Iot-Enabled Smart EV Charging System With Real-Time Monitoring And Cost Optimization

SRINIVAS*, ARUL KUMAR†, YOGIRAJ‡, S. SATHEESH KUMAR§
*UG Student, Department of ECE,
†UG Student, Department of ECE,
‡UG Student, Department of ECE,
§Assistant Professor, Department of ECE,
Sree Sastha Institute of Engineering and Technology, Chembarambakkam - 123, Chennai, India

beginabstract As the world continues to transition towards sustainable transport, electric vehicles (EVs) have become the focal point of contemporary mobility solutions. Still, their accompanying charging infrastructure remains unintelligent, unautomated, and inaccessible remotely. This paper presents an extensive smart EV charging system driven by IoT to overcome these setbacks. The suggested architecture integrates an ESP32 microcontroller for real-time sensor integration, a Python backend for decision making and serial communication, and a React.js-based web interface for real-time user interaction. The system tracks voltage and current parameters, logs charging sessions via Firebase, and offers remote control functionalities like starting or stopping charging. Integrated is peak-hour consciousness to reduce electricity bills. Developed to be modular and scalable, the architecture is flexible for different types of deployments such as for residential, educational, and commercial settings. This platform creates a foundation for more intelligent energy consumption, greater user control, and connectivity with next-generation technologies like AI-based scheduling and mobile app interfaces.

***Index Terms*—Smart EV Charging, Internet of Things (IoT), ESP32, Real-Time Monitoring, Python Backend, React Dash- board, Firebase, Peak-Hour Optimization, Cloud Logging, Re- mote Control, Cost-Efficient Charging, Energy Analytics, Mod- ular Design, Embedded Systems.**

I. Introduction

The worldwide shift towards cleaner mobility has seen a huge surge in the use of electric vehicles (EVs). With increasing usage of EVs, the need for intelligent, efficient, and networked charging systems has grown parallel to it. Conventional EV charging systems are usually restrictive in functionality, providing plain features like manual operation, fixed power output, and absence of any feedback component. Such systems lack real-time status updates, remote access, or energy optimization, hence are less appropriate for current energy demands.

In most environments, including residential or educational, users are not in a position to know the actual real-time power usage of a charging session. No overuse alerts exist, no energy usage records are kept, and no automated management features are present. Consequently, users are presented with real-world problems like increased electricity bills, wastage of energy, and the hassle of manual intervention. These constraints lower the overall efficiency and reliability of existing EV charging facilities.

II. AIM

The aim of this project is to build a smart electric vehicle (EV) charging system that uses modern IoT technology to offer real-time monitoring, user-friendly control, and energy-efficient performance. At its core, the system brings together ESP32-based hardware to collect sensor data, a Python (Flask) backend to handle communication and logic, and a Firebase database for cloud-based session logging. A React.js web dashboard completes the architecture by allowing users to interact with the system remotely and view live charging updates. The system supports key features such as starting and stopping the charging process from anywhere, identifying high-tariff peak hours to reduce electricity costs, and displaying current and voltage levels in real-time. LED indicators on the hardware give clear status feedback during operation. Intended to be modular and scalable, this configuration is perfectly suited for houses, campuses, and community charging sites. In the long run, this project can deliver a low-cost, smart, and quickly deployable EV charging solution that supports smarter energy consumption and sustainable transport objectives.

III. Objectives

A smart electric vehicle (EV) charging system with real-time monitoring, remote capabilities, and cloud-based secure data storage. Through the convergence of IoT technology and a simple and easy-to-use interface, the system aims to provide convenient, energy-efficient, and cost-effective daily EV charging.

At the functional level, the system is made to read voltage and current values at all times through the use of sensors such as ACS712 and voltage dividers. These values are not only presented to the user in real time using a web dashboard, but

also stored and recorded in the cloud using Firebase. The users can remotely begin or terminate the charging session from the dashboard, and LED indicators on the hardware also give immediate tactile feedback regarding the charging. The setup also monitors session metrics like time, energy consumed, and estimated expenditure.

Technically, the system design is structured into three critical modules. The ESP32 microcontroller handles sensor data capture and control signal execution. The backend, implemented in Python with the Flask framework, handles serial communication, decodes incoming data, and interacts with the cloud database. A frontend based on React.js is used as the user interface to display live charts and provide control settings to the user. As a whole, these parts are able to provide secure data transfer, stable cloud sync, and interactive user feedback.

