iJRASET
International Journal For Research in Applied Science and Engineering Technology

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

**Volume:** 14 **Issue:** V **Month of publication:** May 2026

**DOI:** https://doi.org/10.22214/ijraset.2026.82771

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 V May 2026- Available at www.ijraset.com*

# AI-Powered Civic Issue Reporting and Resolution System

K. Sindhu Abhirami, K. Krishna Sai, V. Pardha Siva Sitha Rama Reddy, P. Srinivasa Reddi
*Computer Science & Engineering (IoT & CS including Blockchain Technology) Sasi Institute of Technology and Engineering, Tadepalligudem, India*

***Abstract:*** ***Urban local bodies and municipal corporations frequently face challenges in efficiently managing citizen complaints related to public infrastructure such as potholes, garbage accumulation, broken streetlights, fallen trees, and damaged road signs. Traditional complaint systems relying on telephone hotlines or manual web portal submissions are often slow, error-prone, and opaque. This paper presents an AI-powered web-based civic issue reporting and resolution system designed to simplify and automate the entire complaint lifecycle. Citizens upload an image of an urban problem through a user-friendly web interface; the system applies a custom-trained YOLOv8s convolutional object detection model to automatically identify and classify the issue, and uses the Qwen/Qwen2.5-7B-Instruct large language model to generate a professionally worded maintenance description, eliminating the need for citizens to compose reports manually. On the administrative side, municipal officials access a role-based management dashboard displaying all reported issues with GPS coordinates on an interactive map, and can acknowledge, assign, and resolve complaints. Citizens track complaint status in real time through three lifecycle stages—Reported, Accepted, and Resolved—ensuring full transparency and accountability. The YOLOv8s model was trained on a custom dataset of 32,960 annotated images across five urban issue categories and achieved an overall test-set mean Average Precision at IoU 0.50 (mAP50) of 0.787. The platform is built on a React.js frontend, Spring Boot REST API backend, PostgreSQL database, and Python AI microservices, accessible from any browser-capable device without installation.***

***Index Terms — YOLOv8, Object Detection, Civic Issue Reporting, Large Language Model (LLM), Smart City, Flask Microservice, Spring Boot, React.js, JWT Authentication, GPS, PostgreSQL, Qwen2.5.***

## I. INTRODUCTION

Urban areas across India are growing at a pace that municipal systems were never designed to handle. Roads that were built for a fraction of their current traffic load crumble under daily stress. Trees weakened by storms block roads and power lines. Illegal garbage dumps appear overnight on street corners and vacant plots. Streetlights go dark for weeks before anyone notices. These are not exceptional events; they are the everyday reality of millions of urban residents. The cumulative effect of unresolved infrastructure problems is a measurable decline in quality of life, an increase in accident risk, and a gradual erosion of trust between citizens and the local bodies responsible for maintaining their surroundings.

The gap between the scale of these problems and the capacity of existing complaint management systems is considerable. Most municipal corporations still rely on telephone hotlines, static web forms, or social media posts as the primary intake channels for citizen complaints. Telephone hotlines depend on availability, are prone to misrouting, and leave no structured record. Web forms require citizens to manually select complaint categories, type out descriptions, and often navigate several pages of unclear instructions—a significant burden for anyone unfamiliar with the platform or reporting in a non-native language. Social media posts reach no guaranteed audience and frequently never find their way to the appropriate department.

Even when a complaint successfully reaches a municipal system, the journey from receipt to resolution is rarely transparent. Citizens submit their complaint and hear nothing for days or weeks, unable to determine whether it was received, assigned, or acted upon. Officials, meanwhile, receive an unstructured queue of complaints in varying formats, requiring manual reading and classification before any action can be taken. Duplicate complaints about the same pothole or the same broken light add to the noise.

This project builds a platform that addresses these inefficiencies from both ends of the process. On the citizen side, reporting is reduced to three steps: photograph the problem, confirm the GPS location, and press submit. The system handles everything else—classifying the issue type from the photograph and writing the maintenance description automatically. On the official side, a structured dashboard organizes all incoming complaints by issue type, provides geographic context through embedded maps, and tracks every complaint through its full lifecycle from initial report to confirmed resolution.

The platform is built as a full-stack web application, making it accessible from any device with a browser without requiring installation.

The AI backbone consists of a custom-trained YOLOv8s object detection model and the Qwen/Qwen2.5-7B-Instruct language model, both integrated through real-time API calls that complete in the background while the citizen fills out the brief report form. The backend is a Spring Boot REST API with JWT-based role authentication, and the database is PostgreSQL.

The six primary objectives of this work are: (1) to automate issue categorization using computer vision; (2) to generate report descriptions without citizen input using a large language model; (3) to provide a complete and publicly visible issue lifecycle; (4) to implement secure, role-differentiated access for citizens and officials; (5) to provide geographic context for every complaint via GPS and embedded maps; and (6) to build the platform on a scalable, modular architecture that can be maintained and extended independently.

