Open Access and Peer Review Journal ISSN 2394-2231 https://ijctjournal.org/

Urban Waste Profiling and Reporting System using Mask R-CNN Model

Aswin T Sunil Kumar *Department Of Computer Applications*
1MS24MC015
Ramaiah Institute Of Technology
Bangalore, India
Ms. Geetanjali R *Department Of Computer Applications*
Ramaiah Institute Of Technology
Bangalore, India

***Abstract*—Garbage piles up faster than any Indian municipality can deal with it. The usual approach to understanding what people throw away still involves workers tearing open bags by hand, which is both slow and genuinely hazardous. Most attempts to automate the process have relied on convolutional neural networks that slap a single label onto an entire photograph. That is adequate for a picture of one bottle on a table. It is useless for a picture of actual roadside garbage where a squashed cola can sits on top of yesterday's newspaper next to a torn polythene bag. What you really want is a system that can look at the heap and individually outline each piece of rubbish, separately. This paper describes EcoMask, a system we built around Mask R-CNN to do exactly that. We trained on the TACO litter dataset for 20,000 iterations, threw in aggressive augmentation (random flips, brightness jitter, ±15° rotations), and ended up with a bounding-box AP50 of 27.60% and a segmentation AP50 of 25.29%, with a mean inference time of 141.59 ms per frame on an NVIDIA T4 GPU. On the front end, a Flutter app lets citizens photograph waste, get disposal advice mapped to BBMP bin categories, and submit GPS-tagged reports. On the back end, a React dashboard gives municipal administrators a dual-mode heatmap with a one-kilometre radius scanner for planning targeted cleanups. The entire backend sits inside a Docker container on Hugging Face Spaces, with MongoDB Atlas handling persistence and Cloudinary hosting images so the database does not choke on base64 blobs.**

***Index Terms*—Instance Segmentation, Mask R-CNN, Solid Waste Management, Smart Cities, Deep Learning, Geospatial Analysis.**

I. Introduction

Stand at any BBMP ward boundary in Bangalore at seven in the morning. The truck comes at the same time regardless of whether the bin is overflowing or nearly empty. Residents dump plastic water bottles into the green organic bin because nobody told them otherwise at the moment it mattered. And if you ask the municipality how much recyclable material ended up contaminated last week, they have no data. None. The entire monitoring apparatus consists of a few inspectors who physically sort through bags, weigh things on a bathroom scale, and scribble numbers into a register. The process is slow, unsafe (broken glass, medical waste, decomposing organics), and fundamentally reactive. By the time anyone notices a segregation failure, it has already happened hundreds of times.

Researchers noticed this bottleneck years ago and attacked it from different angles. The IoT crowd stuck ultrasonic sensors inside bins so that fleet managers could see fill levels on a dashboard and route trucks accordingly [2], [3]. Genuinely helpful for logistics. But a full bin tells you nothing about whether its contents are properly segregated. Is it clean cardboard, or cardboard soaked in curry gravy? The sensor has no opinion.

Computer vision was the next attempt. Several groups trained image classifiers on curated datasets and reported strong numbers. Ruiz et al. got 88.66% top-1 accuracy on TrashNet using ResNet-50 [4]. Others pushed transfer learning further with VGG and MobileNet variants [5], [6]. These results are real, but they share a critical flaw: every one of these systems treats a photograph as indivisible. One picture, one label. If the picture happens to contain three different types of rubbish stacked on top of each other, which, outdoors, it almost always does. The classifier picks one winner and ignores the rest.

That gap is what led us to build EcoMask. We needed something that could stare at a messy pile, draw a separate mask around each individual item, assign each item its own category label, tell the user which BBMP bin to put it in, and log the GPS coordinates so the municipality could see contamination hotspots on a map. Mask R-CNN [11] was the obvious backbone because it performs instance segmentation at the pixel level. But the model alone does not solve the problem. The actual contribution of this work is wrapping that model into a practical, three-tier mobile-first pipeline that connects citizen reporting, real-time AI inference, and administrative analytics into a single working system. Citizens use a Flutter app. Administrators use a React web dashboard. The Flask backend, running Detectron2 inside a Docker container, sits between them.

