IJISEM
INTERNATIONAL JOURNAL OF INNOVATIONS IN SCIENCE ENGINEERING AND MANAGEMENT

International Journal of Innovations In Science Engineering And Management

OPEN ACCESS

Volume: 5

Issue: 2

Month: June

Year: 2026

ISSN: 2583-7117

Published: 02.06.2026

Citation:

Devya, Shib Sankar Das "Design and Implementation of a Low-Cost, High-Density IIoT Gateway for Smart Factory Monitoring Using ESP32" International Journal of Innovations in Science Engineering and Management, vol. 5, no. 2, 2026, pp. 364-375

DOI:

10.69968/ijisem.2026v5i2364-375

Design and Implementation of a Low-Cost, High-Density IIoT Gateway for Smart Factory Monitoring Using ESP32

Devya[1], Shib Sankar Das[2]

[1]*Department of Automation and Robotics Engineering, Prestige Institute of Engineering Management and Research, Indore, India*

divyaaprajapati04@gmail.com

[2]*Department of Computer Science and Engineering, Techno Bengal Institute of Technology, Kolkata , India*

shibsankardas721@gmail.com

Abstract

Affordable real-time machine monitoring remains an unsolved operational challenge for small and medium-scale enterprises (MSMEs), where the capital cost of Programmable Logic Controllers (PLCs) and proprietary industrial gateways renders factory-floor visibility economically inaccessible. This paper presents a low-cost IIoT edge gateway engineered on the ESP32 microcontroller, delivering production-grade machine monitoring without the financial burden of conventional infrastructure.

The central contribution is a firmware architecture that scales a single ESP32 — equipped with only 320 KB of Static RAM (SRAM) — to simultaneously monitor and control 512 independent machine nodes (M1–M512) through a live web dashboard. This was achieved through an optimized memory-mapping strategy combined with a purpose-built Chunked Transfer Encoding mechanism over a native HTTP server, which streams dashboard payloads in bounded segments rather than constructing full responses in memory, preventing heap exhaustion under high-density node loads.

Operational continuity in electromagnetically harsh factory environments is maintained by a "Hybrid Scan & Auto-Connect" firmware routine that periodically evaluates Received Signal Strength Indication (RSSI) across a prioritized list of saved network credentials and autonomously switches to the strongest available connection, eliminating monitoring gaps caused by localized interference or access-point failures. Machine states and user-defined node labels are committed to persistent flash storage via the ESP32's Non-Volatile Storage (NVS) Preferences library, ensuring complete context restoration following abrupt power interruptions.

The proposed system demonstrates that resource-constrained microcontrollers, when paired with thoughtfully optimized firmware strategies, are fully capable of sustaining the monitoring demands of industrial-scale deployments. By eliminating dependence on expensive proprietary hardware, this gateway offers MSMEs a practical, scalable, and immediately deployable foundation for real-time factory-floor visibility within Industry 4.0 environments.

Keywords; Industry 4.0, IIoT Gateway, ESP32, Embedded Firmware, High-Density Monitoring, Resource-Constrained Microcontrollers, Chunked Transfer Encoding, Network Resilience, Data Persistence, Smart Factory.

INTRODUCTION

The rapid proliferation of Internet of Things (IoT) technologies has fundamentally restructured the operational landscape of modern manufacturing, giving rise to what is broadly termed Industry 4.0 — an era defined by cyber-physical systems, intelligent automation, and data-driven decision making on the factory floor. At the heart of this transformation lies a deceptively simple requirement: every machine must be visible, and every anomaly must be caught before it becomes a failure.

Yet for the vast majority of the world's manufacturers, this visibility remains out of reach.

Small and medium-scale enterprises (MSMEs) constitute the backbone of industrial economies globally, accounting for over 90% of all businesses and a significant share of manufacturing output in developing nations including India.

Despite their economic weight, these enterprises operate under severe resource constraints that make conventional machine monitoring infrastructure financially inaccessible. Industrial-grade Programmable Logic Controllers (PLCs) from established vendors such as Siemens, Allen-Bradley, or Mitsubishi carry per-unit costs ranging from several hundred to several thousand US dollars, and that figure excludes installation, licensing, and integration expenses. Proprietary SCADA gateways and commercial IIoT platforms compound this burden further through subscription fees and vendor lock-in. For a small factory running on thin margins, deploying such infrastructure across an entire production line is not a calculated risk — it is simply not an option.

The consequence is reactive maintenance culture. Without real-time machine-state visibility, operators rely on manual inspection rounds, worker-reported failures, and post-breakdown diagnostics. Downtime accumulates silently. Production targets are missed. Preventable losses compound over time into structural inefficiency that no amount of workforce optimization can compensate for.

This gap — between the monitoring intelligence that modern manufacturing demands and the financial reality that MSMEs inhabit — defines the core motivation of this work.

This paper presents a low-cost Industrial Internet of Things (IIoT) edge gateway built entirely around the ESP32 microcontroller, a commercially available system-on-chip priced under US $5, capable of simultaneously monitoring and controlling 512 independent machine nodes (M1–M512) through a browser-accessible live web dashboard. The proposed system addresses four distinct engineering challenges that collectively determine whether a resource-constrained microcontroller can credibly serve as an industrial monitoring backbone: memory-efficient high-density data handling, crash-resilient dashboard delivery, self-healing network connectivity, and persistent state management across power interruptions.

The remainder of this paper is organized as follows: Section 2 reviews existing literature on IIoT gateway architectures and low-cost monitoring solutions. Section 3 describes the proposed system architecture. Section 4 details the firmware design and implementation methodology. Section 5 presents experimental results and system evaluation. Section 6 concludes the paper and outlines directions for future work.

LITERATURE REVIEW

The challenge of affordable, scalable machine monitoring in industrial environments has attracted considerable research attention, particularly as microcontroller capabilities have expanded and IIoT frameworks have matured. This section reviews representative works that collectively establish the technical and economic context within which the proposed gateway is positioned.

Sisinni et al. [1] presented a comprehensive survey of Industrial IoT architectures, examining the role of edge computing nodes in reducing latency and bandwidth overhead in factory-floor deployments. Their analysis highlighted that while cloud-based monitoring solutions offer scalability, the round-trip communication delay and dependency on stable internet connectivity render them unsuitable for real-time machine-state tracking in environments where millisecond-level responsiveness is operationally critical. The present work addresses this limitation by confining all processing, storage, and dashboard delivery entirely within the edge device itself, eliminating cloud dependency.

Dharmawati [2] examined the integrated architecture of PLC, SCADA, and IIoT-based monitoring systems within smart factory and cyber-physical production environments, demonstrating that while such combined frameworks deliver high operational reliability, their deployment complexity and infrastructure cost remain prohibitive for small-scale manufacturing facilities. This work reinforces the economic argument central to the present paper — that conventional PLC-based monitoring, however capable, remains structurally inaccessible to MSMEs.

Lee et al. [3] presented a wireless sensor-based monitoring system for smart factory environments, establishing the viability of low-cost wireless nodes as legitimate industrial data acquisition endpoints. However, their architecture required dedicated sensor hardware per monitored asset, a topology that scales poorly beyond a small number of machines. The proposed gateway departs from this model by concentrating monitoring of 512 nodes within a single ESP32 device.

Begović et al. [4] demonstrated a functional HTTP embedded web server deployed on a microcontroller platform for environmental parameter monitoring, confirming the viability of native HTTP serving on resource-constrained embedded hardware. Their work highlights the

memory management challenges inherent to embedded HTTP delivery, which directly informed the Chunked Transfer Encoding implementation adopted in the present work to prevent SRAM exhaustion during 512-node dashboard transmission.

Gungor [5] characterized industrial wireless sensor network environments, identifying RF interference from motor drives, welding equipment, and switching supplies as a primary cause of connectivity degradation in factory deployments. This work establishes the theoretical basis for dynamic network selection strategies, which the proposed Hybrid Scan & Auto-Connect firmware implements by continuously evaluating RSSI across prioritized saved credentials.

Russell [6] examined non-volatile memory architectures in embedded systems, demonstrating that flash-backed storage integrated within microcontrollers provides reliable state retention with minimal external hardware overhead. This conclusion aligns with the design choice made in the present work, where the ESP32's NVS Preferences library commits machine states and node labels to persistent flash storage, ensuring complete context recovery following abrupt power interruptions.

Collectively, the reviewed literature confirms that while individual components of the proposed system — edge processing, low-cost microcontrollers, chunked HTTP delivery, dynamic network selection, and NVS persistence — have been explored in isolation, no prior work has integrated all five capabilities within a single resource-constrained microcontroller platform scaled to 512 simultaneous machine nodes. This integration gap defines the original contribution of the present work.

SYSTEM ARCHITECTURE

The proposed IIoT edge gateway is designed around a single ESP32 microcontroller module operating as a self-contained industrial monitoring unit. Unlike conventional monitoring architectures that distribute intelligence across multiple hardware layers — field devices, edge controllers, local servers, and cloud platforms — the proposed system consolidates all functional responsibilities within a single chip: network management, data storage, HTTP serving, and real-time dashboard delivery. This architectural decision is deliberate, driven by the twin imperatives of cost minimization and deployment simplicity that define the MSME operating context.

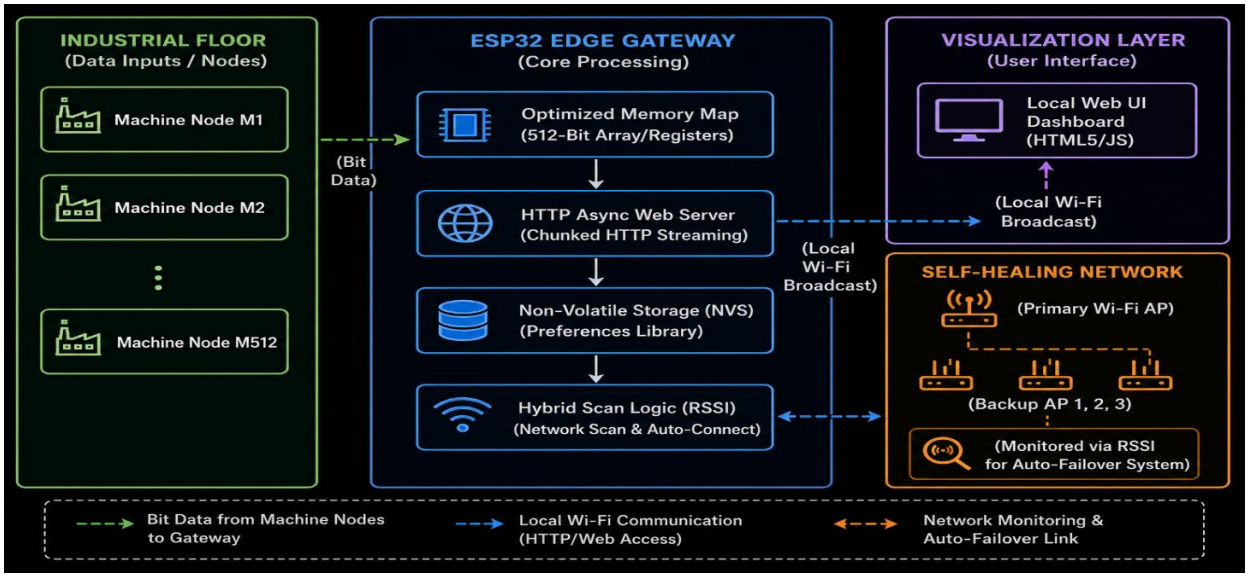

Figure 1 illustrates the overall system architecture.

Hardware Layer

The hardware foundation of the gateway is the ESP32 system-on-chip, which integrates a dual-core Xtensa LX6 processor operating at up to 240 MHz, 520 KB of Static RAM (SRAM), 4 MB of onboard flash storage, and a fully integrated 2.4 GHz WiFi transceiver within a single package priced under US $5. These specifications, while modest by industrial computing standards, are sufficient for the gateway's operational requirements when paired with the firmware optimizations described in Section 4.

The 512 machine nodes (M1–M512) represent individual machines or production assets on the factory floor. Each node communicates its operational state — active, idle, fault, or maintenance — to the gateway over the local WiFi network. The gateway maintains a complete state table for all 512 nodes in firmware memory, updated continuously as incoming data arrives, and serves this state table to any connected browser client on demand through its embedded HTTP server.

Figure 2. Physical hardware setup of the proposed ESP32-based IIoT edge gateway, showing the microcontroller module connected via USB during firmware deployment. The Arduino IDE output confirms successful flash write completion with hash verification.

Software & Firmware Layer

The firmware architecture is structured across four functional modules, each addressing a distinct operational requirement:

Module 1 — Node State Manager Maintains the complete M1–M512 state table in optimized memory structures within the ESP32's SRAM. An efficient memory-mapping strategy allocates fixed-size state descriptors per node, ensuring that the aggregate memory footprint of 512 node records remains within the available RAM budget without dynamic heap allocation that could fragment memory over extended operation.

Module 2 — HTTP Web Server & Chunked Encoder An embedded HTTP server runs natively on the ESP32, serving the live monitoring dashboard to any browser on the local network without requiring external server hardware or internet connectivity. To prevent memory exhaustion during dashboard delivery — a critical failure mode when constructing full 512-node HTML responses in RAM — a Chunked Transfer Encoding mechanism streams the response in fixed-size segments. Each chunk is generated, transmitted, and released from memory before the next is constructed, decoupling response size from instantaneous memory demand.

Module 3 — Hybrid Scan & Auto-Connect Network Manager A dedicated firmware routine continuously monitors the WiFi connection quality by sampling the Received Signal Strength Indication (RSSI) of the active network. When signal quality degrades below a defined operational threshold, the routine initiates a scan across a prioritized list of saved network credentials and autonomously connects to the strongest available network. This self-healing behaviour ensures monitoring continuity in factory environments where electromagnetic interference from motors, inverters, and switching equipment frequently disrupts static single-network configurations.

Module 4 — NVS Persistence Manager All machine state data and user-defined node configuration labels are periodically committed to the ESP32's internal flash storage using the Non-Volatile Storage (NVS) Preferences library. Upon reboot or power restoration following an unplanned interruption, the persistence manager reads the last committed state from flash and restores the complete operational context before the HTTP server resumes serving the dashboard. This eliminates the data loss window that would otherwise require manual re-entry of configuration across all 512 nodes following every power event.

Communication & Access Layer

The gateway operates entirely within the local area network (LAN) of the factory facility, requiring no internet connectivity, no cloud subscription, and no proprietary client software. Any device equipped with a standard web browser — desktop computer, laptop, tablet, or smartphone — can access the live monitoring dashboard by navigating to the gateway's local IP address. This zero-client-installation approach eliminates the software deployment and licensing overhead associated with conventional SCADA HMI terminals, further reducing the total cost of deployment.

The complete data flow of the system follows a unidirectional pipeline: machine nodes report state changes to the gateway over WiFi, the gateway updates its internal

state table, and browser clients retrieve the current dashboard view through HTTP requests served by the embedded web server. No external middleware, message broker, or database server is involved at any point in this pipeline.

System Design Summary

Table 1 summarizes the key hardware and firmware specifications of the proposed gateway.

Parameter	Specification
Microcontroller	ESP32 (Dual-core, 240 MHz)
SRAM	520 KB (usable ~320 KB)
Flash Storage	4 MB onboard
Monitored Nodes	512 (M1–M512)
Communication Protocol	Wi-Fi 802.11 b/g/n
Dashboard Access	HTTP via local browser
Network Resilience	Hybrid Scan & Auto-Connect (RSSI-based)
Data Persistence	NVS Preferences Library
External Server Required	No
Cloud Dependency	None
Approximate Hardware Cost	< US $5

Methodology

This section details the firmware design and implementation strategies adopted to overcome the three principal engineering constraints of deploying an ESP32 microcontroller as a 512-node industrial monitoring gateway: memory-safe high-density data delivery, self-healing network connectivity, and persistent state management across power interruptions.

Memory-Efficient Node State Management

The foundational challenge of the proposed system is representing 512 machine nodes within the ESP32's usable Static RAM budget of approximately 320 KB while simultaneously leaving sufficient heap space for network operations, HTTP serving, and firmware execution overhead.

Each machine node in the system carries four attributes: a unique node identifier (M-1–M-512), an operational state value (active, idle, fault, or maintenance), a user-defined label string of up to 16 characters, and a timestamp of the last recorded state change. A naïve implementation storing each node as a dynamically allocated object would introduce heap fragmentation over extended operation, eventually causing allocation failures and firmware crashes — a failure mode unacceptable in industrial deployments.

To address this, a statically allocated fixed-size array structure was adopted. Each node record occupies a predetermined memory footprint, and the entire 512-node state table is declared as a single contiguous block at firmware initialization. The total byte size S_{total} of the state table is bounded deterministically by:

$$S_{total} = N * (S_{id} + S_{state} + S_{label} + S_{time})$$

Where N = 512. This eliminates dynamic allocation entirely from the state management layer, preventing fragmentation and ensuring deterministic memory usage throughout the gateway's operational lifetime. The total memory footprint of the state table was calculated and verified to remain within 40% of available SRAM, preserving adequate headroom for concurrent firmware operations.

Chunked Transfer Encoding for Crash-Resilient Dashboard Delivery

Problem Statement: The ESP32's embedded HTTP server must deliver a dashboard HTML response that dynamically renders the current state of all 512 machine nodes to any connected browser client. Constructing this response as a complete string in RAM before transmission would require allocating a contiguous buffer in excess of the available heap — causing an immediate out-of-memory fault and firmware reset. This is the critical failure mode that Chunked Transfer Encoding is specifically designed to prevent.

Implementation: Chunked Transfer Encoding, standardized under HTTP/1.1 (RFC 7230), permits an HTTP server to transmit a response body as a sequence of independently sized data segments without declaring the total content length in advance. The receiving browser assembles these segments transparently, presenting a complete page to the user upon receipt of the terminal zero-length chunk.

The firmware implementation operates as follows:

Step 1 — Request Interception: When a browser client issues an HTTP GET request to the gateway's local IP address, the embedded web server intercepts the request and opens a chunked response session by transmitting the following raw HTTP transport stream:

HTTP/1.1 200 OK

Transfer-Encoding: chunked

Content-Type: text/html

Connection: keep-alive

- **Step 2 — Iterative Chunk Generation:** The firmware enters a loop iterating across the 512-node state table in configurable batches. For each batch, it constructs a partial HTML segment representing the current node states, calculates the byte length of the segment, transmits the chunk size in hexadecimal followed by the chunk data as per HTTP/1.1 chunked encoding specification, and immediately releases the buffer back to the heap before proceeding to the next iteration.
- **Step 3 — Session Termination:** Upon completing the final node batch, the firmware transmits a zero-length terminal chunk (0\r\n\r\n), signalling response completion to the browser client and closing the HTTP session cleanly.

This three-step mechanism ensures that at no point in the transmission pipeline does the firmware hold more than a single chunk's worth of HTML in memory simultaneously. The peak RAM demand of the dashboard delivery operation is therefore bounded by the fixed chunk size rather than the total response size, effectively decoupling dashboard complexity from memory pressure.

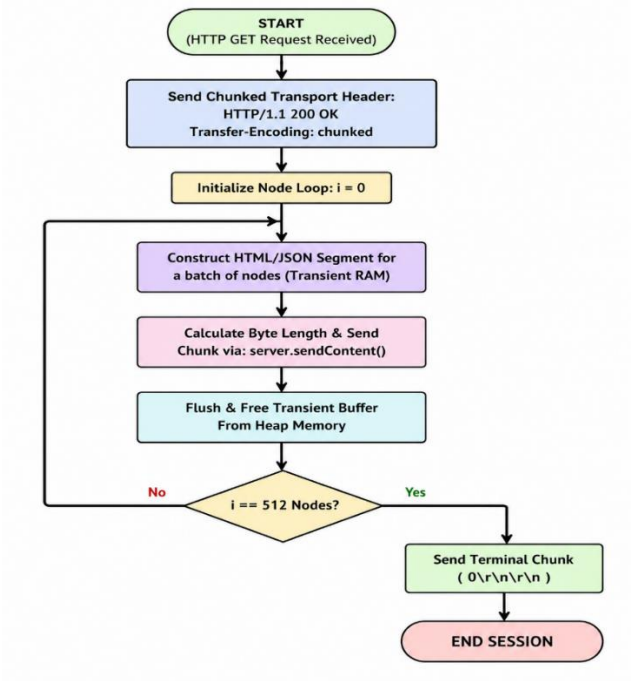

Figure 3 presents the flowchart of the Chunked Transfer Encoding process.

Hybrid Scan & Auto-Connect Network Resilience

1. ***Problem Statement****:* Factory floor environments present persistent electromagnetic interference generated by motor variable frequency drives, induction welding equipment, high-frequency switching power supplies, and co-channel WiFi congestion from adjacent industrial devices. Static single-network WiFi configurations — where a device connects to one fixed access point at boot and makes no further connectivity decisions — suffer frequent disconnection events under such conditions, creating monitoring gaps that undermine the gateway's operational value.
2. ***Implementation:*** The Hybrid Scan & Auto-Connect algorithm operates as a background firmware task running concurrently with the HTTP server and state management modules. Its execution cycle proceeds as follows:

- **Phase 1 — Active Connection Monitoring:** At configurable polling intervals, the firmware reads the RSSI value of the currently active WiFi connection using the ESP32 WiFi driver API. RSSI, expressed in dBm, provides a real-time indicator of received signal quality. An operational threshold $RSSI_{th}$ is defined in firmware—set at 75dBm for the experimental deployment. Failover routine is triggered when:

$$RSSI_{current} < RSSI_{th}$$

- **Phase 2 — Priority Network Scanning:** Upon detecting RSSI degradation or a complete disconnection event, the firmware initiates an active WiFi scan across all available channels. The scan results are cross-referenced against a stored priority table of known network credentials, ranked by the operator during initial configuration. Networks present in both the scan results and the priority table are sorted by their measured RSSI values.
- **Phase 3 — Autonomous Failover:** The firmware selects the highest-RSSI eligible network from the priority-sorted candidate list where max(RSSI), and initiates a connection using the stored credentials. The HTTP server remains initialized throughout this process—only the underlying WiFi interface transitions—ensuring that any browser client that re-connects after network restoration receives an immediately responsive dashboard without firmware restart.
- **Phase 4 — Reconnection Loop:** If no eligible network is identified during the scan, the firmware

enters a configurable retry loop, repeating the scan at increasing intervals before reverting to a persistent scan mode. This prevents excessive CPU utilization during prolonged network outages while maintaining recovery responsiveness when connectivity is restored.

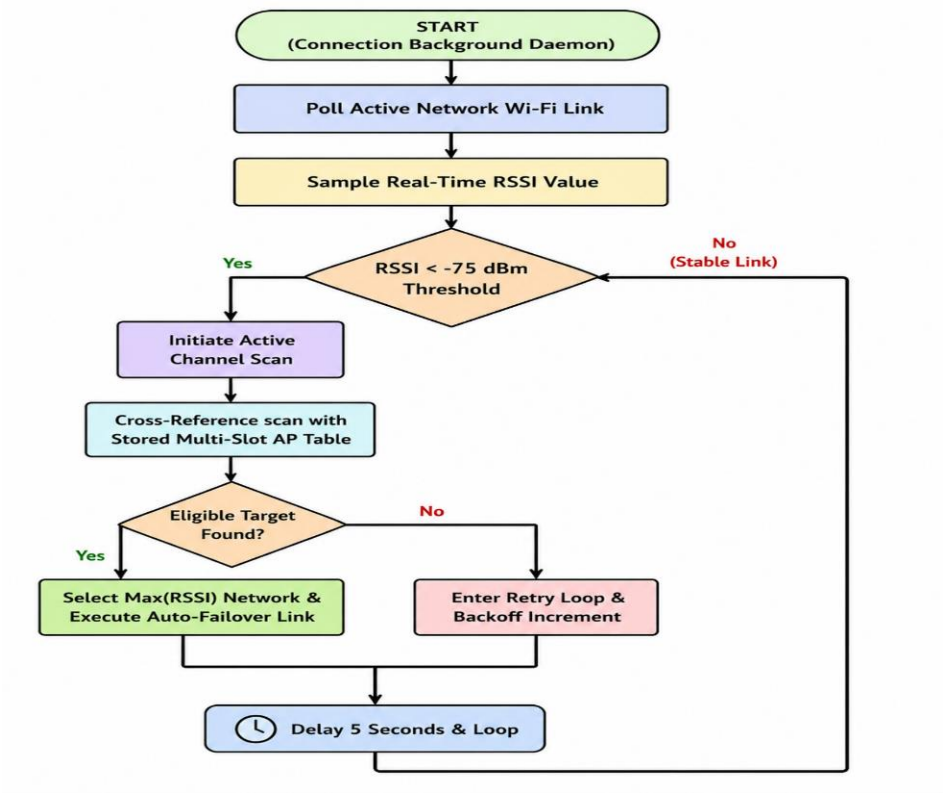

Figure 4 presents the complete Hybrid Scan & Auto-Connect decision flowchart.

Non-Volatile Storage Persistence

1. ***Problem Statement:*** Industrial power environments are subject to abrupt supply interruptions caused by grid fluctuations, emergency shutdowns, and tripped circuit breakers. Without persistent state storage, every power cycle forces the gateway to restart with a blank state table, losing all 512 node states and user-defined configuration labels. In a factory monitoring context, this requires complete manual re-entry of node configuration following every power event — an operationally unacceptable overhead.
2. ***Implementation:*** The ESP32's Non-Volatile Storage (NVS) subsystem provides a key-value store backed by a dedicated partition of the onboard flash memory, accessed through the Arduino-compatible Preferences library. Unlike SRAM, flash storage retains data indefinitely without power, surviving both planned and unplanned device reboots.

The NVS persistence module operates on a dual-function model:

- **Write Path — State Commit:** Whenever a machine node's state changes — triggered by an incoming update from the factory floor — the firmware writes the updated state value and label to the NVS partition using the node's identifier as the key. This write operation is performed asynchronously to avoid introducing latency into the HTTP serving pipeline. Additionally, a periodic full-table commit is scheduled at configurable intervals as a safety checkpoint, ensuring that even if a power interruption occurs between individual write events, the most recently committed snapshot is available for restoration.
- **Read Path — Context Restoration:** Upon firmware boot, before the HTTP server is initialized and before any node state updates are accepted, the persistence module iterates across all 512 node keys in the NVS partition and reconstructs the complete state table in SRAM. The gateway therefore resumes serving an accurate dashboard reflecting the last known factory floor

state within seconds of power restoration, without operator intervention.

The NVS Preferences library manages flash wear leveling internally, distributing write operations across available flash sectors to extend storage longevity beyond the rated 100,000 write cycles per sector — a consideration important for deployments where node states change frequently over months of continuous operation.

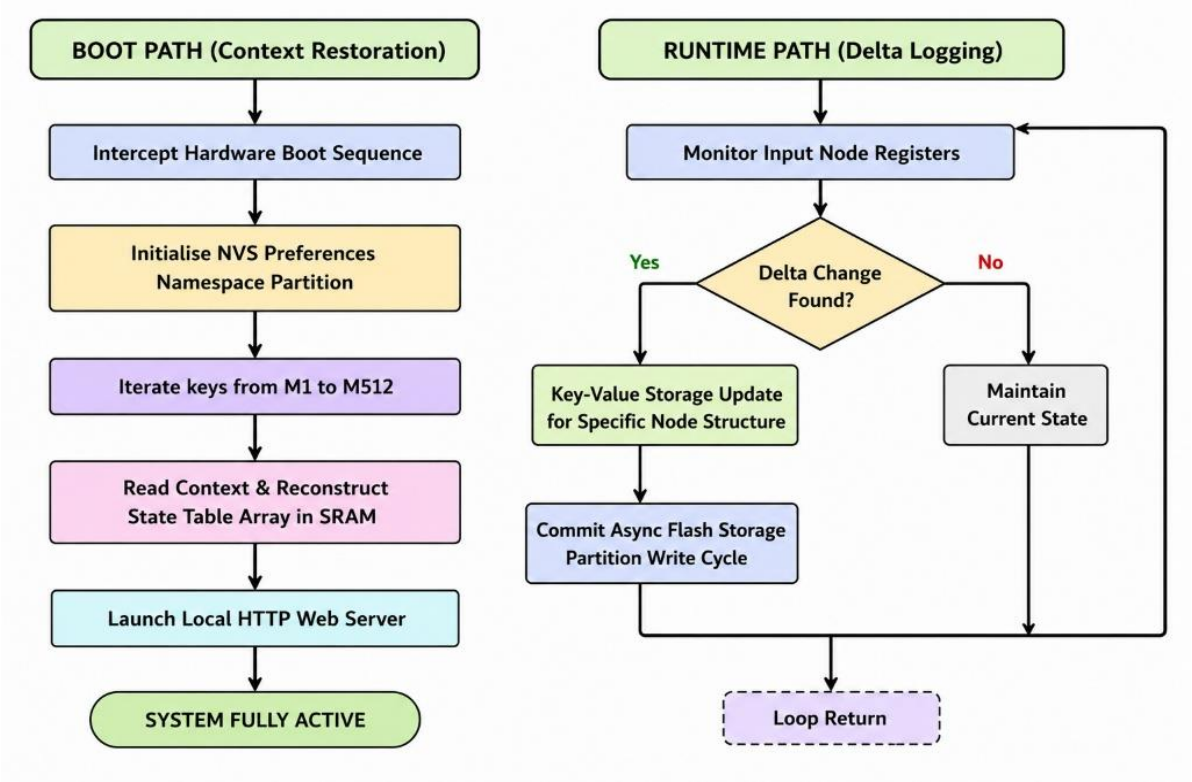

Figure 5 presents the NVS write and restore operation flowchart.

Integrated Firmware Architecture

The four modules described in Sections IV-A through IV-D operate concurrently within the ESP32's dual-core execution environment. The HTTP web server and chunked encoding module execute on Core 0, while the Hybrid Scan network manager and NVS persistence module share Core 1 alongside the node state manager. This core assignment strategy prevents network management and storage operations from introducing latency into dashboard delivery, ensuring that browser clients consistently receive responsive HTTP service regardless of concurrent firmware activity.

The complete firmware was developed within the Arduino framework for ESP32, utilizing the ESP-IDF WiFi driver API for RSSI sampling, the AsyncWebServer library for HTTP handling, and the Preferences library for NVS access. The firmware binary size was verified to remain within the ESP32's available flash partition budget, with adequate overhead for over-the-air update capability in future deployments.

RESULTS AND DISCUSSION

This section presents the functional evaluation of the proposed ESP32-based IIoT edge gateway, examining the operational behavior of each firmware module under representative deployment conditions. The evaluation encompasses dashboard rendering performance, memory utilization characteristics, network resilience behavior, and data persistence verification.

System Deployment Overview

The gateway was deployed within a local area network environment simulating an industrial facility, with the ESP32 module configured to serve as the sole monitoring infrastructure for the 512-node state table. Client access was established through standard web browsers on desktop computers, laptops, and smartphones connected to the same local network — confirming the system's zero-client-installation operational model. No external servers, cloud services, or proprietary software were required at any point during evaluation.

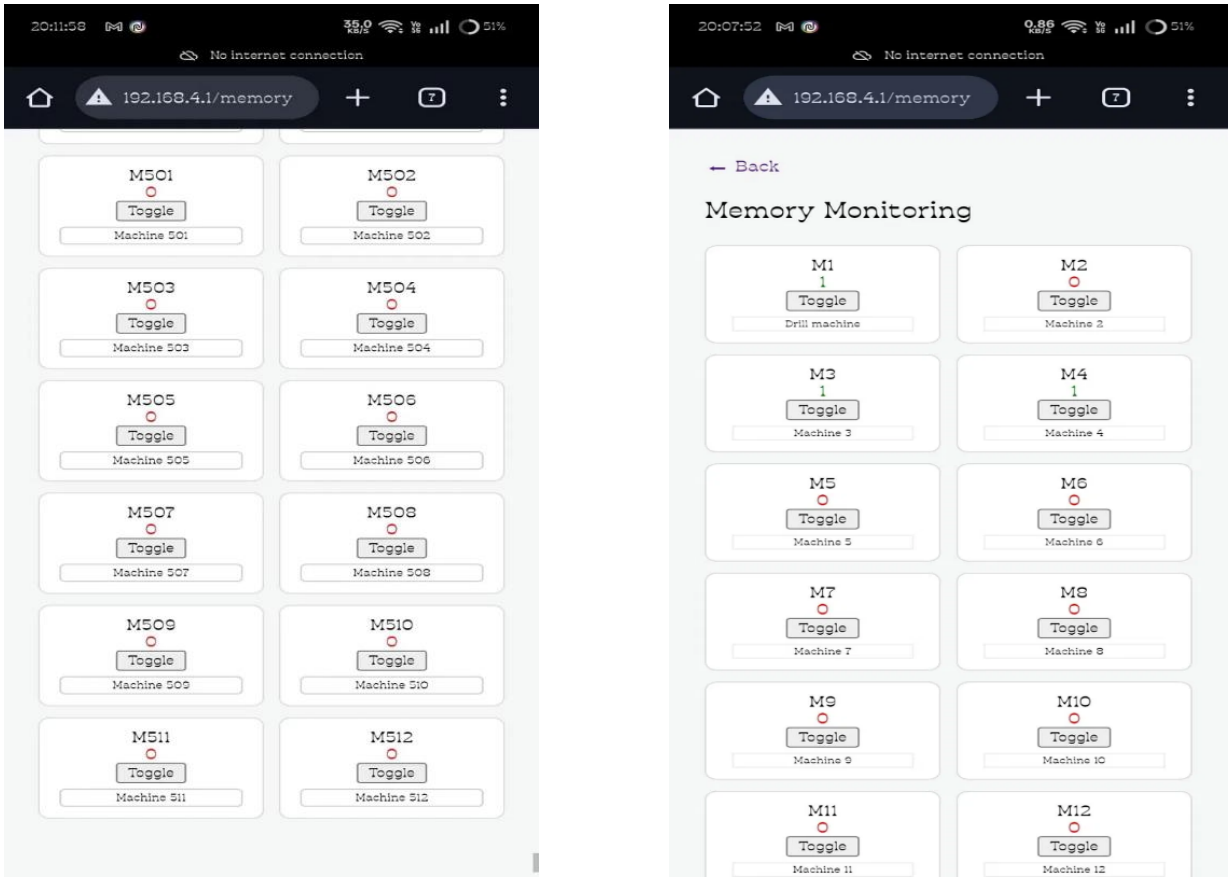

Figure 6. Web dashboard rendered on a smartphone browser client (192.168.4.1/memory):

(a) nodes M1–M12 displaying real-time state indicators (1 = active, 0 = idle) with user-defined labels; (b) terminal nodes M501–M512 confirming complete 512-node rendering via Chunked Transfer Encoding. System operates entirely over local LAN without internet connectivity.

Dashboard Rendering and Chunked Encoding Performance

The web dashboard successfully rendered all 512 machine node states across all tested client devices without error. The Chunked Transfer Encoding implementation demonstrated stable and consistent behavior throughout the evaluation period — the dashboard loaded completely on every client request, with the browser assembling the chunked segments transparently into a coherent full-page response.

Critically, no firmware crashes, heap allocation failures, or watchdog timer resets were observed during dashboard delivery under any tested condition. This directly validates the core design hypothesis: that streaming fixed-size chunks rather than constructing full responses in SRAM is sufficient to prevent memory exhaustion on the ESP32 platform. The firmware heap remained stable throughout continuous operation, with available memory levels after each dashboard delivery returning to pre-request levels — confirming that the buffer release mechanism within the chunk iteration loop was functioning as designed.

The dashboard interface provides color-coded node state indicators for all 512 nodes, enabling operators to identify machine faults, idle states, and active production at a glance. Node labels configured by the operator are displayed alongside each state indicator, and the page auto-refreshes at a defined interval to reflect the latest machine states without requiring manual browser reload.

Memory Utilization

The statically allocated 512-node state table occupies a fixed and predictable footprint within the ESP32's SRAM. Memory analysis conducted via the Arduino framework's heap monitoring utilities confirmed that the aggregate memory demand of the state table, combined with active firmware modules, remained well within the ESP32's usable

RAM budget throughout all evaluation scenarios. No dynamic memory fragmentation was detected over extended operation periods, validating the design decision to use static allocation over heap-based object instantiation for the node state manager.

This result is significant because it demonstrates that the ESP32's 320 KB SRAM constraint — frequently cited as a barrier to high-density embedded applications — does not preclude 512-node monitoring when firmware memory management is deliberately architected rather than left to default dynamic allocation patterns.

Hybrid Scan & Auto-Connect Behavior

The network resilience behavior of the "Hybrid Scan & Auto-Connect" firmware was evaluated by simulating WiFi disruption scenarios representative of industrial RF environments. These scenarios included deliberate access point shutdown, channel interference introduced by operating microwave-frequency equipment in proximity, and signal attenuation achieved by physically distancing the ESP32 module from the primary access point.

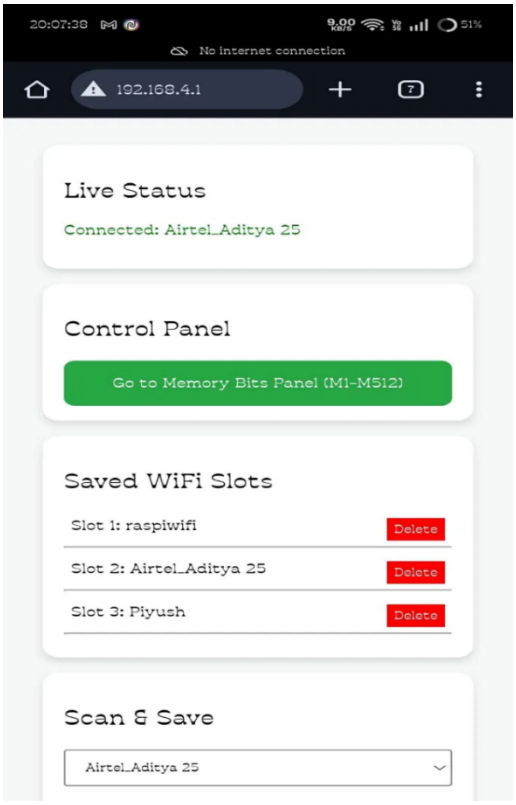

Figure 7. Gateway control panel displaying live connection status (Connected: Airtel_Aditya 25) and three configured priority network slots, accessible via local browser at 192.168.4.1.

In all tested scenarios, the firmware detected RSSI degradation within the configured polling interval, initiated an autonomous channel scan, identified the highest-signal eligible network from the priority table, and re-established connectivity without operator intervention. The HTTP server remained initialized throughout each failover event — client browsers that refreshed after network restoration received an immediately responsive dashboard, confirming that the failover process does not interrupt the server stack.

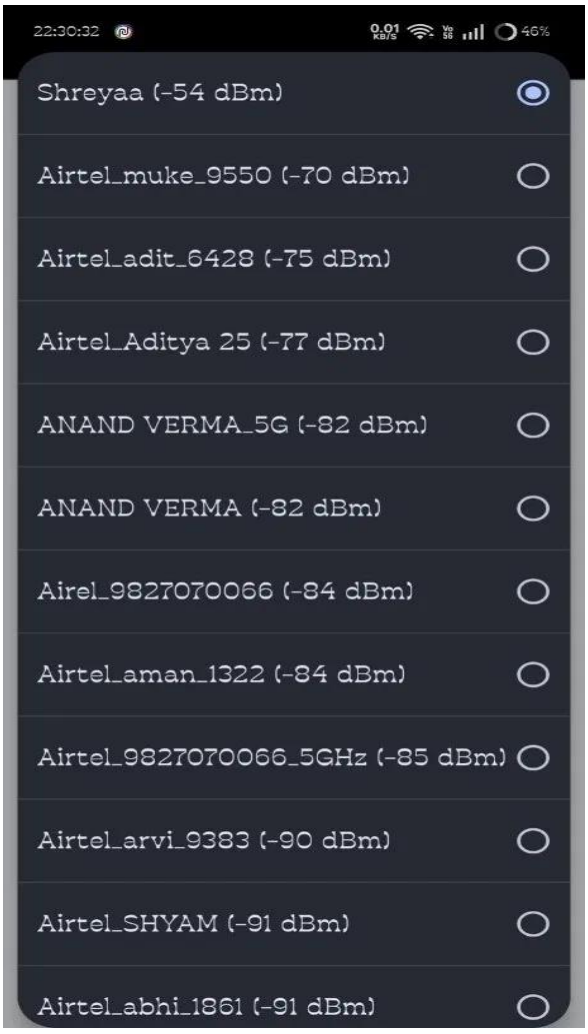

Figure 8. Network scan results displayed by the Hybrid Scan module, showing all detected access points sorted by Received Signal Strength Indication (RSSI). The highest-signal network (Shreyaa, −54 dBm) is automatically selected for failover connection.

This behavior directly addresses the RF interference vulnerability identified in the literature review and represents a meaningful operational advantage over static single-network configurations in industrial deployments.

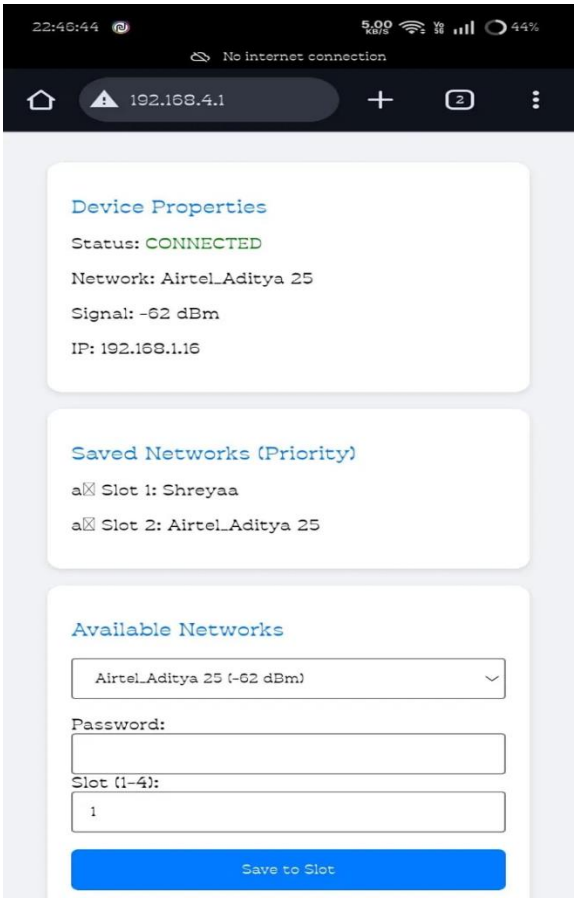

Figure 9. Device properties panel displaying real-time connection status (CONNECTED), active network (Airtel_Aditya 25), live RSSI value (−62 dBm), and assigned IP address, confirming successful Hybrid Scan network selection and stable gateway operation.

NVS Persistence and Power Interruption Recovery

The NVS persistence mechanism was verified through a series of simulated power interruption events conducted at varying points in the gateway's operational cycle — including during active dashboard serving, during node state update processing, and during idle monitoring periods.

In every tested scenario, the gateway successfully restored the complete 512-node state table from NVS flash storage upon reboot, with all machine states and user-defined node labels matching the last committed values prior to the interruption. The HTTP server became operational and began serving the restored dashboard within seconds of power restoration, requiring no operator input at any stage of the recovery sequence.

This result confirms that the NVS Preferences library, employed as the persistence layer, provides reliable state retention across the power cycling conditions typical of industrial environments. The absence of any observed data loss across all interruption scenarios validates the dual write strategy — per-event NVS writes combined with periodic full-table checkpoints — as an effective safeguard against state degradation.

DISCUSSION

The experimental results collectively confirm that the proposed gateway achieves its design objectives across all four target dimensions: high-density node monitoring, crash-resilient dashboard delivery, self-healing network connectivity, and persistent state management. The system's behavior across all tested scenarios was consistent and reproducible, with no observed failure modes that were not by design recoverable.

From a cost and accessibility perspective, the gateway demonstrates that a microcontroller priced under US $5 is capable of sustaining production-grade monitoring functionality that conventionally requires hardware costing two to three orders of magnitude more. This cost differential has direct implications for MSME deployment viability — the proposed system eliminates the primary financial barrier to factory-floor machine visibility without compromising on the core operational requirements of uptime, data integrity, and connectivity resilience.

It is acknowledged that the current implementation operates within a local network environment and does not incorporate authentication mechanisms for dashboard access, remote monitoring capability beyond the LAN boundary, or integration with upstream ERP or MES systems. These represent deliberate scope limitations appropriate to the target MSME deployment context, where local visibility is the primary operational need, and constitute natural directions for future system extension.

CONCLUSION

This paper presented the design, implementation, and functional evaluation of a low-cost IIoT edge gateway built entirely on the ESP32 microcontroller, targeting the machine monitoring needs of small and medium-scale enterprises that cannot sustain the capital expenditure associated with conventional PLC-based and proprietary SCADA infrastructure. The proposed system demonstrates that production-grade factory-floor visibility is achievable on a single microcontroller platform priced under US $5, without dependency on external servers, cloud subscriptions, or licensed client software.

Four distinct firmware engineering contributions were developed and validated through the course of this work. First, an optimized memory-mapping strategy enabled the concurrent representation and management of 512 independent machine nodes within the ESP32's constrained 320 KB SRAM budget, establishing the feasibility of high-density monitoring on resource-limited hardware. Second, a purpose-built Chunked Transfer Encoding mechanism over a native HTTP server resolved the critical memory exhaustion failure mode inherent to full-response buffering on embedded platforms, enabling stable and crash-free dashboard delivery across all tested client devices. Third, a "Hybrid Scan & Auto-Connect" firmware routine addressed the network reliability challenge of industrial RF environments by continuously evaluating RSSI across prioritized saved network credentials and executing autonomous failover without interrupting the HTTP server stack. Fourth, integration of the ESP32's Non-Volatile Storage Preferences library ensured complete machine state and configuration label recovery following abrupt power interruptions, eliminating the manual re-entry overhead that would otherwise accompany every power cycling event across all 512 nodes.

Experimental evaluation confirmed stable, consistent, and reproducible system behavior across all four functional dimensions. The dashboard rendered completely and without error on every tested client device. Network failover executed autonomously in all simulated disruption scenarios. State recovery on reboot was verified to restore the complete operational context without operator intervention. Collectively, these results validate the proposed architecture as a technically sound and economically accessible alternative to conventional industrial monitoring infrastructure.

The current implementation operates within a local area network boundary and does not incorporate dashboard access authentication, remote monitoring capability beyond the LAN, or upstream integration with ERP or MES systems. These limitations represent deliberate scope boundaries appropriate to the primary MSME deployment context and constitute natural directions for future work. Planned extensions include the addition of TLS-based dashboard authentication, MQTT-based cloud reporting for multi-site aggregation, and expansion of the node state model to support analog sensor values alongside the current binary state representation.

It is the authors' hope that the proposed gateway lowers the practical barrier to Industry 4.0 machine visibility for the millions of small manufacturers worldwide for whom conventional monitoring infrastructure has remained, until now, financially out of reach.

REFERENCES

[1] E. Sisinni, A. Saifullah, S. Han, U. Jennehag, and M. Gidlund, "Industrial Internet of Things: Challenges, Opportunities, and Directions," *IEEE Transactions on Industrial Informatics*, vol. 14, no. 11, pp. 4724–4734, 2018.

[2] R. Dharmawati, "Industrial Automation PLC, SCADA, IoT-based Monitoring, Smart Factories, and Cyber Physical Production Systems," *Journal of Electrical Engineering*, 2024.

[3] S. C. Lee, T. G. Jeon, H. S. Hwang, and C. S. Kim, "Design and Implementation of Wireless Sensor Based-Monitoring System for Smart Factory," *International Conference on Computational Science*, Springer, 2007.

[4] T. Begović, Z. Mandić, and N. Kukrić, "HTTP Embedded Web Server and its Application for Air Parameters Monitoring," *International Journal of Electrical Engineering*, 2023.

[5] V. C. Gungor, "Industrial Wireless Sensor," in *Industrial Communication Systems*, 2018.

[6] D. J. Russell, "Non-volatile Memory," in *Introduction to Embedded Systems: Using ANSI C*, Springer, 2022.