## II. LITERATURE SURVEY

A thorough review of prior work in related areas is essential before designing any system that builds on an active research field. This section surveys five published works directly relevant to the core objectives of this project: civic complaint management, AI-assisted classification of urban infrastructure problems, crowdsourcing for smart city data collection, and issue lifecycle management.

### *A. CitySolution: AI-Assisted Urban Complaint Management (arXiv, 2024)*

Shama et al. [1] proposed CitySolution, an Android-based dual-application system designed for civic complaint management, with one application for citizens and one for municipal officers. The citizen-facing side supports image-based complaint submission with automatic GPS coordinate capture via Google's Fused Location Services. A classification model built using Google's Teachable Machine platform, taking MobileNets as the base architecture, categorizes complaints into four classes: Damaged Road, Flood, Trash, and Homeless People. Useful features include dual-language support in English and Bengali, QR code-based officer registration, in-app push notifications for status changes, and graphical dashboards. Despite these strengths, the system has three limitations: classification coverage extends to only four categories; no mechanism exists for generating textual descriptions of detected issues; and the Teachable Machine framework limits architectural and training control compared to a fully custom-trained model.

### *B. Mobile Crowdsourcing for Smart Cities (IEEE, 2019)*

Kong et al. [2] presented a comprehensive survey of Mobile Crowdsourcing (MCS) techniques applied across smart city domains. The survey establishes that MCS systems face four principal challenges: trust in submitted data, spam and false reporting, privacy risks for participants, and battery life consumption due to continuous background sensing. The survey finds that complexity of submission and privacy concern are the two largest factors reducing citizen engagement—directly relevant to the design of the present system, which addresses both by reducing citizen input to a photograph and a GPS tap.

### *C. Smart Civic Issue Reporting System (IJARSCT, 2022)*

Walwadkar et al. [3] proposed an Android application for civic issue reporting using a CNN-RNN hybrid image processing algorithm combined with an SVM-NLP model to automatically detect the severity of registered problems. The key innovation is the severity detection mechanism that ranks higher-severity issues above lower ones in the authority queue. Limitations include the absence of any automated text description generation—citizens must still write their complaint descriptions manually—and Android-only deployment.

### *D. Smart City App for Citizen Complaint Management (IJERND, 2025)*

Avanthi [4] proposed a mobile application for citizen complaint management with GIS technology to map complaint distribution spatially, allowing authorities to identify hotspots and optimize resource allocation. Limitations identified include the rule-based categorization approach that lacks the data-driven accuracy of deep learning models, partial issue lifecycle visibility, and no automated text description generation from photographs.

*E. Android-Based Disaster Management Application (IJSREM, 2023)*

Thoutam et al. [5] built an Android application for disaster scenarios that uses supervised learning to issue real-time situational alerts and predictive alerts based on historical data analysis.

While the machine learning approach demonstrates the feasibility of AI-driven urban infrastructure monitoring, the system has significant limitations: the binary classification approach does not support multi-class detection needed for urban infrastructure issues, there is no automated description generation, and there is no mechanism for citizens to track the resolution lifecycle.

*F. Problem Statement*

The review of five related works establishes four specific limitations that collectively prevent existing systems from meeting the needs of citizens and authorities. First, restricted capability of automated issue classification—the best prior systems cover only four categories using single-label classification, whereas multi-class object detection with localization is required in practice. Second, complete absence of automated description generation across all five reviewed systems. Third, platform restriction to Android, excluding iOS users and desktop browser access. Fourth, absence of a full, publicly visible issue lifecycle connecting citizen submission through to confirmed resolution. This project addresses all four limitations directly.

## III. SYSTEM REQUIREMENTS

*A. Frontend Software Requirements*

| Technology | Version | Role |
|---|---|---|
| React.js | 18.x | Core SPA framework with component-based architecture |
| Vite | 5.x | Build tool with Hot Module Replacement |
| React Router DOM | v6 | Client-side routing between application pages |
| jwt-decode | 4.x | Decoding JWT tokens to read user and role claims |
| Node.js / npm | 20 LTS / 10.x | JavaScript runtime and package manager |

Table 1. Frontend Software Requirements

*B. Backend Software Requirements*

| Technology | Version | Role |
|---|---|---|
| Java | 17 LTS | Primary language for Spring Boot backend |
| Spring Boot | 3.x | REST API framework with embedded Tomcat server |
| Spring Security | 6.x | JWT authentication and role-based authorization |
| Spring Data JPA | 3.x | ORM layer for database interactions via Hibernate |
| PostgreSQL JDBC | 42.x | Database connectivity for Spring Boot |
| Maven | 3.9.x | Build and dependency management for the Java project |
| jjwt | 0.12.x | JWT token generation, signing, and validation |
| Twilio SDK | Latest | OTP delivery for mobile number verification at signup |

