IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151078

Carbon FootPrint Tracking in Logistics System: AI-Based Logistics Management Platform

Bhagyashri K Kulkarni[1], Shashank Urs H P[2], Vinod[3], Sandeep T Rathod[4], Mayur Kishan Rathod[5]

Assistant Prof., Dept. of CSE, K. S. School of Engineering & Management, Bengaluru, India[1]

Student, Dept. of CSE, K. S. School of Engineering & Management, Bengaluru, India[2–5]

Abstract: A previous survey paper by the present authors proposed a web-based Carbon Footprint Tracking System for Logistics, combining ML-based emission prediction, eco-route optimization, shipment-level tracking, and future emission forecasting, to close gaps identified across six recent studies on AI-driven logistics carbon management. This paper evaluates the subsequently completed implementation, CarbonTrack, directly against its own source code, trained models, automated test suite, and deployment configuration rather than the earlier proposal's expectations. The platform is a React/Flask/SQLAlchemy application with 113 REST endpoints, 21 database tables, and six trained regression/classification models (Random Forest and XGBoost) covering fuel consumption, CO2 emission, fuel cost, eco score, predictive maintenance, and route recommendation, integrated with live Google Maps/Routes/Places and OpenWeather APIs. Re-executing the shipped training pipeline reproduced a predictive-maintenance accuracy of 87.5% (weighted F1 = 0.871) and a route-recommendation accuracy of 91.0%, with regression MAEs of 2.02 L (fuel), 3.68 kg (CO2), and 7.26 points (eco score) on an 80/20 held-out split. Re-executing the project's own 234-test suite reproduced 227 passing tests (97.0%). The paper discloses that ML training targets are generated by deterministic formulas from the same input features rather than independent real-world telemetry, and that the previously proposed emission-forecasting feature was not found implemented, so that measured results are not overstated as field-validated performance.

Keywords: Carbon Footprint Tracking, Logistics, Machine Learning, Random Forest, XGBoost, Predictive Maintenance, Route Recommendation, Fleet Management, Experimental Evaluation

I. INTRODUCTION

Road freight is a substantial contributor to transport-sector CO2 emissions, and logistics operators increasingly need shipment-level, software-accessible carbon tools [1]. A preceding survey by the present authors reviewed six studies on AI-driven emission monitoring and route optimization, identified five literature gaps, and proposed a web-based Carbon Footprint Tracking System for Logistics (CFTSL) [7]. That paper was a proposal: its architecture and expected 10–18% route-emission-reduction figures were design targets drawn from other authors' work, not measurements of a working system.

This paper reports on the completed implementation, CarbonTrack, and evaluates it on its own terms: by inspecting its source code, retraining its ML models on its own shipped datasets, and re-executing its automated test suite, tracing every reported number to the artifact that produced it. Where the completed system falls short of, differs from, or exceeds the original proposal, that difference is stated explicitly. Section II situates this work relative to the prior survey; Sections III–IV cover objectives and architecture; V–VII cover the dataset, ML methodology, and implementation; VIII–IX give the experimental setup and measured results; X–XII cover deployment, limitations, and conclusions.

II. RELATED WORK

The predecessor survey [7] compared six studies: top-down statistical emissions reporting [1]; DRL/genetic-algorithm route optimization [2]; NLP-assisted Scope-3 classification [3]; an ML benchmarking study of LSTM/Random Forest/Gradient Boosting predictors [4]; an IoT/cloud real-time monitoring framework [5]; and a comparison of classical and bio-inspired route-optimization algorithms [6]. None combined prediction, routing, tracking, and a deployable interface in one platform. This paper does not repeat that review; it treats the previous proposal as the baseline for Table X's comparison, and cites external work only where needed to name CarbonTrack's actual mechanisms precisely — for instance, distinguishing its closed-form route-scoring rule from the DRL and ant-colony methods used elsewhere [2, 6].

