Eco-Returns – Reverse Logistics for return orders

A Project Work-I Report

Submitted in partial fulfillment of requirement of the

Degree of

BACHELOR OF TECHNOLOGY in COMPUTER SCIENCE & ENGINEERING

BY

Anushka Paharia
EN22CS301183

Under the Guidance of
Prof. Mohammed Mazhar & Prof. Roshni Verma

Department of Computer Science & Engineering
Faculty of Engineering
MEDICAPS UNIVERSITY, INDORE- 453331

August - December 2025

Report Approval

The project work“ **Eco-Returns – Reverse Logistics for return orders”** is hereby approved as a creditable study of an engineering/computer application subject carried out and presented in a manner satisfactory to warrant its acceptance as a prerequisite for the Degree for which it has been submitted.

It is to be understood that by this approval the undersigned do not endorse or approve any statement made, opinion expressed, or conclusion drawn therein; but approve the “Project Report” only for the purpose for which it has been submitted.

Internal Examiner
Name:
Designation
Affiliation

External Examiner
Name:
Designation
Affiliation

Declaration

I hereby declare that the project entitled **“Eco-Returns – Reverse Logistics for return orders”** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in Computer Science completed under the supervision of **Prof. Mohammed Mazhar & Prof. Roshni Verma, Computer Science and Engineering,** Faculty of Engineering, Medicaps University Indore is an authentic work.

Further, I declare that the content of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for the award of any degree or diploma.

Signature and name of the student

Certificate

We, **Prof. Mohammed Mazhar & Prof. Roshni Verma,** certify that the project entitled **"Eco-Returns – Reverse Logistics for return orders"** submitted in partial fulfillment for the award of the degree of Bachelor of Technology by **Anushka Paharia, EN22CS301183** is the record carried out by her under our guidance and that the work has not formed the basis of award of any other degree elsewhere.

____

Prof. Mohammed Mazhar .

Prof. Roshni Verma

Computer Science & Engineering

Medicaps University, Indore

____

Dr. Kailash Chandra Bandhu

Head of the Department

Computer Science & Engineering

Medicaps University, Indore

\

Acknowledgements

I would like to express my deepest gratitude to Honorable Chancellor, **Shri R C Mittal,** who has provided me with every facility to successfully carry out this project, and my profound indebtedness to **Prof. (Dr.) D K Patnaik,** Vice Chancellor, Medicaps University, whose unfailing support and enthusiasm has always boosted up my morale. I also thank **Prof. (Dr.) Ratnesh Litoriya,** Associate Dean, Computer Science and allied branches, Medicaps University, for giving me a chance to work on this project. I would also like to thank my Head of the Department **Dr. Kailash Chandra Bandhu** for his continuous encouragement for the betterment of the project.

It is their help and support, due to which we became able to complete the design and technical report.

Without their support this report would not have been possible.

Anushka Paharia
EN22CS301183
B.Tech. IV Year
Department of Computer Science & Engineering
Faculty of Engineering
Medicaps University, Indore

Abstract

The rapid growth of e-commerce has led to a significant rise in product returns, creating operational challenges and contributing to environmental impact due to transportation emissions and waste generation. *Eco-Returns* is an AI-enabled return optimization system designed to streamline reverse logistics by predicting the optimal disposition for returned products and estimating the carbon footprint saved through sustainable decisions.

The system integrates a **Random Forest–based Machine Learning model** to recommend dispositions such as *resell, refurbish, recycle, donate*, or *returnless refund*. It operates through a **microservice architecture** comprising a React-based customer interface, a Node.js SaaS backend, and a Python Flask ML service. Additional modules include a heuristic fraud-detection score and a rule-based carbon emission estimation engine that computes environmental benefits based on distance, product weight, and emission factors.

The platform provides a unified workflow covering return initiation, prediction generation, admin approval, and sustainability analytics. Experimental evaluation demonstrates that the system improves decision consistency, reduces manual workload, identifies high-risk returns, and provides quantifiable insights into carbon savings. *Eco-Returns* showcases how intelligent automation can enhance operational efficiency while promoting sustainable reverse logistics practices.

Keywords: Reverse Logistics, Machine Learning, Sustainability, Carbon Footprint, E-commerce Returns, Fraud Detection, SaaS Architecture

Table of Contents

Report Approval..**2**
Declaration..**3**
Certificate..**4**
Acknowledgements..**5**
Abstract..**6**
Table of Contents..**7**
List of Figures..**9**
List of Tables..**10**
Abbreviations..**11**
Notations & Symbols..**12**
Chapter-1..**1**
Introduction..**1**
1.1 Introduction..1
1.2 Literature Review..1
1.3 Objectives..3
1.4 Significance..3
1.5 Research Design..4
1.6 Source of Data..5
Chapter-2..**6**
Experimental Set-Up And Procedures..**6**
2.1 Experimental Set-up..6
2.2 Software Environment..6
2.2 Procedures Adopted..9
Chapter-3..**11**
System Design..**11**
3.1 System Architecture Diagram..11
3.2 Use Case Diagram..11
3.3 DFD Level-0..12
3.4 DFD Level-1..13
3.5 ER Diagram..14
3.6 Component Diagram..14
3.7 Deployment Diagram..15
3.8 State Machine Diagram..16
3.9 Control Flow Diagram..16
Chapter-4..**19**
Database Design..**19**
4.1 Objectives of Database Design..19