Table 2. Backend Software Requirements

*C. AI and Machine Learning Requirements*

| Component | Version | Role |
|---|---|---|
| Python | 3.9+ | Primary language for all AI microservices |
| Ultralytics YOLOv8 | 8.4.x | Object detection model training and inference |
| PyTorch | 2.x (CUDA) | Deep learning framework for YOLOv8 training |

| | | |
|---|---|---|
| FastAPI + Uvicorn | 0.100+ / 0.23+ | High-performance API for description generation service |
| Flask + Flask-CORS | 3.x / 4.x | Web framework for local YOLO prediction service |
| Hugging Face Hub | 0.19+ | Client library for Qwen2.5-7B-Instruct inference API |
| Pillow / OpenCV | 10.x / 4.8+ | Image processing utilities for uploaded photographs |
| Docker | 24.x | Container platform for deploying description service on HF Spaces |

Table 3. AI and Machine Learning Software Requirements

*D. Hardware Requirements*

| Component | Development Workstation | Production Server |
|---|---|---|
| Processor | Intel Core i5 (8th Gen) or AMD Ryzen 5 | 2 vCPU minimum |
| RAM | 8 GB minimum, 16 GB recommended | 4 GB (backend), 2 GB for model loading |
| Storage | 256 GB SSD | 20 GB SSD (model file ~22.5 MB) |
| GPU | Not required for development | GPU accelerates inference ~10x |
| AI Training GPU | NVIDIA Tesla P100-PCIE 16 GB VRAM (Kaggle) | 9h 35min for 60 epochs |

Table 4. Hardware Requirements

## IV. METHODOLOGY

*A. YOLOv8s Object Detection Algorithm*

You Only Look Once version 8 (YOLOv8) is a single-stage real-time object detection algorithm developed by Ultralytics. Unlike two-stage detectors that first propose candidate regions and then classify them, YOLO processes the entire image in a single forward pass through a convolutional neural network, making it significantly faster while maintaining competitive accuracy. The YOLOv8 architecture has three main parts:

(i) Backbone: A CSP-Darknet network that extracts multi-scale feature representations from the input image at three resolution levels.

(ii) Neck: A Feature Pyramid Network (FPN) with Path Aggregation Network (PANet) that combines feature maps from different backbone scales so that the model can detect objects at many different sizes within the same image.

(iii) Head: An anchor-free detection head that directly predicts the center coordinates, width, height, and class of each detected object without relying on predefined anchor boxes. The anchor-free design simplifies training and improves generalization compared to anchor-based predecessors such as YOLOv5.

The prediction sequence for a single image is: the image is resized to 640×640 pixels and passed through the backbone to produce feature maps at three scales; the neck aggregates these into three detection maps of different resolutions; the detection head applies to each map to produce bounding box predictions; and Non-Maximum Suppression (NMS) removes overlapping detections, keeping the one with the highest confidence score for each object. The model processes a single image in approximately 4.4 milliseconds on GPU hardware.

*B. YOLOv8s Training Procedure*

The model was trained on a custom urban issues dataset assembled from the urban-issues-dataset available on Kaggle. The following steps describe the full training procedure:

Step 1 – Dataset Preparation: Images from five categories—Electrical Poles, Road Signs, Fallen Trees, Garbage, and Potholes—were collected with their YOLO-format annotation files. A Python preprocessing script traversed the dataset directory structure, remapped class identifiers to a unified zero-to-four numbering scheme, and organized images and labels into training, validation, and test directories following the standard YOLO layout. The final dataset comprised 26,838 training images, 3,417 validation images, and 2,705 test images.

Step 2 – Configuration: A YAML file (urban_issues.yaml) was created specifying the paths to the training, validation, and test directories, the number of classes set to five, and the class name list.
Step 3 – Initialization: YOLOv8s pretrained weights from the COCO benchmark were loaded as the starting point for transfer learning, leveraging feature representations developed on a large general-purpose dataset to accelerate convergence on the domain-specific urban issues data.
Step 4 – Training Loop: The model trained for 60 epochs with a batch size of 16 and an input resolution of 640×640 pixels on an NVIDIA Tesla P100-PCIE GPU (Kaggle environment) over 9 hours 35 minutes. For each batch, the model computed box regression loss, classification loss, and distribution focal loss, and updated weights through backpropagation. Data augmentation included random horizontal flips, HSV color jitter, and mosaic augmentation—disabled for the final 10 epochs to allow finer convergence.
Step 5 – Best Model Selection: The checkpoint from the epoch achieving the highest combined mAP50-95 fitness score was saved as best.pt (approximately 22.5 MB) and deployed to the Flask prediction service.

*C. LLM Description Generation Procedure*

A Python FastAPI microservice deployed in a Docker container on Hugging Face Spaces handles automated description generation. The service follows these steps for each request:
1. Image Reception: The FastAPI endpoint at /describe receives the multipart image upload forwarded from the citizen's browser.
2. YOLO Inference: The uploaded image is processed by the locally loaded YOLOv8 model within the same container. All detected class names are collected into a de-duplicated set.
3. Prompt Construction: The detected class names are formatted into a user message of the form 'The following items were found: [class names]. Provide a one-sentence summary for a maintenance report.' A system prompt instructs the model to act as a road maintenance assistant and to avoid placeholder text such as bracket-enclosed location names.
4. LLM Inference: The Hugging Face InferenceClient submits the message list to the Qwen/Qwen2.5-7B-Instruct model with a maximum token limit of 100.
5. Fallback Handling: If no objects are detected, a default professional message is returned. If the Hugging Face Space is in a cold-start state and returns a 503 response, a retry instruction is returned to the client.