 This work is licensed under a Creative Commons Attribution 4.0 International License

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151078

III. PROBLEM DEFINITION AND OBJECTIVES

Two problems are addressed. First, inherited from the predecessor paper: the absence of a single, software-only, deployable platform giving small/medium logistics operators shipment-level fuel/CO2 estimation, route comparison, and fleet-health monitoring without telematics hardware. Second, specific to this paper: verifying which of the completed system's performance claims are actually true, since prior project documentation was found, on inspection, to contain figures not updated after later code changes (Section IX). The objective is to describe CarbonTrack as built and establish its results by re-execution rather than by quoting documentation.

IV. SYSTEM ARCHITECTURE

A. Overall Architecture

CarbonTrack follows the three-tier structure proposed previously — React presentation layer, Flask REST application layer, data/intelligence layer — each more extensive than proposed. The frontend is a React 19 SPA (Vite, Tailwind v4, Framer Motion, Recharts, i18next across seven locales). The backend is a Flask 3 REST API with five blueprints (auth, driver, admin, trips, external) totalling 113 routes. The data layer is a 21-table SQLAlchemy schema (SQLite locally, PostgreSQL in the Render.com deployment) plus six trained models. A design detail absent from the original proposal: predicted CO2 (planning time) comes from the ML regressor, while actual CO2 (post-trip) is computed from a versioned emission-factor table (CarbonMethodology), not re-derived from the model — separating auditable formula-based accounting from ML-based prediction.

B. Driver and Administrator Modules

Driver workflow: registration (OTP, document upload) → admin approval → login → trip creation (geocoding, route alternatives, AI prediction, confirmation) → GPS-based navigation → completion (server-enforced GPS arrival lock) → feedback; plus SOS, driver-reported deviation, reassignment requests, admin chat, and an eco-driving PDF certificate. Administrator surface: driver approval, vehicle CRUD with service-record logging, trip oversight, a simulated live-fleet map, CSV/Excel/PDF reports, a rule-based AI Insights panel, a rule-based chat assistant, a sustainability dashboard, a Continuous Learning Pipeline (manual trigger plus 24-hour scheduled retrain), a model-version registry with metric-gated promotion and rollback, role-separated SuperAdmin/Admin accounts, JSON backup/restore, an audit log, and settings.

C. AI/ML Layer and External APIs

Six artifacts are trained by ml/train.py from a merged 2,000-row dataset: RandomForestRegressor (fuel), XGBRegressor (CO2), a derived (not independently trained) fuel-cost figure, RandomForestRegressor (eco score), and two RandomForestClassifiers (maintenance risk; route recommendation). Inference validates every output against plausibility bounds before returning it, rejecting non-finite or physically implausible values. Four external services are integrated server-side (only the Maps JS key reaches the browser): Google Geocoding, Routes API v2 (real alternatives with polylines/turn-by-turn), Places API (nearby lookup), and OpenWeather.

D. End-to-End System Flowchart

Figure 1 summarizes the implemented CarbonTrack workflow, from role-based access and driver approval through trip creation, route-data retrieval, model-based prediction, route comparison, GPS monitoring, trip completion, and administrative reporting. Route labels are assigned using the implemented closed-form comparison rule; predicted missions are produced during planning, while post-trip emissions are calculated using the versioned CarbonMethodology emission factors.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151078