4.2 Logical Database Design..19
4.3 Normalization..21
4.4 Data Dictionary..22
4.5 Storage Strategy..22
4.6 Data Security Measures..22
4.7 Backup & Recovery..23
Chapter-5..24
Machine Learning Models..24
5.1 Return Disposition Prediction Model..24
5.2 Carbon Footprint Estimation Model..26
5.3 Fraud Risk Score Model (Heuristic)..26
5.4 Model Integration..27
Chapter-6..28
System Implementation..28
6.1 Overview of Implementation..28
6.2 Frontend Implementation (E-Commerce User Interface)..28
6.3 Backend Implementation (Node.js SaaS API)..29
6.4 Machine Learning Implementation (Flask API)..30
6.5 Database Implementation (MongoDB)..30
6.6 Security Implementation..30
6.7 Deployment..31
Chapter-7..32
Results and Discussions..32
Chapter-8..35
Summary and Conclusions..35
Chapter-9..36
Future Scope..36
Appendix..37
A.1 Core System Components..37
A.2 Frameworks and Key Libraries..37
Bibliography..39

List of Figures

Fig No.	Title	
3.1	System Architecture Diagram	11
3.2	Use Case Diagram	12
3.3	DFD Level-0	13
3.4	DFD Level-1	13
3.5	ER Diagram	14
3.6	Component Diagram	15
3.7	Deployment Diagram	16
3.8	State Machine Diagram	17
3.9	Control Flow Diagram	18
7.1	ML Recommendation Output (Admin Dashboard)	32
7.2	Carbon Savings Dashboard	33
7.3	Return Initiation Form	34
7.4	Return Detail View	34

List of Tables

Table No.	Title	
Table 4.1	User Table	19
Table 4.2	Order Table	20
Table 4.3	Return Table	20
Table 4.4	Carbon Metrics Table	21
Table 4.5	Data Dictionary Summary	22

Abbreviations

Abbreviation	**Full Form**
API	Application Programming Interface
CORS	Cross-Origin Resource Sharing
CO_2	Carbon Dioxide
DBMS	Database Management System
DFD	Data Flow Diagram
EF	Emission Factor
ER	Entity–Relationship
ETL	Extract, Transform, Load
GPS	Global Positioning System
HTTP	Hypertext Transfer Protocol
ML	Machine Learning
RF	Random Forest
REST	Representational State Transfer
UI	User Interface
UUID	Universally Unique Identifier

Notations & Symbols

Symbol	**Description**
d	Distance between customer and warehouse (km)
w	Product weight (kg)
EF	Emission factor (kg CO_2 per kg·km)
$CO_{2transport}$	Carbon emission from transport
CO_{2saved}	Carbon saved based on disposition
S	Disposition-specific saving percentage
FS	Fraud score
t	Days since purchase
P	Product price

Chapter-1

Introduction

1.1 Introduction

The emergence of digital commerce has significantly altered global retail dynamics by enabling high-volume transactions, personalized purchasing experiences, and efficient supply chain processes. However, one persistent challenge that continues to impose substantial financial and operational strain on e-commerce companies is the management of product returns.
Return processing involves inspection, validation, repackaging, fraud detection, and final dispositioning of the product. These processes not only increase operational costs but also contribute to environmental degradation through transportation emissions and landfill accumulation.

To address these concerns, this project presents an **AI-enabled Return Optimization System** that integrates seamlessly with an e-commerce workflow through a cloud-based SaaS backend. The system utilizes a Machine Learning model to recommend an optimal disposition for each returned product—such as **refurbish, resell, recycle, donate, or returnless refund**.

The project comprises:

- A **React-based e-commerce storefront** where users place orders and initiate returns
- A **SaaS admin portal** that enables administrators to review return requests, view ML recommendations, and finalize dispositions
- A **Node.js backend** that manages return records, integrates with an ML model, and performs carbon footprint analysis
- A **Python-based ML microservice** that predicts the most appropriate disposition using cleaned and modified retail return data

The system emphasizes **sustainability**, offering administrators the ability to track **carbon emissions saved** through eco-friendly decisions, making it both operationally efficient and environmentally conscious.

1.2 Literature Review

The increasing growth of e-commerce has led to a corresponding rise in product returns, making reverse logistics a critical component of modern retail systems. Ketzenberg et al. highlight that return flows now represent a significant share of total logistics activity, introducing challenges related to cost, fraud, and sustainability management [1]. As return volumes increase, retailers seek analytical tools capable of supporting disposition decisions, fraud detection, and operational optimisation.

Early studies primarily examined rule-based approaches for return disposition. Rogers and Tibben-Lembke provided a foundational analysis of reverse logistics processes, demonstrating that inefficient disposition decisions significantly inflate operational costs and environmental burdens [2]. However, traditional rule-based methods were shown to be limited in their adaptability, especially with the increasing complexity of modern retail ecosystems.

With the expansion of machine learning research, the emphasis shifted toward predictive modelling for return classification and fraud detection. Breiman's seminal work on Random Forests [3] established the basis for ensemble methods widely used in classification tasks, including return disposition prediction. More recent work by Ramanathan et al. analysed online customer behaviour to forecast return likelihood and detect anomalies in return activity, illustrating the potential of behavioural features and multivariate models for fraud identification [4]. Similarly, Xu et al. demonstrated that machine learning techniques outperform statistical baselines in predicting e-commerce return outcomes, recommending hybrid feature sets that integrate financial, behavioural, and product-level data [5]. These studies collectively establish machine learning as a robust tool for analysing and predicting return patterns.

In parallel, sustainability research has increasingly focused on the environmental implications of reverse logistics. Mangla et al. emphasised that poor return management contributes substantially to carbon emissions due to transportation and product disposal activities, and they identified technology-assisted disposition optimisation as an essential sustainability strategy [6]. A comprehensive review by de Oliveira et al. further explored sustainable practices in logistic chains, concluding that predictive systems integrated with green logistics policies significantly reduce waste and emissions across product lifecycles [7]. At a broader level, Bouchard et al. found that supply-chain-level carbon optimisation requires accurate modelling of transport distances, item weight, and recovery strategies, reinforcing the need for automation and data-driven carbon estimation tools [8].