II. Literature Review

We went through the recent literature across three areas and found a recurring theme: each wave of systems solved one piece of the puzzle while leaving the rest wide open.

A. Digitisation of Waste Workflows

Saptaputra et al. built an app called Pilahin for Indonesian waste banks [1]. It replaced handwritten ledgers with digital

forms. Participation went up, errors went down. But there was zero automation in the loop, meaning every waste category was typed in by a person. That works for a small community waste bank. Try scaling it to a city the size of Bangalore and the data entry alone becomes a bottleneck.

B. IoT Fill-Level Monitoring

Pavithra et al. wired bins with HC-SR04 ultrasonic sensors and sent readings to a cloud server over cellular links [2]. Perera et al. got creative and added weight and temperature sensors, reasoning that decomposing organic waste would register warmer than inert recyclables [3]. In theory, clever. In practice, a bin sitting in Bangalore's 35°C afternoon sun reads hot no matter what is inside. Neither system could answer the only question that matters for segregation compliance: is the stuff in this bin actually sorted correctly?

C. CNN-Based Image Classification

Ruiz et al. benchmarked TrashNet across several backbones and peaked at 88.66% with ResNet-50 [4]. Yi and Kim coupled a classifier with on-screen disposal advice and showed that the combination actually changed recycling behaviour [5]. Dedania et al. quantised a MobileNetV2 model and ran it on a Raspberry Pi for edge sorting [6]. All solid work.

But Majchrowska et al. took classifiers trained on studio-quality datasets and tested them on actual photographs of street litter, crumpled, muddy, stacked, and accuracy dropped hard [8]. The reason is obvious once you think about it. Image-level classification assumes one object per frame. Real garbage does not sit neatly in the centre of a white background.

Fuse et al. recognised this and applied Mask R-CNN to multi-object waste detection [7]. Their model handled overlapping items. But they shipped it as a web-only tool, which means a sanitation worker standing next to a bin needs a laptop and Wi-Fi to use it. Alourani et al.'s survey named the missing capability "compositional intelligence", referring to the ability to understand not just that a bin is full, but what it is full of [9]. That phrase stuck with us.

III. Problem Statement

We identified four problems. None of the systems we surveyed solves all four.

Manual auditing cannot scale. Workers open bags, sort contents by hand, and weigh materials. It is expensive. It is hazardous: needles, shattered glass, biological contamination. And it is so slow that most municipalities audit fewer than 2% of their collection routes.

IoT sensors cannot see composition. Ultrasonic and weight-based monitors are good at answering "when should we empty this bin?" They cannot answer "is this bin correctly segregated?" You cannot enforce a segregation policy without composition data.

Single-label classifiers break on real waste. Forcing a model to pick one label per photograph works in a lab. In the field, every photograph contains multiple items. The classifier either picks one winner or hedges with a confidence distribution that nobody downstream can use.

No integrated citizen pipeline exists. Some papers classify waste in isolation [4], [6]. Others detect it through a browser [7]. Others monitor fill levels [2], [3]. Nobody has stitched instance-level detection, mobile disposal guidance, GPS-tagged reporting, and administrative heatmaps into one application that a non-technical person can use while standing next to a dumpster.

EcoMask is our attempt to close all four gaps at once.

IV. Proposed Methodology and System Design

We split EcoMask into three tiers out of necessity. The Mask R-CNN model file is 351 MB. Running that on a budget Android phone would eat through RAM and kill the battery in minutes. So the phone stays thin. It captures images and displays results. The server does the heavy computation. And a separate web dashboard gives administrators the macro view.

A. System Architecture

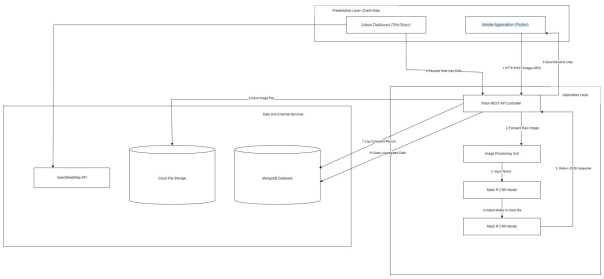

Fig. 1. Three-tier architecture of the EcoMask system, showing the Flutter client, Flask backend, and React administrative dashboard.