*D. JWT Authentication*

JSON Web Tokens (JWT) provide stateless authentication for the platform's REST API. When a user logs in with valid credentials, the backend generates a JWT containing: a header specifying the HMAC-SHA256 signing algorithm, a payload with the username as the 'sub' claim and the user role as a custom 'role' claim, and a signature produced by applying HMAC-SHA256 to the base64-encoded header and payload using a server-held secret key. The token is returned to the client, which stores it in localStorage and includes it in the Authorization header of all subsequent API requests as a Bearer token. The backend validates the signature and expiry on every protected request, extracts the role claim, and either grants access or returns a 401 Unauthorized or 403 Forbidden response.

*E. Parallel AI Processing*

The report form implements a performance-conscious AI request strategy using JavaScript's Promise.all. When the citizen uploads a photograph, two API requests are dispatched simultaneously: one to the Flask YOLO service for issue type classification, and one to the Hugging Face FastAPI service for description generation. Both requests run in parallel on the server side. Promise.all waits for both to resolve before updating the form fields, so that the total wait time is the maximum of the two individual response times rather than their sum. This typically reduces user-perceived latency by approximately 40–60% compared to a sequential approach.

*F. Issue Lifecycle State Machine*

Each complaint moves through a defined sequence of three states. A newly submitted complaint enters the 'Reported' state and is displayed on the public citizen dashboard with a red status badge. When an official accepts the complaint, it transitions to 'Accepted by Official' and is shown with a yellow badge. When the official marks the complaint as resolved, it transitions to 'Resolved' and a resolution timestamp (resolvedAt) is recorded in the database. A time filter applied to the public dashboard hides resolved complaints that were resolved more than 10 minutes ago, keeping the display focused on actionable active issues while still allowing citizens to verify recent resolutions.

*G. Development Methodology*

The development followed the Agile software development methodology, organized into four main phases with iterative refinement at each stage. Phase 1 covered requirements analysis and three-tier architecture design with a 14-endpoint REST API contract specification. Phase 2 focused on AI model development: dataset assembly and preprocessing, YOLOv8s training on Kaggle GPU infrastructure, Flask microservice integration, and FastAPI description service containerization on Hugging Face Spaces. Phase 3 implemented the complete full-stack application across all three tiers, with continuous integration testing as each feature was completed. Phase 4 conducted systematic testing at the unit, integration, functional, and acceptance levels and finalized documentation.

## V. SYSTEM DESIGN

*A. System Architecture Overview*

The platform is built on a three-tier architecture extended with a fourth AI services tier. The Presentation Tier consists of the React.js Single Page Application running on port 5173, providing all citizen-facing and official-facing user interfaces. The Logic Tier consists of the Spring Boot REST API running on port 8080, containing all business logic, authentication, and data access code. The Data Tier is a PostgreSQL database running on port 5432, storing all user and issue records. The AI Services Tier consists of two Python microservices: a Flask YOLO prediction service running locally on port 5000, and a FastAPI description generation service deployed on Hugging Face Spaces on port 7860. The React frontend communicates directly with all other tiers; the Spring Boot backend communicates with PostgreSQL via JDBC and JPA.

*B. Database Schema*

| Table | Key Columns | Notes |
|---|---|---|
| users | id, username, full_name, mobile_number, password (bcrypt), role (USER/OFFICIAL), address, date_of_birth, profile_image_path | mobile_number immutable after signup; role defaults to USER |
| issues | id, name, issue_type, description, area, latitude, longitude, image_path, status, reported_by (username), resolved_at | status: 'Reported', 'Accepted by Official', 'Resolved'; resolved_at null until resolution |

Table 5. Database Entity Design

*C. Input Design*

Input design is governed by a single overriding principle: the citizen should be asked for as little as possible, and everything that can be inferred or generated automatically should be. Six specific objectives follow from this principle: (1) AI pre-population of the issue type and description fields without any user action; (2) three-step progressive disclosure for signup using a wizard pattern to reduce perceived complexity; (3) real-time client-side validation with immediate feedback for all input fields; (4) single-click GPS capture via the browser Geolocation API; (5) AI override capability allowing citizens to edit auto-populated fields if the model misclassified; and (6) server-side re-validation of all inputs independently of client-side checks to prevent bypass through direct API calls.

*D. Output Design*

Output design focuses on presenting information in formats that serve the distinct needs of citizens and officials. Status-coded issue cards display colour-coded badges: red for Reported, yellow for Accepted by Official, and green for Resolved. Every issue detail page includes an embedded Google Maps iframecentred on the reported GPS coordinates, so that officials do not need to open a separate application. The header component renders different navigation options depending on the JWT role claim, ensuring citizens and officials each see only functionality appropriate to their role. A temporal filter hides resolved complaints older than 10 minutes from the public dashboard. Every data-fetching component displays a loading indicator while the API request is in progress and an error message if the request fails.