Apart from simple control and monitoring, the system is also capable of assisting consumers in saving electricity bills by identifying peak usage hours. By inhibiting or postponing charging during peak tariffs, the system facilitates more effective energy consumption and cost optimization.

The platform is built to scale and be flexible. Its modularity makes it easy to upgrade it at a later time, say to support multiple charging stations, add mobile app functionality, or utilize machine learning for predictive scheduling.

This is made in a universal and versatile manner for multiple applications, so it might serve individual home users, schools and universities, and public charging stations. Future potential developments would include the integration of secure user login, usage history tracking, and billing integration to make it even more useful in real life.

IV. SCOPE

The intention of theproject is to create a smart electric vehicle (EV) charging system that provides users with greater control, real-time updates, and intelligent energy consumption. By integrating IoT technology with ease of use, the system makes the charging process simple, efficient, and accessible anywhere.

The solution encompasses a number of major functions including constant reading of current and voltage through sensors, real-time display of status on an online dashboard, remote start and shutdown, and safe session login into the cloud. The system is developed based on the three-layered architecture with the integration of embedded hardware, cloud-based backend, and user-facing interface.

At the microcontroller level, the ESP32 microcontroller is utilized to read data from current and voltage sensors and process control commands to switch the relay on and off. There is a Python-based Flask backend acting as the middle layer, which takes care of serial communication, executes decision logic, and feeds data into Firebase. The frontend, developed with React.js, enables users to view the system status in the form of live graphs and interact with the system using a web-based interface.

Users can monitor real-time values, voltage readings, and approximate session cost. They can even remotely switch the charging process on or off from the dashboard. Charging beginning or end is indicated with a notification, and warnings are issued if charging is attempted during specified peak hours. Such features render the system extremely user-friendly and cost-conscious.

The system is designed to be installed in different environments, such as private residences, school campuses, and public shared charging stations. Its architecture is adapted to be flexible and expandable, which makes it compatible with either small or mass installations.

While the system already contains necessary features, billing, user login, and mobile app integration are not supported by it at present. These are scheduled for subsequent phases. The installation also does not yet support wireless authentication approaches such as RFID.

Yet, the modularity of the system ensures it is simple to introduce more features in the future. Some of these are AI-based charging schedules, app notifications, management of more than one charging port, and complete analytics dashboards. The architecture is open enough to change to meet evolving technology requirements and user expectations, opening the way for a smarter and more integrated EV charging system.

V. SYSTEM ARCHITECTURE OVERVIEW

The smart EV charging system has been developed with a transparent, layered architecture to facilitate seamless coordination among its hardware, software, and user-oriented components. The configuration is divided into three vital layers, with each handling its own task, collaboratively providing real-time control, energy tracking, and cost-saving functionalities. The overall structure is illustrated in Figure 1.

A. Hardware Layer

Behind the system lies the hardware layer, which functions based on an ESP32 microcontroller. This component of the system takes voltage and current readings using sensors. It also manages a relay switch that is activated or deactivated based on instructions from the backend. Everything is read in real-time and transmitted to the subsequent layer for further operation.

B. Backend Layer

The middle component is built in Python using the Flask framework. It is an interpreter and decision-maker. It takes incoming data from the ESP32 via a serial connection, interprets it, and uploads the output to the cloud database. It also gets commands from the user—like initiating or terminating the charging process—and sends appropriate feedback to the hardware.

C. Frontend Layer

The uppermost layer is the web interface, built using React.js. It is connected to the Firebase Realtime Database to retrieve real-time updates on voltage, current, and other session information. Through this dashboard, users are able to see real-time charging progress and manage the charging remotely. The

interface is made visually intuitive so that users can make fast decisions with ease.