Mobile client (Flutter/Dart). We picked Flutter because one Dart codebase compiles to native Android and iOS binaries. The app has nine screens. The splash screen checks `SharedPreferences` for a cached JWT token and auto-navigates to the home screen if one exists, skipping the login flow entirely, a small thing, but early testers complained about having to log in every time they opened the app. The home screen shows six quick-action cards arranged in a grid (AI Scanner, Segregation Guide, Community Map are live; three others are placeholders marked "Coming Soon"). Navigation uses a `BottomAppBar` with a `CircularNotchedRectangle` and a centre-docked FAB that launches the camera. We borrowed this pattern from Google's Material 3 guidelines because it puts the primary action (scanning waste) right under the user's thumb.

The camera screen was the screen that gave us the most trouble. Early testers photographing cluttered scenes found that autofocus would hunt back and forth without locking. We fixed this with a tap-to-focus feature: tapping the viewfinder converts the pixel position to normalised 0.0–1.0 coordinates and calls both `setFocusPoint` and `setExposurePoint` on the camera controller simultaneously. A yellow animated reticle appears at the tap location (60×60 box, `Colors.amber` border), shrinks from scale 1.2 to 1.0 over 300 ms via a `TweenAnimationBuilder`, and fades after two seconds. We also added pinch-to-zoom, clamped between

`getMinZoomLevel()` and `getMaxZoomLevel()` from the camera controller. Users can alternatively pick images from the phone gallery through an `ImagePicker` widget at 85% JPEG quality, which is useful when someone has already taken a photo and wants to classify it later.

When the user captures a photo, the app sends it as a multipart POST to the `/analyze` endpoint with a Bearer token in the header. A loading dialog appears ("EcoMask AI is analyzing. . . "). Once results come back, the result screen groups detections by class using a `Map<String, int>` and displays quantities ("Plastic: Qty 3, Paper: Qty 1"). This grouping was added because of a bug we hit during testing: if somebody scanned a pile with five plastic bottles, the UI originally rendered five separate "Plastic" rows. Cluttered and confusing. We fixed it with a HashMap that consolidates duplicates and shows a count badge instead.

Below the grouped list, a "How to Dispose" section maps each detected class to BBMP's bin system: Plastic, Metal, Glass, and Paper route to the Blue dry-waste bin; Bio/Unclassified routes to the Green wet-waste bin. Tapping "Submit Report" triggers a GPS fix via the `Geolocator` package at high accuracy with a ten-second timeout, bundles everything into a JSON payload, and posts it to `/submit-report`.

The history screen (the largest file in the app at 447 lines of Dart) lets users scroll through past submissions. Each entry is reverse-geocoded using the `geocoding` package (`placemarkFromCoordinates`), so instead of raw coordinates, users see "MG Road, Bangalore, Karnataka." Tapping an entry opens a `DraggableScrollableSheet` (initial size 70% of screen, expandable to 90%) showing the full Cloudinary image, all detected items with confidence percentages, and the formatted address. Swipe-to-delete calls the backend's `/delete-report/` endpoint. We also kept backward compatibility with old records that stored images as base64 strings before we migrated to Cloudinary. The screen checks for both `image_url` and a legacy `image` field.

The heatmap screen plots the user's past reports on an OpenStreetMap layer via `flutter_map`. Default centre is `LatLng(13.0305, 77.5649)` (Bangalore). Markers show item counts. A "My Location" button calls the GPS and moves the map with an animated transition to zoom level 16.

For security, we wrote a global `AuthGuard` utility. It holds a `GlobalKey<NavigatorState>` so that any part of the app, even non-widget code like an HTTP interceptor, can trigger a redirect to the login screen. If any API call returns 401, the `handleUnauthorized` function clears all shared preferences and forces a logout. This saved us from a class of bugs where expired tokens would cause cryptic errors deep in the widget tree.

Backend (Flask/Python). The entire backend lives in a single 421-line `app.py` file. Seven route groups. We will not pretend the code is elegant. It is a monolith, deliberately. We tried splitting it into blueprints early on and spent two days fighting circular import issues with the Detectron2 predictor object. Merging everything back into one file fixed it instantly.