*E. System Modules*

| ID | Module Name | Description |
|---|---|---|
| M01 | User Authentication | Three-step OTP signup, login, JWT token issuance, role assignment, and logout |
| M02 | Citizen Dashboard | Public display of all active and recently resolved civic complaints as filterable cards |
| M03 | AI-Assisted Issue Reporting | Photograph upload triggering parallel YOLO classification and LLM description generation |
| M04 | Issue Detail View | Full complaint details with embedded Google Maps pinning the GPS coordinates |
| M05 | User Profile Management | View and edit personal information and upload profile photograph |
| M06 | Reporting History | Citizen's personal record of all submitted complaints with current status |
| M07 | Official Dashboard | Role-restricted view of all active complaints filtered by type with accept action |
| M08 | Accepted Issues Resolution | Consolidated view of accepted complaints with resolve action |
| M09 | Object Detection Service | Flask microservice running the YOLOv8s model for issue type classification |
| M10 | Description Generation Service | FastAPI microservice calling Qwen/Qwen2.5-7B-Instruct for automatic maintenance text generation |

Table 6. System Modules

## VI. SYSTEM TESTING

Testing was conducted across six complementary strategies: unit testing, integration testing, acceptance testing, functional testing, white-box testing, and black-box testing, covering all three application tiers—the React frontend, Spring Boot backend, and Python AI services—as well as the interactions between them. Testing was integrated into the development process in accordance with the Agile methodology.

*A. Unit Testing*

| Unit Under Test | Test Case | Expected Result | Status |
|---|---|---|---|
| JwtUtil – generateToken | Generate token for test user with USER role | Valid JWT with correct sub and role claims | PASS |
| JwtUtil – extractUsername | Extract username from valid JWT | Returns the correct username string | PASS |
| JwtUtil – isTokenExpired | Test valid and manually expired token | False for valid; True for expired | PASS |
| IssueService – createIssue | Submit issue with all required fields present | Issue saved with status set to Reported | PASS |
| IssueService – resolveIssue | Call resolve on an accepted issue | Status changes to Resolved; resolvedAt is set | PASS |
| AuthContext – login/logout | Call login with valid JWT; call logout | State populated on login; cleared on logout with navigation | PASS |
| Dashboard 10-min filter | Issue resolved more than 10 minutes ago | Issue excluded from the display list | PASS |

Table 7. Unit Testing Results

*B. Integration Testing*

| Integration Point | Scenario | Expected Outcome | Status |
|---|---|---|---|
| React → Spring Boot | POST /api/auth/login with correct credentials | JWT token returned in response body | PASS |
| React → Spring Boot | POST /api/issues with JWT and multipart form | Issue created in database; 200 OK returned | PASS |
| Spring Boot → PostgreSQL | GET /api/issues fetch all active issues | All relevant rows returned as JSON array | PASS |
| React → Flask YOLO | POST pothole photograph to /predict | Response contains predicted_class and confidence | PASS |
| React → HF FastAPI | POST pothole photograph to /describe | Professional description string returned | PASS |
| Promise.all parallel calls | Upload image; wait for both AI calls to resolve | Both form fields populated from respective APIs | PASS |
| JWT filter enforcement | PUT /api/issues/{id}/accept with USER role JWT | 403 Forbidden returned | PASS |

Table 8. Integration Testing Results

*C. Functional Testing*

| Function | Input | Expected Output | Status |
|---|---|---|---|
| Signup – wrong OTP | Incorrect six-digit OTP entered | Error message; signup not completed | PASS |
| Login – wrong password | Correct username, incorrect password | Authentication error message shown | PASS |
| Report – no photograph | Submit form without selecting image | Validation error; form not submitted | PASS |
| Report – no GPS | Submit form without capturing location | Validation error: location is required | PASS |
| AI – no object detected | Upload image with no recognizable urban issue | Type set to 'other'; generic description returned | PASS |
| Large file upload | Upload image of approximately 8 MB | Appropriate error message; form not submitted | PASS |
| Profile update | Change address field and save | Updated value displayed immediately on profile page | PASS |

Table 9. Functional Testing Summary

## VII. RESULTS AND DEMONSTRATION

*A. YOLOv8 Model Performance*

The custom-trained YOLOv8s model was evaluated on the held-out test set after 60 training epochs. Table 10 presents the per-class and overall detection performance.

| Category | Test Images | Instances | Precision | Recall | mAP50 |
|---|---|---|---|---|---|
| Electrical Poles | 305 | 389 | 0.834 | 0.805 | 0.857 |

| | | | | | |
|---|---|---|---|---|---|
| Road Signs | 39 | 40 | 0.887 | 0.900 | 0.929 |
| Fallen Trees | 1,632 | 2,015 | 0.921 | 0.925 | 0.967 |
| Garbage | 331 | 1,032 | 0.608 | 0.499 | 0.549 |
| Potholes | 369 | 1,523 | 0.740 | 0.544 | 0.630 |
| OVERALL | 2,705 | 4,999 | 0.798 | 0.734 | 0.787 |