Collectively, existing work demonstrates three important trends. First, reverse logistics represents a growing operational and sustainability challenge for retailers. Second, machine learning models—including Random Forests and other ensemble approaches—are increasingly effective for supporting decision-making in return management. Third, integrating environmental metrics such as carbon emission savings aligns disposition optimisation with sustainability goals. Yet, despite these contributions, relatively few studies address the combined problem of personalised return classification, fraud detection, and carbon estimation within a unified system architecture.

This project therefore contributes to the literature by developing an integrated SaaS-based return management platform that: (i) processes return requests with machine learning-based disposition prediction; (ii) identifies high-risk or atypical return patterns; and (iii) computes carbon emissions avoided through sustainable dispositions such as resell or refurbish. The system aligns with prior academic work while addressing the gap in multi-objective optimisation frameworks that jointly consider operational efficiency and environmental impact.

1.3 Objectives

The objectives of this project are:

1. **To develop a machine learning model** capable of predicting the optimal disposition for returned products based on historical order and return data.

2. **To implement a SaaS-enabled backend** that communicates with both the ML service and the e-commerce interface.

3. **To create a user-facing e-commerce frontend** allowing order placement and return initiation.

4. **To design an admin dashboard** where return data, ML recommendations, carbon savings, and final decisions can be managed.

5. **To compute carbon emissions saved** through sustainable dispositioning and visualise them using analytical dashboards.

6. **To evaluate the system's performance** in optimizing return workflows and supporting sustainability goals.

1.4 Significance

The project presents a practical demonstration of how machine learning can be applied to the modern challenges of retail returns and reverse logistics. It integrates multiple interdisciplinary components—including predictive modelling, full-stack web development, cloud-based APIs, and sustainability analytics—making it a strong academic case study in applied machine learning systems. Such a system also contributes to research in environmental optimisation, highlighting how data-driven decision-making can reduce waste and improve operational processes. Overall, it provides a comprehensive example suitable for academic exploration in areas such as supply chain optimisation, sustainable technologies, and intelligent automation.

From an industry perspective, the system addresses one of the most costly aspects of e-commerce operations: product returns. By automating disposition decisions, identifying risky or potentially fraudulent returns, and streamlining the return-approval workflow, the solution can significantly reduce reverse-logistics expenses for merchants. Additionally, the carbon-footprint module introduces measurable sustainability benefits, helping organisations track and reduce emissions associated with returned goods. The modular SaaS architecture ensures that the system is highly scalable and can be integrated into existing commercial platforms, offering real-world applicability for retailers, logistics companies, and sustainability-driven enterprises.

1.5 Research Design

This project employed a structured, phased methodology that blends data analysis, full-stack system design, and rigorous validation to address inefficiencies in product return processes.

1.5.1 Problem Identification and Scope

The initial effort established the core challenges this system addresses: the **high cost of handling returns** and the pervasive **limited automation** and lack of **sustainability insights** within conventional return workflows. The primary objective was defined as creating a system that mitigates financial loss while simultaneously promoting environmentally conscious decision-making across the return process.

1.5.2 Data Acquisition and Preparation

Data acquisition commenced with a publicly available **Orders and Returns dataset from Kaggle**. This raw data underwent essential cleaning, transformation, and modification to align with project requirements, ensuring all entries were consistent and complete for modeling. Crucially, extensive **feature engineering** was performed to enrich the dataset with relevant fields, including specific product attributes, geographical **distance** between shipping locations, and detailed **temporal features** (e.g., time since purchase) relevant to analyzing return delays and fraud risk.

1.5.3 Model Development

The preparation for model development involved standard data preprocessing techniques, including encoding categorical variables, scaling numerical features, and systematic handling of outliers. Further advanced **feature engineering** was dedicated to indicators for fraud, distance, and delay, which served as direct inputs to the classifier. The final **Model Selection** favored the **Random Forest Classifier** due to its inherent robustness, high predictive accuracy, and strong model interpretability. The trained model was rigorously evaluated and saved through defined **ML pipelines** to ensure its stability, reproducibility, and deployment readiness.

1.5.4 System Architecture

The solution employs a scalable **microservices architecture** centered around three primary components. The machine learning component is deployed as a dedicated **Flask ML microservice**. This service interacts with an **Express.js backend**, which is responsible for managing core return logic, data persistence, and running analytical queries. The user experience is handled by two distinct **React.js frontends**—one tailored for the customer experience and a separate administrator portal. All communication between these services is standardized via **REST API integration**.

1.5.5 Implementation and Validation

The implementation focused on delivering user-centric applications: a seamless **Customer UI** incorporating a product catalog, order history, and a guided return form; and a comprehensive **Admin Portal** presenting the system's core intelligence, including real-time ML fraud predictions and sustainability **carbon dashboards**. A critical implementation feature was the custom logic for calculating **carbon footprint savings**. This calculation utilized integrated distance and product weight metrics to quantify the environmental benefit of successful return reduction or alternative processing suggestions. The final **Evaluation Phase** involved testing ML predictions against known fraudulent and non-fraudulent patterns, followed by conducting **end-to-end functional testing** of the complete return lifecycle and meticulous validation of the carbon computation logic to ensure its accuracy and reliability.

1.6 Source of Data

The project exclusively utilized **secondary data** sourced from a publicly available **Kaggle Orders and Returns dataset**, a common benchmark in retail analytics research. The raw dataset was subject to rigorous **cleaning, modification, and extension** to align with the specific requirements of the fraud and sustainability analysis. Modifications involved analytical derivations to include fields relevant to **product characteristics**, **return reasons**, **pricing and weights**, **customer return history**, and critical **distance and sustainability attributes**. No primary survey or manual data collection was conducted, meaning all extensions and changes were analytical in nature and derived solely from the foundational secondary data.