The AI inference endpoint (`/analyze`) works like this: it reads the uploaded file bytes with `np.frombuffer`, decodes

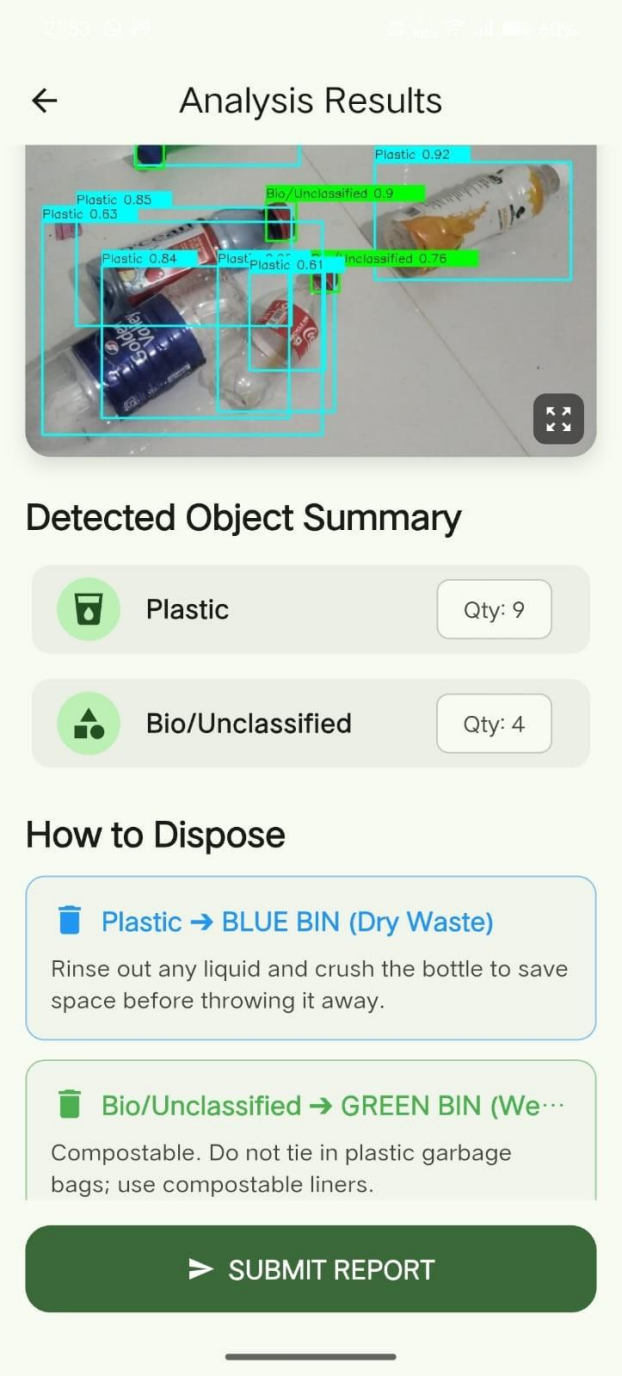

Fig. 2. EcoMask mobile interface displaying AI segmentation results and BBMP-aligned disposal guidance.

them into an OpenCV BGR array via `cv2.imdecode`, and feeds the array into Detectron2's `DefaultPredictor`. The predictor returns an `Instances` object. We iterate through `pred_classes`, `scores`, and `pred_boxes`, drawing colour-coded bounding boxes onto the original image. Each class gets a specific BGR colour tuple: Plastic is cyan `(255, 255, 0)`, Metal is magenta `(255, 0, 255)`, Paper is yellow `(0, 255, 255)`, Glass is red `(0, 0, 255)`, Bio is green `(0, 255, 0)`. Label text width is estimated crudely as `len(label) * 10` pixels, which is not pretty, but func-

tional. The annotated image is JPEG-encoded and base64-wrapped for the response.

Report submission (`/submit-report`) uploads the image to Cloudinary using its Python SDK. We originally stored raw base64 strings directly in MongoDB documents. That was a mistake we discovered the hard way: after a few hundred scans, the database had ballooned to several gigabytes, and read queries were crawling. Switching to Cloudinary, where we store only a lightweight `secure_url` and `public_id` in Mongo, fixed the bloat overnight.

