iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** X **Month of publication:** October 2026

DOI: https://doi.org/10.22214/ijraset.2026.85561

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

UrbanGlow: An AI-Based Adaptive Street Lighting

Mrs. C Akshaya[1], Ranjithkumar M[2], Sanjith R[3], Sivaperumal K[4]
[1]*Assistant Professor,* [2, 3, 4]*UG Scholars, Department of Computer Science and Engineering, Adhiyamaan College of Engineering (An Autonomous Institution), Hosur, India*

Abstract: ***UrbanGlow is an intelligent, AI-powered streetlight management system designed to improve urban lighting efficiency, optimize energy consumption, and support smarter streetlight operation by combining environmental data, traffic patterns, weather conditions, and machine learning. Conventional streetlight systems often operate at fixed brightness levels, regardless of changes in traffic, ambient light, or weather conditions. This can lead to unnecessary energy consumption, inefficient lighting, and limited visibility into streetlight performance. UrbanGlow addresses these challenges through an integrated approach that uses data-driven predictions to support adaptive lighting decisions and energy-efficient management. The system incorporates five major modules: Traffic Prediction, Brightness Optimization, Weather-Aware Lighting, Energy Consumption Forecasting, and Monitoring and Dashboard. The Traffic Prediction module analyzes historical traffic patterns and time-related information to estimate traffic levels. The Brightness Optimization module determines suitable lighting levels using factors such as ambient light intensity, traffic density, and time of day. The Weather-Aware Lighting module considers weather conditions when recommending lighting adjustments, while the Energy Consumption Forecasting module estimates energy usage based on brightness levels, operating hours, and traffic patterns. These modules are integrated through a Python-based backend that loads trained machine learning models and processes input data, while a React-based web dashboard presents current conditions, predicted traffic, recommended brightness levels, energy forecasts, and system status in an accessible interface. The IoT-oriented architecture supports the integration of environmental sensors, traffic detection devices, and connected controllers for streetlight monitoring and control. By combining predictive analytics with centralized monitoring, UrbanGlow aims to help improve lighting decisions, reduce avoidable energy consumption, support maintenance planning, and provide a scalable foundation for intelligent streetlight management in smart cities.***

Index Terms: ***Adaptive lighting, Machine Learning, Smart City, Energy Management, Web Dashboard, Internet of Things (IoT), ESP32, LDR, PIR sensor.***

I. INTRODUCTION

A. Background

Street lighting improves night-time visibility, pedestrian movement, and road use. However, fixed-timer or constant-brightness systems cannot adapt to changes in ambient light, traffic, or weather. Reviews of smart streetlighting highlight adaptive control, communication, monitoring, and maintenance as key design concerns (Ref. 1, Ref. 2).

UrbanGlow was developed as a project prototype that brings these features together in an accessible interface. The system uses sensors and an ESP32 controller, while a web dashboard displays lighting conditions and machine-learning-based brightness recommendations. Input data is sent to a backend prediction service, which returns a suggested brightness level.

B. Objectives

The work is motivated by three practical questions. First, can ambient-light and movement sensing provide a simple local rule for day/night operation and activity-aware brightness? Second, can a lightweight web service expose a model-based brightness recommendation using interpretable contextual inputs? Third, how can the application present the recommendation without overstating energy savings that have not yet been measured in the field?

C. Contributions

The main contributions are: (1) a layered architecture that separates edge sensing, prediction service, and monitoring interface; (2) an implementation description grounded in the supplied source code and presentation; (3) an explicit distinction between implemented software behaviour and planned physical or predictive functions; and (4) a reproducible evaluation plan covering control response, model quality, power use, and reliability. This boundary-focused reporting is important because a working interface is not, by itself, proof of field energy savings.

©IJRASET: All Rights are Reserved

II. LITERATURE SURVEY

A. Smart Streetlight Architectures

Mahoor et al. review smart streetlight systems, including control strategies, communication, smart-city applications, and cybersecurity (Ref. 1). They describe centralized, decentralized, and hybrid architectures. Centralized systems support overall monitoring but rely on connectivity, while local control allows faster responses. Hybrid systems combine local decisions with network-based monitoring. UrbanGlow follows this approach conceptually, assigning sensing and lamp control to the edge and visualization and model-based recommendations to the web layer.

Omar et al. discuss motion sensing, communication, fault diagnosis, weather data, and AI-based control in intelligent streetlighting (Ref. 2). Their work highlights streetlights as connected urban assets. UrbanGlow uses a limited prototype scope, with ambient-light and motion sensing as intended physical inputs and traffic, weather, time, and LDR values as dashboard prediction features.

B. Adaptive Control and Energy Considerations

Agramelal et al. review streetlight control methods, from scheduled switching and sensor-based dimming to computer vision and learning-based approaches (Ref. 3). Rule-based control is simple and predictable, while data-driven methods can respond to more context. However, lighting must remain suitable for road and operating conditions. UrbanGlow's 0–100% brightness output is therefore a recommendation, not a guarantee of photometric compliance.