Chapter-2

Experimental Set-Up And Procedures

2.1 Experimental Set-up

The experimental set-up for this project involves the coordinated interaction of multiple technological components—including a machine learning environment, a cloud-inspired backend service, and two separate web-based frontends. The infrastructure was designed to replicate a real-world e-commerce return management system while enabling modular experimentation, monitoring, and controlled evaluation.

2.1.1. Hardware Environment

The entire project was developed and executed on a standard computing environment with the following specifications:

- Processor: Intel i5/i7 or equivalent
- RAM: Minimum 8 GB
- Operating System: Windows / macOS / Linux
- Storage: ~5 GB required for dependencies, datasets, and logs

Although the system is platform-independent, the development environment was optimized for local execution using Node.js and Python.

2.2 Software Environment

The development and deployment of the Eco-Returns system required a structured multi-layered software environment comprising machine learning infrastructure, backend services, and frontend user interfaces.

Machine Learning Environment

The machine learning subsystem is responsible for building, training, and serving the **Random Forest–based disposition prediction model**. This environment was designed and executed using **Python 3.10+**. The core libraries utilized for data processing and modeling include Pandas 1.5.3, NumPy 1.23.5, and Scikit-learn 1.2.2, with Joblib 1.3.2 used for model persistence. This subsystem serves two main functions: the preparation and preprocessing of the modified Kaggle orders dataset, and the training of the Random Forest classification model. In deployment, the model is exposed as a lightweight **Flask 2.2.5 REST API** (the ML microservice) through a dedicated /predict endpoint, ensuring reproducibility via pinned dependency versions.

● **Language**	● Python 3.10+
● **Key Libraries**	● Scikit-learn 1.2.2, Pandas 1.5.3
● **API Framework**	● Flask 2.2.5

SaaS Simulation Backend (Node.js Service Layer)

The backend layer simulates the SaaS platform, coordinating return data, invoking the ML model, performing carbon emission estimation, and enabling admin-level analytics. This layer is built on the **Node.js 18.x LTS** platform using the **Express.js 4.19+** framework. While the system architecture uses custom modules (computeCarbonSaved.js, geo.js) for core computations, the development mode utilized an in-memory Map store for the database. This layer is tasked with receiving return requests, communicating with the Flask ML API for predictions, persisting return records, computing estimated and confirmed carbon savings, and providing authenticated REST endpoints for the admin dashboard.

Platform	Node.js 18.x LTS
Framework	Express.js 4.19+
Database (Dev)	In-memory Map store
Security	Authenticated REST endpoints

Frontend Interfaces
Two separate **React applications** were developed using **React.js 18.x** and the **Vite 4.x** build tool: the **Eco-Returns Customer Portal** and the **SaaS Admin Dashboard**. Both applications employ a modern, component-driven frontend architecture utilizing JavaScript (ES2020+) and standard **React Hooks** for state management. API integration is managed via fetch-based asynchronous data communication with the Node backend.

The **Customer-facing Ecommerce UI** allows users to browse products, place orders, submit returns, and track return history. A critical feature is the collection of user geolocation data to facilitate logistics-based carbon computation. Conversely, the **Admin Dashboard** allows staff to inspect incoming returns, view ML recommendations, approve dispositions, and view the comprehensive **carbon footprint dashboard** summarizing confirmed and estimated savings.

Customer Portal	React.js 18.x
Admin Dashboard	React.js 18.x
Build Tool	Vite 4.x

2.1.3. Data Environment

The system uses a **modified version of the Kaggle Orders and Returns dataset**.
The dataset was preprocessed to:

- Remove inconsistencies
- Generate additional return features
- Add weight and category attributes
- Map disposition labels for supervised training

This processed dataset is then used to train the ML model and to simulate realistic return events.

2.1.4 System Architecture

This system is orchestrated by the **Node.js Backend**, which receives return data generated by the **Customer App**. The Backend queries the **ML Microservice**, a dedicated service that predicts the optimal return disposition using a trained Random Forest model. The results are stored by the Backend, which then feeds relevant metrics to two secondary components: the **Admin Dashboard** for final decision-making and sustainability monitoring, and the **Carbon Footprint Engine** for computing CO_2 savings based on product weight, distance, and the determined disposition.

2.1.5 Experimental Controls

To maintain consistency across experiments:

- the same dataset split was used for model testing
- all model evaluations were conducted with fixed random seeds
- the backend always called the same ML endpoint for predictions

- carbon calculations remained standardized using predefined emission constants

This set-up ensures that results are reproducible and comparable.

2.2 Procedures Adopted

The overall development and testing process was executed through six structured phases, ensuring systematic progress and alignment with established academic research methodologies.

Phase A: Data Preparation and Feature Engineering

This initial phase focused on preparing the secondary Kaggle dataset for model ingestion. The dataset was imported, and missing or erroneous values were addressed through cleaning procedures. Product categories and return reasons were normalized to ensure consistency. Crucial engineered features were added, including temporal attributes (*days_since_purchase*), customer behavior metrics (*customer_return_count_90d*), order valuation heuristics (*order_value_ratio*), and a preliminary fraud score heuristic. The resulting ML-compatible dataset was then partitioned into training and testing subsets.

Phase B: Machine Learning Model Development

Model development commenced with testing multiple classification algorithms, including Logistic Regression and Decision Trees. Models were rigorously evaluated using standard metrics such as accuracy, precision, recall, and confusion matrices. The Random Forest Classifier was ultimately selected due to its superior predictive performance and strong interpretability. Hyperparameters were tuned for optimization, and the final trained model was saved as a deployable pipeline. A dedicated Flask API was subsequently built to expose the model predictions via the /predict HTTP endpoint.

Phase C: Backend API Development

The core service layer was built using Express.js, providing robust REST routes necessary for posting new returns, fetching records for display, updating admin decisions, re-running ML predictions, and computing carbon metrics. Security was enforced through CORS protection and API key authorization for all administrative routes. Critical system integration involved asynchronous REST calls to the ML microservice. Furthermore, a dedicated carbon footprint calculation module was created, which synthesized product weight, return distance, emission factors, and disposition-specific coefficients to quantify environmental savings.