JWT authentication uses PyJWT with HS256 sign-ing. The secret key defaults to a hardcoded fallback (`"ecomask_fallback_secret_key_2026"`) if the environment variable is missing. This is not ideal for production, but acceptable for a prototype. Tokens expire after 24 hours. The `@token_required` decorator extracts the Bearer token from the `Authorization` header, decodes it, and looks up the user by `ObjectId`. Password hashing uses Werkzeug's `generate_password_hash`.

Three admin endpoints power the dashboard. `/api/admin/summary` runs a MongoDB aggregation pipeline: an `$unwind` on the items array followed by a `$group` on `items.class` to get per-category counts. `/api/admin/volume` queries the last seven days of scans (`datetime.utcnow() - timedelta(days=6)`), groups them by day name using `strftime("%a")`, and maps the AI class "Bio/Unclassified" to a `wetWaste` key in the response. `/api/admin/reports` pulls the 50 most recent scans sorted by timestamp descending.

The heatmap data endpoint does something worth mentioning: it checks the user's role from the JWT payload. Admins see all scans. Regular users see only their own. This role-based filtering happens server-side via different MongoDB queries, such as `find({})` for admins versus `find({"user_id": str(current_user["_id"])})` for regular users.

Report deletion has a similar guard. Non-admin users can only delete their own reports because the query adds a `user_id` filter: `query["user_id"] = str(current_user["_id"])`. Admins get an unrestricted query.

One honest limitation we should flag: the admin-specific API endpoints (`/api/admin/summary`, `/api/admin/volume`, `/api/admin/reports`) have no server-side role verification. The access control exists only on the React frontend, which checks `data.role !== "admin"` before saving the token to `localStorage`. A determined user who manually crafted API requests with a valid non-admin token could theoretically hit these endpoints. For a prototype this is acceptable. For production it would need middleware.

Data layer (MongoDB Atlas + Cloudinary). We chose MongoDB over PostgreSQL because scan records have variable-length item arrays. One scan might contain one detected object; another might contain twelve. Storing ragged arrays of `{class, confidence}` pairs is awkward in a fixed-schema SQL table but trivial in a document store. Each scan document contains: `user_id`, `user_email`, an `items` array, `total_items_reported`, a Cloudinary `image_url` and `image_public_id`, a GeoJSON `Point` with `[lng, lat]` coordinates (longitude first. This is the GeoJSON standard, which is the opposite of what most people expect and a source of bugs we caught during testing), and a UTC timestamp.

Admin dashboard (React/Vite). The web dashboard is a separate React 19 application scaffolded with Vite, styled with Material UI, and deployed on Vercel. It has three pages behind a protected route: Overview, Live Heatmap, and Citizen Reports.

The Overview page shows four KPI summary cards (Total Scans, Items Processed, Plastic Detected, Wet Waste) and a weekly volume area chart rendered with Recharts. The chart plots two gradient-filled areas (plastic in blue and organic in green) over a seven-day window.

The heatmap page is the component we are most pleased with. It has two view modes toggled by a switch: "Density Map" and "Interactive Pins." In density mode, we render a `leaflet.heat` layer with a 30-pixel radius, 25-pixel blur, and a five-step gradient from blue through cyan, lime, and yellow to red. The cursor changes to a crosshair because clicking anywhere on the map triggers the radius scanner. The scanner calculates the Haversine distance from the click point to every scan record using Leaflet's built-in `map.distance()` function and shows a modal listing all contamination reports within a one-kilometre circle. This feature was directly inspired by how BBMP divides Bangalore into wards. Administrators think in terms of local areas, not individual pins. In pin mode, each scan appears as a coloured `CircleMarker` (radius 8, white border, 70% fill opacity) with a popup showing the waste class, confidence percentage, and a clickable image thumbnail.