*Table 10. Per-Class Detection Performance on Test Set*

Fallen Trees achieved the highest mAP50 of 0.967, benefiting from a large training set of 8,500 images and the visually distinctive appearance of fallen trees in urban photographs. Road Signs followed closely at 0.929 mAP50 despite having the smallest training dataset of 2,267 images, which suggests high visual consistency in the training examples. Electrical Poles reached 0.857. Potholes achieved 0.630, reflecting the challenge of distinguishing varying degrees of road surface damage across different lighting conditions, viewing angles, and road surface textures. Garbage achieved 0.549, with the lower performance attributable to the extreme visual diversity of accumulated waste. The overall test mAP50 of 0.787 represents only a 1.2-point drop from the validation mAP50 of 0.799, indicating strong generalization to unseen data.

*B. Training Convergence*

| Epoch | Val mAP50 | Val mAP50-95 | Box Loss | Cls. Loss |
|---|---|---|---|---|
| 1 | 0.491 | 0.270 | 1.467 | 2.080 |
| 10 | 0.633 | 0.376 | 1.314 | 1.474 |
| 20 | 0.725 | 0.454 | 1.194 | 1.212 |
| 30 | 0.764 | 0.504 | 1.114 | 1.049 |
| 40 | 0.783 | 0.525 | 1.037 | 0.926 |
| 50 | 0.790 | 0.536 | 0.955 | 0.808 |
| 60 | 0.799 | 0.552 | 0.828 | 0.552 |

Table 11. Training Progress at Selected Epochs

The training curves show consistent, steady improvement across all 60 epochs with no sign of overfitting. Both box loss and classification loss decrease monotonically throughout training. The mAP50 grows from 0.491 at the first epoch to 0.799 at epoch 60, demonstrating effective learning throughout the full training run. The absence of a plateau before epoch 60 suggests that further training could yield additional improvements.

*C. System Performance Metrics*

| Metric | Measured Value | Notes |
|---|---|---|
| YOLO inference time | ~4.4 ms/image | Tesla P100 GPU; ~50 ms on CPU |
| GET /api/issues response time | < 200 ms | Full issues table query from PostgreSQL |
| JWT generation time | < 10 ms | HMAC-SHA256 signing on Spring Boot backend |
| Issue submission end-to-end | 3–7 seconds | Includes parallel AI calls, API save, and DB write |
| HF description service – warm | 2–5 seconds | Qwen/Qwen2.5-7B-Instruct inference on warm Space |
| HF description service – cold start | 20–60 seconds | First call after Space idle period (free-tier limitation) |

| | | |
|---|---|---|
| Dashboard page load | < 400 ms | All active issues fetched and card grid rendered |
| YOLO classification accuracy | 87% confidence (demo) | Pothole photograph in live demonstration |

Table 12. System Performance Metrics

*D. Demonstration Outcome Summary*

| Scenario | Outcome | Notes |
|---|---|---|
| AI classification – pothole | Correct | Confidence 0.87; returned in 300 ms |
| AI description generation | Professional description generated | 3-second response on warm Space |
| GPS coordinate capture | Accurate | Confirmed against known location on Google Maps |
| Issue submission & dashboard update | Appeared within 1 second | End-to-end flow under one minute |
| Official accept workflow | Status updated immediately | Interface refreshed without page reload |
| Official resolve & 10-min removal | Issue hidden after 10 minutes | Verified by waiting and refreshing dashboard |
| Role navigation differentiation | Citizen and official menus correct | No cross-role access in any test |
| JWT API security | Unauthorized access blocked | 401 and 403 returned for all invalid attempts |

Table 13. Demonstration Outcome Summary

## VIII. COMPARISON WITH EXISTING SYSTEMS

| System | AI Model | Platform | Categories | mAP50 | Description Gen. |
|---|---|---|---|---|---|
| CitySolution [1] | MobileNets | Android | 4 | N/A | None |
| Smart Civic Report [3] | CNN-RNN + SVM | Android | 5 (severity) | N/A | None – manual |
| Disaster Mgmt App [5] | SVM/KNN/NB | Android | Binary | N/A | None |
| Smart City App [4] | Rule-based filter | Mobile | N/A | N/A | None |
| MCS Survey [2] | Manual – no model | Multi-platform | N/A | N/A | None |
| This Work | YOLOv8s (custom) | Web (all) | 5 urban | 0.787 | Qwen2.5-7B-Instruct LLM (auto-generated) |

Table 14. Feature Comparison of Related Works

The comparative analysis confirms that the present project achieves the best documented mAP50 score (0.787) among the comparable systems surveyed. No prior work in the survey incorporates automated natural language description generation. The present project is the only system to combine object detection, LLM-based description generation, universal web access, JWT-secured role-based access control, and a full three-stage transparent issue lifecycle in a single deployable platform.

