DESIGN AND IMPLEMENTATION OF A SMART URBAN LIGHTING SYSTEM USING AN ADAPTIVE TINYML AND SELF-RECOVERY OF LORA COMMUNICATION

NGUYEN VAN LANH*, NGUYEN KHAC SON, NGUYEN TUAN PHAT

Faculty of Electronics Technology, Industrial University of Ho Chi Minh City, HCMC, Viet Nam

** Corresponding author: nguyenvanlanh@iuh.edu.vn*

DOIs: https://www.doi.org/10.46242/jstiuh.v83i5.6179

Abstract. Conventional urban lighting systems often rely on fixed schedules or astronomical sunrise and sunset times, which limits their ability to react to local weather, support reliable field communication, and recover from peripheral faults. This paper presents an adaptive smart urban lighting system based on a distributed gateway-node architecture and focuses on three measurable contributions. First, we implement a gateway-side TinyML module that uses only two normalized inputs, average ambient illuminance and normalized API time, to estimate an offset for sunrise and sunset switching times. Second, we realize a hybrid ISR/Timer-based AC dimming mechanism that separates zero-cross detection from TRIAC gating to obtain stable and flicker-free brightness control for 220 V LED luminaires. Third, we add a communication self-recovery framework that detects and recovers from LoRa and I2C faults without restarting the whole system. The TinyML module is intentionally lightweight: the deployed linear regression model contains only six float parameters in total, requires 24 bytes for weights and biases, and performs a single dot-product-style inference at the gateway. Experimental validation on real ESP32-based hardware shows stable bi-directional LoRa communication, deterministic dimming behavior, successful peripheral recovery, and adaptive sunrise/sunset switching under varying environmental conditions. Compared with fixed-schedule lighting and cloud-dependent control approaches, the proposed design shifts adaptation to the gateway edge and keeps runtime overhead minimal, making it suitable for deployable smart city lighting systems.

Keywords. AC Dimming, Communication Self-recovery, LoRa, Smart Urban Lighting, TinyML.

1 INTRODUCTION

Urban public lighting systems play an important role in traffic safety, public security, and nighttime visibility in modern cities. With the development of smart city applications, lighting systems have gradually integrated sensing, communication, and control technologies to improve energy efficiency and operational flexibility. Castro, Jara, and Skarmeta [1] reviewed smart lighting solutions for smart cities and pointed out the advantages of using sensors and centralized monitoring platforms. However, many existing systems still operate based on fixed schedules or astronomical sunrise and sunset times, which do not fully reflect real environmental conditions such as cloud cover, rain, or local shading. As a result, lighting efficiency and performance are often not optimal in practical operation. In large-scale deployments, communication reliability and hardware stability remain key challenges. Gungor and Hancke [2] discussed the design requirements of industrial wireless sensor networks and highlighted the importance of reliability and long-term stable operation. At the hardware level, AC-powered luminaires require accurate control to operate reliably. Rashid [3] and Texas Instruments [4] showed that TRIAC-based phase-angle control is commonly used for AC dimming, but its performance strongly depends on timing accuracy. In multitasking embedded systems, timing delays caused by concurrent tasks can lead to unstable brightness and visible flicker. Recent developments in embedded systems have enabled lightweight machine learning to be deployed at the edge. Warden and Situnayake [5] introduced TinyML and demonstrated that machine learning models can run on low-resource microcontrollers, while Banbury et al. [6] analyzed the trade-offs between model accuracy, resource usage, and system reliability. However, most existing smart lighting systems still treat communication, intelligence, and hardware control as separate components, and lack adaptive edge intelligence, stable AC dimming, and automatic fault recovery. To overcome these limitations, this paper presents a smart urban lighting system based on a distributed gateway–node architecture, focusing on practical deployment. Adaptive lighting behavior is implemented using a lightweight edge-based TinyML

© 2026 Industrial University of Ho Chi Minh City

module at the gateway to adjust sunrise and sunset switching times based on real illuminance data. At the node level, a hybrid ISR/Timer-based AC dimming mechanism is used to achieve stable and flicker-free brightness control for 220 V LED luminaires. In addition, a lightweight health monitoring and self-recovery mechanism is designed to restore communication and peripheral operation without requiring a full system reboot. The proposed system is evaluated through experimental results obtained from a prototype deployed under real operating conditions. The main contributions of this work are summarized as follows: Firstly, an edge-based TinyML module for adaptive adjustment of sunrise and sunset switching times. Secondly, a hybrid ISR/Timer-based AC dimming architecture for stable and flicker-free brightness control. Thirdly, a lightweight health monitoring and self-recovery mechanism to improve system robustness. Novelty and Differentiation from Existing Approaches: Compared with conventional smart lighting systems that follow fixed astronomical schedules, the proposed solution adds a local adaptation layer at the gateway and keeps it independent from cloud inference. Compared with cloud-managed lighting schemes, the proposed system avoids network-dependent model execution and reduces control latency. Compared with general TinyML deployments reported in the literature, the present work is narrower but more deployable: the model is deliberately compact, the inference target is clear, and the output is a schedule correction offset rather than a high-dimensional prediction. The key novelty is not the use of TinyML alone, but the combination of edge-side adaptive scheduling, deterministic dimming, and self-recovery within one deployable gateway-node stack.

2 SYSTEM DESIGN METHOD

2.1 Overall System Architecture

The proposed smart urban lighting system adopts a distributed gateway–node architecture consisting of a central gateway and multiple lighting nodes deployed at lamp locations. Each lighting node performs local sensing and actuation, including ambient illuminance measurement, real-time AC dimming control, and status reporting. Nodes do not communicate directly with one another; instead, all data exchange and control commands are coordinated through the gateway. This centralized communication model simplifies network management and avoids the complexity of distributed routing. The gateway acts as the system coordination unit, aggregating data from lighting nodes, managing command scheduling, and hosting lightweight intelligence components such as the TinyML-based adaptive scheduling module. Time-critical control tasks are executed locally at the nodes, while system-level coordination and decision making are handled at the gateway, providing a balanced trade-off between real-time performance, scalability, and robustness. This integrated system is hereafter referred to as the SmartLight system.

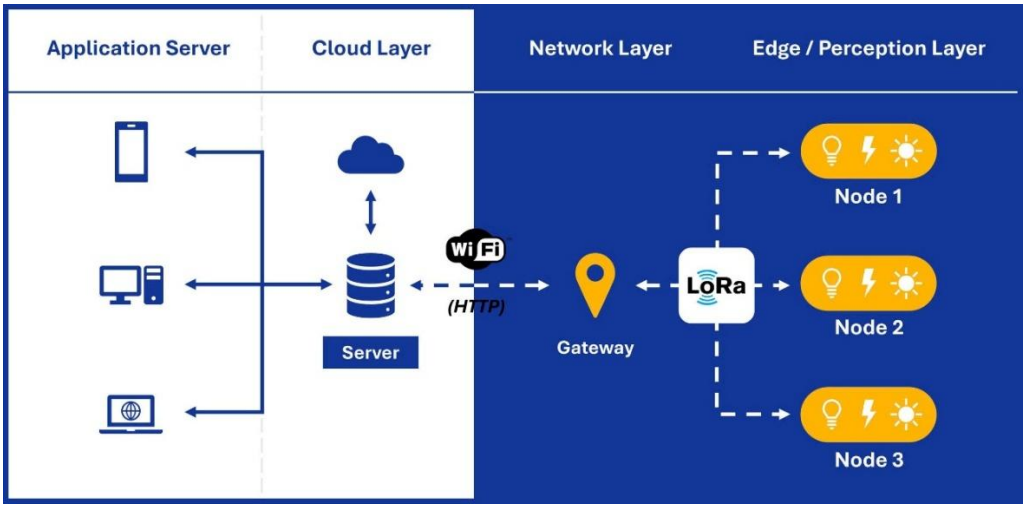


Figure 1: Overall architecture of the proposed smart urban lighting system

2.2 Layered Functional Design

To clearly organize system responsibilities and improve modularity, the proposed smart lighting system

adopts a layered functional design. As illustrated in Fig. 2, the architecture separates application interaction, coordination and intelligence at the gateway, state and reliability management, and hardware control at the node level. This separation allows each functional layer to be designed and evaluated independently while maintaining clear interaction boundaries.