Phase D: Customer-Facing Application Development

The development of the Customer App involved designing user interface components for product listings, order history, and the login interface. A specific workflow was implemented for return creation, which included reason selection, generation of the fraud heuristic, and calculation of return distance based on the user-entered location. The application featured a clean multi-page navigation system and utilized local storage to ensure persistent user experiences across orders and returns.

Phase E: Admin Dashboard Development

The Admin Dashboard provided key operational and sustainability intelligence. It incorporated table-based dashboards displaying pending and processed returns, ML suggestions, and calculated fraud scores. Administrative actions, such as "Re-run ML" and "Accept Disposition," were added to facilitate human oversight. A separate carbon analytics dashboard was developed to present confirmed and estimated CO_2 savings, including visual indicators and charts to clearly interpret the sustainability impact broken down by category and final disposition.

Phase F: Testing, Validation, and Iteration

The final phase involved rigorous quality assurance. Unit testing was performed for backend routes, followed by manual, end-to-end testing of the full workflow, from order submission and return request through ML prediction and final admin acceptance. A key validation step ensured that carbon savings updated correctly upon final disposition approval. This phase also incorporated necessary bug fixes and refinements, focusing on achieving realistic handling of weight and distance data, and cross-validating consistency between the UI behaviour and backend logic.

Chapter-3

System Design

3.1 System Architecture Diagram

The system architecture illustrates the overall structural design of the application, depicting how different components interact with each other. It highlights the flow of data between the frontend, backend, machine learning model, and the database. This architecture ensures modularity, scalability, and efficient communication between system components. It also helps identify key layers such as the presentation layer, application layer, and data layer.

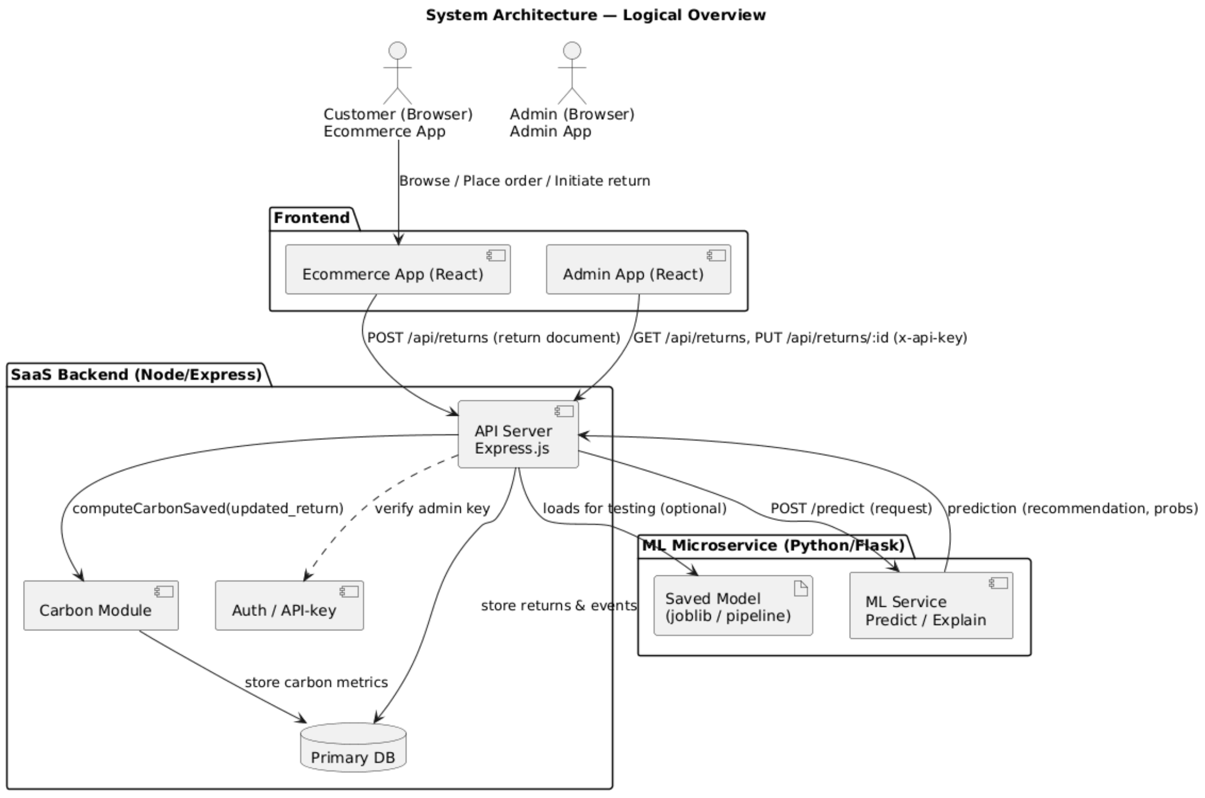
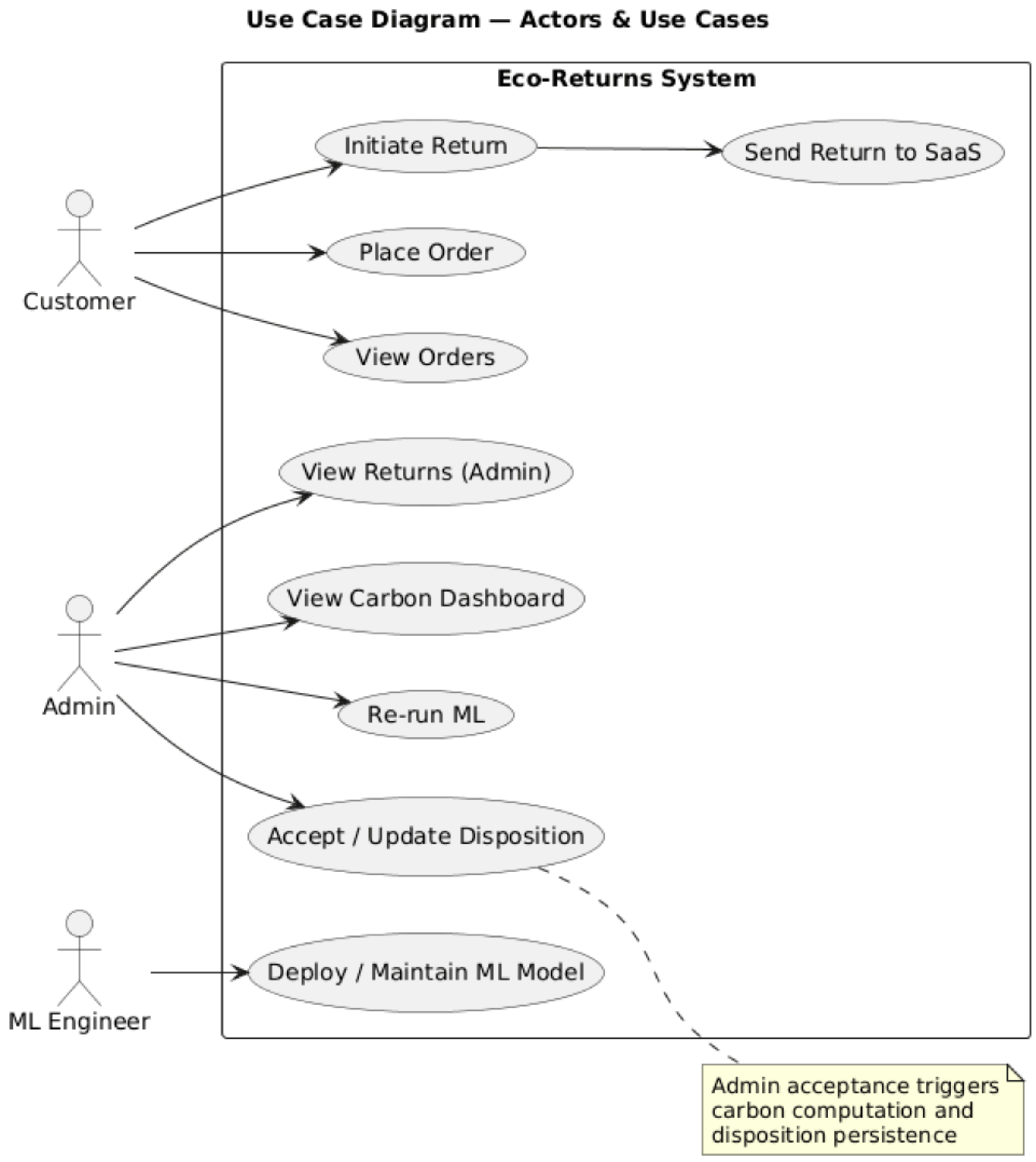