Bachanek et al. discuss intelligent street lighting as a potential energy-saving approach for cities (Ref. 4). Elejoste et al. present a wireless streetlight control system using communication and LED technology, demonstrating the value of remote management and modular design (Ref. 5). These studies motivate UrbanGlow's monitoring and extensibility, but their results cannot be directly applied because the hardware, road conditions, luminaires, traffic, and evaluation methods differ.

C. Research Gap Addressed

Existing work already establishes that adaptive lighting and remote monitoring are useful design patterns. UrbanGlow does not claim a new sensing principle or a new machine-learning algorithm. Its project-level contribution is the integration of a simple sensor-based operating concept with a small prediction API and a dashboard that exposes the contextual inputs and output. A second contribution is reporting the current implementation boundary: the provided source includes the brightness-prediction endpoint and chart endpoint, but does not include a complete database-backed telemetry pipeline, a fault-detection model, or a hardware-in-the-loop test record. These items are therefore treated as future work rather than completed experimental outcomes. Table I summarises the surveyed literature and the gap addressed.

Table I. Summary of surveyed work and identified gap

Ref.	Focus	Relationship / Limitation Addressed in UrbanGlow
[1]	Smart streetlight architectures and cybersecurity review	Motivates hybrid separation between local pole control and web monitoring
[2]	IoT intelligent street lighting and AI-assisted control survey	Supports combining ambient-light/motion sensing with contextual AI features
[3]	Classification of rule-based vs. learning-based dimming	Highlights trade-off between rule predictability and data-driven context sensitivity
[4]	Smart-city energy-saving case studies for street lighting	Motivates energy monitoring while requiring site-specific baseline calibration
[5]	Wireless LED streetlight control and modular deployment	Demonstrates remote management; requires local fallback during connectivity loss

ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue X Oct 2026- Available at www.ijraset.com

III. EXISTING SYSTEM

In conventional municipal and campus deployments, street lighting networks rely on fixed time switches, astronomical clocks, or manually configured illumination schedules. While simple to operate, these legacy systems cannot adapt to real-time environmental or traffic variations. The principal disadvantages are summarised below.

1) Fixed-schedule inefficiency. Lamps remain at full output throughout low-activity overnight hours, consuming unnecessary electrical energy when no pedestrians or vehicles are present.

2) Lack of weather and ambient adaptability. Static timers do not respond to early dusk, heavy cloud cover, rain, or fog, nor do they dim appropriately when ambient light is sufficient.

3) Absence of centralized visibility. Conventional poles operate as isolated circuits without a unified web dashboard to inspect contextual variables, recommended PWM levels, or sector-wide trends.

4) Unverifiable optimization claims in basic prototypes. Many demonstration prototypes present dashboard heuristics as achieved energy savings without separating software simulation from calibrated hardware telemetry.

IV. PROPOSED SYSTEM

A. System Overview

UrbanGlow has four main layers: field sensing and control, feature preparation, brightness recommendation, and dashboard monitoring. The intended hardware includes an ESP32, LDR, PIR sensor, and controllable lamp driver. The supplied software consists of a Flask backend and a React/TypeScript frontend. The backend loads joblib model files and provides JSON endpoints, while the frontend sends contextual inputs and displays the results. Figure 1 illustrates the logical architecture of UrbanGlow, showing the relationship between field sensing and control, feature preparation, brightness recommendation, and dashboard monitoring.

B. Field Sensing and Actuation Concept

The LDR is intended to provide a relative ambient-light reading. When daylight is present, the control concept keeps the streetlight off; after the ambient level falls below a configured threshold, the system enters a night-time state. The PIR sensor is intended to indicate movement in its field of view.

During night-time, movement can trigger a temporary increase in brightness, while the no-motion state can use a lower standby level.

This is a proposed control policy described in the project presentation; the uploaded software package does not contain ESP32 firmware, wiring diagrams, sensor calibration records, or oscilloscope/current measurements. Accordingly, the physical control sequence is specified as the target design and should be validated on the actual circuit.

C. Software Stack

The frontend uses React, TypeScript, and Vite. It includes condition cards, a brightness gauge, an optimizer panel, energy insights, and traffic, energy, and weather charts.

The backend uses Python Flask, cross-origin support, and joblib to load the models. The /api/predict_brightness endpoint accepts LDR, traffic, weather, and hour values.

The /api/chart_data endpoint provides chart data. This structure keeps the dashboard separate from the prediction logic and allows the backend to be tested independently. Table I summarises the related studies, their main focus, and the research gap addressed by UrbanGlow.

D. End-to-End Information Flow

In the intended deployment, sensors provide local readings to the controller, which can pass contextual data to the analytics service over a network. The service prepares the input features, runs the brightness model, limits the output to a valid range, and returns a recommended brightness level and status. The dashboard then displays the recommendation alongside the input values.