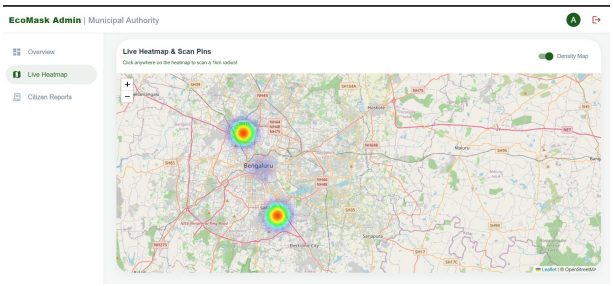

Fig. 3. The React-based administrative dashboard rendering the density heatmap and 1-kilometre radius scanner.

The Citizen Reports page renders a sticky-header table with 60×60 px image thumbnails, UTC timestamps, coordinates in monospace font to four decimal places, and coloured chips for waste categories.

We ran into a Vite compatibility issue with `@mui/icons-material`. The tree-shaking was not working correctly, and the production bundle was enormous. Our workaround was inlining SVG paths directly into `SvgIcon` components. Not the cleanest solution, but it cut the bundle size dramatically and the icons render fine.

Deployment. The backend Dockerfile builds on `python:3.10-slim`, installs CPU-only PyTorch wheels from the official index, compiles Detectron2 from Facebook's GitHub repository, and exposes port 7860. We hosted it on

Hugging Face Spaces. The React dashboard is deployed separately on Vercel.

B. Dataset and Preprocessing

We trained exclusively on TACO (Trash Annotations in Context) [10]. TrashNet would have been easier to work with. The images are clean studio shots, but that cleanliness is exactly the problem. TACO photographs are messy: litter in parks, gutters, beaches, and streets under natural lighting, with multiple items frequently overlapping. That mess is what our model needs to handle.

TACO's original annotations are fine-grained, with dozens of sub-categories like "Crisp packet," "Cigarette," and "Plastic film." We collapsed these into five macro-categories that map directly to BBMP's segregation bins: Plastic, Metal, Paper, Glass, and Bio/Unclassified. The mapping was done at the COCO JSON annotation level before feeding anything into the training pipeline.

Augmentation was aggressive. Random horizontal and vertical flips (garbage has no "correct" orientation, as a bottle on its side is still a bottle). Random rotation within ±15°. Brightness jitter between 0.7x and 1.3x of the original intensity. This was especially important because early experiments revealed the model was latching onto colour temperature rather than shape, causing it to confuse brown glass with organic waste on overcast days. Random scale jitter rounded out the augmentation set.

C. Model Configuration and Training

We used Detectron2's `mask_rcnn_R_50_FPN_3x` config: ResNet-50 backbone paired with a Feature Pyramid Network. The FPN matters because waste items span a huge range of sizes in the same frame. A cardboard box might fill half the image while a bottle cap is barely visible in the corner. Without multi-scale feature maps, the detector consistently missed smaller objects during our early experiments.

`NUM_CLASSES` was set to 5. We loaded COCO-pretrained weights as the initialisation point and fine-tuned for 20,000 iterations with a step-down learning rate scheduler. `SCORE_THRESH_TEST` sits at 0.50 in production, so anything below 50% confidence gets suppressed. We tried lowering it to 0.30, hoping to catch more items, but that introduced too many false positives. The model started "detecting" wood grain patterns on park benches as paper waste. So 0.50 it stayed.

One practical detail: in deployment, the model runs on CPU (`cfg.MODEL.DEVICE = "cpu"`) because our Hugging Face Spaces environment did not guarantee GPU availability. The inference benchmarks reported here were measured on an NVIDIA Tesla T4 to give a fair performance baseline.

D. User Workflow

The citizen opens EcoMask, gets auto-logged in if their token is still valid (the splash screen checks this via `SharedPreferences`), and taps the central scan button. The camera launches. They frame the waste, optionally zoom in with a pinch gesture or tap to lock focus, and press the shutter. The app ships the image to the backend, shows a spinner, and a few seconds later the result screen appears with grouped detections and bin-specific disposal guidance.

Tapping "Submit Report" grabs a GPS fix and sends everything to the server. The backend uploads the annotated image to Cloudinary, writes the scan record to MongoDB with GeoJSON coordinates, and sends back a confirmation. The report now shows up in the user's history screen with a reverse-geocoded address and in the admin's heatmap.

