J. System Workflow

The complete monitoring process begins with the acquisition of battery measurements. The ESP32 processes the sensor inputs, calculates battery indicators, updates the local display, and manages the configured safety outputs. It also provides local HTTP data and transmits selected measurements to ThingSpeak. The React dashboard retrieves the latest cloud feed for remote visualization.

The proposed AI framework can use historical cloud measurements for further data analysis after suitable training data and reference health labels become available.

Fig. 3 illustrates the monitoring and safety-processing workflow implemented in the proposed system.

K. Implementation Methodology

The implementation of the proposed prototype was organized into the following development phases:

1) **Hardware Development:** Integration of the ESP32, voltage sensor, ACS712 current sensor, DS18B20 temperature probe, flame sensor, TFT touchscreen, relay, buzzer, and indicator LEDs.
2) **Sensor Integration:** Configuration of the analog and digital interfaces for acquiring battery voltage, current, temperature, and flame-sensor information.
3) **Embedded Firmware Development:** Development of the ESP32 program for sensor acquisition, measurement processing, battery indicator calculation, display updates, and safety-related output control.
4) **Local Interface Development:** Implementation of the embedded HTTP server and JSON endpoint for providing selected battery information over the local network.
5) **Cloud Integration:** Configuration of ThingSpeak fields and firmware-based transmission of selected battery measurements.
6) **Web Dashboard Development:** Development of the React interface for retrieving the latest ThingSpeak feed and displaying battery parameters.

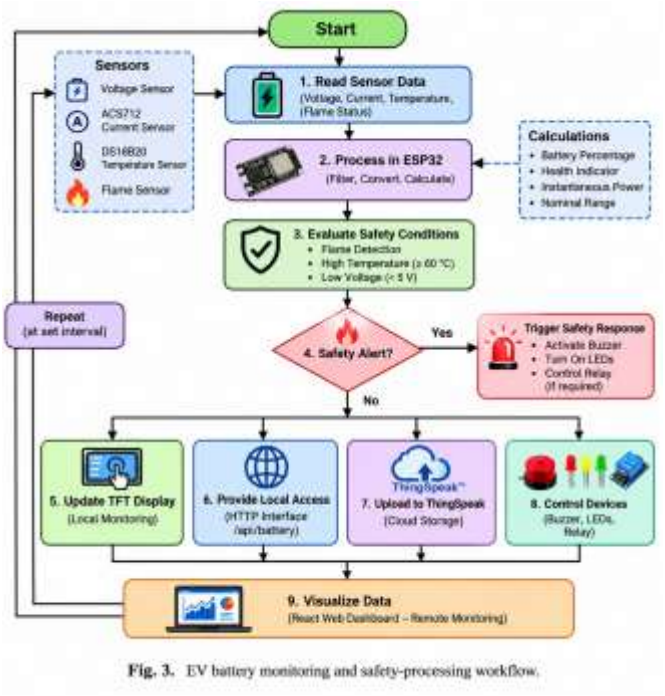


Fig. 3. EV battery monitoring and safety-processing workflow.

7) **AI Framework Design:** Definition of a proposed cloud-based machine-learning workflow for historical battery data processing and future health prediction.
8) **System Integration:** Integration of hardware, firmware, local display, cloud communication, and web dashboard components.
9) **System Testing:** Verification of sensor readings, display operation, communication, calculated indicators, and safety-related responses under controlled conditions.

L. Advantages of the Proposed Design

The proposed architecture offers the following advantages:

- Continuous observation of selected battery parameters through embedded sensing.
- Local visualization using the TFT touchscreen.
- Remote access to uploaded battery measurements through the ThingSpeak platform.
- A React-based dashboard for graphical presentation of selected battery information.
- Integration of flame and temperature inputs for safety-related monitoring.
- Modular hardware and software architecture that supports future improvements.
- Programmed battery indicators that provide an accessible summary of monitored conditions.
- A proposed cloud AI framework that provides a direction for future data-driven battery health analysis.