©IJRASET: All Rights are Reserved

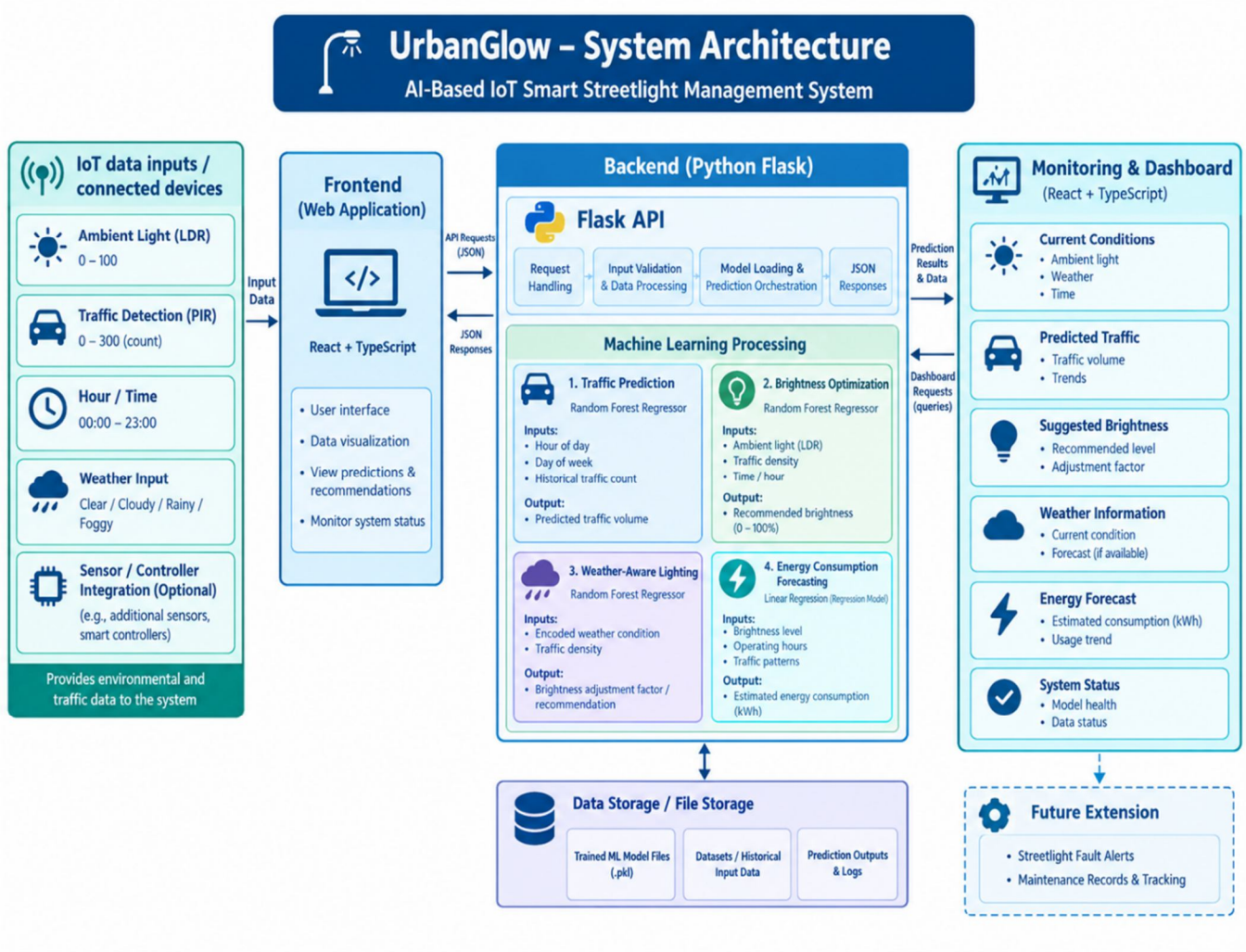


Fig. 1: UrbanGlow logical architecture.

Table II. Existing approach versus proposed UrbanGlow framework

Aspect	Conventional / Existing Approach	Proposed UrbanGlow Framework
Day/Night Switching	Fixed timer or manual relay	Ambient-light (LDR) threshold with hysteresis
Night-time Dimming	Constant 100% output all night	Standby dimming with PIR motion-triggered boost
Contextual Optimization	None	ML brightness recommendation from LDR, traffic, weather, and hour
Operator Interface	Manual field inspection	React + TypeScript web dashboard with live API integration
Resilience	Single point of failure on timer	Local ESP32 deterministic fallback if network/API is unreachable

V. METHODOLOGY AND IMPLEMENTATION DETAILS

Table III summarises the technologies and components used in UrbanGlow, along with their respective roles in the proposed framework.

Table III. UrbanGlow technology stack

Layer	Technology / Component	Role in Framework
Edge Controller	ESP32 Microcontroller	Target local sensor polling, PWM lamp control, and network telemetry
Field Sensors	LDR & PIR Sensor	Ambient light level assessment and pedestrian/vehicle movement detection
Frontend	React, TypeScript, Vite	Interactive control center, condition cards, brightness gauge, and charts
Backend API	Python, Flask, Flask-CORS	REST endpoints (/api/predict_brightness, /api/chart_data) and fallback logic
ML Inference	scikit-learn, joblib	Serialized brightness, traffic, and energy estimators with category encoders

A. Input Representation

The current API accepts four inputs: ambient light (LDR), traffic volume, weather category, and hour of day. The frontend defines their ranges as LDR 0–100, traffic 0–300 vehicles per hour, weather, and hour 0–23. The backend converts LDR and traffic to numeric values and the hour to an integer. Weather is processed using a saved encoder when available, with a fixed category mapping as a fallback.

