WWW.IJRPR.COM

International Journal of Research Publication and Reviews

Journal Homepage: www.ijrpr.com ISSN 2582-7421

Hybrid Solar-Wind Monitoring with ESP32 and Local Energy Analytics

Rovaan KG[1], Sanjay Dhurson V[2], Sanjeevkumar P[3], Shrinivaas SB[4]

[1,2,3,4] Department of Computer Science and Engineering, Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India

ABSTRACT

Abstract This paper presents the design and implementation of a local-first hybrid renewable-energy monitoring platform developed for a small solar-wind prototype. The system combines ESP32 firmware, INA219 electrical sensing, authenticated data ingestion, SQLite persistence, a React/TypeScript dashboard, energy analytics, weather-aware baseline prediction, and an on-device energy assistant. Unlike a conventional demonstration dashboard, the imple mentation explicitly distinguishes simulator data from live hardware data and preserves unavailable measurements as null rather than treating missing values as zero. The ESP32 firmware samples configured INA219 paths every five seconds, calculates power from measured voltage and current, synchronizes time through NTP, and submits timestamped JSON readings over a trusted local network. The server validates ranges, timestamps, operating mode, duplicate records, and bearer-token authentication before storage. Energy is integrated with a trapezoidal method while gaps longer than 30 minutes are excluded. Repository inspection identified seven test suites containing 25 executable test cases covering protocol validation, analytics, persistence, API behavior, weather fallback, simulator separation, and assistant behavior. The software prototype is therefore functionally substantial, but physical sensor calibration, verified wiring, battery charging, and measured solar-wind performance remain future experimental work.

Keywords: Hybrid renewable energy; ESP32; INA219; IoT monitoring; SQLite; Energy analytics; Local AI

1. Introduction and Literature Review

The increasing use of distributed renewable-energy sources has created a need for low-cost monitoring systems that can record generation, identify unavailable channels, and present interpretable information to users [9], [10]. Small solar-wind prototypes are particularly difficult to evaluate because solar irradiance, airflow, electrical loading, battery state, and sensor availability change over time. A useful engineering platform must therefore do more than display a few instantaneous voltage values. It must preserve source identity, timestamp observations, validate incoming data, calculate derived quantities, and make the distinction between measured and simulated values visible to the user. Recent hybrid renewable-energy studies combine solar, wind, storage, and IoT monitoring to address intermittency and improve observability in distributed energy systems [1], [2], [3]. The platform described in this paper applies that general architecture to a small local-first prototype. The ESP32 platform is appropriate for this type of connected instrumentation because it integrates Wi-Fi and Bluetooth connectivity with I2C, UART, and other interfaces [4]. The INA219 is a bidirectional digital power monitor with a 26 V common-mode range, I2C/SMBus communication, programmable gain, and direct voltage, current, and power reporting [5]. These capabilities make it suitable for small prototype paths, provided that the actual shunt, board limits, polarity, isolation, and wiring are verified before operation. Existing small-scale energy dashboards often blur the boundary between demonstration data and physical mea surements. Such a design can make a prototype appear more mature than the underlying experiment. The present system takes a different approach. It assigns every reading a source mode, keeps simulator endpoints separate from live hardware endpoints, uses null values for unavailable channels, and avoids claiming system efficiency or battery state of charge when the necessary measurements are absent. This design is important because a value of zero, an estimated value, and an unavailable value have different engineering meanings. The contribution of this work is a reproducible application architecture rather than a claim of completed electrical validation. It documents the firmware-to-server protocol, the measurement and analytics pipeline, the simulator and live-mode separation, the weather integration, the local language-model assistant, and the current software validation evidence. The paper also states the remaining experimental limitations so that software functionality is not confused with hardware performance.

2. Objectives and Research Design

The main objective is to develop a monitoring and analysis platform for a small hybrid solar-wind energy proto type. The specific objectives are to: (i) acquire electrical observations from ESP32-connected INA219 sensors; (ii) transmit timestamped readings to a local server; (iii) validate and persist the readings in SQLite; (iv) calculate source-level power, energy, peaks, averages, and daily totals; (v) expose the data through a responsive dashboard; (vi) provide a simulator for repeatable demonstrations; and (vii) support local conversational assistance without sending telemetry to a hosted inference service. The research design is an implementation-oriented engineering study. The repository itself is treated as the primary artifact, and claims in this paper are derived from its firmware, TypeScript server modules, tests, configuration, and documentation. The evaluation is therefore divided into two layers. The first layer is software verification, including protocol checks, analytics tests, storage behavior, and API tests. The second layer is physical

validation, including panel and turbine characterization, INA219 calibration, battery charging, and end-to-end field measure ments. The repository contains substantial evidence for the first layer but does not yet provide a complete measured dataset for the second layer.