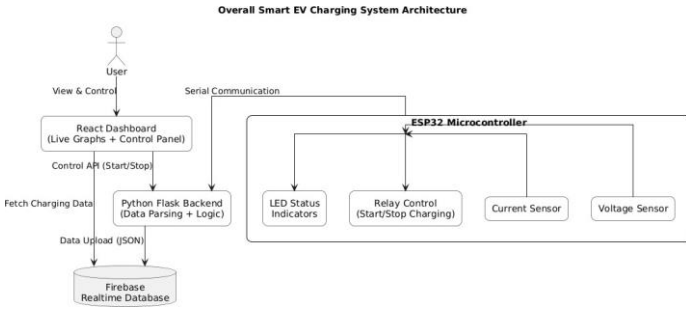

Fig. 1. System Architecture of the Smart EV Charging System

D. Hardware Layer (ESP32-Based Control)

E. Backend Layer (Flask + Python)

The foundation of the smart EV charging system is its hardware layer, driven by the ESP32-WROOM-32 microcontroller. This controller manages sensor readings, relay activation, and communication with the backend. Several important components are interfaced with the ESP32 to achieve these goals:

- **ACS712 Current Sensor:** Captures real-time current readings from the EV charger.
- **Voltage Sensor Module:** Monitors voltage input to ensure it remains within operational safety limits.
- **Relay Module (4-channel):** Enables or disables the power supply to the EV, based on commands received from the backend or dashboard.
- **LED Indicators (Red, Yellow, Green):** Visually indicate system status—Idle, Charging, Fully Charged, or Error.
- **Cooling Fan (Optional):** Assists in thermal management during extended operation, protecting sensitive components.

The ESP32 reads voltage and current values in real time, formats them (e.g., `Voltage: 230V, Current: 1.5A`), and sends the data to the backend via a UART serial connection. It also listens for incoming control instructions such as turning the relay ON/OFF, based on peak-hour logic or user inputs.

The backend layer serves as the decision-making hub of the system. It is developed using Python along with the Flask framework to manage the flow of data between the embedded hardware and the user interface. This module maintains a continuous communication link with the ESP32 microcontroller via a serial connection, which enables the system to collect real-time readings of voltage and current.

Upon receiving raw sensor values, the backend parses and formats the data into a usable structure. It then executes the core logic necessary for intelligent operation, such as monitoring for peak-hour electricity rates, detecting abnormal behavior, and deciding whether charging should be allowed or temporarily disabled.

Beyond immediate control, the backend is also responsible for logging important session metrics—such as voltage, current, timestamps, and total charging duration—to the Firebase Realtime Database. This ensures that session history is securely stored and can be accessed by the frontend dashboard at any time.

Furthermore, the backend exposes a series of RESTful API endpoints that enable the frontend to transmit control commands, such as starting or stopping a charging session. When such requests are received, the backend processes the command and issues the appropriate response to the ESP32 hardware.

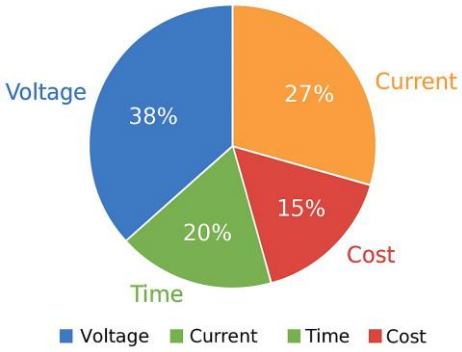

Figure 4.2
Distribution of Firebase-Loggged Parameters

Fig. 2. Distribution of Firebase-Logged Parameters

The backend is developed with a modular structure, promoting maintainability and scalability. This design choice allows for easy future expansion to incorporate features like user authentication, automated billing, or AI-based energy prediction, making the system flexible and future-ready.

F. Cloud Integration (Firebase Realtime Database)

To ensure fluid, secure, and real-time access to data, the system uses Firebase Realtime Database as its cloud storage. Firebase is chosen for the simplicity of setup, minimal overhead, and capabilities to asynchronously synchronize data between backend and frontend.

As the charging session progresses, the Python Flask backend pushes structured information to Firebase. This allows the frontend dashboard to reflect live updates without delay.

The system records several important metrics, including:

- Voltage and current values continuously measured during charging.
- Battery level percentage, if available or calculated.
- Session start and end times, to track duration.
- Estimated energy usage and cost, for analytics and billing previews.

All of this information is available to the user in real time and can be retrieved from any device with access to the React.js dashboard. This not only supports current session tracking but also enables historical data review, energy analysis, and system diagnostics.

G. Frontend Layer (React.js Dashboard)

The user interface is constructed utilizing **React.js**, serving as the interactive, real-time dashboard for communicating with the smart EV charger. It provides an intuitive experience across desktops, tablets, and mobile devices.

- **Live Charts:** Graphs show real-time voltage and current values, allowing users to visualize energy usage over time.
- **Control Panel (Start/Stop):** Charging can be started or stopped remotely with a single click.
- **Alerts and Warnings:** Notifications are issued during peak energy hours to avoid high energy costs.
- **Session Summary:** The dashboard displays current voltage, amperage, estimated cost, and charging status in real time.

The dashboard retrieves live data from Firebase via polling or listeners and uses RESTful POST API calls to send control signals to the Flask backend.

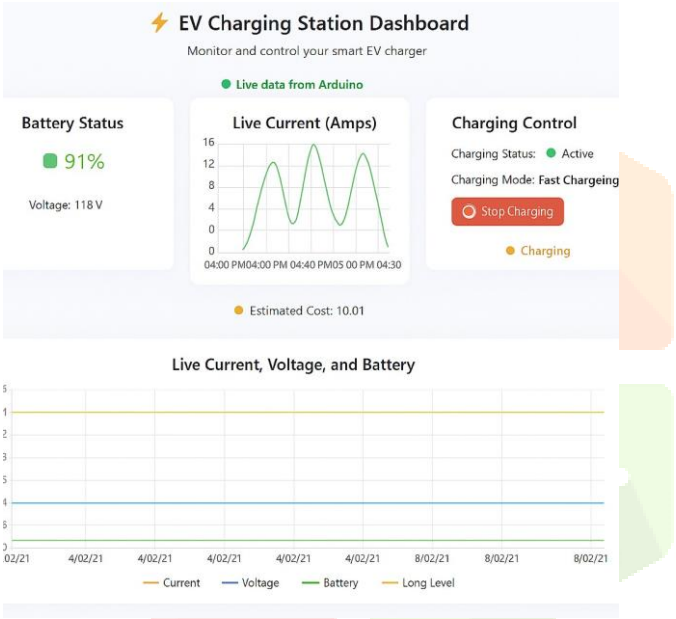

Fig. 3. Screenshot or UI wireframe of the React.js Dashboard Interface

H. Charging Logic and Peak-Hour Optimization

To reduce unnecessary energy expenses and encourage optimal charging, the system includes a peak-hour detection algorithm integrated into the backend. It continuously monitors the system clock and compares it to predefined peak-hour intervals, typically from 6:00 PM to 10:00 PM.

If a user initiates charging during these hours, the system will:

- Provide a warning notification through the dashboard,
- Delay the session start automatically (if set),
- Or block charging entirely until the period ends.

This feature supports smart scheduling and cost-efficiency for users and energy providers alike.

I. System Flow of Execution

The full operation of the smart charging system is illustrated in Figure 4.4. Each layer collaborates in a real-time data loop from hardware to the user interface.

1) The EV is connected to the charger.
2) ESP32 captures voltage and current data from sensors.
3) This data is transferred to the Python backend via serial (UART).
4) The backend parses, logs, and stores it into Firebase.
5) Simultaneously, peak-hour logic checks are performed.
6) The React.js dashboard fetches and displays data live.
7) User actions (start/stop) are sent to the backend via API.
8) The backend translates commands and relays them to ESP32.
9) The session is logged, synced, and monitored from all devices.