The accuracy of the calculated indicators, reliability of the communication links, and effectiveness of the safety responses

require further experimental evaluation.

IV. Experimental Results and Performance Evaluation

A. Experimental Setup

The proposed IoT-Based EV Battery Monitoring System was implemented as a functional prototype using an ESP32 development board and the associated sensing and display modules. The experimental setup included a voltage sensor, ACS712 current sensor, DS18B20 temperature sensor, flame sensor, ILI9341 TFT touchscreen, and configured safety output devices.

The ESP32 acquired the sensor measurements and performed the programmed calculations. The local display presented the processed battery information. The embedded HTTP server provided selected information through a JSON endpoint, while ThingSpeak was used to upload measurements for remote monitoring. A React-based dashboard retrieved the latest cloud feed and displayed the selected parameters.

The experimental evaluation focused on checking the operation and integration of the hardware and software components. It included sensor acquisition, local display updates, cloud communication, dashboard visualization, and the operation of the programmed monitoring functions.

B. Experimental Procedure

The prototype was evaluated through a sequence of functional checks. The procedure followed the main data-processing path of the proposed system:

1) The ESP32 and connected sensor modules were initialized.
2) The voltage sensor and ACS712 current sensor provided analog measurements to the controller.
3) The DS18B20 sensor supplied temperature information through the One-Wire interface.
4) The flame sensor supplied its digital detection status to the ESP32.
5) The firmware processed the measurements and calculated the programmed battery indicators.
6) The TFT touchscreen was updated with the latest measurements and system status.
7) The embedded HTTP interface made selected battery information available in JSON format.
8) The ESP32 transmitted configured measurements to the ThingSpeak cloud platform.
9) The React dashboard retrieved the latest available cloud feed and displayed the corresponding battery information.

The checks were intended to verify the functional operation of the integrated prototype. They do not establish calibrated sensor accuracy, long-term battery health prediction accuracy, or safety certification.

TABLE II
Functional Verification of the Proposed System

Test Case	Expected Behaviour	Result
Voltage Measurement	ESP32 reads voltage sensor input	Verified
Current Measurement	ESP32 reads ACS712 sensor input	Verified
Temperature Measurement	DS18B20 provides temperature data	Verified
Flame Sensor Input	Controller reads flame detection status	Verified
TFT Display	Battery information is presented locally	Verified
Local HTTP Interface	JSON endpoint provides selected data	Verified
ThingSpeak Communication	Configured measurements are uploaded	Verified
React Dashboard	Latest available cloud data is displayed	Verified

C. Functional Verification

Functional verification was conducted to examine the operation of the implemented hardware and software components. The test cases covered sensor acquisition, display operation, local HTTP communication, cloud data transmission, and dashboard updates.

The principal functional checks are summarized in Table II.

The functional checks indicate that the principal monitoring components were integrated and operated under the tested prototype conditions. The verification results represent functional observations rather than quantitative measurements of accuracy or reliability.

D. Sensor Monitoring and Data Visualization

The ESP32 collects the sensor measurements and processes them before updating the local display. Voltage and current measurements are obtained through analog inputs, while temperature and flame information are acquired through their respective sensor interfaces.

The local TFT touchscreen provides immediate access to the measured parameters. The embedded HTTP interface provides selected measurements in JSON format, allowing compatible devices on the local network to retrieve the current information.

The ThingSpeak platform provides a separate path for cloud-based monitoring. The firmware uploads selected measurements to the configured channel fields. The React dashboard retrieves the latest available cloud feed and presents the measurements through its graphical interface.

The firmware is configured to upload data approximately every 20 seconds, whereas the dashboard requests updated cloud information approximately every five seconds. Consequently, the dashboard may display the same cloud measurement across multiple refresh cycles when a new firmware upload has not yet occurred.

E. Performance Evaluation Metrics

The performance of the proposed system can be evaluated using delivery time, sensor measurement error, communication

success rate, and dashboard update delay. These metrics provide a framework for quantitative evaluation of the prototype.