System Architecture — Logical Overview
Customer (Browser) Ecommerce App
Admin (Browser) Admin App
Browse / Place order / Initiate return
Frontend
Ecommerce App (React)
Admin App (React)
POST /api/returns (return document)
GET /api/returns, PUT /api/returns/:id (x-api-key)
SaaS Backend (Node/Express)
API Server Express.js
computeCarbonSaved(updated_return)
verify admin key
loads for testing (optional)
POST /predict (request)
prediction (recommendation, probs)
ML Microservice (Python/Flask)
Saved Model (joblib / pipeline)
ML Service Predict / Explain
Carbon Module
Auth / API-key
store returns & events
store carbon metrics
Primary DB

Fig 3.1 System Architecture Diagram of Eco-Returns

3.2 Use Case Diagram

The use case diagram defines the interactions between external users (actors) and the system. It captures the major functionalities offered by the application, such as submitting return requests, generating return recommendations, managing orders, and updating product status.

This diagram helps visualize user roles and the scope of the system by outlining key operations from the user's perspective.



Use Case Diagram — Actors & Use Cases
Eco-Returns System
Customer
Initiate Return
Send Return to SaaS
Place Order
View Orders
Admin
View Returns (Admin)
View Carbon Dashboard
Re-run ML
Accept / Update Disposition
ML Engineer
Deploy / Maintain ML Model
Admin acceptance triggers carbon computation and disposition persistence

Fig. 3.2 Use Case Diagram of Eco-Returns

3.3 DFD Level-0

The Level-0 Data Flow Diagram represents the system as a single, high-level process with its basic input and output flows. It identifies the main external entities and shows how data enters, is processed, and exits the system. This diagram offers a simplified overview that sets the foundation for more detailed data flow models.

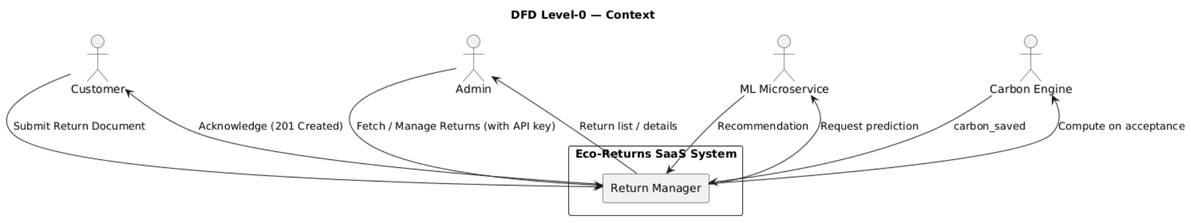


Fig 3.3 DFD Level 0 of Eco-Returns

3.4 DFD Level-1

The Level-1 Data Flow Diagram decomposes the main system process into multiple sub-processes to provide a more detailed view of the internal data movement. It explains how data is validated, processed, and stored across different modules. This level helps understand the logical workflow of the system and the interactions between various functions.