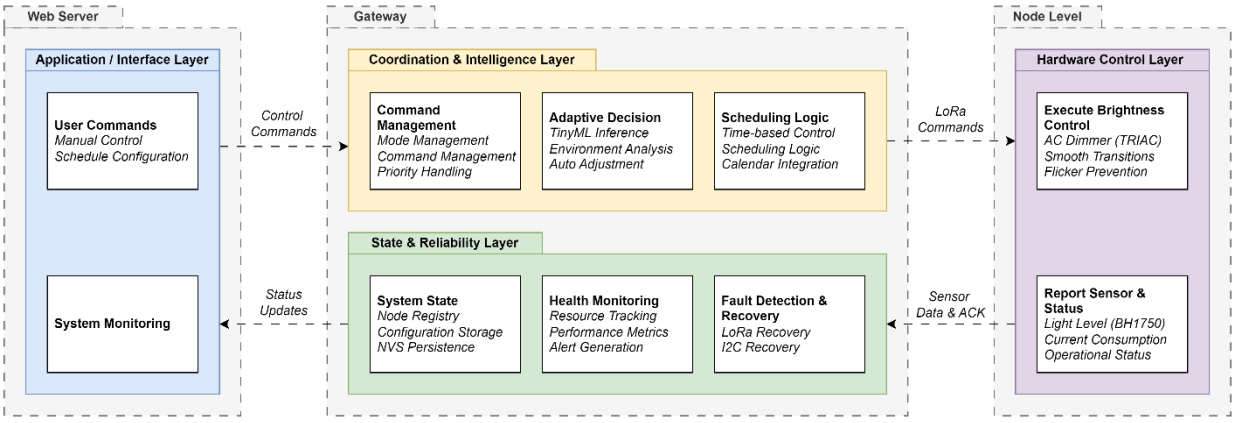


Figure 2: Layered functional design of the proposed smart urban lighting system

The application and interface layer provides user interaction and system supervision functions, including manual control, schedule configuration, and system monitoring. This layer serves as the entry point for user commands and high-level configuration without directly interacting with time-critical control operations. The coordination and intelligence layer, implemented at the gateway, is responsible for command management, scheduling, and adaptive decision-making. In this layer, a lightweight TinyML module analyzes aggregated illuminance data from multiple nodes and adjusts sunrise and sunset switching times accordingly, enabling context-aware lighting behavior with low computational overhead. The state and reliability layer spans both the gateway and lighting nodes. It maintains system state information and continuously monitors operational health. When communication or peripheral faults are detected, targeted recovery actions are triggered to restore normal operation without affecting unaffected components. Finally, the hardware control layer is implemented at the lighting node level and handles time-critical physical control tasks. This layer executes brightness control commands using AC dimming mechanisms and reports sensor measurements and operational status back to the gateway, enabling closed-loop monitoring and adaptive control.

2.3 Adaptive Lighting Design Using Edge-Based TinyML

Conventional lighting schedules based on astronomical sunrise and sunset times are derived from theoretical models and do not reflect real environmental conditions such as weather variation or local shading. To improve adaptability while maintaining system simplicity, the proposed system integrates an edge-based TinyML module to adjust lighting schedules using real-world illuminance measurements.

As illustrated in Fig. 3, lighting nodes periodically collect ambient light data using onboard sensors and perform lightweight local processing. The gateway aggregates the illuminance information from multiple nodes to obtain a representative view of the surrounding lighting conditions, reducing sensitivity to sensor noise and individual node failures. It shows distributed illuminance sensing at lighting nodes, lightweight inference at the gateway, and integration with the automatic scheduling mechanism.

The TinyML inference is executed at the gateway, where system-wide data and timing information are available. A lightweight linear regression model is employed to estimate a time offset relative to the nominal sunrise or sunset time. The model takes normalized illuminance and time inputs and produces a bounded time delta, ensuring deterministic and low-overhead execution suitable for embedded platforms. Rather than directly controlling lamp brightness, the predicted time offset is applied to adjust the existing lighting schedule. Specifically, lamp activation is advanced during sunset conditions and delayed during sunrise conditions. The adjusted schedule is stored persistently and executed by the automatic scheduling mechanism, allowing adaptive behavior to be seamlessly integrated without disrupting the underlying control logic.

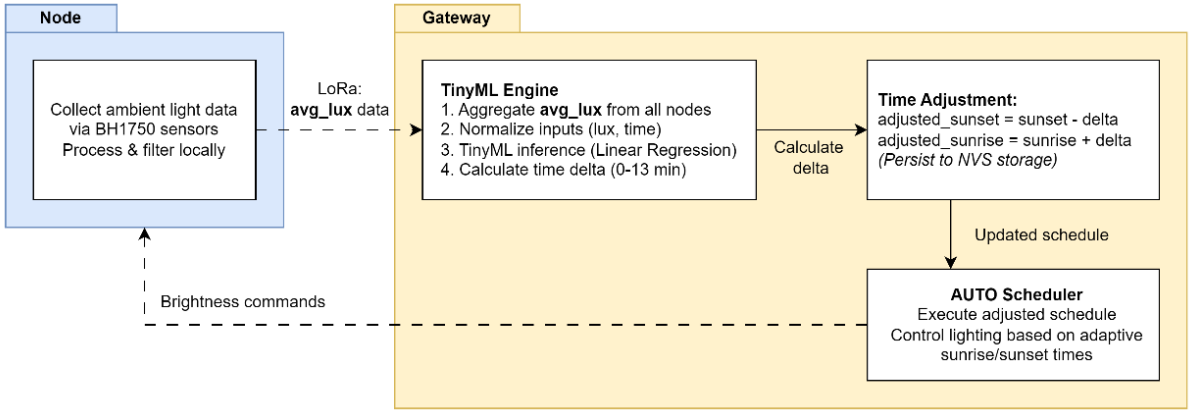


Figure 3: Decision-level TinyML flow for adaptive lighting control

The TinyML model is formulated as a small linear regression that estimates a time offset $\Delta\hat{t}$ (in minutes) relative to the astronomical sunrise or sunset time using:

$$\Delta\hat{t} = \omega_1 * avg_lux_norm + \omega_2 * api_time_norm + b \quad (1)$$

where avg_lux_norm denotes the normalized average illuminance measured within a predefined observation window, api_time_norm represents the normalized astronomical reference time, and ω_1, ω_2, and b are parameters learned during offline training. We use a linear regression model because the deployment goal is not expressive classification but a small, interpretable correction function that can run directly on the gateway without any cloud dependency.

To ensure numerical stability and robustness against outliers, the feature normalization step maps average illuminance into a bounded range as follows:

$$avg_lux_norm = \frac{\min(\bar{L}, L_{\max})}{L_{\max}} \quad (2)$$

where $\bar{L}$ is the measured average illuminance over the observation window and L_{max} is an empirically selected upper bound. This normalization constrains the feature to the range $[0,1]$, limits the influence of extreme lighting conditions, and simplifies embedded deployment.