For reliable deployment, input units and preprocessing must match those used to train the model. A production version should validate inputs, record the model version, and return clear errors for invalid data. The current API uses default values when fields are missing, which is useful for demonstration but should be improved for operational use.

B. Brightness Recommendation

The input consists of ambient light, traffic volume, encoded weather category, and hour. The loaded model uses these inputs to generate a brightness recommendation, which is constrained to a range of 0–100%. The system reports ON when the recommended brightness is above zero and OFF otherwise. The dashboard displays the result as a recommended PWM percentage. This value represents a control setting, not a direct measurement of electrical power or road illuminance. The actual relationship depends on the driver and lamp and must be measured.

C. Weather-Aware Recommendation

Weather is represented by a categorical value rather than a direct visibility measurement. The model may therefore learn associations present in its training data, but a weather label alone cannot quantify fog density, rainfall intensity, road reflectance, or actual visibility. The interface supports Clear, Cloudy, Rainy, and Foggy values, and the backend encodes this category before prediction. In a field deployment, weather data should be time-stamped, sourced reliably, and used with safety constraints so that a low model recommendation cannot override minimum lighting requirements.

D. Energy Estimate and Interpretation

The current prediction route returns an energy estimate computed in code as $\hat{E} = 1.4b + 12.5$, where b is the clipped brightness recommendation. This is an application-level heuristic, not a measured energy model: the route loads an energy-model artifact, but the supplied endpoint does not use that artifact for the returned estimate. The units and coefficients of the formula are not supported by a calibration dataset in the provided package. Consequently, the dashboard's energy number should be labelled an illustrative estimate until it is replaced by a calibrated model using measured wattage, operating duration, and a documented baseline.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

E. Traffic Prediction and Fault Monitoring Status

The backend loads a serialized traffic model, but the current brightness endpoint uses the traffic value supplied in the request rather than invoking that model to forecast traffic. Thus, the traffic-prediction function is a planned capability in the project description, not a demonstrated end-to-end prediction in the current API route. Similarly, fault detection and maintenance alerts are listed as objectives in the presentation, but the supplied backend code contains no lamp-fault classifier, telemetry database, or maintenance-ticket workflow. These functions can be integrated later by adding a time-series input pipeline, lamp-current or voltage feedback, persistence, and explicit fault rules.

F. Dashboard Rendering

The frontend renders four contextual indicators—ambient light, traffic count, weather condition, and hour of day—alongside the suggested brightness and an energy insight panel.

The chart endpoint supplies traffic, weekly energy, and weather series. In the supplied code, the chart values are fixed baseline arrays except for the current traffic and energy estimate used by the prediction response. They are useful for demonstrating the layout and interaction but are not a substitute for historical measurements. A production dashboard should clearly distinguish live telemetry, model output, and illustrative data.

G. Backend API

The Flask application initializes the model directory relative to the backend file and attempts to load brightness_model.pkl, traffic_model.pkl, energy_model.pkl, day_encoder.pkl, and weather_encoder.pkl. If a model cannot be loaded, the service records a warning and can use a fallback brightness heuristic. The prediction route checks that a JSON payload exists, parses its fields, encodes weather, invokes the brightness estimator when available, clips the result, and returns a JSON response containing success, suggested_brightness, status, input values, forecast_energy, and chart data. Exceptions are returned as a server error with a message.

For a research-grade deployment, fallback mode should be visible in the response metadata, not only in console output. Otherwise, the interface may show a valid-looking prediction without indicating that the intended model is unavailable. Recommended changes include model-version metadata, strict schema validation, unit-aware fields, request logging, timeout handling, and automated tests for missing fields, unknown weather categories, and extreme values.

H. Frontend Interaction

The React interface initializes a demonstration state, lets the operator adjust ambient light, traffic, weather, and hour, and sends these values to the backend when the optimization action is triggered. The service layer checks response status and verifies that suggested_brightness is numeric before passing the result to the components. A backend health check uses the chart endpoint to infer whether the service is reachable. This design makes it possible to test the presentation independently of physical hardware.

I. Model Artifact and Deployment Considerations

The presence of serialized model files demonstrates that model artifacts are bundled with the project, but the provided materials do not identify the training dataset, target definition, split strategy, feature distributions, hyperparameters, validation scores, or uncertainty. The models should therefore be documented with a model card containing dataset provenance, feature units, target construction, training code, evaluation protocol, version identifier, and known failure cases. Pickle/joblib artifacts should only be loaded from trusted sources because deserialization can execute code.

J. Proposed Control and Safety Policy

The control system should use the LDR to identify day and night, with a threshold and hysteresis to prevent rapid switching. During daylight, the lamp remains off according to the site policy. At night, it uses a minimum standby brightness, which increases temporarily when the PIR detects movement and gradually returns to standby when movement stops.