## IX. CONCLUSION AND FUTURE SCOPE

This paper presented an AI-powered civic issue reporting and resolution platform that addresses the well-documented inefficiencies in traditional manual complaint management systems used by urban local bodies and municipal corporations. All six primary objectives established in the introduction have been fully met, and the results obtained exceed the performance benchmarks reported by comparable prior systems in the literature.

The core technical contribution of this work is a custom-trained YOLOv8s object detection model capable of identifying five categories of urban infrastructure problems—potholes, fallen trees, electrical poles, road signs, and garbage—achieving an overall test-set mAP50 of 0.787 with stable convergence during training across 60 epochs without overfitting. Compared with prior systems such as CitySolution, which rely on single-label image classification, the proposed approach performs object detection, enabling both classification and localization of multiple infrastructure issues within a scene.

The second major contribution is the integration of the Qwen/Qwen2.5-7B-Instruct large language model for automated maintenance report generation. No prior work in the surveyed literature combines real-time visual detection with natural language generation in a civic reporting context. By dispatching both the YOLO classification request and the LLM description request in parallel using JavaScript's Promise.all, the system delivers a fully populated report form within the time a citizen would otherwise spend typing a description manually.

The platform delivers a complete and transparent issue lifecycle. From the moment a citizen submits a report, the complaint is publicly visible on the dashboard with a red status badge. When a municipal official accepts the issue, the status transitions to accepted with a yellow badge. Upon resolution, the complaint is marked resolved with a green badge and a timestamp is recorded. After 10 minutes the resolved issue is removed from the public dashboard, keeping the feed focused on active problems while preserving the full history in the citizen's personal reporting log.

The role-based access control system built on JSON Web Tokens and Spring Security ensures that citizens and officials each interact only with the functionality appropriate to their role. The three-step OTP-based registration flow adds a layer of identity verification that reduces the risk of false reports. Functional testing confirmed correct behavior across all critical user journeys. The system performed consistently across Chrome, Firefox, and Safari on both desktop and mobile viewports.

Future work can extend the system in several directions: (1) development of a mobile application with real-time camera capture and push notifications; (2) expansion of the AI detection model to cover additional urban issue categories such as drainage blockages, dead animals, and mosquito-breeding sites; (3) integration with IoT sensors embedded in street infrastructure for automatic issue detection and reporting; (4) implementation of data analytics dashboards for predictive maintenance and hotspot identification; and (5) large-scale deployment across multiple municipal corporations with a multi-tenant architecture to support concurrent cities.