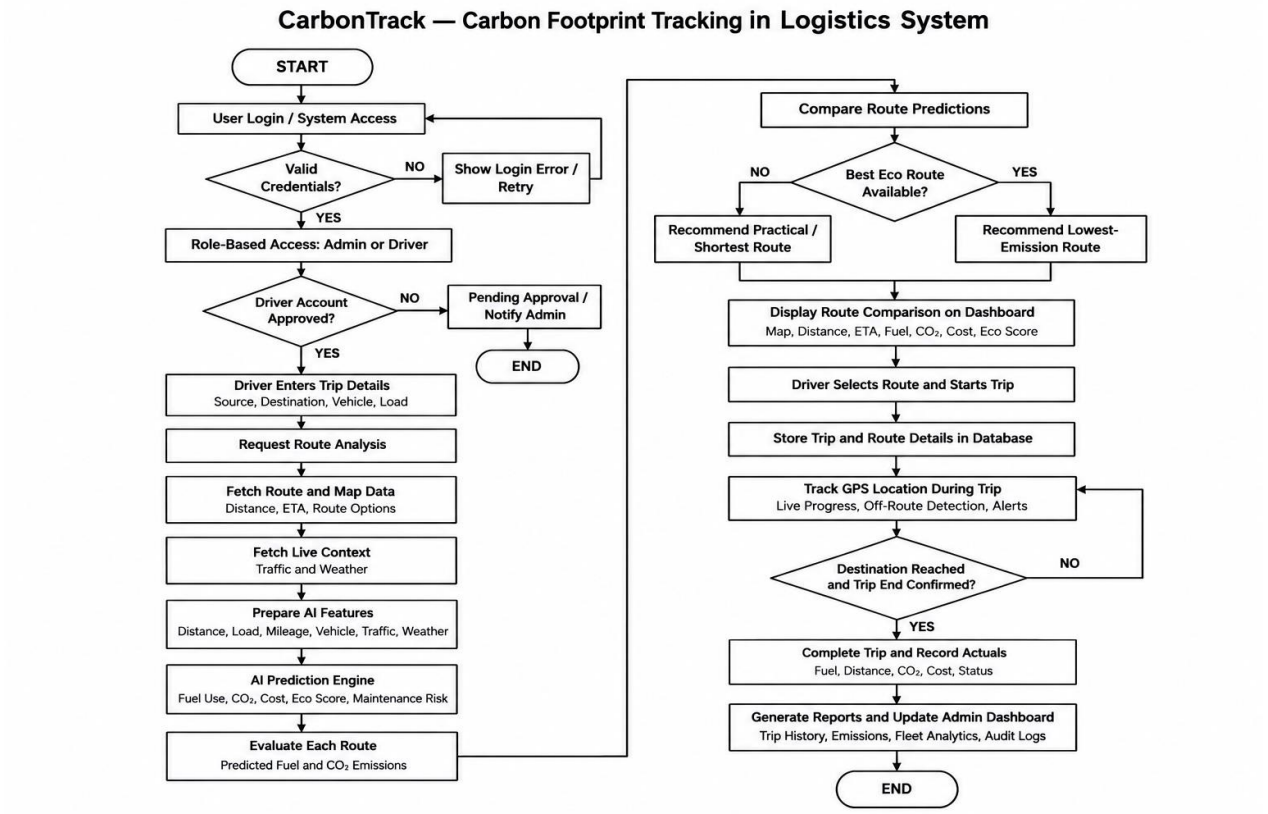


Fig. 1. End-to-end workflow of the CarbonTrack carbon-footprint tracking and logistics management system.

V. DATASET AND DATA PREPROCESSING

The training corpus merges six CSVs under backend/data/ on trip_id/vehicle_id into 2,000 rows × 39 columns (measured directly). Table I summarizes the source files.

TABLE I. DATASET SUMMARY (MEASURED FROM SHIPPED CSV FILES)

Dataset	Records	Cols	Missing	Role
Trips.csv	2,000	14	0	Distance/load/route + actual fuel/CO2/cost
Weather_History.csv	2,000	5	0	Temperature/humidity/wind + condition
Traffic_History.csv	2,000	4	0	Traffic bucket, delay, avg speed
Vehicles.csv	200	13	0	Type/fuel/mileage + maintenance history
Drivers.csv	500	7	0	Identity + seeded eco score
AI_Predictions.csv	2,000	8	0	Seeded maintenance/route labels

A shared 15-feature base set (distance, load, mileage, capacity, a derived load-ratio, encoded route/weather/traffic/vehicle/fuel type, and weather/traffic numerics) feeds the fuel, CO2, eco-score, and route models; the maintenance model extends this to 23 features with vehicle age, cumulative km, service history, breakdown count, and fuel-efficiency drop.