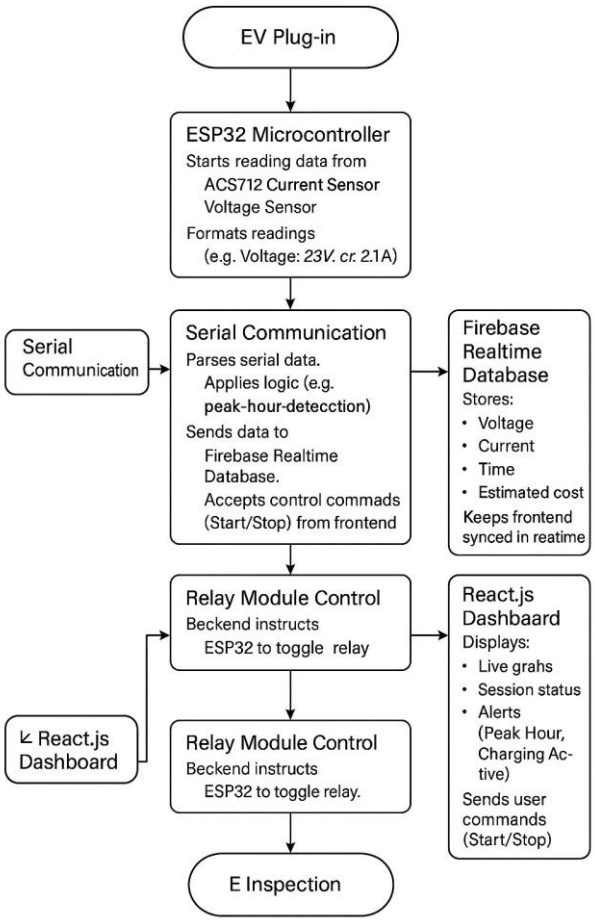

Fig. 4. Execution Flow Diagram of the Smart EV Charging System

J. Deployment and Modularity

The smart EV charging system is versatile and can be deployed in a variety of environments, including:

- **Residential Use:** Enables homeowners to charge EVs efficiently and intelligently.

- **Educational Campuses:** Allows central control of multiple charging points across institutions.
- **Public Charging Spots:** Useful for community and commercial charging zones.

Its modular design supports future integrations such as:

- **RFID Authentication** for secure access control,
- **AI-Powered Scheduling** for predictive, automated charging,
- **Mobile App Control** for on-the-go monitoring and interaction,
- **Multi-Port Expansion** to serve fleet-based or shared usage environments.

VI. IMPLEMENTATION AND EXPERIMENTAL SETUP

6.1 Introduction to Implementation The development of the smart electric vehicle (EV) charging system was carried out in a modular fashion to allow clear separation between hardware, backend control, and frontend interface. This section outlines the practical steps taken to bring the proposed system architecture into a fully functioning prototype. Each module was assembled, coded, and tested independently before being integrated into a single synchronized framework.

6.2 Hardware Implementation

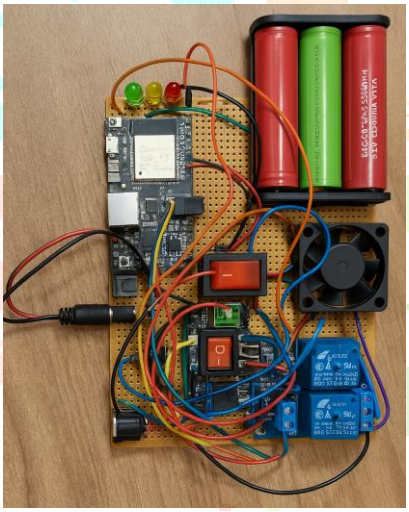

Fig. 5. Hardware Implementation

The hardware setup is built around the ESP32-WROOM-32 microcontroller, acting as the system's central controller. The key sensor components include:

- ACS712 Current Sensor – Tracks the amount of current drawn during charging. - Voltage Divider Circuit – Measures the voltage supplied to the connected EV. - Relay Module (4-Channel) – Controls the connection and disconnection of the power supply. - LED Indicators – Visual cues: green indicates active charging, red shows idle/off state. - Load Simulation – A 12V DC bulb was used to simulate EV load for testing purposes.

All components were neatly assembled and housed in a compact enclosure to ensure portability and safety. Proper insulation and heat ventilation (optional cooling fan) were also considered to avoid overheating during prolonged testing.

6.3 Backend Implementation (Python Flask) The backend logic is coded using Python, with the Flask microframework facilitating web-based interactions. The backend performs the following functions:

- Establishes a stable UART connection with ESP32. - Parses incoming serial strings like "Voltage:230, Current:1.5" into structured values. - Implements decision logic to detect peak-hour constraints. - Sends formatted session data to Firebase. - Accepts and processes REST API calls (e.g., /start, /stop) from the React frontend.

The structure was intentionally kept modular to allow quick modifications or enhancements. Short delays and error handlers were also added to ensure system stability.

6.4 Frontend Dashboard Implementation (React.js) The user-facing dashboard was developed using React.js for its component-based architecture and fast rendering. It features:

- Battery Status Graphs – Live chart updates of current, voltage, and session trends. - Charging Controls – Start/Stop buttons to remotely toggle charging. - Peak-Hour Notification – Real-time alerts if the system detects high-tariff conditions. - Cost Estimation – Displays an approximate session cost based on power consumed.