The training and deployment pipeline is derived from a simple offline workflow. The pipeline is intentionally compact:

1. Collect a 20-minute lux window around sunrise or sunset.
2. Remove spikes using a z-score filter and smooth the remaining series using a moving average.
3. Compute the representative average lux feature avg_lux.
4. Normalize avg_lux and the API time into avg_lux_norm and api_time_norm.
5. Train a separate linear regression model for sunrise and sunset.
6. Export the learned coefficients into ml_models.h for gateway inference.

The gateway does not execute training on-device. It only performs inference when sunrise/sunset context is active and fresh node lux data are available. The deployed model is embedded in the gateway firmware as a header file containing two small linear models. For each event, the gateway performs one dot product and a clamp operation, then shifts the target switching time accordingly. The resulting corrected schedule can be stored in NVS and reused until the next update cycle. The implementation-level benchmark profile is summarized below Table 1.

Table 1: Deployment-level performance evaluation metrics

Metrics	Values
Input features 2	(api_time_norm, avg_lux_norm)
Output	1 offset value $\Delta\hat{t}$ in minutes

Parameters per model	3 floats (2 weights + 1 bias)
Total deployed parameters	6 floats
Model weight memory	24 bytes (float32)
Estimated inference time	Well below 1ms on ESP32-S3-class gateway hardware
Arithmetic per inference	2 multiplications + 2 additions + 1 clamp
Deployment location	Gateway firmware
Inference trigger	Sunrise/sunset update window

Because the model is a single linear layer with event-driven inference, the computational overhead isnegligible relative to the gateway's Wi-Fi, LoRa, and task-management workload.

2.4 AC Dimming Control Design

Conventional phase-angle AC dimming is highly sensitive to timing jitter, particularly in multitasking embedded systems where software delays can lead to visible flicker and unstable brightness. To address this issue, the proposed lighting node adopts a dual-path AC dimming control strategy that separates time-critical operations from non-critical processing. As illustrated in Fig. 4, the fast path is implemented using a zero-cross interrupt service routine (ISR). Upon detecting each AC zero-crossing event, the ISR records the timestamp and computes the required triggering delay corresponding to the target brightness level. Importantly, no TRIAC triggering is performed directly inside the ISR, ensuring minimal execution time and deterministic response. separating fast zero-cross detection from delayed TRIAC triggering to achieve microsecond-level timing accuracy and flicker-free operation. Summary, separating fast zero-cross detection from delayed TRIAC triggering to achieve microsecond-level timing accuracy and flicker-free operation.

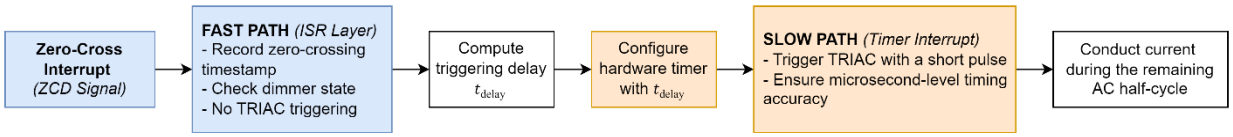


Figure 4: Dual-path AC dimming control strategy

The slow path is responsible for executing the delayed TRIAC triggering. Based on the computed delay, a hardware timer is configured to generate a short triggering pulse at the appropriate moment within the AC half-cycle. By delegating precise delay handling to a timer-based mechanism, the system achieves microsecond-level timing accuracy while avoiding prolonged ISR execution. This dual-path design effectively decouples real-time timing accuracy from higher-level software activities. As a result, stable and flicker-free brightness control is maintained even under concurrent communication and control tasks, making the proposed approach well suited for practical smart lighting deployments.

2.5 Reliability and Self-Recovery Design

Long-term deployment of smart lighting systems requires the ability to tolerate communication disruptions, peripheral faults, and transient software anomalies without manual intervention. To address this requirement, the proposed system incorporates a reliability and self-recovery design that is integrated at the system level rather than treated as an auxiliary feature. As illustrated in Fig. 5, system health is continuously monitored through lightweight runtime checks. At the gateway, health monitoring focuses on high-level indicators such as communication responsiveness and resource utilization trends, avoiding intrusive or computation-heavy diagnostics. When abnormal conditions are detected, recovery requests are generated and forwarded to a centralized recovery coordination mechanism.

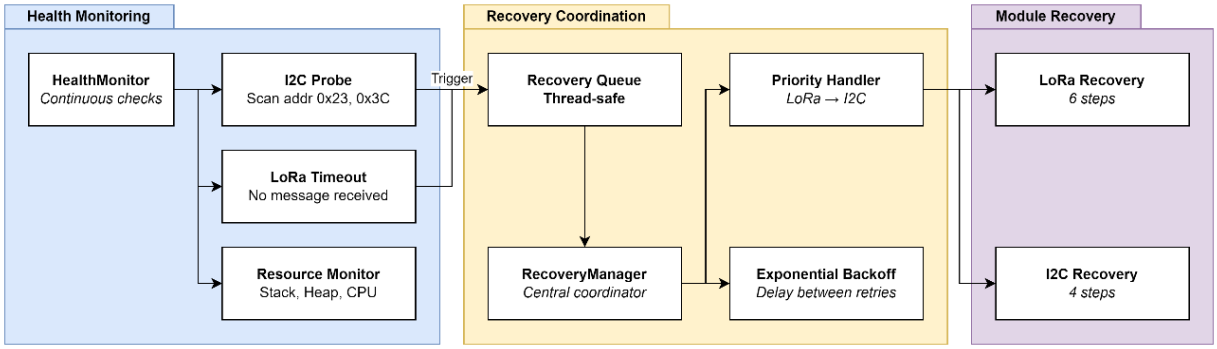


Figure 5: System-level reliability and self-recovery architecture

The recovery coordination mechanism manages recovery actions in a structured and prioritized manner. Recovery requests are queued in a thread-safe structure and processed by a dedicated recovery manager. To prevent repeated failures and recovery storms, retry attempts are regulated using an exponential backoff strategy. In addition, recovery actions are prioritized based on their impact on system operation, with communication recovery taking precedence over peripheral-level recovery. Module-level recovery is performed using software-based procedures without requiring system-wide resets. Communication recovery focuses on restoring LoRa operation through controlled reinitialization, while peripheral recovery targets I2C devices through bus re-probing and reconfiguration. By isolating recovery actions to the affected subsystems, the proposed design maintains continued operation of unaffected components. This layered reliability and self-recovery approach enables the system to sustain stable operation under real-world conditions, including intermittent communication failures and transient hardware faults, thereby improving long-term availability and robustness in practical smart lighting deployments.

3 HARDWARE IMPLEMENTATION

This section presents the hardware implementation of the proposed smart urban lighting system. The system consists of two main components: a lighting node and a central gateway, forming a distributed gateway–node architecture. The hardware is designed to support reliable outdoor operation, with time-critical control executed locally at the lighting node and system-level coordination handled at the gateway. This separation enables stable real-time control, scalable deployment, and robust long-distance communication. Figure 6 illustrates the implemented hardware system and the interaction between the gateway, lighting nodes, and the cloud-based application layer.

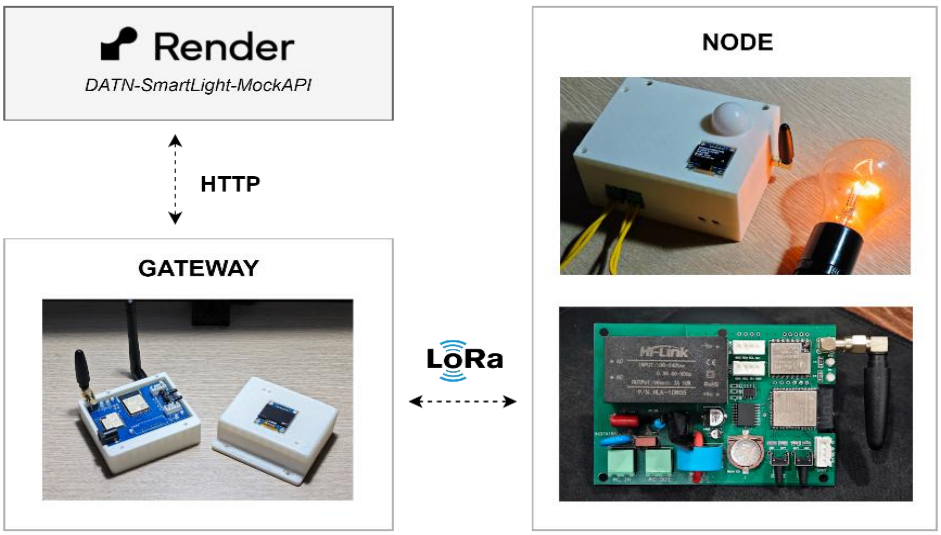
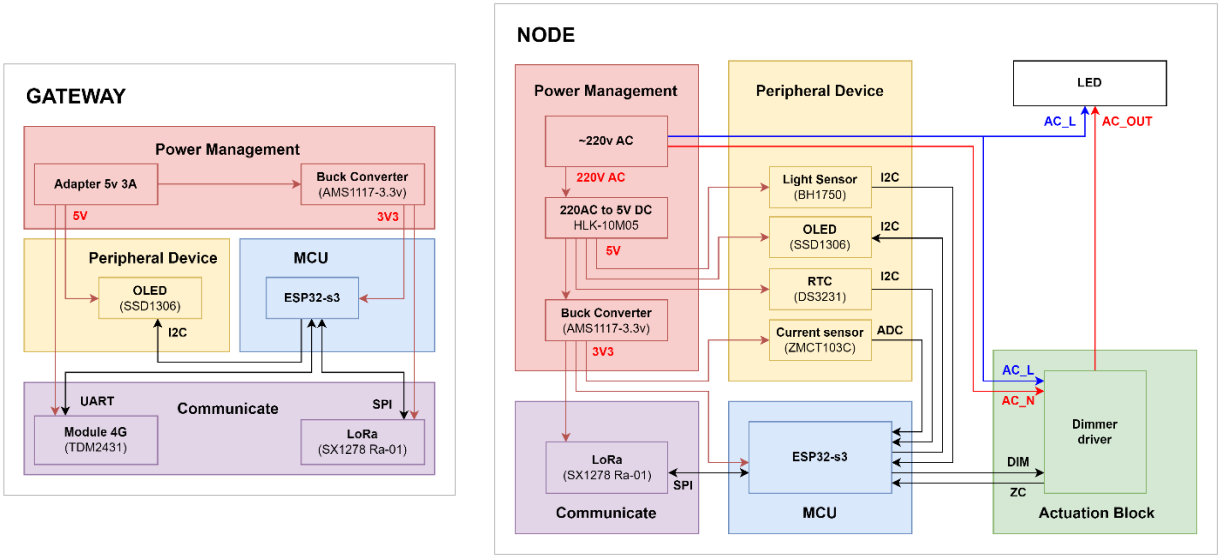


Figure 6: Hardware of the proposed smart urban lighting system

3.1 Hardware Architecture of the Lighting Node and Gateway

The proposed smart urban lighting system employs a distributed hardware architecture consisting of lighting nodes deployed at lamp locations and a central gateway responsible for system-level coordination. While both components are built around embedded microcontrollers, their hardware designs are optimized for distinct functional roles.



(a) Gateway hardware (b) Lighting node hardware

Figure 7: Block-level hardware architecture of the proposed system

Figure 7 illustrates the hardware architecture of the gateway and lighting node, integrating sensing, communication, power management, and AC dimming. The lighting node is an autonomous embedded device responsible for local sensing, real-time brightness control, and bidirectional communication with the gateway. Its hardware comprises power management, sensing, communication, an MCU, and an AC dimming block. The MCU acquires sensor data, manages communication, and generates synchronization signals for the dimmer, while dedicated circuits monitor ambient illuminance and load current. To improve deployment flexibility and maintenance, peripherals such as the ambient light sensor and OLED display are connected through external interfaces, whereas the LoRa transceiver and RTC are integrated directly on the PCB for greater mechanical reliability. A TRIAC-based AC dimmer driver provides isolated and safe control of 220 V luminaires. In contrast, the gateway performs data aggregation, system coordination, and external connectivity. It is built around an ESP32-S3 MCU, with LoRa communication via SPI, a 4G module via UART, and an OLED display via I²C for local monitoring. The gateway is powered from a 5 V adapter with onboard regulation to a stable 3.3 V supply. Overall, the hardware emphasizes communication reliability, modularity, and continuous operation, consistent with IoT smart lighting systems for smart cities and buildings [9], [10]. Schematic diagrams of the gateway and node are shown in Figs. 8 and 9.

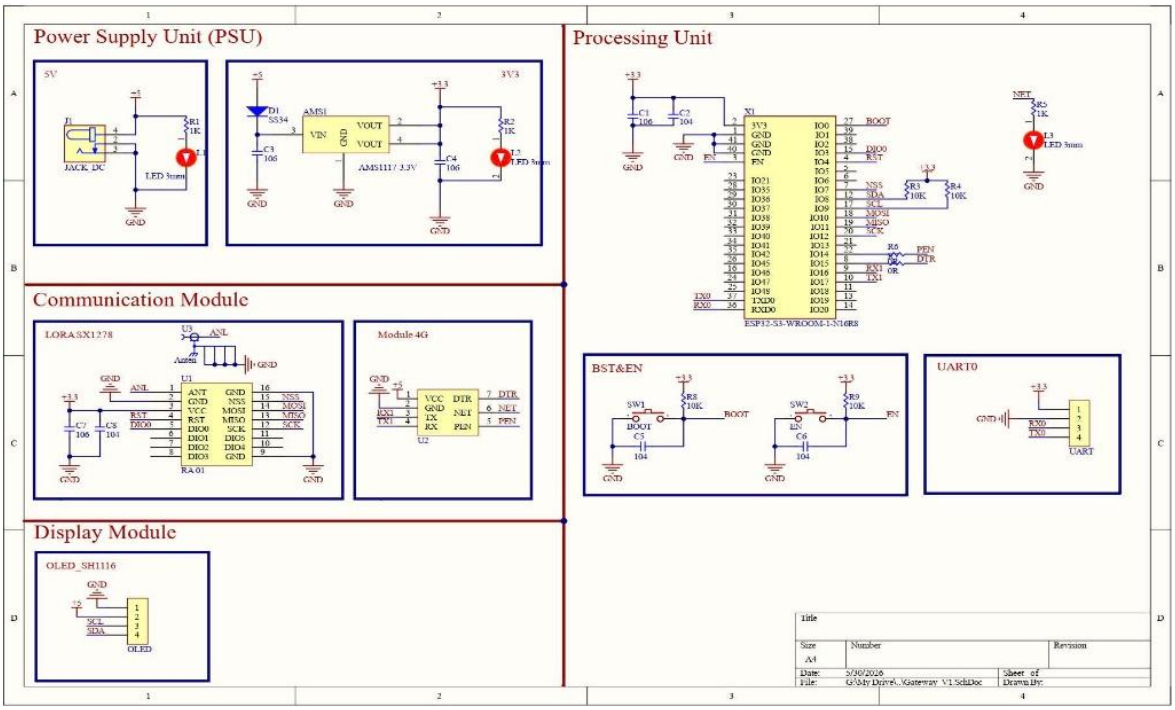


Figure 8: Schematic diagram of Gateway

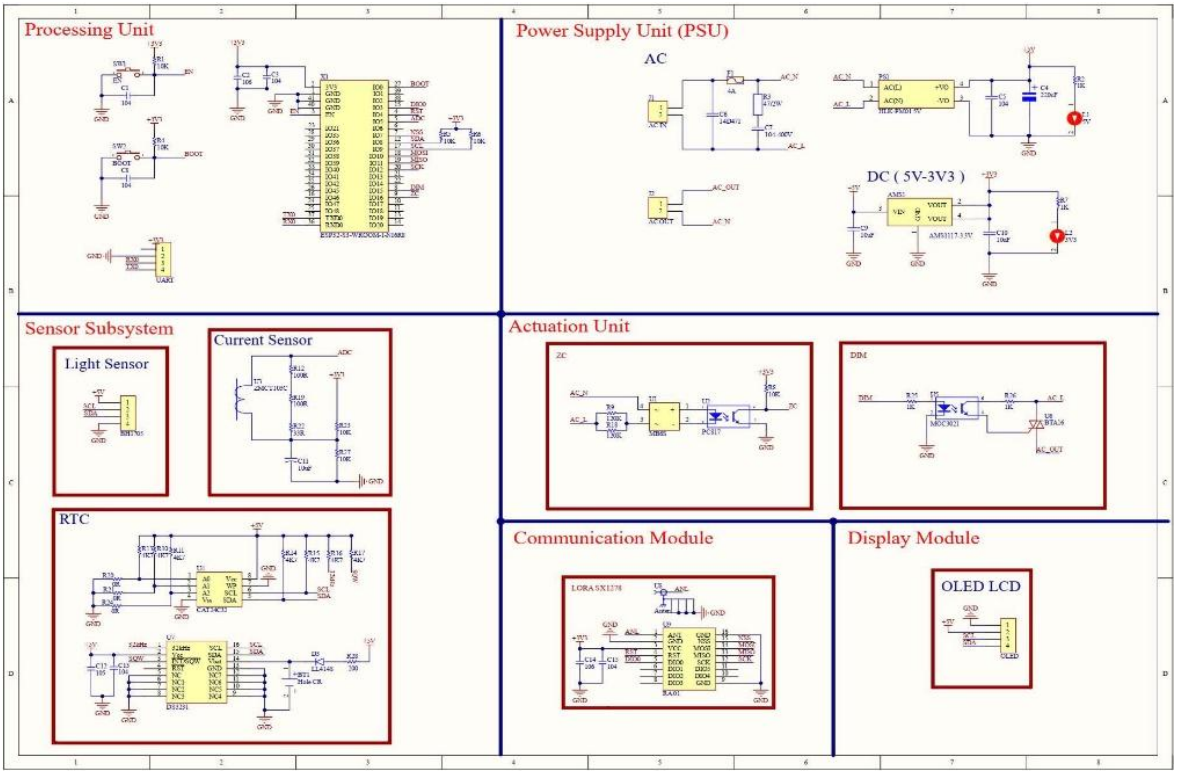
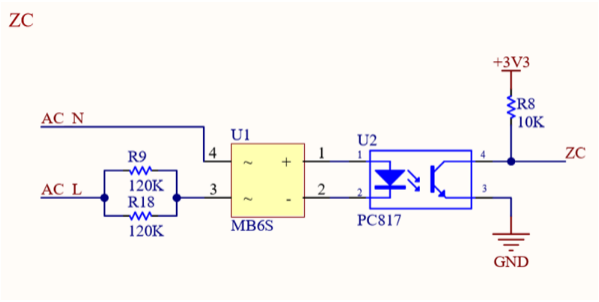


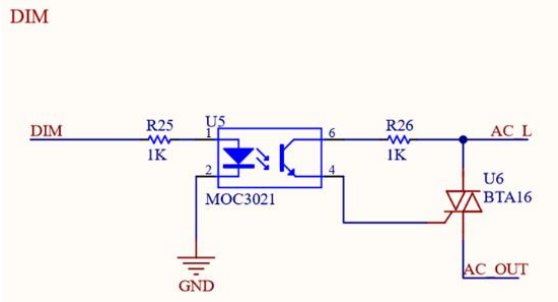
Figure 9: Schematic diagram of Node

3.2 AC Dimming Hardware Implementation

The AC dimming hardware is a key component of the lighting node, enabling stable brightness control for 220 V luminaires and providing the physical basis for the phase-angle dimming strategy described as follows:



(a) The zero-cross detection circuit



(b) TRIAC-based switching circuit for phase-angle control of 220 V luminaires

Figure 10: AC dimming hardware schematic

As shown in Fig. 10, the AC dimmer hardware is composed of two functional blocks: a zero-cross detection circuit and a TRIAC-based switching circuit. The zero-cross detection block provides synchronization signals for each AC half-cycle, which are used as timing references by the microcontroller. The switching block receives delayed control signals from the microcontroller and drives the power TRIAC to regulate the conduction angle within each half-cycle. By separating synchronization and switching into dedicated hardware paths, the dimming circuit supports repeatable and stable operation, contributing to flicker-free brightness control under real deployment conditions.

To provide a simple and lightweight control interface, the desired brightness level P (0–100%) is mapped to the TRIAC triggering delay using a linear relationship:

$$A_{delay} = (1 - \frac{P}{100}) t_{half} \tag{3}$$

For a mains frequency f = 50 Hz, the duration of one AC half-cycle is given by:

$$t_{half} = \frac{1}{2f} \tag{4}$$

3.3 Peripheral Integration and Power Considerations

Peripheral devices in the proposed system are integrated according to their functional role and long-term operational stability. Core components related to communication and timing are mounted directly on the main PCB to improve mechanical robustness and reduce connector-related failures, while selected peripherals are connected via external bus interfaces to support flexible placement and easier maintenance. This integration strategy reflects practical deployment requirements, where certain peripherals require environmental exposure or user visibility, whereas others benefit from permanent onboard mounting. Standard digital interfaces are used consistently to simplify system expansion and hardware replacement without affecting the overall architecture. From a power perspective, the lighting node employs an isolated AC–DC conversion stage to safely supply low-voltage electronics from the mains source. Local voltage regulation ensures stable operation of the microcontroller and peripheral devices, while electrical isolation between high- and low-voltage domains enhances safety and reduces susceptibility to electrical noise in outdoor environments. These design choices provide a stable hardware foundation for the experimental evaluation presented in the following section.

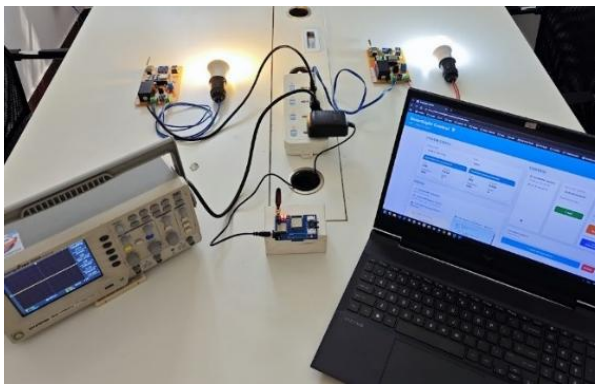
4 EXPERIMENTAL RESULTS

This section presents the experimental results obtained from the prototype smart urban lighting system. The experiments aim to evaluate the practical performance of the proposed design, focusing on AC dimming stability, communication reliability and self-recovery behavior, and adaptive lighting control using edge-based TinyML. All experiments were conducted using real hardware under realistic operating conditions.

4.1 Experimental Setup

This section describes the experimental setup used to validate the proposed SmartLight system using

physical hardware prototypes. The experimental platform consists of one gateway and multiple SmartLight nodes, each integrating an ESP32-S3 microcontroller, a LoRa communication module, an AC dimmer circuit for 220 V lighting control, and basic sensing components. Two hardware versions of the lighting node were implemented during development. The initial version was used for laboratory validation and signal-level testing, where peripheral modules were connected via pin headers to allow flexible modification and direct observation of AC dimming behavior using external measurement instruments. Based on these results, a final hardware version with fully integrated modules on a custom PCB and protective enclosure was developed and used for all subsequent experiments as in Fig. 11. In addition to indoor laboratory tests, outdoor experiments were conducted under real ambient lighting conditions to evaluate system behavior during sunrise and sunset, with a particular focus on validating the TinyML-based adaptive lighting control.



(a) Eearly hardware prototype used for signal-level testing with external measurement instruments



(b) Final hardware prototype with integrated modules used for experimental evaluation

Figure 11: Experimental setup of the SmartLight system during laboratory validation

4.2 Evaluation of AC Dimming Performance

This subsection evaluates the AC dimming performance implemented on the SmartLight node, focusing on brightness controllability, operational stability, and flicker behavior under real hardware conditions. The dimming mechanism is based on phase-angle control synchronized with AC zero-crossing, where brightness levels are adjusted by varying the TRIAC triggering delay within each half-cycle of the 220 V / 50 Hz waveform using a linear delay mapping scheme. During the experiments, dimming levels were adjusted remotely via the web interface while the lamp behavior was directly observed. Figure 12 presents the dimming performance at four representative brightness levels. The results show clear and consistent brightness variation corresponding to the selected dimming commands. Across all tested levels, including low-brightness operation, no visible flicker was observed during steady-state conditions. This confirms that the zero-cross detection and timer-based TRIAC triggering mechanism operates reliably under concurrent tasks such as communication and sensing. To further validate the dimming behavior, the load current was monitored using the integrated current sensing circuit. The relationship between brightness level, triggering delay, and measured load current is summarized in Table 2. As the brightness level decreases, the triggering delay increases and the effective conduction time is reduced, resulting in a gradual decrease in load current. These measurements are consistent with the expected behavior of phase-angle AC dimming and confirm the correctness of the delay mapping used in the implementation.

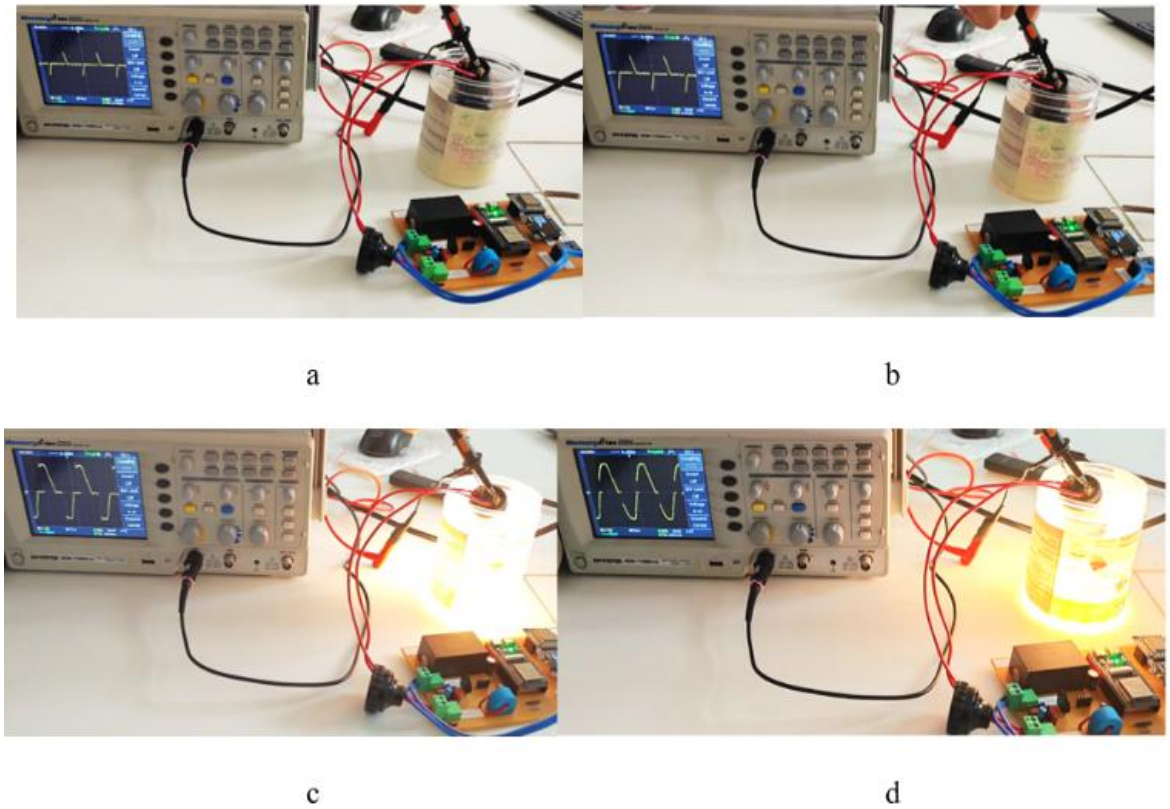


Figure 12: AC dimming performance at different brightness levels: (a) low brightness, (b) medium–low brightness, (c) medium–high brightness, and (d) full brightness.

Table 2: AC dimming delay mapping and conduction characteristics of the SmartLight Node

Brightness (%)	Trigger delay (µs)	On-time (µs)	On-Time %	Status
0%	10,000	0	0%	OFF
1%	8,500	1,500	15.00%	MIN visible
10%	7,995	2,005	20.00%	
20%	7,400	2,600	26.00%	-
30%	6,850	3,150	31.50%	-
40%	6,300	3,700	37.00%	Medium
50%	5,750	4,250	42.50%	-
60%	5,200	4,800	48.00%	-
70%	4,650	5,350	53.50%	-
85%	3,825	6,175	61.75%	-
100%	3,000	7,000	70.00%	MAX safe

4.3 Communication Reliability and Self-Recovery Performance

This subsection evaluates the reliability of the communication subsystem and the effectiveness of the self-recovery mechanism under real hardware conditions. Controlled fault scenarios were intentionally introduced by temporarily disconnecting or disabling the LoRa communication module at the node level, allowing assessment of the system's ability to detect communication failures, execute automatic recovery procedures, and resume normal operation without manual intervention. Figure 13 illustrates the self-recovery behavior of the proposed system under a communication fault. When an abnormal condition is detected through timeout monitoring, the recovery procedure is triggered automatically, reinitializing the communication module and restoring normal message exchange between the node and the gateway without resetting the microcontroller. Repeated fault injection experiments confirm that the recovery behavior is consistent and that lighting operation remains uninterrupted during the recovery process. These results

demonstrate that the proposed self-recovery mechanism enhances system robustness and supports reliable long-term operation.