Evaluation layer Evidence in the project

Software verification 25 executable test cases across seven test suites; source-level

validation of protocol, persistence, calculations, APIs, weather

fallback, and assistant behavior.

Physical validation Sensor wiring, calibration, battery charging, wind generation, and

measured solar-wind performance are not claimed as complete.

Table 1: Evaluation boundary used in this paper.

3. System Architecture

The system is organized as four cooperating layers: the embedded acquisition layer, the authenticated ingestion layer, the persistence and analytics layer, and the presentation and assistance layer. The embedded layer is implemented in Arduino C++ for ESP32. The server is implemented in TypeScript and Express. SQLite stores timestamped readings, while the React client provides dashboard views, charts, simulator controls, history export, weather access, and the Energy Assistant.

Embedded Acquisition Layer

The firmware initializes two Adafruit INA219 instances at I2C addresses 0x40 and 0x41. Each sensor can be mapped to one configured path: solar, wind, battery, or load. The mapping remains unassigned until the physical wiring and address jumpers are verified. For a responding and mapped sensor, the firmware reads bus voltage and current, converts current from milliamperes to amperes, and calculates electrical power as: $P = V \times I$ where P is power in watts, V is measured bus voltage in volts, and I is measured current in amperes. Unavailable channels are transmitted with null voltage, current, and power fields. This prevents an unconnected channel from being interpreted as a zero-power source. The firmware samples at a nominal five-second interval. It first establishes Wi-Fi connectivity, then obtains a UTC timestamp after NTP synchronization. If the clock is not valid, the reading is not transmitted. Each payload contains a device identifier, ISO-8601 timestamp, live mode, firmware version, four channel objects, and INA219 readiness flags. The implementation uses plain HTTP and is explicitly intended for a trusted local development LAN, not direct Internet exposure.

Authenticated Device Protocol

The server accepts live data only when the payload mode is “live” and a bearer token of at least 32 characters is configured. Token comparison uses a timing-safe comparison. Requests are rate limited per client address. The timestamp must be ISO-8601, cannot be more than five minutes in the future, and cannot be older than seven days at ingestion. Each channel must be an object whose electrical fields are finite numbers or null. The default validation limits are 26 V for bus voltage and 3.2 A for current, although the limits can be configured to match the actual INA219 breakout and shunt. If voltage and current are both present, server-side power is derived from their product. A supplied power value is rejected when it conflicts with the derived value beyond the configured tolerance. Battery state of charge is accepted only when null or within 0–100 percent. Duplicate device-timestamp pairs are rejected by the persistence layer.

Persistence and Analytics Layer

The application stores timestamped readings in SQLite. Retention is limited to 90 days, and history queries support ranges, pagination, and source-mode filtering. The software separates simulator history from hardware history so that demonstration data cannot silently become evidence of physical generation. Energy is computed by integrating adjacent samples with the trapezoidal rule: $E \approx \sum n-1\ k=1\ (\ Pk-1+Pk\ 2\ )\Delta tk$ where Δtk is the elapsed time in hours and E is reported in watt-hours. Intervals with non-positive duration, missing power, or a gap greater than 0.5 hours are skipped. For daily summaries, intervals crossing a day boundary are split rather than assigned entirely to one date. This is preferable to treating missing intervals as zero consumption, which would systematically understate energy. The server exposes current status, current readings, history, analytics, daily generation, alerts, baseline prediction, and weather proxy endpoints. System state is classified as surplus, deficit, balanced, or unavailable using generation and load when the required values exist. The code deliberately does not label renewable coverage as system efficiency. It also does not infer battery state of charge from voltage, wind speed from generator voltage, or renewable energy delivered to load without suitable measurements and calibration.

Dashboard and Local Assistant

The React/TypeScript dashboard presents four task-focused areas: dashboard, simulator, climate and output predictor, and daily generation history. Simulator controls modify synthetic irradiance, airflow, load demand, and battery state. The user interface labels simulator values clearly and keeps the dashboard empty when live hardware is disconnected. The weather view requests forecast data from Open-Meteo after location permission or manual coordinates are provided. Open-Meteo provides hourly and daily forecast variables through a coordinate-based API and supports forecasts of up to 16 days [6]. In this project the weather response is used with a transparent time-of-day and persistence baseline; it is not presented as a trained climate or machine-learning generation model. The Energy Assistant uses the ONNX-compatible Qwen2.5-0.5B-Instruct model through Transformers.js.