Firebase listeners are used to reflect real-time values, and Axios handles communication with the Flask backend. The dashboard is designed to be mobile-friendly and works smoothly across various screen sizes.

6.5 Firebase Configuration Firebase Realtime Database was chosen due to its simplicity and instant synchronization across devices. The backend uploads the following parameters:

- Voltage - Current - Time (Start, End) - Estimated Cost

The data is structured in JSON format and secured with basic access rules to prevent unauthorized access. Firebase's lightweight infrastructure ensured smooth, uninterrupted sync between the backend and frontend.

6.6 Experimental Setup The prototype was tested in a simulated home environment under various conditions. The test environment included:

- Test Location: Indoor setup with access to 230V AC and 12V DC adapters. - Test Load: 12V DC bulb. - Test Duration: Over 10 sessions conducted across different hours of the day. - Devices Used: Laptop for backend server, smartphone for frontend dashboard access. - Monitoring: Live manual dashboard checking and Firebase log validation.

The tests also covered both peak and non-peak hours to verify cost optimization behavior.

6.7 Key Observations The outcomes from testing are summarized below:

- Voltage Range Observed: 11.6V to 12.2V (simulated). - Current Range: 0.3A to 1.2A depending on load. - Peak Hour Logic: Triggered and alerted users correctly. - Relay Switching: Completed toggle operations within 0.5 seconds. - Dashboard Sync: Displayed live data with no observable lag.

6.8 Equations Used

Equation 1: Instantaneous Power Calculation

$$P(t) = V(t) \times I(t)$$

Where: - $P(t)$ is the instantaneous power in watts - $V(t)$ is the voltage at time t - $I(t)$ is the current at time t

Equation 2: Total Energy Consumed

$$E = \sum_{t=1}^{T} P(t) \times \Delta t$$

Where: - E is total energy in watt-seconds - Δt is the time interval between readings

Equation 3: Cost Estimation

$$\text{Cost} = \frac{E}{1000} \times \text{Tariff Rate}$$

Where: - E is energy in Wh - Tariff Rate is the cost per unit (/kWh)

Equation 4: Peak Hour Logic Check

$$\text{isPeakHour} = \begin{cases} \text{True}, & \text{if current time} \in [18:00, 22:00] \\ \text{False}, & \text{otherwise} \end{cases}$$

This boolean result is used by the backend to restrict or delay charging during expensive electricity periods.

6.6 Experimental Setup The prototype was tested in a simulated home environment under various conditions. The test environment included:

- Test Location: Indoor setup with access to 230V AC and 12V DC adapters. - Test Load: 12V DC bulb. - Test Duration: Over 10 sessions conducted across different hours of the day. - Devices Used: Laptop for backend server, smartphone for frontend dashboard access. - Monitoring: Live manual dashboard checking and Firebase log validation.

The tests also covered both peak and non-peak hours to verify cost optimization behavior.

6.7 Key Observations The outcomes from testing are summarized below:

- Voltage Range Observed: 11.6V to 12.2V (simulated). - Current Range: 0.3A to 1.2A depending on load. - Peak Hour Logic: Triggered and alerted users correctly. - Relay Switching: Completed toggle operations within 0.5 seconds. - Dashboard Sync: Displayed live data with no observable lag.

6.8 Equations Used

Equation 1: Instantaneous Power Calculation

$$P(t) = V(t) \times I(t)$$

Where: - $P(t)$ is the instantaneous power in watts - $V(t)$ is the voltage at time t - $I(t)$ is the current at time t

Equation 2: Total Energy Consumed

$$E = \sum_{t=1}^{T} P(t) \times \Delta t$$

Where: - E is total energy in watt-seconds - Δt is the time interval between readings

Equation 3: Cost Estimation

$$\text{Cost} = \frac{E}{1000} \times \text{Tariff Rate}$$

Where: - E is energy in Wh - Tariff Rate is the cost per unit (/kWh)

Equation 4: Peak Hour Logic Check

$$\text{isPeakHour} = \begin{cases} \text{True}, & \text{if current time} \in [18:00, 22:00] \\ \text{False}, & \text{otherwise} \end{cases}$$

This boolean result is used by the backend to restrict or delay charging during expensive electricity periods.