On the admin side, logging into the React dashboard triggers three parallel API calls. Summary cards populate. The volume chart draws. And the heatmap loads every georeferenced report. The administrator can toggle between a continuous density surface and individual pins, and click anywhere in density mode to pull up every report within a one-kilometre radius.

V. Experimental Results and Analysis

We evaluated the model on the TACO test split using the standard COCO evaluation protocol.

A. Quantitative Performance

After the full 20,000-iteration training run:

- **Bounding-box AP50: 27.60%.** This measures how well the model draws rectangles around objects at a 50% intersection-over-union threshold. For reference, COCO benchmarks on "clean" object categories like cars and people typically report AP50 above 50%. Waste is harder. Objects are crumpled, deformed, half-buried in mud, smeared with food residue. Getting 27.60% across five diverse categories in those conditions is a genuine signal, not noise.
- **Segmentation mask AP50: 25.29%.** A stricter metric that checks whether pixel-level masks actually trace object contours. The gap between box AP50 and mask AP50 is only 2.31 percentage points, which tells us the mask head is producing tight, shape-aware outlines rather than lazy rectangular fills.
- **Per-class breakdown.** Plastic scored highest at 35.88% AP50. Not surprising. TACO has the most plastic images, and plastic items (bottles, bags, wrappers) have distinctive shapes. Glass and Bio/Unclassified lagged behind due to fewer training examples and higher visual ambiguity. A crushed transparent glass bottle on grey concrete is hard to spot even for a human being.

Inference speed. On the T4 GPU, mean inference was 141.59 ms per image across the test set. That covers the full Detectron2 pipeline: backbone forward pass, region proposal network, ROI pooling, classification head, box regression, and mask prediction.

B. Real-World Latency

The GPU benchmark paints an optimistic picture. In actual deployment on Hugging Face Spaces, where the model runs on

CPU, the end-to-end round trip (image upload over a cellular network, server-side decoding, CPU inference, Cloudinary upload, MongoDB write, and response transmission) takes 15 to 20 seconds. We had targeted a 3-to-5 second window. We missed it. This is test case TC8 in our validation suite, and it failed. The bottleneck is CPU inference: Mask R-CNN with a ResNet-50 backbone is simply too heavy to run quickly without a GPU. Solving this would require either paying for GPU-backed hosting or compressing the model for edge deployment. Both of which we discuss in future work.

C. Edge-Case Failures

We ran a specific test (TC7) where we photographed artificial fruit, specifically a plastic banana peel, alongside a real banana peel. The model classified both as Bio/Unclassified with similar confidence. It could not tell the difference. This makes sense in retrospect: the model learned to recognise shape and colour patterns, and a well-made artificial banana looks like a real banana. Solving this would probably require either multi-spectral imaging or training on a dataset that explicitly includes synthetic imitations, neither of which we attempted.

Small, dark objects against dark backgrounds were another weak spot. A black bottle cap on asphalt often fell below the 0.50 confidence threshold and got missed entirely. Heavily crushed aluminium cans that had been flattened to near-disc shapes sometimes received correct class labels but sloppy mask boundaries, because the anchor boxes in the RPN do not expect aspect ratios that extreme.

D. Qualitative Observations

Where the model works well, it works convincingly. Spatially adjacent objects, such as a plastic bottle lying next to leaf litter, get two clean, independent masks with correct labels even when the items are physically touching. That is the whole point of instance segmentation, and it delivers.

Fig. 4. Qualitative results from the fine-tuned Mask R-CNN model, demonstrating pixel-level separation of overlapping waste items.

VI. Conclusion and Future Work

We set out to build a waste monitoring system that could do four things no existing tool handles together: look at a mixed pile, individually outline each object, tell a citizen which bin to use, and give a municipal planner a spatial map of contamination hotspots. EcoMask does all four.

The Mask R-CNN backbone, fine-tuned on TACO for 20,000 iterations, handles the hard part: pixel-level separation of overlapping objects. The Flutter app handles the human part: making the AI accessible to somebody who has never heard of instance segmentation and just wants to know where to throw their garbage. The Flask backend does the computation, manages authentication, and persists data to MongoDB and Cloudinary. The React dashboard closes the loop by turning individual scan reports into geospatial intelligence: density heatmaps, weekly volume trends, and a one-kilometre radius audit scanner that lets ward officers focus on the worst-affected neighbourhoods.