1) End-to-End Data Delivery Time: The end-to-end delivery time represents the time required for a measurement to travel from acquisition to its availability on the remote dashboard. It can be expressed as

$$T_{total} = T_{acq} + T_{proc} + T_{cloud} + T_{dash} \quad (11)$$

where T_{acq} represents sensor acquisition time, T_{proc} represents ESP32 processing time, T_{cloud} represents cloud communication and storage time, and T_{dash} represents the dashboard retrieval and display time.

The total delivery time can be measured using timestamps recorded at the corresponding stages. Repeated measurements are required to determine average delay and variation under different network conditions.

2) Sensor Measurement Error: The accuracy of the voltage and current measurements can be evaluated by comparing the sensor readings with a calibrated reference instrument. The percentage error can be calculated as

$$E(\%) = \frac{|X_{measured} - X_{reference}|}{X_{reference}} \times 100 \quad (12)$$

where $X_{measured}$ is the value reported by the monitoring system and $X_{reference}$ is the corresponding reference measurement.

This evaluation should be performed at multiple operating points to identify measurement deviations across the intended range.

3) Communication Success Rate: The communication success rate measures the proportion of successful data transmissions or retrieval requests. It can be calculated as

$$R_{success}(\%) = \frac{N_{success}}{N_{total}} \times 100 \quad (13)$$

where $N_{success}$ represents the number of successful communication operations and N_{total} represents the total number of attempted operations.

Repeated tests under different network conditions are necessary to evaluate communication reliability.

4) Dashboard Update Delay: The dashboard update delay represents the time between a new cloud measurement becoming available and its presentation on the React dashboard. It can be evaluated using

$$T_{update} = t_{display} - t_{upload} \quad (14)$$

where t_{upload} is the time at which the cloud measurement is uploaded and $t_{display}$ is the time at which the updated value is displayed on the dashboard.

The actual delay depends on the firmware upload interval, cloud service response time, network latency, and dashboard polling interval.

F. Discussion of Experimental Results

The functional evaluation demonstrates the integration of the ESP32 controller, sensor modules, TFT touchscreen, embedded HTTP interface, ThingSpeak cloud service, and React dashboard. The implemented system provides local access to battery measurements and supports remote visualization of the latest uploaded information.

The voltage and current sensor interfaces provide the electrical measurements used by the firmware. The DS18B20 provides temperature information, while the flame sensor contributes a digital safety-related input. The ESP32 processes these inputs and updates the local display and configured output devices.

The ThingSpeak integration provides cloud storage for selected measurements. The React dashboard retrieves the latest available feed and presents the corresponding battery parameters. This arrangement allows the system to provide both local and remote monitoring functions.

The calculated battery percentage is based on the configured voltage mapping. The health indicator is generated using fixed threshold-based penalties, while the remaining-life estimate is derived proportionally from the health indicator. The nominal range estimate is based on the configured full-range value and battery percentage. These values are useful as programmed prototype indicators, but they should not be interpreted as validated state-of-charge, electrochemical state-of-health, remaining-useful-life, or actual vehicle-range predictions.

The proposed AI framework provides a methodology for future analysis of historical battery measurements. Its actual prediction performance cannot be established until a suitable labelled dataset is obtained, a model is trained, and independent testing is performed.

The functional tests demonstrate the operation of the implemented components under the evaluated conditions. However, quantitative measurements are still required to establish sensor accuracy, end-to-end communication delay, cloud delivery reliability, and the effectiveness of safety-related responses. Further testing with calibrated instruments and repeated measurements would provide a stronger basis for assessing the performance of the proposed system.

V. Discussion

A. Technical Discussion

The proposed IoT-Based EV Battery Monitoring System integrates sensor acquisition, embedded processing, local visualization, and cloud-based monitoring within a single prototype. The ESP32 serves as the central controller and coordinates the connected sensors, TFT touchscreen, communication interfaces, and configured output devices.