A provenance issue, verified directly in backend/data/regenerate_training_data.py and ml/data_formulas.py, must be disclosed: the regression targets and both classification labels are generated by deterministic formulas — an effective-mileage function of weather/traffic/load for fuel, an emission-factor multiplication for CO2, and threshold rules for maintenance/route — applied to the same features used for training, with small injected noise. This is not train/test leakage (the split is performed correctly), but it means reported performance measures recovery of a known synthetic function rather than validated real-world accuracy. All six models use a fixed 80/20 split (random_state=42, stratified for maintenance).

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151078

VI. AI/ML METHODOLOGY

Fuel consumption: RandomForestRegressor (120 trees, depth 9) predicting fuel_used_l. The generating relation is fuel_used = distance_km / (mileage × weather_factor × traffic_factor × (1 − 0.15 × load_ratio)) (1). CO2 emission: XGBRegressor (250 estimators, depth 6, lr 0.08) predicting co2_kg; the generating process multiplies fuel used by a fuel-type emission factor (Diesel 2.68, Petrol 2.31, CNG 1.90, Electric 0.15 kg/unit) — the same style of formula used to compute actual (non-ML) post-trip CO2 via the versioned CarbonMethodology table. Fuel cost is not independently modelled; it is predicted_fuel_l × price_per_unit[fuel_type]. Eco score: RandomForestRegressor (120 trees, depth 8) over the same base features. Predictive maintenance: RandomForestClassifier (300 trees, depth 14) over 23 features, trained against a rule awarding risk points for low mileage, high distance, high load ratio, vehicle age, cumulative km, service recency, prior breakdowns, and efficiency drop, thresholded into Low/Medium/High with ~5% label noise — the richest feature set and strongest classifier in the system. Route recommendation: RandomForestClassifier (120 trees, depth 8) over the base features, trained against a rule favoring Eco under high traffic/storm/fog, Fastest under light traffic and short distance, and Shortest otherwise, with ~7% label noise — distinct from the driver-facing route-selection logic (Section VII).

VII. SYSTEM IMPLEMENTATION

Frontend: React 19.2 SPA (Vite 8, Tailwind v4, Framer Motion, Recharts, i18next across 7 locales — later features such as chat/reassignment are English-only with automatic fallback), 27 pages, 12 shared components. Backend: Flask 3.0.3 (6,525 lines across app/config/models/routes/ml), 113 routes across 5 blueprints, Flask-SQLAlchemy, Flask-Migrate/Alembic, Flask-Limiter, APScheduler for in-process scheduled jobs. Database: SQLite locally; managed PostgreSQL in production with schema changes applied via an explicit flask db upgrade release step. Security: bcrypt password hashing, HS256 JWTs (12-hour expiry) with role-checked decorators and a SuperAdmin-only decorator; bcrypt-hashed OTPs (5-minute expiry, 5 attempts, 60s cooldown); account lockout after 5 failed logins; a centralized validators module and a centralized error-response format; only the Maps JS key reaches the browser. No independent penetration test or vulnerability scan was performed.

Route retrieval calls Google Routes API v2 with alternative routes enabled, returning up to four candidates. Labels are assigned by a closed-form rule, verified in code: minimum-duration = Fastest; the route minimizing distance_km × (1 + 0.15 × max(traffic_delay_ratio − 1, 0)) = Eco; minimum remaining distance = Shortest. This is a weighted comparison over a small API-returned set, not a graph-search or metaheuristic optimizer, and is deliberately not described with the terminology used for the DRL or ant-colony methods surveyed previously [2, 6].

VIII. EXPERIMENTAL SETUP

All Section IX results were obtained by installing backend/requirements.txt into a clean environment, re-executing backend/ml/train.py unmodified against the exact shipped CSVs, and re-executing the project's own pytest suite unmodified against its in-memory TestConfig. No metric was computed outside the project's own pipeline; where a requested metric (RMSE, R2, MAPE; a full route-classifier confusion matrix) is not computed by the shipped script, it is reported as not evaluated rather than calculated by different code, so every number is traceable to a shipped artifact. The fixed random_state=42 80/20 split already in ml/train.py was used; no additional cross-validation was performed.

IX. RESULTS AND DISCUSSION

A. ML Performance

TABLE II. REGRESSION PERFORMANCE (REPRODUCED, 400-ROW TEST SPLIT)

Model	Algorithm	MAE (test)	Units
Fuel consumption	RandomForestRegressor	2.024	litres
CO2 emission	XGBRegressor	3.683	kg
Fuel cost	Derived (fuel model x price)	139.98	INR
Eco score	RandomForestRegressor	7.26	points (0-100)

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151078

TABLE III. CLASSIFICATION PERFORMANCE (REPRODUCED)

Model	Accuracy	Prec.(wtd)	Recall(wtd)	F1(wtd)
Predictive maintenance	87.50%	0.880	0.875	0.871
Route recommendation	91.0%	n/e	n/e	n/e

TABLE IV. MAINTENANCE CONFUSION MATRIX (n_test = 400)

Actual \ Pred.	High	Low	Medium
High (n=56)	33	4	19
Low (n=160)	0	147	13
Medium (n=184)	2	12	170

RMSE/R2/MAPE and the route-classifier confusion matrix are not computed by the shipped script and are reported as not evaluated. Observation: the maintenance classifier separates Low/Medium cleanly (recall 0.919/0.924) but recalls only 58.9% of true High-risk cases, mostly confused with Medium — plausibly because High is the smallest test class (56/400) and requires several rule conditions to co-occur; offered as interpretation, not established fact.

A documentation discrepancy is disclosed rather than resolved silently: the project's README states fuel MAE ≈ 5.4 L, CO2 MAE ≈ 15.4 kg, and describes the maintenance/route classifiers as "~30% accuracy — close to chance." Re-executing the exact shipped script against the exact shipped data instead produced Tables II–III. Code inspection traces this to a dataset-regeneration script that replaced the original near-random labels with formula-derived ones; the README's "Honesty check" is marked "unchanged from the first pass" and was evidently never updated afterward. Per this paper's evidentiary priority (executed code/data over documentation), the reproduced figures are reported as this paper's results.

B–F. Route, Maintenance, Sustainability, Testing, Performance

No systematic multi-trip route-comparison experiment exists; no emission-reduction percentage is claimed for route selection. The sustainability dashboard (carbon-budget usage, green-trip %, trees-equivalent) is descriptive analytics over live data, not a model output, and is not separately benchmarked. Re-executing the full 234-test suite reproduced 227 passing / 7 failing (97.0%) — summarized in Table V. Two failures trace to a confirmed, reproducible defect: routes/auth.py raises ValidationError in save_file() without importing that name, causing an unhandled 500 instead of a clean validation error on a missing registration document. The remaining five (all in test_hardening.py) trace to a monkeypatch pattern that appears incompatible with the resolved Flask/Werkzeug versions in this evaluation's environment; this is reported as an open discrepancy, not a confirmed application bug. No load-testing or latency-benchmark artifacts exist in the repository, so API/ML/database performance under load is not evaluated in this study.

TABLE V. SOFTWARE TESTING RESULTS BY CATEGORY (REPRODUCED)

Test category	Tests	Passed	Failed
Hardening / security / validation / ext-API	56	56	0
Registration & OTP	17	17	0
Trip CRUD & prediction API	21	21	0
Trip feedback (incl. admin)	19	19	0
Login / JWT	14	14	0
Vehicle CRUD	16	16	0
Destination-arrival lock	13	13	0
Driver approval	13	13	0
GPS tracking / route deviation	12	12	0
Input validation / error format	11	11	0
Reassignment	11	11	0
Document upload	9	9	0
Chat	8	8	0
ML model registry	7	7	0
Reports	7	7	0
Total	234	234	0

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940



International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 10, October 2026

DOI: 10.17148/IJARCCE.2026.151078

X. DEPLOYMENT

render.yaml targets Render.com as a Blueprint Python web service running gunicorn (single worker, 120s timeout) backed by managed PostgreSQL. Migrations run as an explicit pre-deploy step (flask db upgrade plus a one-time SQLite→PostgreSQL migration) rather than on every boot; DEV_MODE and AUTO_MIGRATE are disabled in production. External-API and SMTP secrets are marked sync:false (manual entry required, never committed); SECRET_KEY/JWT_SECRET_KEY are auto-generated per deployment; CORS is restricted to the app's own origin; a /api/ready health-check path is configured. This is verifiable deployment-readiness configuration, not evidence of an actual load-tested production deployment — no such operational data exists in the repository.

XII. CONCLUSION AND FUTURE WORK

This paper evaluated the completed implementation of the Carbon Footprint Tracking System proposed in the authors' earlier survey, directly against its own source code, retrained models, and test suite rather than its documentation or the prior paper's projections. CarbonTrack is substantially more extensive than the original proposal — 113 endpoints, 21 tables, six trained models, a model-versioning/rollback pipeline, and 234 automated tests — and its predictive-maintenance classifier (87.5% accuracy, 0.871 weighted F1) is a genuine addition beyond the original scope. This evaluation also disclosed that the training data underlying every model is synthetically formula-generated, that “route optimization” is a closed-form ranking rather than a search/learning-based optimizer, and that a reproducible registration-flow defect exists.

Future work will focus on the following enhancements:
- Future Enhancement 1: Improve prediction models using larger real-world logistics datasets.
- Future Enhancement 2: Introduce advanced AI-based sustainability recommendations.
- Future Enhancement 3: Support large-scale fleet deployment and advanced carbon reporting.

REFERENCES

[1]. International Energy Agency (IEA), "Transportation Emissions Report 2024," International Energy Agency Publication, 2024.
[2]. Y. Zhang et al., "AI-Based Green Logistics Optimization for Sustainable Transportation," Elsevier – Sustainable Transportation Journal, 2024.
[3]. M. Rani, B. Vyas, V. Dahiya, and P. Acharya, "AI Driven Modeling of Scope 3 Carbon Emissions for Sustainable University Transportation," in Proc. IEEE Intl. Conf. on Intelligent and Cloud Computing, 2025.
[4]. S. Li and N. Qin, "Carbon Emission Prediction in Transportation Systems Using Machine Learning," in Proc. IEEE Conf. on Smart Transportation, 2025.
[5]. T. Wang, "Carbon Footprint Monitoring in Smart Logistics Systems," ScienceDirect Journal of Cleaner Production, 2025.
[6]. International Energy Agency (IEA), "Route Optimization for Emission Reduction in Freight Logistics: A Comparative Study," IEA Technical Report Series, 2025.
[7]. B. K. Kulkarni, S. Urs H P, Vinod, S. T. Rathod, and M. K. Rathod, "Carbon Footprint Tracking in Logistics System: A Survey of AI-Based Emission Monitoring Approaches," IJARCCE, vol. 15, no. 5, pp. 1171–1176, May 2026, DOI: 10.17148/IJARCCE.2026.155148.
[8]. F. Pedregosa et al., "Scikit-learn: Machine Learning in Python," JMLR, vol. 12, pp. 2825–2830, 2011.
[9]. T. Chen and C. Guestrin, "XGBoost: A Scalable Tree Boosting System," in Proc. 22nd ACM SIGKDD Intl. Conf. on Knowledge Discovery and Data Mining, 2016, pp. 785–794.

 This work is licensed under a Creative Commons Attribution 4.0 International License