Transformers.js runs supported transformer models in web environments through ONNX Runtime [7], and the referenced model is published with ONNX weights for Transformers.js use [8]. The assistant receives clear simulator/live labels and preserves missing telemetry fields. It is a local convenience feature, not an evaluated forecasting model.

4. Implementation and Research Methodology

The implementation follows a source-to-dashboard pipeline. At the device side, the firmware constructs a JSON document every five seconds. The server validates the document before converting external field names into internal TypeScript structures. The normalized reading then becomes the input to persistence, summaries, alerts, and charts. The same internal representation is used for hardware and simulator data, but a mode field remains attached throughout the pipeline. The simulator is not a cosmetic front-end mock. It generates prototype-scale values from irradiance, wind speed, load demand, weather scenario, and battery-level controls. It also supports explicit fault injection for solar, wind, battery, load, or all channels. Fault injection changes the affected channel to unavailable rather than silently substituting a zero. This enables demonstrations of missing-data behavior, delayed streams, battery alerts, and generation deficits. The software stack uses React 19, TypeScript, Express, SQLite through Node's built-in sqlite interface, Drizzle configuration, Vitest tests, Vite, and pnpm scripts for checking, testing, building, and starting the application. The repository provides separate firmware and simulated-device examples, an environment template, database backup script, schema files, and test modules. The seven test suites are named for authentication logout, device protocol, energy semantics, energy API, energy mathematics, energy storage, and weather behavior. Component Technology or interface Role Embedded controller ESP32, Arduino C++ Wi-Fi, NTP, I2C acquisition, HTTP telemetry Electrical sensing Two INA219 devices Bus voltage, current, and power on mapped paths Backend TypeScript, Express Validation, APIs, alerts, analytics, weather proxy Persistence SQLite Timestamped readings, duplicate pro tection, retention Frontend React, TypeScript, Vite Dashboard, simulator, charts, exports Assistant Transformers.js and ONNX Qwen model Local text generation using project context Table 2: Principal implementation components.

5. Results and Discussion

The repository provides measurable software-validation evidence. Seven test files contain 25 executable test cases. The cases cover calculation of power only when voltage and current are available, trapezoidal integration, long-gap rejection, daily-boundary splitting, runtime and unit conversion, forecast metrics, duplicate persistence, malformed and out-of-range payloads, future timestamps, invalid range configuration, authenticated ingestion, simulator/live separation, history ranges, weather coordinates, daily mode separation, injected sensor faults, atomic simulator configuration, status labeling, local assistant responses, conversation context, connection ques tions, and invalid chat inputs. The validation design produces several practical results. First, it prevents missing sensor channels from becoming false zero readings. Second, it prevents a simulator endpoint from switching the application into live hardware mode. Third, it rejects inconsistent power values when the submitted value disagrees with voltage multiplied by current. Fourth, it makes duplicate device timestamps idempotent at the storage boundary. Fifth, it separates the retained histories of simulation and hardware modes. Sixth, it skips large temporal gaps during energy integration instead of fabricating unobserved generation. The software's most important result is therefore semantic reliability. A dashboard reading is not treated as a physical fact solely because it is displayed. The source label, sensor status, timestamp, and null fields remain available for interpretation. This is valuable for student prototypes because early demonstrations commonly include disconnected sensors, incomplete wiring, synthetic data, or partially implemented battery paths. The current evidence does not justify a numerical claim for solar efficiency, wind output, battery runtime, or prediction accuracy. The repository documents the need for panel and turbine calibration, sensor mapping, battery protection verification, experiment logging, and time-based forecast evaluation. The correct conclusion is that the platform is software-complete enough to support those experiments, not that the experiments have already been completed.

6. Safety, Limitations, and Future Scope

The INA219 measures only the electrical path through which its shunt is wired. It does not measure wind speed or solar irradiance, and two INA219 devices cannot simultaneously provide four independent physical paths. The actual sensor mapping must therefore be verified before assigning sensors to solar, wind, battery, or load channels. The battery path has an additional safety boundary. A nominal 3.7 V, 2400 mAh lithium-ion cell has a theoretical nominal energy of approximately 8.88 Wh, but usable energy depends on cell condition, protection, charging limits, conversion losses, and load behavior. A TP4056 module is a single-cell charger, while an MT3608 is a boost converter; neither statement alone proves that a complete solar-wind charging system is safe. Unregulated generation must not be connected directly to a lithium-ion cell. The current application performs no charging, switching, or load-control operation. The firmware uses plain HTTP and is intended for a trusted local LAN. A production deployment would require HTTPS, device enrollment, credential rotation, access control, secure network configuration, and hardware- specific calibration. The repository also lacks user/account authentication, a calibration wizard, validated coulomb counting, relay outputs, verified wiring records, and a trained or evaluated generation model. Future work should first establish a measurement campaign with calibrated voltage and current references. It should then characterize the solar panel and wind generator under controlled irradiance, airflow, and load condi tions. The battery subsystem should be redesigned and tested with an appropriate charge controller, protection, fuse, isolation, and temperature monitoring. Later software work should add experiment metadata, calibration profiles, forecast holdout evaluation, persistent user settings, device key rotation, and deployment-grade security.

7. Conclusion

This paper analyzed and documented a repository-based hybrid solar-wind monitoring platform that combines ESP32 firmware, INA219 sensing, authenticated telemetry, SQLite persistence, a React dashboard, energy analytics, weather-aware baseline prediction, and a local language-model assistant. The design's central strength is explicit data semantics: simulator and live hardware modes are separated, unavailable channels are preserved as null, calculated values are derived only from available measurements, and energy integration excludes unsupported time gaps. The repository contains seven test suites and 25 executable test cases that provide meaningful software evidence for protocol validation, analytics, storage, APIs, simulator behavior, weather fallback, and assistant behavior. At the same time, the project does not yet provide sufficient physical measurements to claim validated hybrid gener ation, battery charging performance, wind conversion efficiency, or forecast accuracy. The work should therefore be presented as a functional monitoring and analytics platform that is ready to support controlled hardware experiments. Completing calibration, safe energy storage, field measurement, and forecast evaluation will convert the current engineering prototype into a fully evidenced renewable-energy research system.

Acknowledgement

The authors acknowledge Sri Shakthi Institute of Engineering and Technology, Coimbatore, Tamil Nadu, India, for providing the academic context for this project.

References

[1] A. H. Bagdadee, M. S. Rahman, I. Al Mamoon, D. A. Dewi, A. K. M. Islam, and L. Zhang, "Empowering smart homes by IoT-driven hybrid renewable energy integration for enhanced efficiency," Scientific Reports, vol. 15, art. no. 41491, 2025, doi: 10.1038/s41598-025-25328-2. [

2] W. He, M. J. A. Baig, and M. T. Iqbal, "An open-source supervisory control and data acquisition architecture for photovoltaic system monitoring using ESP32, Banana Pi M4, and Node-RED," Energies, vol. 17, no. 10, art. no. 2295, 2024, doi: 10.3390/en17102295.

[3] S. A. Sarkar, A. D. Nimbalkar, S. K. More, O. S. Kesarkar, and A. S. Patil, "Design of solar panel monitoring system using ESP32 and IoT," in 2024 IEEE International Conference on Smart Power Control and Renew able Energy (ICSPCRE), 2024, doi: 10.1109/ICSPCRE62303.2024.10675117.

[4] Espressif Systems, "ESP32: A feature-rich MCU with integrated Wi-Fi and Bluetooth connectivity," Espressif Systems. Available: https://www.espressif.com/en/products/socs/esp32

[5] Texas Instruments, "INA219 Zerø-Drift, Bidirectional Current/Power Monitor With I2C Interface," datasheet, Rev. G. Available: https://www.ti.com/lit/gpn/INA219

[6] Open-Meteo, "Weather Forecast API documentation," Open-Meteo. Available: https://open-meteo.com/en/ docs

[7] Hugging Face, "Transformers.js: State-of-the-art machine learning for the web," documentation. Available: https://huggingface.co/docs/transformers.js/index

[8] ONNX Community, "Qwen2.5-0.5B-Instruct," Hugging Face model repository. Available: https:// huggingface.co/onnx-community/Qwen2.5-0.5B-Instruct

[9] S. A. Kalogirou, Solar Energy Engineering: Processes and Systems, 2nd ed. Academic Press, 2014.

[10] G. M. Masters, Renewable and Efficient Electric Power Systems, 2nd ed. Wiley, 2013.