One of the main features of the system is the integration of local and remote monitoring. The TFT touchscreen provides immediate access to the measured battery parameters, while the embedded HTTP interface provides selected information to devices connected to the local network. ThingSpeak supports

cloud storage of the configured measurements, and the React dashboard provides a graphical interface for remote observation.

The modular software architecture separates sensor acquisition, data processing, display management, local HTTP communication, and cloud transmission. This separation supports maintenance and future modification of individual system components.

The battery percentage calculation uses a configured voltage range. Although this provides a simple indicator for the prototype, voltage alone does not fully represent battery state of charge under changing load and temperature conditions. More accurate state-of-charge estimation would require suitable battery characterization, calibration, and consideration of current and operating history.

Similarly, the health indicator is calculated through programmed threshold penalties. It provides a summary of selected operating conditions, but it does not measure actual battery degradation. The remaining-life value is derived from this indicator and therefore inherits its limitations. A validated battery health or remaining-useful-life prediction system would require historical battery data, a suitable estimation model, and testing against reference measurements.

The nominal range estimate is proportional to the calculated battery percentage and a configured full-range constant. It does not account for vehicle efficiency, driving style, road conditions, or environmental factors. Therefore, it should be considered a nominal estimate rather than a prediction of actual driving distance.

The proposed cloud AI framework provides a possible approach for extending the system beyond programmed indicators. It requires reliable historical battery measurements and reference state-of-health values. The model must be trained and evaluated independently before its predictions can be used as reliable battery health information.

B. Practical Applications

The proposed system can provide a foundation for battery monitoring in small electric vehicle prototypes, laboratory battery test arrangements, and educational IoT demonstrations. Its local display and cloud connectivity can help users observe selected battery parameters without depending entirely on manual measurements.

Potential application areas include:

- **Electric Vehicle Prototypes:** Monitoring battery voltage, current, and temperature during controlled operation.
- **Battery Testing Laboratories:** Collecting and displaying selected battery measurements during experimental testing.
- **Educational Demonstrations:** Demonstrating sensor integration, embedded processing, IoT communication, and cloud visualization.
- **Remote Monitoring:** Providing access to uploaded battery measurements through a web dashboard.
- **Safety Monitoring Demonstrations:** Demonstrating temperature and flame-sensor inputs with programmed local alerts.

The suitability of the prototype for practical vehicle deployment would require further electrical, thermal, mechanical, and safety validation.

C. Advantages of the Proposed System

The proposed system provides several advantages:

- **Continuous Monitoring:** Sensor measurements provide ongoing information about selected battery operating parameters.
- **Local Visualization:** The TFT touchscreen presents battery information directly on the embedded device.
- **Remote Accessibility:** ThingSpeak and the React dashboard support remote observation of uploaded measurements.
- **Integrated Sensing:** Voltage, current, temperature, and flame-sensor information are collected through a single controller.
- **Modular Architecture:** The separation of hardware and software functions supports future system improvements.
- **Low-Cost Prototype:** The system uses commonly available embedded hardware and software platforms.
- **Expandable Design:** Additional sensors, communication features, and improved estimation methods can be incorporated in future versions.
- **AI-Oriented Architecture:** The proposed cloud framework provides a basis for future machine-learning-based battery health analysis.

D. Limitations and Challenges

Despite the successful integration of the prototype components, certain limitations remain.

- **Sensor Calibration:** The accuracy of voltage and current measurements depends on sensor calibration and the quality of the electrical connections.
- **Battery Percentage Estimation:** The implemented voltage-based calculation is a simplified indicator and does not provide a fully calibrated state-of-charge measurement.
- **Health Estimation:** The health indicator uses programmed threshold rules rather than a trained and validated battery degradation model.
- **AI Model Validation:** The proposed cloud AI framework requires suitable labelled battery ageing data, model training, and independent testing.
- **Remaining-Life Estimation:** The remaining-life value is a proportional estimate and has not been verified through long-term battery ageing experiments.
- **Range Estimation:** The nominal range calculation does not include real vehicle operating conditions or energy efficiency variations.
- **Cloud Dependency:** Remote monitoring depends on Wi-Fi connectivity and the availability of the cloud service.