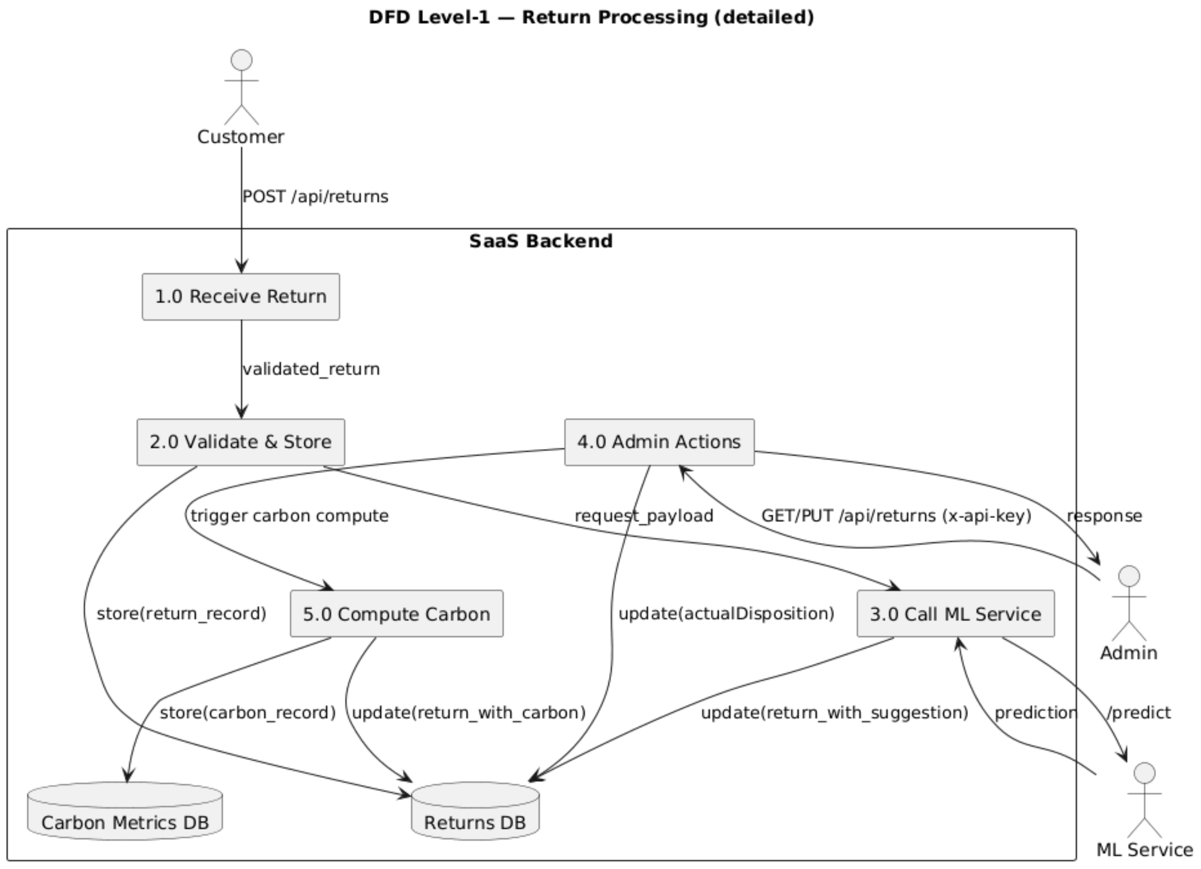


Fig 3.4 DFD Level 1 of Eco-Returns

3.5 ER Diagram

The Entity–Relationship (ER) diagram presents the database structure by identifying key entities and their relationships. It includes details such as customer, product, order, return request, and classification results. The ER diagram helps ensure that the database is normalized, logically structured, and capable of supporting system operations efficiently.