The controller should maintain a safe local state during sensor failure, restart, or communication loss. Machine-learning recommendations should remain within defined brightness limits and must not replace local control. The physical control sequence, hardware protection, and lighting levels require testing before public-road installation.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

K. Model Evaluation and Reproducibility

For reproducibility, future model releases should include the training script, dataset source, preprocessing steps, feature order, and evaluation results. The report should also state whether the training data came from real measurements, simulated values, or manually generated examples. Model evaluation should include prediction errors and a comparison with a simple baseline, such as fixed brightness or rule-based control. Performance should be examined across different weather, time, and traffic conditions. The current source package does not fully document the model-training process, so these details are recommended for future work.

L. Security and Future Deployment

A field deployment should use secure communication, authenticated devices, and access controls. Invalid or stale data should be rejected, while local lighting control should continue during network failure.

Motion data should be limited to what is needed for lighting and maintenance. Dashboard access should be controlled, and configuration changes should be logged. These are recommendations for future deployment, as the supplied source code does not demonstrate a complete security or fleet-management system.

M. Proposed Testing and Evaluation

Testing should begin with the sensors and controller. The LDR should be compared with a reference lux meter, and the PIR should be tested at different distances and angles. Lamp output should be measured at different brightness levels using suitable electrical and photometric instruments. The API should be tested with valid and invalid inputs, boundary values, model failure, and backend unavailability. The system should also be tested under network loss, sensor failure, and controller restart to verify that local lighting control remains safe. A field test should compare adaptive control with a fixed-brightness or scheduled baseline under comparable conditions. It should record energy use, illumination, response time, reliability, and missing data. Energy savings should only be claimed when supported by measured energy values and adequate illumination. Table IV presents the requirements traceability matrix, linking the project objectives with the evidence currently available and the additional validation required. Table II compares the conventional streetlighting approach with the proposed UrbanGlow framework in terms of day/night switching, night-time dimming, contextual optimization, operator interface, and resilience.

Table IV. Requirements traceability matrix

Objective	Current Evidence	Evidence Still Required
Adaptive day/night control	Specified in project slides; hardware code/logs not supplied.	LDR threshold calibration and day/night transition test.
Motion-aware brightness	PIR-based behavior described as target logic.	Repeated motion test with response-time and hold-time measurements.
AI brightness recommendation	Flask route calls brightness model or fallback.	Dataset provenance, holdout metrics, baseline comparison.
Traffic forecasting	Traffic model artifact loaded but not invoked by current prediction route.	Dedicated forecast endpoint and chronological evaluation.
Energy forecasting	API returns brightness-based linear estimate.	Metered power data and calibrated forecast evaluation.
Remote monitoring	React dashboard and chart API present.	Device telemetry integration, timestamps, database persistence.
Fault detection	Objective in project presentation.	Fault signals, injected-fault test set, false-alarm/miss rates.

VI. SYSTEM REQUIREMENTS

The software prototype requires a 64-bit dual-core or quad-core workstation or server running Windows 10/11, Linux (Ubuntu 20.04/22.04 LTS), or macOS, equipped with at least 4 GB RAM, Python 3.9+ (with Flask, Flask-CORS, scikit-learn, joblib, pandas, and numpy), and Node.js 18+ with Vite and TypeScript for building and serving the React dashboard. The intended field hardware layer requires an Espressif ESP32 development board, a light-dependent resistor (LDR) voltage-divider circuit for ambient-light sensing, a passive infrared (PIR) motion sensor module, a PWM-compatible LED luminaire driver with electrical isolation, and a 2.4 GHz Wi-Fi link for communicating with the Flask analytics service.

VII. RESULTS AND DISCUSSION

A. Evidence Available from the Project Artifacts

The supplied presentation contains dashboard screenshots showing current conditions, brightness recommendations, energy insights, and traffic, energy, and weather charts. The source code includes a React interface and Flask prediction endpoints.

These artifacts confirm that a software demonstration and model-serving pathway have been developed. However, they do not demonstrate continuous ESP32 sensor data transmission or field testing of streetlight hardware.

Figure 2 presents dashboard evidence extracted from the supplied UrbanGlow project presentation, illustrating the interface and its visualization components and Figure 3 shows the UrbanGlow AI dashboard, highlighting the brightness recommendation and associated operating conditions.

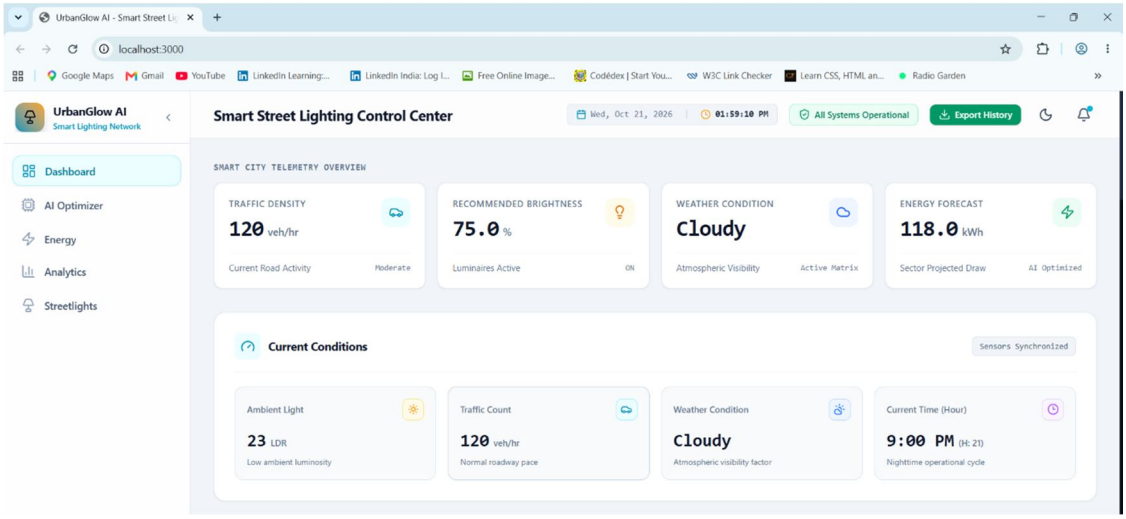


Fig. 2: Dashboard evidence extracted from the supplied UrbanGlow presentation

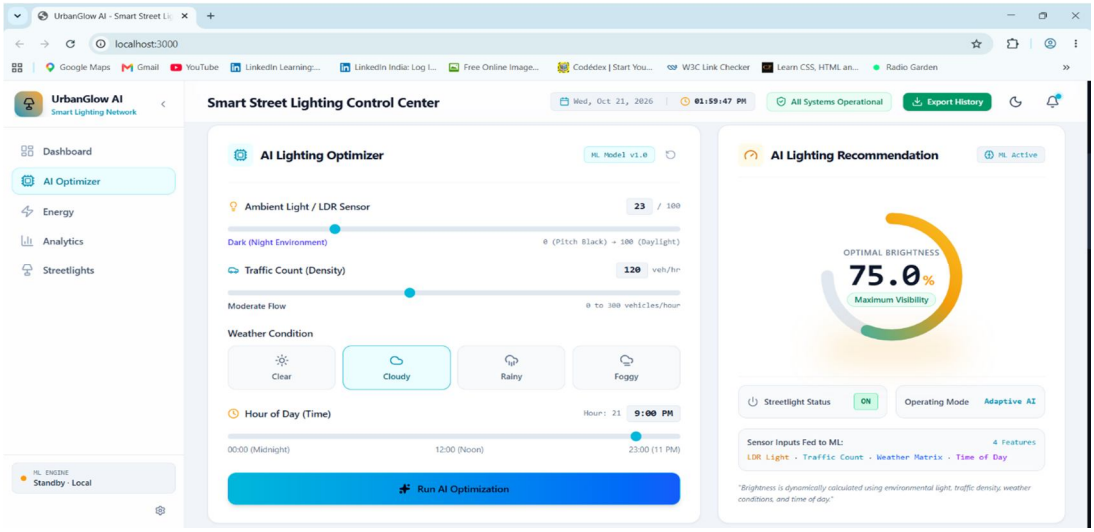


Fig. 3: UrbanGlow AI dashboard showing brightness recommendations.

B. Functional Assessment

Table V summarizes the functions supported by the supplied project artifacts and distinguishes implemented software from planned features.

Table V. Functional assessment of UrbanGlow prototype

Function	Evidence in Supplied Artifacts	Status for This Paper
Brightness recommendation	Flask route invokes the loaded brightness model or fallback heuristic; output is clipped to 0–100%.	Implemented in software; model evaluation not supplied.
Weather-aware input	Weather category is accepted and encoded before inference.	Implemented as an input feature; field benefit unmeasured.
Traffic forecasting	Traffic model artifact is loaded, but current route accepts traffic as an input.	Not demonstrated end-to-end.
Energy forecasting	Returned estimate is computed from a fixed linear expression in the route.	Illustrative heuristic; not validated.
ESP32 + LDR + PIR control	Specified in the presentation; firmware and hardware logs are absent from the uploaded ZIP.	Target architecture; hardware validation required.
Database-backed monitoring	SQL storage is mentioned in the presentation; no database integration is visible in the supplied backend.	Not verified in code package.
Fault detection and maintenance	Listed as an objective; no fault model or ticket workflow is visible in the supplied backend.	Future extension.

C. Energy Savings and Model Evaluation

Energy savings cannot be quantified without measured energy values from both a baseline and the adaptive system under comparable conditions. The dashboard's brightness-based estimate cannot replace actual power measurements because the lamp's electrical response is unknown.

A future evaluation should record power, brightness command, ambient light, motion events, and timestamps for both fixed-output and adaptive operation.

Model performance also requires a defined target and test dataset. Metrics such as MAE, RMSE, and R^2 can evaluate brightness predictions, while traffic forecasting requires a suitable held-out dataset. The current materials do not provide ground-truth test data or evaluation results, so no model accuracy values are reported.

D. Dashboard Interpretation

The dashboard presents contextual inputs alongside the brightness recommendation, helping users understand how the input values influence the output. Its cards, gauge, and charts provide a convenient way to inspect the system.

However, demonstration charts and estimated values may be mistaken for live measurements. Brightness percentages may also be interpreted as energy savings. To avoid confusion, the interface should label values as live measurements, model predictions, simulated examples, or unavailable data.

The dashboard screenshot demonstrates the interface design only; it does not represent experimental field results.