- **Safety Validation:** The flame sensor and configured output devices require controlled testing to establish their response and reliability.

These limitations define important areas for further development and experimental investigation.

VI. Conclusion and Future Work

A. Conclusion

This paper presented the design and implementation of an IoT-Based EV Battery Monitoring System with a cloud-connected architecture using an ESP32 microcontroller. The prototype integrates voltage, current, temperature, and flame-sensor inputs to provide information about selected battery operating conditions.

The system uses an ESP32 controller to acquire and process sensor measurements. An ILI9341 TFT touchscreen provides local visualization, while an embedded HTTP interface makes selected battery information available over the local network. ThingSpeak supports cloud data transmission, and a React-based web dashboard retrieves the latest uploaded measurements for remote monitoring.

The firmware calculates battery percentage, a health-related indicator, estimated remaining life, instantaneous power, and nominal range using programmed logic. These calculations provide indicative information for the prototype and are not presented as validated machine-learning predictions or direct measurements of battery degradation.

A cloud AI health prediction framework has also been proposed to investigate the use of historical battery measurements for future battery health estimation. The framework describes data collection, preprocessing, feature selection, model training, validation, and cloud-based prediction. Actual AI prediction requires suitable labelled battery ageing data and experimental validation.

Functional verification was used to examine the operation of the sensor interfaces, display, local communication interface, cloud transmission, and dashboard. The results demonstrate the integration of the principal hardware and software components under the evaluated conditions. However, further quantitative testing is required to establish measurement accuracy, communication performance, and the effectiveness of safety-related responses.

Overall, the proposed prototype demonstrates a practical approach to combining embedded sensing, local monitoring, and cloud connectivity for electric vehicle battery observation. Its modular architecture provides a foundation for further development towards more accurate battery estimation and cloud-based intelligent monitoring.

B. Future Work

Several improvements can be considered to enhance the accuracy, reliability, and functionality of the proposed system.

- **Advanced State-of-Charge Estimation:** Future work can investigate calibrated voltage-based methods, coulomb counting, and model-based estimation techniques to improve battery-percentage accuracy under different operating conditions.
- **Machine-Learning-Based Battery Health Prediction:** A trained machine-learning model can be developed using suitable battery ageing datasets and validated measurements. Such a model could investigate battery degradation and state-of-health estimation more accurately than the current threshold-based indicator.
- **Cloud AI Model Integration:** The trained model can be deployed in a cloud-based processing environment. Historical battery measurements uploaded through ThingSpeak can then be analysed to generate health estimates for presentation on the web dashboard.
- **Remaining Useful Life Prediction:** Future development can incorporate historical battery measurements, charge-discharge cycles, and ageing characteristics to investigate validated remaining-useful-life estimation.
- **Improved Sensor Calibration:** The voltage and current sensors can be calibrated against reference instruments at multiple operating points. This would help reduce measurement errors and improve the reliability of calculated parameters.
- **Enhanced Safety Monitoring:** Further work can investigate improved temperature sensing, fault detection, alert mechanisms, and verified relay-control behaviour under controlled test conditions.
- **Advanced Cloud Analytics:** Additional cloud-based processing can be explored to analyse historical battery measurements, identify trends, and provide improved monitoring information.
- **Mobile Application Development:** A dedicated mobile application can be developed to provide convenient access to battery measurements, alerts, and historical monitoring information.
- **Extended Experimental Evaluation:** Future experiments can measure end-to-end delivery time, sensor accuracy, communication success rate, dashboard update delay, and system behaviour under different network and operating conditions.

These improvements could help transform the current functional prototype into a more accurate and reliable battery monitoring platform suitable for extended experimental evaluation and future electric vehicle applications.