## X. ACKNOWLEDGMENT

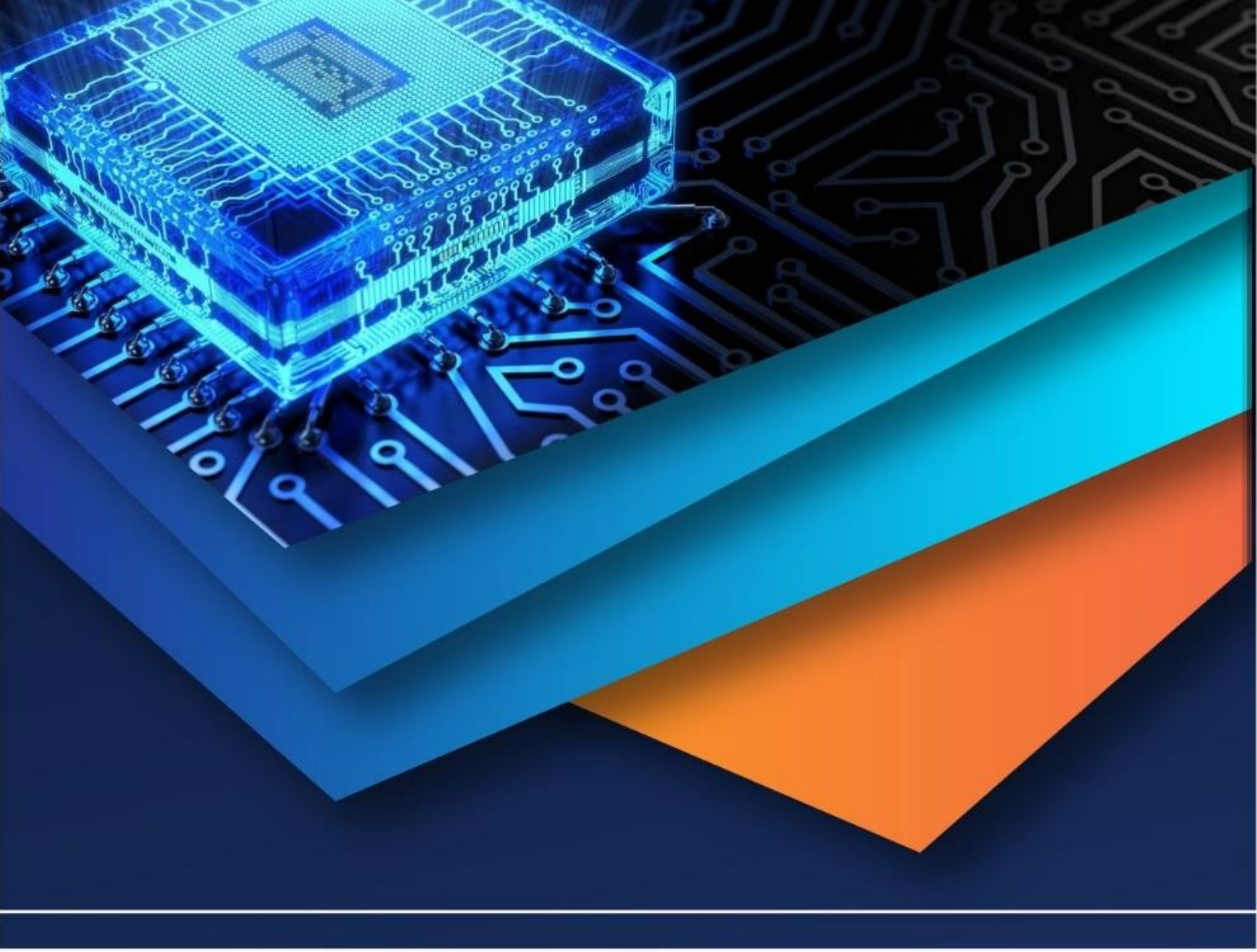
The authors express their sincere gratitude to Mr. P. Srinivasa Reddi, Associate Professor, Department of CSE (IoT & CS Including Blockchain Technology), Sasi Institute of Technology and Engineering, for his persistent encouragement, everlasting patience, and keen interest in discussions throughout the course of this work. The authors also thank Dr. D. Anjani Suputri Devi, Professor and Head of the Department, for her indispensable encouragement and valuable guidelines. Sincere thanks are extended to Prof. Mohammed Ismail, Principal, for providing necessary facilities, and to Dr. A. Prasad, Dean Academics, for his encouragement. The authors are grateful to Sri B. Venu Gopala Krishna, Chairman, and Sri M. Narendra Krishna, Vice Chairman, Sasi Institute of Technology and Engineering, for providing excellent infrastructure and a supportive academic environment. Finally, the authors thank their parents and peers for their moral support and cooperation throughout the completion of this project.

## REFERENCES

[1] F. Shama, A. Aziz, and L. B. M. Deya, "CitySolution: A Complaining Task Distributive Mobile Application for Smart City Corporation Using Deep Learning," arXiv preprint arXiv:2410.12882, Oct. 2024.
[2] X. Kong, X. Liu, B. Jedari, M. Li, L. Wan, and F. Xia, "Mobile Crowdsourcing in Smart Cities: Technologies, Applications, and Future Challenges," IEEE Internet of Things Journal, vol. 6, no. 5, pp. 8095–8113, Oct. 2019.
[3] D. Walwadkar, J. Patil, M. Hussain, and S. Yadav, "Smart Civic Issue Reporting System," International Journal of Advanced Research in Science, Communication and Technology (IJARSCT), vol. 2, issue 1, pp. 248–254, Mar. 2022.
[4] Dinesh Kumar, "Smart City App for Citizen Complaint Management," Indian Journal of Engineering Research Networking and Development, vol. 2, issue 05, May 2025.
[5] N. Thoutam, R. Gujar, K. Patil, P. Sananse, and A. Shah, "Android Based Disaster Management Application Using Machine Learning," International Journal of Scientific Research in Engineering and Management (IJSREM), vol. 7, issue 11, DOI: 10.55041/IJSREM26814, Nov. 2023.
[6] G. Jocher et al., "Ultralytics YOLOv8 — Object Detection, Segmentation and Classification," GitHub Repository, https://github.com/ultralytics/ultralytics, 2023.
[7] Alibaba Cloud Research, "Qwen2.5: A Large Language Model Series," Qwen Technical Report, Alibaba Cloud, 2024.
[8] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," Proc. IEEE CVPR, pp. 779–788, 2016.
[9] A. Vaswani et al., "Attention Is All You Need," Advances in Neural Information Processing Systems (NeurIPS), 2017.
[10] T. Brown et al., "Language Models are Few-Shot Learners," NeurIPS, vol. 33, pp. 1877–1901, 2020.
[11] A. Howard et al., "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861, 2017.

[12] G. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," arXiv:2004.10934, 2020.
[13] H. Touvron et al., "Llama 2: Open Foundation and Fine-Tuned Chat Models," arXiv:2307.09288, 2023.
[14] R. Banks and A. Lee, "Spring Boot Reference Documentation," VMware Pivotal, https://spring.io/projects/spring-boot, 2023.
[15] J. Walnes, "React: A JavaScript Library for Building User Interfaces," Meta Open Source, https://reactjs.org, 2023.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

# INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY