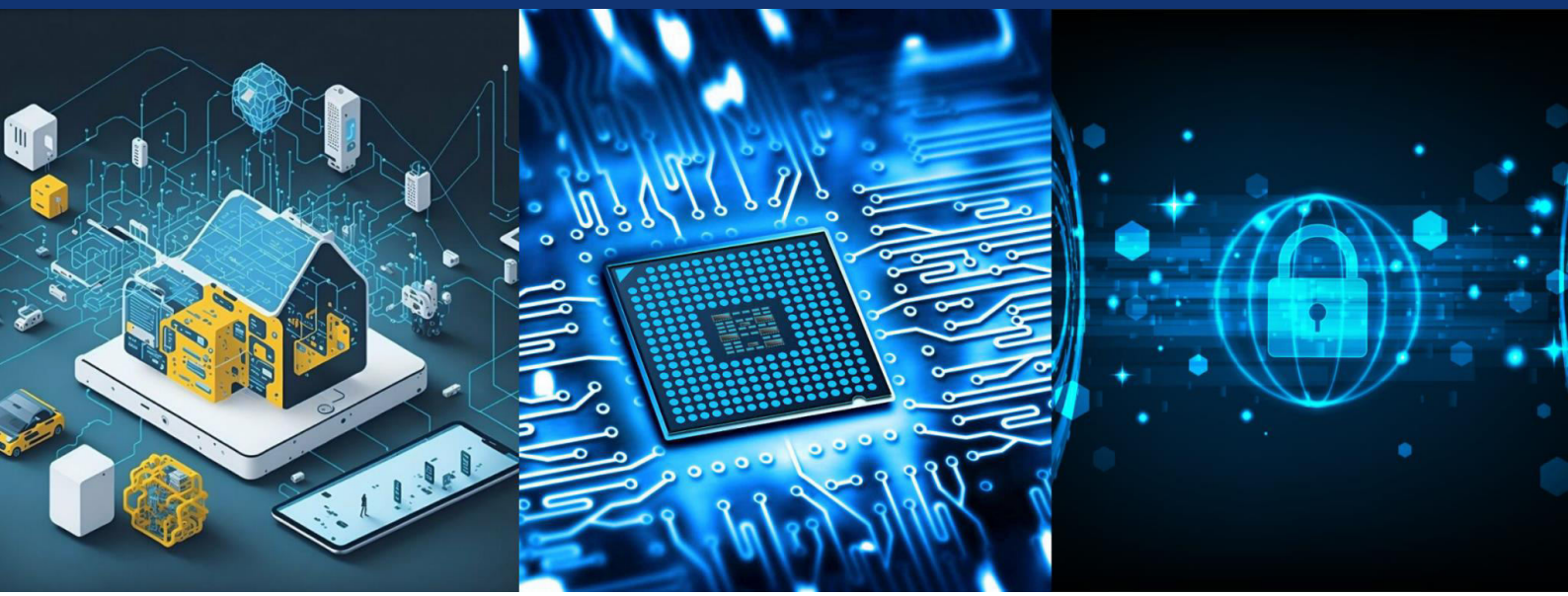
IJIRCCE

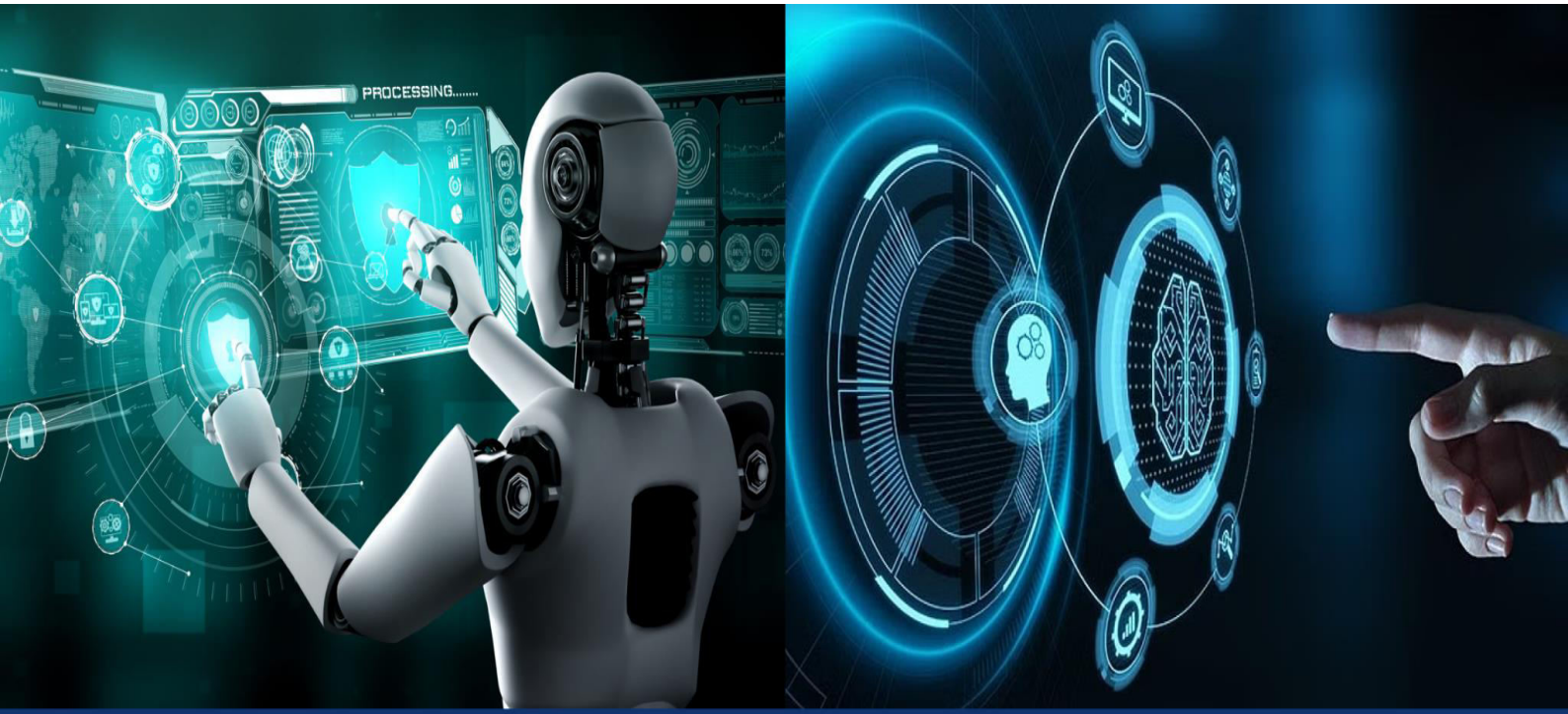
ISSN(O): 2320-9801

ISSN(P): 2320-9798



International Journal of Innovative Research in Computer and Communication Engineering

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)



Impact Factor: 9.927

Volume 14, Issue 10, October 2026

© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410044

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

AI-Driven Design of Advanced Smart ICU Monitoring with Early Sepsis Risk Analysis

Diwakar Reddy[1], Divyam Singh[1], Haripriya MP[1], Shivani RK[1], Dr. M Selvi[2]

UG Students, Dept. of Computer Science and Design, Rajarajeshwari College of Engineering, Bengaluru, Karnataka, India[1]

HOD, Dept. of Computer Science and Design, Rajarajeshwari College of Engineering, Bengaluru, Karnataka, India[2]

ABSTRACT: This paper represents the design and development of a prototype real-time intensive care unit (ICU) and monitoring system that combines sensor-based with vital-sign acquisition, a machine-learning-based sepsis risk assessment module, the web dashboard and locally hosted language-model assistance. The ESP32 microcontroller interact with a MAX30102 pulse-oximetry sensor, the MLX90614 infrared temperature sensor and the MPU6050 motion sensor. The firmware samples and processes sensor readings, display status and measurements on an I2C LCD, and transmits structured JSON data to a Python application over a serial connection. Then Python backend uses Flask and Flask-SocketIO to receive the stream, derive the estimated blood-pressure value using a programmed heuristic, run the Random Forest classifier and broadcast the resulting measurements and risk category to the web interface. The model-training pipeline is configured to uses the PhysioNet 2019 Sepsis Challenge patient-record files and uses heart rate, systolic blood pressure, oxygen saturation and temperature as input features. The web application provides live monitoring, trend history, patient-contextual AI chat, medication-safety assessment, AI-generated summaries and structured shift-handover reports through a locally hosted Ollama model. The prototype also supports Firebase Firestore logging and a buzzer-based alert path for critical classifications. The uploaded implementation establishes the software and hardware workflow; however, no verified model-training output, measured predictive-performance results or clinical validation data were supplied for this manuscript. The system is therefore presented as a development prototype and not as a clinically validated diagnostic or treatment device.

KEYWORDS: ICU monitoring, Internet of Things, ESP32, vital-sign monitoring, sepsis risk assessment, Random Forest, Flask, Firebase, edge-connected healthcare, local large language model

I. INTRODUCTION

Intensive care units require continuous observation of patients because changes in vital signs can occur between routine clinical assessments. Heart rate, oxygen saturation and body temperature are among the measurements that help clinicians follow a patient's condition. When these readings are collected from separate devices or reviewed manually, it can be difficult to view the current measurements, their recent trends and the associated alerts in one place.

The project described in this paper addresses this problem by developing a connected monitoring prototype. The system combines an ESP32-based sensor unit with a Python backend and a browser-based dashboard. The hardware collects physiological and motion-related readings, while the backend processes the incoming data and runs a machine-learning model that produces a sepsis-risk estimate. The dashboard presents the incoming measurements and risk information, and additional AI-assisted modules support patient-contextual queries, medication-safety review, summaries and shift handover.

The system is designed as a monitoring and decision-support prototype. It is not intended to replace bedside medical equipment, clinical judgement or established sepsis screening procedures. In particular, the current implementation estimates blood pressure from a programmed relationship involving heart rate and oxygen saturation; it does not measure blood pressure using a dedicated cuff or validated cuffless instrument.

The main contributions of the project are:

- Development of an ESP32-based sensor unit using MAX30102, MLX90614 and MPU6050 modules.
- Serial transmission of structured readings from the microcontroller to a Python backend.

© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410044

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

- Integration of a Random Forest sepsis-risk classifier with a preprocessing pipeline.
- Real-time browser dashboard updates using Flask-SocketIO and a live data stream.
- Firebase Firestore logging and patient-oriented web modules for summaries, safety review and shift handover.
- Integration of a locally hosted Ollama language model for selected text-based clinical-support functions.

II. SYSTEM OVERVIEW AND ARCHITECTURE

The prototype is organised into four connected layers: the sensor and embedded layer, the communication and processing layer, the data-storage layer, and the web and AI-support layer. The ESP32 is responsible for acquiring readings and sending them through its USB serial interface. The Python application receives the serial stream, prepares the data, performs risk inference and emits a real-time event. The browser application receives the event and updates the monitoring interface.

The embedded unit communicates with the backend at 115200 baud. The firmware sends a JSON record approximately once per second. The fields include heart rate (hr), oxygen saturation (spo2), temperature (temp), and two gyroscope values (gyroX and gyroY). The Python backend adds estimated systolic and diastolic blood-pressure fields, a sepsis-risk score, a risk category and a short assessment before broadcasting the updated record.

The Python backend is implemented using Flask, Flask-CORS and Flask-SocketIO. It attempts to identify the ESP32 serial port from common USB-serial adapter descriptions and uses COM3 as a fallback when automatic detection fails. The backend also initialises Firebase Admin for Firestore access and loads the saved classifier and preprocessing pipeline from the models directory.

The browser application is divided into separate HTML pages and JavaScript modules. The dashboard module receives the live Socket.IO event, updates the displayed vital signs and records selected readings in browser local storage. Other modules provide AI chat, medication-safety assessment, summaries and shift-handover generation. Firebase's browser SDK is used by the frontend for authentication and patient-record access, while the Python backend uses the Firebase Admin SDK for server-side logging.

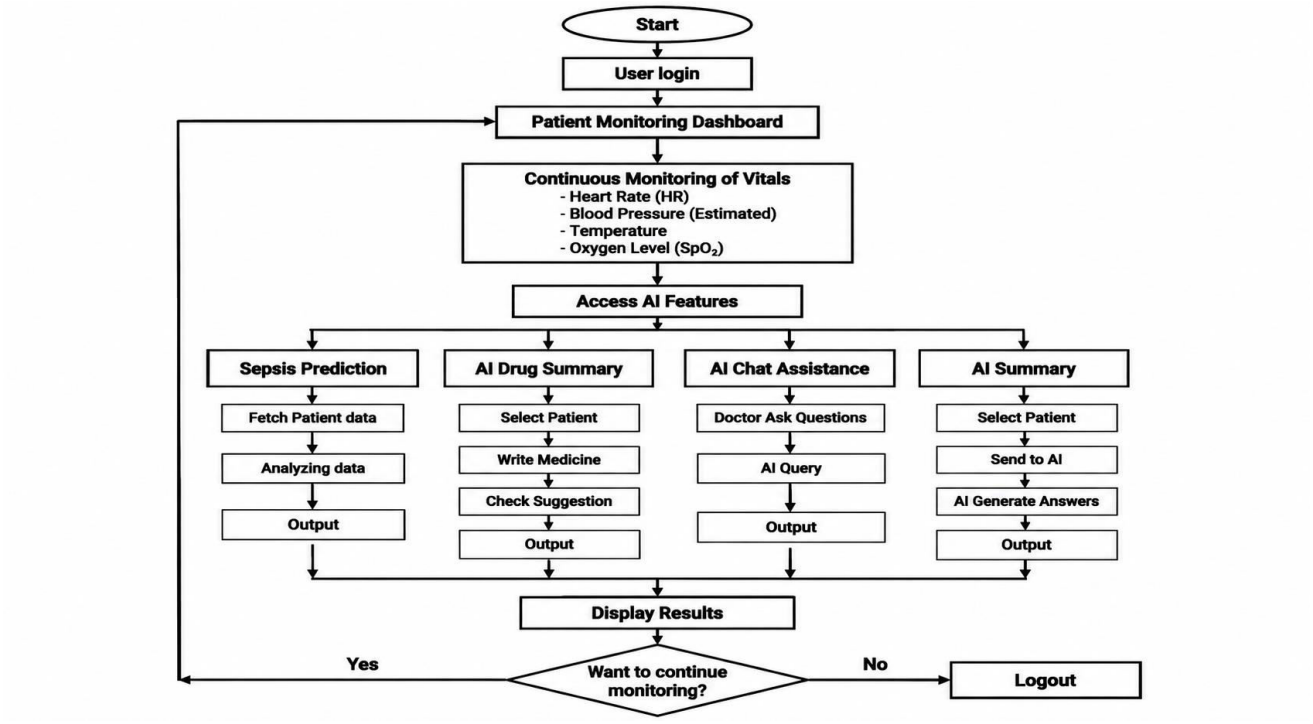


Figure 1 Architecture

© 2026 IJIRCCE | Volume 14, Issue 10, October 2026 DOI: 10.15680/IJIRCCE.2026.1410044

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

III. HARDWARE DESIGN AND DATA ACQUISITION

A. ESP32 microcontroller

The ESP32 DevKit V1 acts as the embedded controller. It coordinates the sensor interfaces, updates the LCD and sends the measurements to the host computer. The firmware uses the I2C bus with SDA on GPIO 21 and SCL on GPIO 22. Serial communication is configured at 115200 baud.

B. MAX30102 pulse-oximetry sensor

The MAX30102 is used for pulse-related measurement. The firmware reads the red and infrared channels, maintains running DC estimates and tracks signal maxima and minima over a measurement interval. A ratio-based calculation is used to obtain an SpO2 estimate. Heart beats are detected from the infrared signal using the heartRate library, and valid beat intervals are used to calculate beats per minute. A rolling buffer of up to eight readings is used to calculate the displayed average heart rate. The firmware also applies signal checks and rejects selected implausible or unstable readings.

The firmware waits for finger placement and a warm-up interval before transmitting a valid heart-rate record. When a finger is not detected or the signal is still being acquired, the firmware sends a negative heart-rate value to indicate an unavailable reading. These states are important because an unavailable sensor reading should not be interpreted as a confirmed patient event.

C. MLX90614 infrared temperature sensor

The MLX90614 infrared sensor is used to obtain an object-temperature reading. The firmware applies a programmed adjustment to readings above 20 °C and then applies an exponential smoothing step, using 95 percent of the previous filtered value and 5 percent of the new reading. The adjusted value is transmitted as the temperature field. This is the implemented prototype calculation; the uploaded code does not provide clinical calibration data establishing that the resulting value is equivalent to a core body temperature.

D. MPU6050 motion sensor

The MPU6050 is initialised through its I2C address and configured for the ±2 g accelerometer range. The current firmware reads gyroscope X and Y registers, converts the raw values using the programmed scale factor and applies a small-value dead-zone filter. Peak values are retained during the one-second reporting interval and included in the JSON record. The firmware uses the motion values in the LCD status display when the absolute value of either axis exceeds the programmed threshold.

E. Local indicators and display

A 16×2 I2C LCD is connected at address 0x27 through the LcdDisplay module. A green LED indicates the monitoring state, while a red LED is used with the buzzer during an alert. The passive buzzer is connected to GPIO 18. When the backend sends the CRITICAL_ALERT command over serial, the firmware runs a non-blocking buzzer sequence consisting of five short beeps. The non-blocking implementation allows the sensor-reading loop to continue while the alert sequence is active.

IV. SOFTWARE DESIGN AND IMPLEMENTATION

A. Serial ingestion and real-time communication

The Flask application runs a background serial-reader thread. It reads incoming lines, checks that a line begins with a JSON object, parses the record and extracts the available vital fields. The backend then computes the estimated blood-pressure values when a positive heart-rate value is available. The enriched record is emitted to connected browser clients using the Socket.IO event named vital_update.

The dashboard JavaScript listens for vital_update. It copies the incoming heart rate, SpO2, temperature, motion values, risk score, risk category and assessment into the dashboard state. It refreshes the visible interface and dispatches a vitalsUpdated browser event so that the sepsis-risk display module can update its presentation. The dashboard also contains a separate hardware-simulation mode that generates synthetic values for interface demonstration. Simulation readings must be distinguished from live sensor readings during testing and reporting.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

B. Blood-pressure estimation

The backend does not receive blood-pressure measurements from a dedicated blood-pressure sensor. Instead, calculate_cuffless_bp() estimates systolic and diastolic values from heart rate, with an additional adjustment based on low SpO2 and a small time-varying component. The estimate is bounded within programmed limits. These values are subsequently used as the SBP input to the machine-learning pipeline and are displayed by the dashboard.

This distinction is important for interpreting the system. The calculated values are algorithmic estimates generated by the current code, not measured blood pressure. The uploaded project files do not include a calibration study, reference-cuff comparison or validation dataset for this estimation method. Consequently, the manuscript does not claim clinical accuracy for the estimated blood pressure.

C. Machine-learning model

The model-training script loads a compiled CSV file when available. If the compiled file is not present, it searches the training subdirectory for PhysioNet-format PSV files, reads them and combines their rows into a single dataset. The selected input features are HR, SBP, O2Sat and Temp, while SepsisLabel is used as the target. A preprocessing pipeline applies median imputation to missing values and standard scaling to the input features.

The classifier is a RandomForestClassifier configured with 100 trees, a fixed random state of 42, parallel execution and balanced class weights. The script creates an 80:20 train-test split and prints the test-set score after fitting the model. The trained classifier and preprocessing pipeline are saved as sepsis_model.pkl and preprocessing_pipeline.pkl in the models directory. At runtime, the backend loads both files, transforms each incoming feature record through the saved preprocessing pipeline and obtains the positive-class probability using predict_proba().

The backend converts the model probability into a percentage and assigns a category using the thresholds present in the code: values greater than 75 percent are labelled Critical, values greater than 40 percent are labelled Moderate, and the remaining values are labelled Low. A text assessment is also assembled from programmed vital-sign checks, including high heart rate, low SpO2, low systolic pressure and elevated temperature.

The project includes a script that downloads up to 30,000 PSV files from the PhysioNet Open Data S3 bucket and a separate script that generates a synthetic CSV containing balanced example classes. The training script uses an existing compiled CSV before searching for raw PSV files. Therefore, the actual source and size of the dataset used for any particular model run depend on which files were present when training was performed. No dataset snapshot, trained model binary or training log was included with the uploaded source files, so this paper does not report a verified sample count or model accuracy.

D. Firebase logging

The backend uses the Firebase Admin SDK to connect to Cloud Firestore. It writes records under the patients collection, using the configured bed node and a vitals_log subcollection. Each stored record contains a server timestamp, heart rate, SpO2, temperature, sepsis-risk score, risk category and a type field distinguishing a critical event from a stable interval log. The backend accumulates readings and is configured to write an averaged stable record at 30-minute intervals, while critical classifications are logged immediately. In the uploaded backend code, the target bed node is currently set to bed-1.

The frontend uses Firebase for user authentication and patient-record operations. It also uses browser local storage to retain a rolling vital-sign history for dashboard and AI modules. The dashboard code stores a reading every ten live update ticks and retains up to 720 entries for a bed. The code comments describe this as approximately two hours of history, based on the one-second incoming stream.

E. Local language-model modules

The project includes AI chat, medication-safety review, AI summary and shift-handover modules. These modules send prompts to a locally hosted Ollama API at localhost:11434. The configured primary model name is llama3:8b-instruct-q3_K_M, with a fallback to llama3 when the requested model is unavailable. The shift-handover module requests a structured report using Situation, Background, Assessment and Recommendation headings. The AI summary module prepares patient information and compressed vital trends as context. The medication-safety module prompts the

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

language model to assess whether a listed drug is both safe and indicated based on the patient information supplied to it.

The local language model is separate from the Random Forest inference path. The sepsis probability is calculated in the Python backend and passed to the browser; the AI modules use the language model to generate text-based assistance from the available patient context. The presence of these modules does not establish that their generated medical suggestions are clinically correct. Their outputs require review by qualified clinical staff.

V. SYSTEM WORKFLOW AND ALGORITHM

A. Operational workflow

- The ESP32 initialises the I2C bus, LCD and sensor modules.
- The MAX30102, MLX90614 and MPU6050 are read and processed according to the firmware timing and filtering logic.
- Approximately once per second, the ESP32 sends a JSON record through the serial connection.
- The Python background thread parses the record and distinguishes unavailable or initializing readings from positive heart-rate readings.
- For a positive heart-rate reading, the backend calculates estimated systolic and diastolic blood pressure using the programmed heuristic.
- The saved preprocessing pipeline transforms HR, estimated SBP, SpO2 and temperature before the Random Forest model calculates a risk probability.
- The backend assigns Low, Moderate or Critical risk according to its programmed probability thresholds and creates a text assessment.
- The enriched record is broadcast to the web dashboard through the vital_update Socket.IO event.
- Eligible records are stored in the browser history and selected stable or critical records are logged to Firestore.
- When the risk category is Critical, the backend sends CRITICAL_ALERT to the ESP32, which activates the buzzer and red LED.

B. Risk inference formulation

Let the incoming feature vector be represented as x = [HR, SBP, O2Sat, Temp]. The saved preprocessing pipeline applies median imputation and standardisation before the vector is passed to the Random Forest classifier. The positive-class probability returned by the classifier is converted to a percentage:

$$Risk\ (\%) = 100 \times P(SepsisLabel = 1 \mid x)$$

The risk category is then assigned using the backend thresholds:

- Critical, when Risk (%) > 75.
- Moderate, when 40 < Risk (%) ≤ 75.
- Low, when Risk (%) ≤ 40.

These are implementation thresholds in the uploaded code. They should not be described as clinically validated decision boundaries. The system's assessment text also checks for selected vital-sign patterns to provide a short explanation alongside the model score.

VI. RESULTS AND DISCUSSION

The uploaded source code demonstrates an implemented end-to-end software and hardware workflow: sensor acquisition on the ESP32, serial data transfer, Python-side processing, machine-learning inference, real-time dashboard updates, Firestore logging and a serial return path for a critical buzzer alert. The frontend also contains patient-oriented tools for AI chat, medication-safety review, summaries and shift-handover generation using a local Ollama service.

The firmware contains sensor initialisation checks, a finger-detection and warm-up path, heart-rate averaging, SpO2 estimation, temperature smoothing, motion filtering, LCD output and non-blocking alert signalling. The backend includes serial-port discovery, model loading, preprocessing, probability-based classification, event broadcasting and periodic or critical-event logging. The dashboard supports both live-stream and simulation modes, and stores a bounded history in local storage.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

The available files are source code and configuration files rather than a complete experimental record. They do not include a verified dataset used for the final training run, a saved model artifact, a test-set report, confusion matrix, sensitivity, specificity, precision, recall, F1-score, ROC-AUC, latency measurements or a documented clinical trial. It is therefore not possible to make a defensible quantitative claim about predictive performance from the uploaded materials alone.

Several implementation limitations should be considered before clinical use. First, the blood-pressure values are estimated from a heuristic and have not been validated against a reference device. Second, the temperature adjustment and SpO2 calculation are firmware-level prototype calculations, and the uploaded files do not include calibration evidence. Third, the training script splits rows after combining patient records; it does not show a patient-wise separation strategy. Its preprocessing pipeline is fitted before the train-test split, so the evaluation procedure may expose the test data to information from preprocessing. Fourth, the synthetic-data generator creates class-separated example values and should only be used for software pipeline testing, not as evidence of clinical performance.

The firmware's no-finger and warm-up states are represented using a negative heart-rate value. The Python backend also contains a separate handler for a zero heart-rate state, but the current firmware's ordinary unavailable-reading path sends a negative value. The paper therefore describes the implemented unavailable-reading handling and does not claim a validated asystole-detection function.

The prototype should be evaluated in a controlled, non-clinical test environment before any clinical study. Further work should include sensor calibration against reference instruments, patient-wise dataset splitting, independent test data, class-imbalance analysis, threshold evaluation, confusion-matrix reporting, measurement of end-to-end latency and review of AI-generated text by qualified clinicians. A formal clinical validation study and regulatory assessment would be required before considering clinical deployment.

VII. CONCLUSION AND FUTURE WORK

This paper presented the design and implementation of a real-time ICU monitoring prototype that connects an ESP32 sensor unit to a Python backend, a Random Forest sepsis-risk module, Firebase Firestore and a browser-based monitoring application. The hardware reads pulse-related, temperature and motion signals, displays local status information and transmits structured readings to the backend. The backend processes the stream, estimates a sepsis-risk probability and broadcasts the resulting information to the dashboard. Critical classifications can trigger a buzzer response through a serial command.

The web application extends the monitoring workflow with patient records, live trends, local language-model chat, medication-safety review, AI summaries and SBAR-based shift handover. The implementation provides a foundation for integrating embedded sensing, machine-learning inference and clinician-facing software in one prototype. However, the present work is a development-stage system. The estimated blood pressure, sensor calculations and machine-learning output have not been established as clinically validated measurements or predictions.

Future work will focus on validating the sensor readings against reference equipment, improving the blood-pressure measurement approach, refining the model-training and evaluation methodology, reporting standard classification metrics, testing performance under continuous operation and assessing the reliability and safety of the local language-model features. Until those steps are completed, the system should be used only for development and demonstration, not for autonomous clinical decisions.

REFERENCES

[1] PhysioNet, “2019 Sepsis Prediction Challenge,” dataset referenced by the project training and download scripts. Available: https://physionet.org/content/challenge-2019/1.0.0/

[2] Project implementation files: ESP32 firmware (sepsis_monitor.ino), Python backend (app.py), model-training pipeline (train_model.py), and frontend JavaScript modules. Internal project source code, supplied for preparation of this manuscript.

www.ijircce.com | e-ISSN: 2320-9801, p-ISSN: 2320-9798| Impact Factor: 9.927| ESTD Year: 2013|

International Journal of Innovative Research in Computer and Communication Engineering (IJIRCCE)

(A Monthly, Peer Reviewed, Refereed, Scholarly Indexed, Open Access Journal)

[3] M. A. Reyna, C. S. Josef, R. Jeter, S. P. Shashikumar, M. B. Westover, S. Nemati, G. D. Clifford, and A. Sharma, "Early Prediction of Sepsis From Clinical Data: The PhysioNet/Computing in Cardiology Challenge 2019," Critical Care Medicine,
[4] L. Breiman, "Random Forests," Machine Learning,
[5] Analog Devices, "MAX30102: High-Sensitivity Pulse Oximeter and Heart-Rate Sensor for Wearable Health," Datasheet,
[6] Melexis, "Datasheet for MLX90614: Single and Dual Zone Infrared Thermometer," Datasheet.
[7] InvenSense Inc., "MPU-6000 and MPU-6050 Product Specification,"
[8] Espressif Systems, "ESP32 Technical Reference Manual," Espressif Systems.
[9] Pallets Projects, "Flask Documentation."
[10] Flask-SocketIO Contributors, "Flask-SocketIO Documentation."
[11] Google, "Cloud Firestore Documentation
[12] Chart.js Contributors, "Chart.js Documentation."



INTERNATIONAL STANDARD SERIAL NUMBER INDIA

INNO SPACE SJIF Scientific Journal Impact Factor

doi crossref

निस्केयर NISCAIR

INTERNATIONAL JOURNAL OF INNOVATIVE RESEARCH

IN COMPUTER & COMMUNICATION ENGINEERING

9940 572 462 6381 907 438 ijircce@gmail.com

www.ijircce.com

Scan to save the contact details