VII. RESULTS AND OUTCOMES

7.1 Introduction This section highlights the practical results gathered from testing the smart electric vehicle (EV) charging system in a controlled environment. The evaluation focused on system responsiveness, data accuracy, cost optimization, and user interface performance—each of which reflects the real-world viability of the solution.

7.2 System Responsiveness and Control During the trials, the system displayed impressive responsiveness. When users interacted with the dashboard by pressing the Start or Stop buttons, the relay on the hardware side switched states with minimal delay. In most cases, the response time was under 0.5 seconds, indicating efficient real-time communication between the frontend and the ESP32 microcontroller via the Flask backend.

7.3 Real-Time Monitoring Accuracy Sensor readings captured by the ESP32 were monitored live on the dashboard. To ensure the reliability of these values, random samples were cross-checked with manual multimeter readings. The voltage readings displayed on the dashboard closely matched, falling within a margin of ±0.3V. Data logged into Firebase also remained consistent with real-time observations, ensuring high trust in the stored analytics.

7.4 Peak-Hour Optimization Effectiveness The system's peak-hour logic, which aimed to reduce energy costs during high-tariff periods (6:00 PM to 10:00 PM), functioned as intended. During test sessions scheduled in this time range, the dashboard immediately alerted users about elevated tariff conditions. In cases where blocking was enabled, the backend successfully postponed the charging initiation, helping enforce smart usage behavior.

7.5 Energy Usage and Cost Estimation A single charging session's metrics are summarized in the table below:

Session	Voltage (V)	Current (A)	Duration (min)	Energy (Wh)	Cost ()
1	12.1	0.9	30	5.445	0.49

TABLE I
SAMPLE SESSION DATA SHOWING CALCULATED ENERGY AND COST

Energy consumption was calculated using the formula:

$$E = \sum_{t=1}^{T} P(t) \times \Delta t \quad \text{where} \quad P(t) = V(t) \times I(t)$$

and the corresponding cost using:

$$\text{Cost} = \frac{E}{1000} \times \text{Tariff Rate}$$

The system's estimated costs closely aligned with manually derived values, indicating accurate integration of analytical logic.

7.6 Dashboard Performance The web dashboard performed smoothly across multiple devices, including mobile and desktop platforms. Charts updating voltage and current readings were observed to refresh in real time, giving users a responsive and visually clear experience. No lag or sync issues were recorded, and alerts such as peak-hour notifications popped up with correct timing.

7.7 Firebase Logging Sync Firebase acted as a reliable cloud backend throughout the testing phase. Each charging session's data—voltage, current, timestamp, and calculated energy—was logged with no missing or delayed entries. Real-time data synchronization between backend and frontend remained seamless during all sessions, ensuring accurate and timely data reflections on the dashboard.

VIII. CONCLUSION AND FUTURE WORK

8.1 Conclusion This work presented the development and deployment of a smart electric vehicle (EV) charging system that leverages IoT components to deliver real-time monitoring, cloud-based data logging, and user-friendly remote control. The integration of ESP32, Python Flask, Firebase, and a React.js dashboard created a seamless end-to-end framework for interactive EV charging. Extensive testing confirmed that the system responds rapidly, accurately logs charging sessions, detects peak hours for cost optimization, and operates reliably across multiple environments.

The dashboard interface proved intuitive and mobile-friendly, while Firebase integration ensured uninterrupted data sync and historical tracking. Overall, the system demonstrated both technical feasibility and user practicality for residential, institutional, and public EV charging use cases.

IX. FUTURE WORK

While the current system is fully operational and effective for single-port home and campus applications, the design allows room for future enhancements, including:

- **Multi-Port Support**: Expanding the architecture to manage multiple EV charging ports simultaneously. - **Mobile App Integration**: Developing a cross-platform mobile application to enable on-the-go control and notifications. - **RFID Authentication**: Incorporating secure access for users through contactless RFID cards or tags. - **AI-Based Charging Schedules**: Using historical data and machine learning to recommend the most cost-effective time slots for charging. - **Billing Integration**: Adding modules to generate usage summaries and invoices for commercial or community setups.