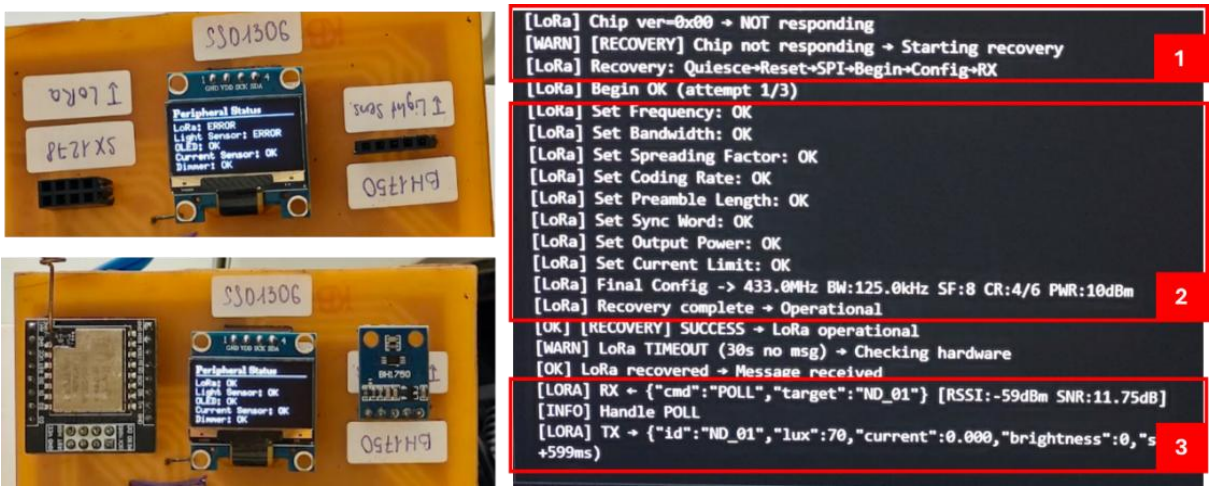
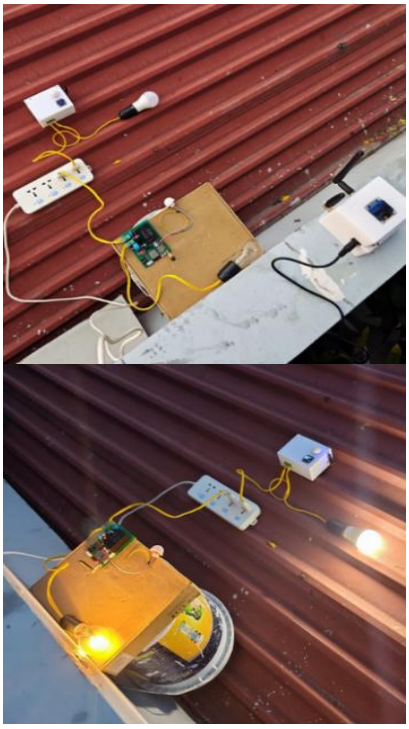


Figure 13: Experimental observation of communication fault detection and automatic self-recovery behavior in the SmartLight system:(1) communication failure detected at runtime, (2) recovery procedure executed automatically, and (3) communication restored with normal message exchange.

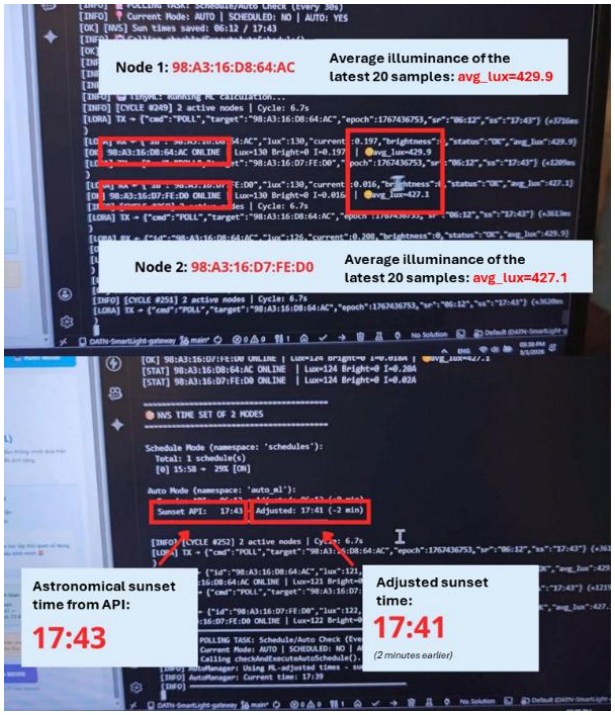
4.4 Evaluation of TinyML-Based Adaptive Lighting Control

This subsection evaluates the effectiveness of the TinyML-based adaptive lighting control implemented at the gateway. Instead of relying solely on fixed astronomical sunset schedules, the proposed approach adjusts lamp activation times based on aggregated ambient illuminance measurements from multiple lighting nodes. A lightweight TinyML model estimates a time offset relative to the API-provided sunset time, which is then applied to the existing schedule-based control. To illustrate system behavior under real operating conditions, two representative outdoor experimental cases are presented.

Figure 14 illustrates the system behavior under clear weather conditions near sunset. In this case, the astronomical sunset time provided by the API is 17:43, while the TinyML-adjusted lighting activation time is shifted slightly earlier to 17:41. Due to the relatively sufficient ambient illumination, the TinyML model generates only a minor time offset, allowing the system to closely follow the astronomical schedule while preserving adaptive behavior. Figure 15 presents a contrasting case under low-visibility conditions near sunset. Here, the astronomical sunset time is 17:44, whereas the TinyML-adjusted lighting activation time is advanced to 17:35. The significantly reduced ambient illumination leads the TinyML model to produce a larger negative time offset, enabling earlier lamp activation to improve lighting availability under unfavorable environmental conditions. Across both cases, the TinyML inference is executed reliably at the gateway without introducing observable latency or instability. The adaptive control mechanism effectively complements the astronomical schedule by dynamically adjusting lighting behavior according to real ambient conditions, while remaining lightweight and suitable for embedded gateway deployment.



(a) Outdoor experimental setup of the lighting node near sunset

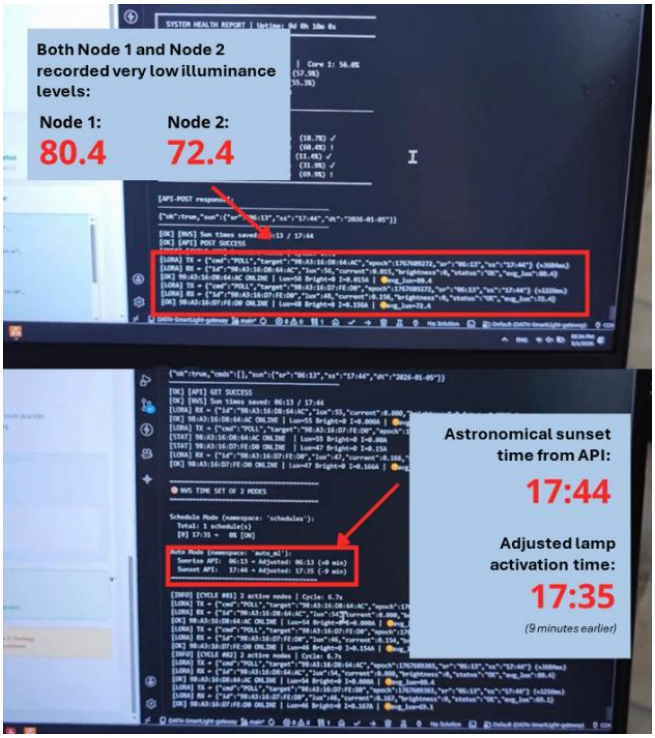


(b) Gateway log showing aggregated ambient illuminance measurements and a minor sunset time adjustment generated by the TinyML model

Figure 14: TinyML-based adjustment of sunset lighting time under clear weather conditions



(a) Outdoor experimental setup with reduced ambient illumination



(b) Gateway log showing low aggregated illuminance values and a significant sunset time adjustment generated by the TinyML model

Figure 15: TinyML-based adjustment of sunset lighting time under low-visibility conditions

The comparative discussion and TinyML suitability can be understood by comparing the proposed module with two common alternatives: fixed astronomical scheduling and cloud-based adaptive lighting as in Table 3.