E. Comparison with Related Approaches

Compared with timer-based lighting, UrbanGlow's concept can consider ambient light and movement. Compared with sensor-only control, its software layer provides model-assisted recommendations and a central dashboard. Its intended local ESP32 control can also preserve basic lighting behaviour during network loss.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

These are architectural differences, not proven performance improvements. The prototype uses a limited set of inputs, which makes it suitable for demonstration but restricts what can be inferred. For example, PIR sensing indicates movement but does not provide an accurate traffic count.

The system must also balance energy efficiency with road safety and suitable illumination. Brightness recommendations should support decision-making rather than replace lighting standards or approved operating policies.

F. Scalability and Maintainability

The current Flask endpoint is suitable for a local prototype. A larger deployment would require reliable telemetry handling, data storage, service monitoring, and device management.

Each device should have a unique identifier and documented firmware and configuration versions. Threshold changes should be controlled, and deployment should proceed gradually to reduce the risk of widespread errors.

Maintenance should include sensor calibration, lamp inspection, and monitoring of missing data. Model performance should also be reviewed over time, particularly when seasonal conditions, traffic, or sensor placement change.

G. Reproducibility and Limitations

For reproducibility, future project releases should include source code, model artifacts, dependency versions, sensor calibration records, sample API requests, and test results. Numerical claims should be traceable to measured data or a clearly defined calculation.

The current evaluation is based on project artifacts rather than field experiments. The supplied files do not include sensor logs, lamp power measurements, a hardware test report, or a labelled model benchmark. The model-training process is also not fully documented.

The energy estimate is heuristic, the traffic model is not used by the current prediction route, and the chart endpoint returns predefined values. Planned database storage and fault detection are not demonstrated in the supplied backend.

Physical limitations also remain. PIR sensors may miss some movement, LDR readings can be affected by nearby light sources, and weather categories provide only coarse context. The control system requires calibration, minimum illumination limits, and safe fallback behaviour before deployment. Road-lighting compliance must be evaluated separately.

VIII. CONCLUSION AND FUTURE SCOPE

A. Conclusion

UrbanGlow presents a prototype framework for adaptive street lighting that combines a sensor-based control concept, a machine-learning-assisted brightness API, and a web dashboard. The supplied source code demonstrates a React/TypeScript interface, a Flask prediction service, weather encoding, brightness recommendations, and visualization components. The project presentation also describes an ESP32, LDR, and PIR-based design, along with planned traffic forecasting, energy prediction, and fault monitoring.

However, physical sensor integration, persistent telemetry storage, validated traffic forecasting, model-based energy prediction, and fault monitoring are not confirmed by the supplied code. Therefore, no numerical claims are made about energy savings or model accuracy. UrbanGlow provides a foundation for a smart-lighting demonstrator, while hardware testing, model evaluation, and reliable data collection are needed to establish its performance.

B. Future Scope

The Future development should focus on connecting the software with real hardware and collecting reliable data. This includes implementing ESP32 firmware for sensor readings and lamp control, connecting the device to the backend, and storing timestamped sensor values, brightness commands, power measurements, and device status.

The dashboard should clearly distinguish measured, simulated, and predicted values. Traffic forecasting should be evaluated using historical data, while energy prediction should be calibrated against actual power measurements and compared with fixed-brightness operation.

Weather-aware control should use reliable weather information and maintain appropriate safety limits. Fault monitoring can begin with basic checks for missing data, unusual sensor readings, and differences between commanded and measured lamp behaviour.

Finally, a multi-night campus trial should evaluate energy use, illumination, response time, missed-motion events, false activations, and system reliability.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

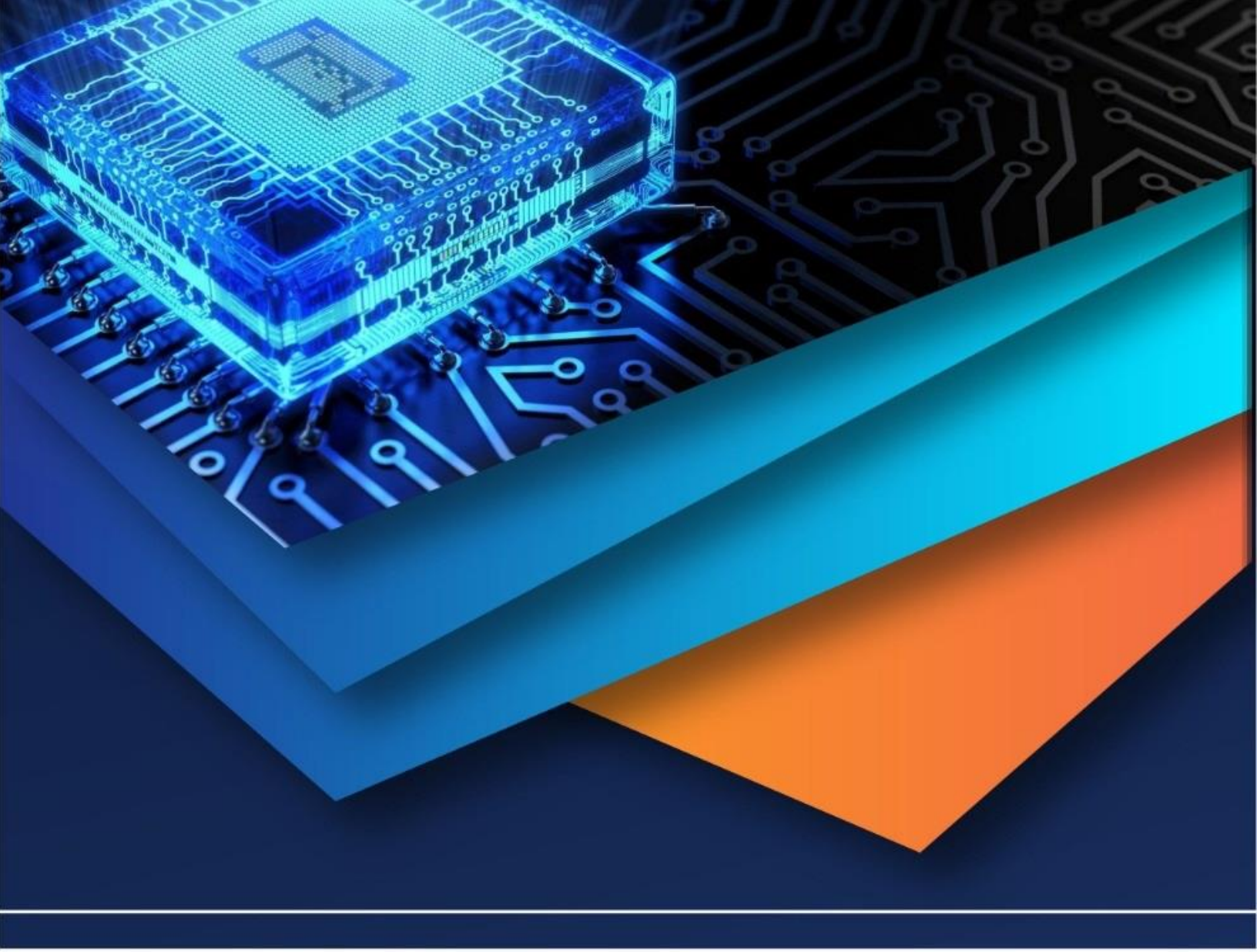
IX. ACKNOWLEDGMENT

The The authors thank Dr. R. Radhakrishnan, Principal, and Dr. G. Fathima, Professor and Head, Department of Computer Science and Engineering, Adhiyamaan College of Engineering, Hosur, for the opportunity and encouragement, and the project coordinator Mrs. C. Akshaya AP\CSE and faculty of the department for their support.

REFERENCES

[1] M. Mahoor, Z. S. Hosseini, A. Khodaei, A. Paaso, and D. Kushner, "State-of-the-art in smart streetlight systems: a review," IET Smart Cities, vol. 2, no. 1, pp. 24–33, 2020, doi: 10.1049/iet-smc.2019.0029.
[2] A. Omar, S. AlMaeeni, H. Attia, M. Takruri, A. Altunaiji, M. Sanduleanu, R. Shubair, M. S. Ashhab, M. Al Ali, and G. Al Hebsi, "Smart City: Recent Advances in Intelligent Street Lighting Systems Based on IoT," Journal of Sensors, vol. 2022, Article ID 5249187, 2022, doi: 10.1155/2022/5249187.
[3] F. Agramelal, M. Sadik, Y. Moubarak, and S. Abouzahir, "Smart Street Light Control: A Review on Methods, Innovations, and Extended Applications," Energies, vol. 16, no. 21, Article 7415, 2023, doi: 10.3390/en16217415.
[4] K. H. Bachanek, B. Tundys, T. Wiśniewski, E. Puzio, and A. Maroušková, "Intelligent street lighting in a smart city concepts—a direction to energy saving in cities: an overview and case study," Energies, vol. 14, no. 11, Article 3018, 2021, doi: 10.3390/en14113018.
[5] P. Elejoste, I. Angulo, A. Perallos, A. Chertudi, I. I. Zuazola, A. Moreno, L. Azpilicueta, J. J. Astrain, F. Falcone, and J. Villadangos, "An easy to deploy street light control system based on wireless communication and LED technology," Sensors, vol. 13, no. 5, pp. 6492–6523, 2013, doi: 10.3390/s130506492.
[6] IEEE, "IoT Based Street Light Controller and Monitoring System," in Proc. 2021 IEEE 8th International Conference on Adaptive Science and Technology (ICAST), 2021, doi: 10.1109/ICAST52759.2021.9682127.
[7] Espressif Systems, "ESP32 Series Datasheet," Espressif Systems, ESP32 Series Datasheet, Accessed: Oct. 20, 2026.
[8] Arduino, "Arduino Documentation," Arduino Documentation, Accessed: Oct. 20, 2026.
[9] Pallets Projects, "Flask Documentation," Flask Documentation, Accessed: Oct. 20, 2026.
[10] scikit-learn developers, "Model persistence," scikit-learn User Guide, scikit-learn Model Persistence, Accessed: Oct. 20, 2026.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)