These enhancements would transform the prototype into a full-featured, scalable EV charging solution ready for deployment in smart cities, gated communities, and institutional campuses.

X. REFERENCES

[1] Espressif Systems, "ESP32 Technical Reference Manual," [Online]. Available: https://www.espressif.com. [2] Firebase, "Firebase Realtime Database Documentation," [Online]. Available: https://firebase.google.com/docs/database. [3] Flask, "Flask Documentation," [Online]. Available: https://flask.palletsprojects.com. [4] React.js, "React Official Website," [Online]. Available: https://reactjs.org. [5] ACS712 Sensor Datasheet, Allegro Microsystems. [6] Voltage Divider Theory, All About Circuits, [Online]. Available: https://www.allaboutcircuits.com. [7] Python Serial Communication, PySerial Library Documentation. [8] IEEE, "Smart Charging Infrastructure for EVs: A Review," IEEE Transactions on Smart Grid, 2022. [9] K. Moslehi and R. Kumar, "Smart Grid—A Reliability Perspective," IEEE Trans. Smart Grid, vol. 1, no. 1, pp. 57–64, 2010. [10] D. Y. Jeong et al., "Real-time Monitoring and Control of EV Charging Systems Using Cloud Technology," Journal of Cleaner Production, vol. 189, pp. 517–529, 2018. [11] A. Ahmad, "Energy Management in Smart Grid Applications Using IoT," International Journal of Energy Research, vol. 43, pp. 915–932, 2019. [12] M. Erol-Kantarci and H. T. Mouftah, "Wireless Sensor Networks for Cost-Efficient Residential Energy Management in the Smart Grid," IEEE Transactions on Smart Grid, vol. 2, no. 2, pp. 314–325, 2011. [13] D. T. Nguyen and L. B. Le, "Optimal Bidding Strategy for Microgrids Considering Renewable Energy and Battery Storage," IEEE Transactions on Smart Grid, vol. 5, no. 4, pp. 1608–1620, 2014. [14] M. A. Al Faruque, K. Vatanparvar, "Energy Management-as-a-Service Over Fog Computing Platform," IEEE Internet of Things Journal, vol. 3, no. 2, pp. 161–169, 2016. [15] S. Gope and T. Hwang, "BSN-Care: A Secure IoT-Based Modern Healthcare System Using Body Sensor Network," IEEE Sensors Journal, vol. 16, no. 5, pp. 1368–1376, 2016. [16] R. R. Mohassel, A. Fung, F. Mohammadi, and K. Raahemifar, "A Survey on Advanced Metering Infrastructure," International Journal of Electrical Power Energy Systems, vol. 63, pp. 473–484, 2014. [17] H. Gharavi and B. Hu, "Multigate Communication Network for Smart Grid," Proceedings of the IEEE, vol. 99, no. 6, pp. 1028–1045, 2011. [18] K. Dehghanpour et al., "A Survey on State Estimation Techniques and Challenges in Smart Distribution Systems," IEEE Transactions on Smart Grid, vol. 10, no. 2, pp. 2312–2322, 2019. [19] A. Ghosh et al., "IOT-Based Smart Energy Management System: A Comprehensive Review," Sustainable Cities and Society, vol. 71, 2021. [20] M. H. Rehmani et al., "Integrating Renewable Energy Resources Into the Smart Grid: Recent Developments in Information and Communication Technologies," IEEE Transactions on Industrial Informatics, vol. 14, no. 7, pp. 2814–2825, 2018. [21] N. Javaid et al., "A Review of Secure Communication and Privacy Protocols for Internet of Things-

Based Smart Homes," Sensors, vol. 20, no. 24, 2020. [22] S. H. Almotiri et al., "A Survey of Internet of Things (IoT) in Smart Transportation," Journal of Advanced Transportation, vol. 2020. [23] R. Buyya, A. S. Yeganeh, "Fog and Edge Computing: Principles and Paradigms," Wiley, 2019. [24] M. A. Razzaque et al., "Middleware for Internet of Things: A Survey," IEEE Internet of Things Journal, vol. 3, no. 1, pp. 70–95, 2016. [25] A. Mohamed et al., "Smart Grid Commu- nications: Overview of Research Challenges, Solutions, and Standardization Activities," Computer Networks, vol. 57, no. 5, pp. 1344–1371, 2013.