Are the numbers impressive? Honestly, not by computer vision competition standards. AP50 of 27.60% for boxes and 25.29% for masks is modest. But waste detection is fundamentally harder than detecting cars or pedestrians. Objects are amorphous. Textures are inconsistent. Occlusion is normal, not exceptional. And the model reliably separates touching objects and produces usable masks under these conditions. For a civic application, that is the result that matters.

We see four directions for future work. First, edge deployment: compressing the model through knowledge distillation or converting to TensorFlow Lite so that inference can run directly on the phone, eliminating the network dependency and the 15-to-20 second latency. We looked into ONNX Runtime Mobile but did not get far enough to report numbers. Second, video stream processing: extending from single frames to continuous video would open up applications at transfer stations and sorting facilities, though it would need a tracking layer like DeepSORT to avoid counting the same bottle twice across consecutive frames. Third, dataset rebalancing: the underrepresentation of Glass and Metal in TACO is a real bottleneck, and targeted data collection for these categories in Indian urban settings would likely push per-class AP numbers up. Fourth, gamification: the app currently computes a trivial eco-points score (`totalReports * 10`) on the client side with no backend persistence. Building a proper reward system with leaderboards, streak tracking, and redeemable incentives could sustain long-term citizen engagement, which ultimately determines whether a civic tool like this gets used or abandoned within a month.

Data Availability The custom-mapped dataset, trained Mask R-CNN model weights, and inference scripts are publicly available under the CC BY-NC 4.0 license via Kaggle, to support reproducibility and further research.

References

[1] E. H. Saptaputra, N. Bonafix, and A. S. Araffanda, “Mobile App as Digitalisation of Waste Sorting Management,” in *IOP Conf. Ser.: Earth Environ. Sci.*, vol. 1169, no. 1, p. 012007, 2023.

[2] S. Pavithra, S. Aishwarya, and V. A. Senthil, "IoT Based Automated Waste Collection System for Smart Cities," in *Proc. 2023 Int. Conf. Comput. Commun. and Informatics (ICCCI)*, Coimbatore, India, 2023, pp. 1–6.

[3] C. Perera et al., "Smart Waste Management System using IoT," in *Proc. 2022 Int. Conf. Adv. Inf. Technol. (ICAIT)*, Yangon, Myanmar, 2022, pp. 112–118.

[4] V. Ruiz, A´ . Sa´nchez, J. F. Ve´lez, and B. Raducanu, "Automatic Image-Based Waste Classification," in *From Bioinspired Systems and Biomedical Applications to Machine Learning*, vol. 11487. Cham: Springer, 2019, pp. 422–431.

[5] C. J. Yi and C. F. Kim, "AI-Powered Waste Classification Using Convolutional Neural Networks (CNNs)," *Int. J. Adv. Comput. Sci. Appl.*, vol. 15, no. 10, pp. 67–75, 2024.

[6] A. Dedania, D. Panchal, J. Patel, and J. Jani, "AI-Powered Garbage Classification System with IoT Implementation," *Int. J. Innov. Res. Multidisciplinary Phys. Sci.*, vol. 13, no. 2, Mar. 2025.

[7] S. G. Fuse, R. N. Kathkoriya, D. A. Lahane, R. S. Nagane, and K. K. Chhajed, "AI-Based Waste Classification & Reporting System," *Int. J. Res. Publ. Rev.*, vol. 6, no. 4, pp. 13817–13822, Apr. 2025.

[8] S. Majchrowska et al., "Deep learning-based waste detection in natural and urban environments," *Waste Manage.*, vol. 138, pp. 274–284, Feb. 2022.

[9] A. Alourani, M. S. Al-Shammari, and A. A. Al-Zahrani, "Smart waste management and classification system using advanced IoT and AI technologies," *PeerJ Comput. Sci.*, vol. 8, p. e1042, 2022.

[10] P. Proenc¸a and P. Simo˜es, "TACO: Trash Annotations in Context for Litter Detection," GitHub Repository, 2020.

[11] Facebook AI Research (FAIR), "Detectron2 Documentation," Facebook Research, 2019.