Table 3: Evaluation of TinyML-Based Adaptive Lighting Control

Criterion	Fixed schedule	Cloud-based adaptation	Proposed TinyML gateway module
Local ambient-lightinput	No	Sometimes	Yes
Inference location	None	Cloud/server	Gateway edge
Network dependence	Low	High	Low
Parameters stored on device	0	Variable	6 floats
Model memory	0B	Variable	24B
Per-event compute	0	Server-side	1 dot product + clamp
Recovery from network outage	N/A	Limited	Yes, schedule can reuse cached state

The proposed module is therefore suitable for an embedded gateway because it introduces only minimal memory and compute overhead while still improving schedule accuracy under local weather variability. The current prototype still has several limitations: the TinyML module is lightweight and interpretable, but it is not a high-capacity model and therefore cannot capture all environmental factors; the evaluation is performed on a prototype-scale deployment, not a city-scale network with hundreds of nodes; and long-term energy savings and maintenance cost reduction are not yet measured over a multi-month field trial. These limitations motivate future work on richer features, larger deployments, and longer observation periods.

5 CONCLUSIONS

This paper presented a smart urban lighting system based on a distributed gateway-node architecture, with an emphasis on practical hardware design, reliability, and adaptive control. Experimental results from real hardware deployments demonstrate stable and flicker-free AC dimming, reliable long-range communication, and effective self-recovery from communication and peripheral faults without manual intervention. In addition, the edge-based TinyML module enables adaptive adjustment of lighting activation times using real ambient illuminance measurements, and case studies under different weather conditions show improved responsiveness compared with fixed astronomical schedules while maintaining lightweight computation suitable for embedded platforms. The novelty of the system lies in its deployable combination of edge-side adaptive scheduling, deterministic AC dimming, and targeted self-recovery. Comprehensive evaluations, including benchmark results for memory and inference overhead, along with quantitative comparisons against fixed scheduling, cloud-dependent adaptation, and naïve full-reset recovery, demonstrate the effectiveness of the proposed design. Overall, the proposed system provides a practical and extensible solution for smart urban lighting applications, with future work focusing on large-scale deployment, long-term performance evaluation, and further optimization of adaptive control strategies to enhance energy efficiency and scalability.

ACKNOWLEDGMENT

The authors would like to thank the Skytech Solution Technology Co., LTD, for providing facilities and technical support during the development and experimental evaluation of this research. The authors also appreciate the valuable feedback from colleagues, which contributed to the completion of this work.

REFERENCES

[1] M. Castro, A. J. Jara, and A. F. Skarmeta, Smart lighting solutions for smart cities, *Sensors*, vol. 16, no. 8, pp. 1–20, 2016.
[2] V. C. Gungor and G. P. Hancke, Industrial wireless sensor networks: Challenges, design principles, and technical approaches, *IEEE Transactions on Industrial Electronics*, vol. 56, no. 10, pp. 4258–4265, 2009.
[3] M. H. Rashid, Power Electronics: Circuits, Devices, and Applications, 4th ed. *Pearson Education*, 2014.

[4] Texas Instruments, Phase control using TRIACs, *Application Report*, 2016.
[5] P. Warden and D. Situnayake, TinyML: Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers, *O'Reilly Media*, 2020.
[6] C. Banbury et al., Benchmarking TinyML systems: Challenges and direction, *Proceedings of the IEEE*, 2021, vol. 109, no. 8, pp. 1295–1318.
[7] Semtech Corporation, LoRa modulation basics, *Technical White Paper*, 2015.
[8] A. Augustin, J. Yi, T. Clausen, and W. M. Townsley, A study of LoRa: Long range & low power networks for the Internet of Things, *Sensors*, vol. 16, no. 9, pp. 1–18, 2016.
[9] P. Chiradeja and S. Yoomak, Development of public lighting system with smart lighting control systems and Internet of Thing (IoT) technologies for smart city, *Energy Reports*, vol. 10, p. 3355–3372, 2023.
[10] W. Xu, J. Zhang, J. Y. Kim, W. Huang, S. S. Kanhere, S. K. Jha and W. Hu, The design, implementation, and deployment of a smart lighting system for smart buildings, *IEEE Internet of Things Journal*, vol. 6, no. 4, p. 7266–7281, 2019.

THIẾT KẾ VÀ THỰC HIỆN HỆ THỐNG CHIẾU SÁNG ĐÔ THỊ THÔNG MINH SỬ DỤNG TINYML THÍCH ỨNG VÀ CƠ CHẾ TỰ PHỤC HỒI KẾT NỐI LORA

NGUYỄN VĂN LÀNH*, NGUYỄN KHẮC SƠN, NGUYỄN TUẤN PHÁT
Khoa Công nghệ Điện tử, Trường Đại học Công nghiệp Thành phố Hồ Chí Minh
** Tác giả liên hệ: nguyenvanlanh@iuh.edu.vn*

Tóm tắt. Các hệ thống chiếu sáng đô thị truyền thống thường vận hành dựa trên lịch trình cố định hoặc thời gian mặt trời mọc/lặn theo thiên văn, điều này hạn chế khả năng thích ứng với thời tiết cục bộ, hỗ trợ truyền thông thực địa tin cậy và khôi phục sau khi bị các lỗi thiết bị ngoại vi. Bài báo này giới thiệu một hệ thống chiếu sáng đô thị thông minh có khả năng thích ứng, dựa trên kiến trúc phân tán gồm các nút và cổng kết nối (gateway), đồng thời tập trung vào ba đóng góp có thể định lượng được. Thứ nhất, chúng tôi triển khai một mô-đun TinyML tại phía cổng kết nối, mô-đun này chỉ sử dụng hai đầu vào đã chuẩn hóa gồm độ rọi môi trường trung bình và thời gian API đã chuẩn hóa để ước tính độ lệch thời gian cho việc bật/tắt đèn vào lúc mặt trời mọc và lặn. Thứ hai, chúng tôi hiện thực hóa cơ chế điều chỉnh độ sáng (dimming) dòng xoay chiều (AC) lai kết hợp giữa ngắt (ISR) và bộ định thời (timer), trong đó tách biệt việc phát hiện điểm không (zero-cross detection) khỏi quá trình kích hoạt TRIAC nhằm đảm bảo kiểm soát độ sáng ổn định và không bị nhấp nháy cho các đèn LED 220V. Thứ ba, chúng tôi bổ sung một cơ chế tự khôi phục truyền thông giúp phát hiện và khắc phục các lỗi LoRa và I2C mà không cần khởi động lại toàn bộ hệ thống. Mô-đun TinyML được thiết kế tối giản một cách có chủ đích: mô hình hồi quy tuyến tính được triển khai chỉ bao gồm tổng cộng sáu tham số kiểu số thực (float), chiếm 24 byte cho các trọng số và hệ số điều chỉnh (bias), đồng thời thực hiện suy luận theo dạng tích vô hướng ngay tại cổng kết nối. Quá trình kiểm chứng thực nghiệm trên phần cứng ESP32 cho thấy hệ thống đạt được khả năng truyền thông LoRa hai chiều ổn định, cơ chế điều chỉnh độ sáng hoạt động chính xác, khả năng khôi phục thiết bị ngoại vi thành công và khả năng thích ứng thời gian bật/tắt đèn theo điều kiện môi trường biến đổi. So với các phương pháp chiếu sáng theo lịch trình cố định hay điều khiển phụ thuộc vào đám mây, thiết kế được đề xuất chuyển dịch khả năng thích ứng về phía cổng kết nối (edge) và giảm thiểu chi phí tài nguyên khi vận hành, qua đó trở thành giải pháp phù hợp để triển khai cho các hệ thống chiếu sáng đô thị thông minh.
Từ khóa: Điều chỉnh độ sáng đèn AC, Tự khôi phục truyền thông, LoRa, Chiếu sáng đô thị thông minh, TinyML.

Received on August 24 – 2026
Revised on October 08 – 2026