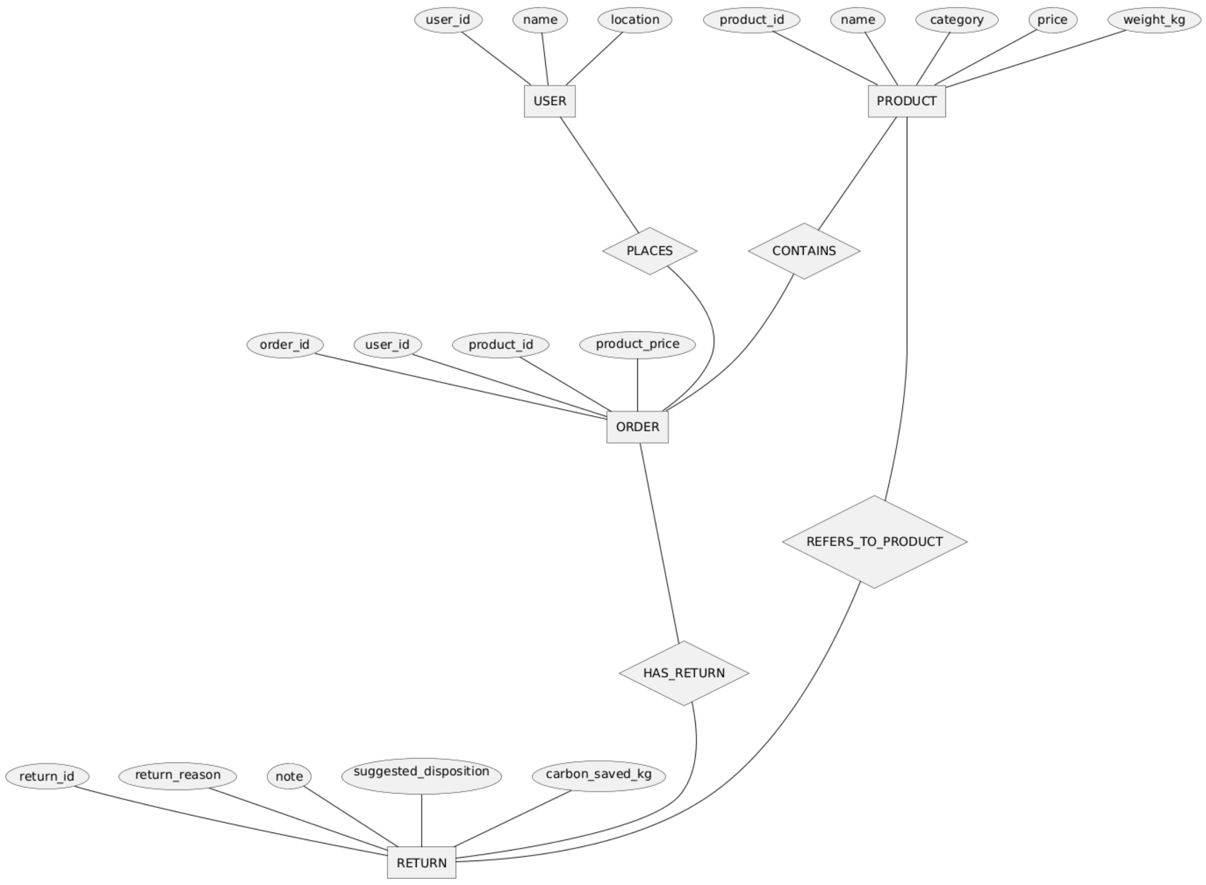


Fig 3.5 ER Diagram of Eco-Returns

3.6 Component Diagram

The component diagram describes the major software components of the system and how they connect. It breaks the system into logical modules such as the user interface, backend service, machine learning engine, and data storage. This diagram emphasizes the modular design approach, allowing individual components to be modified or upgraded independently.

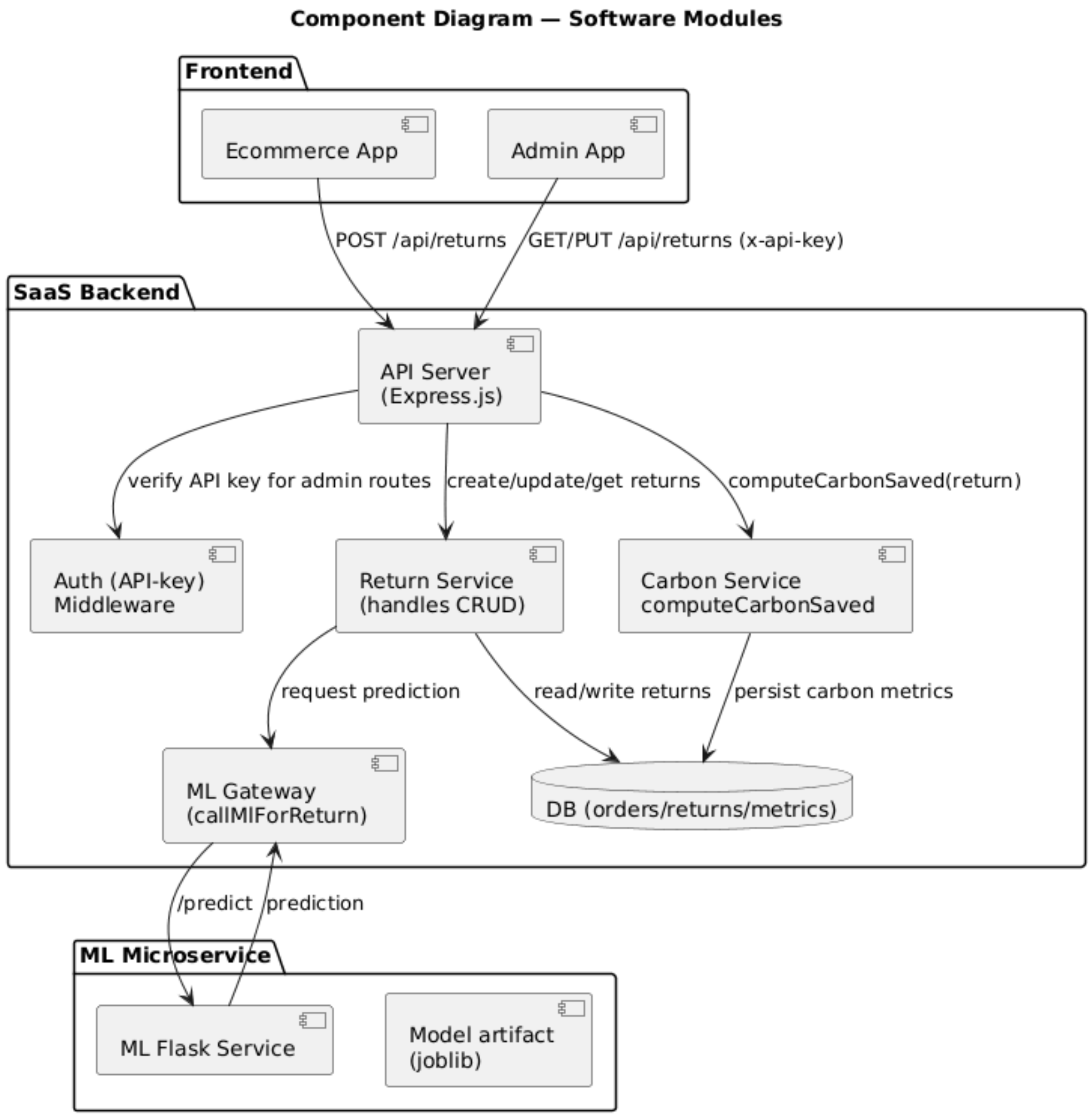
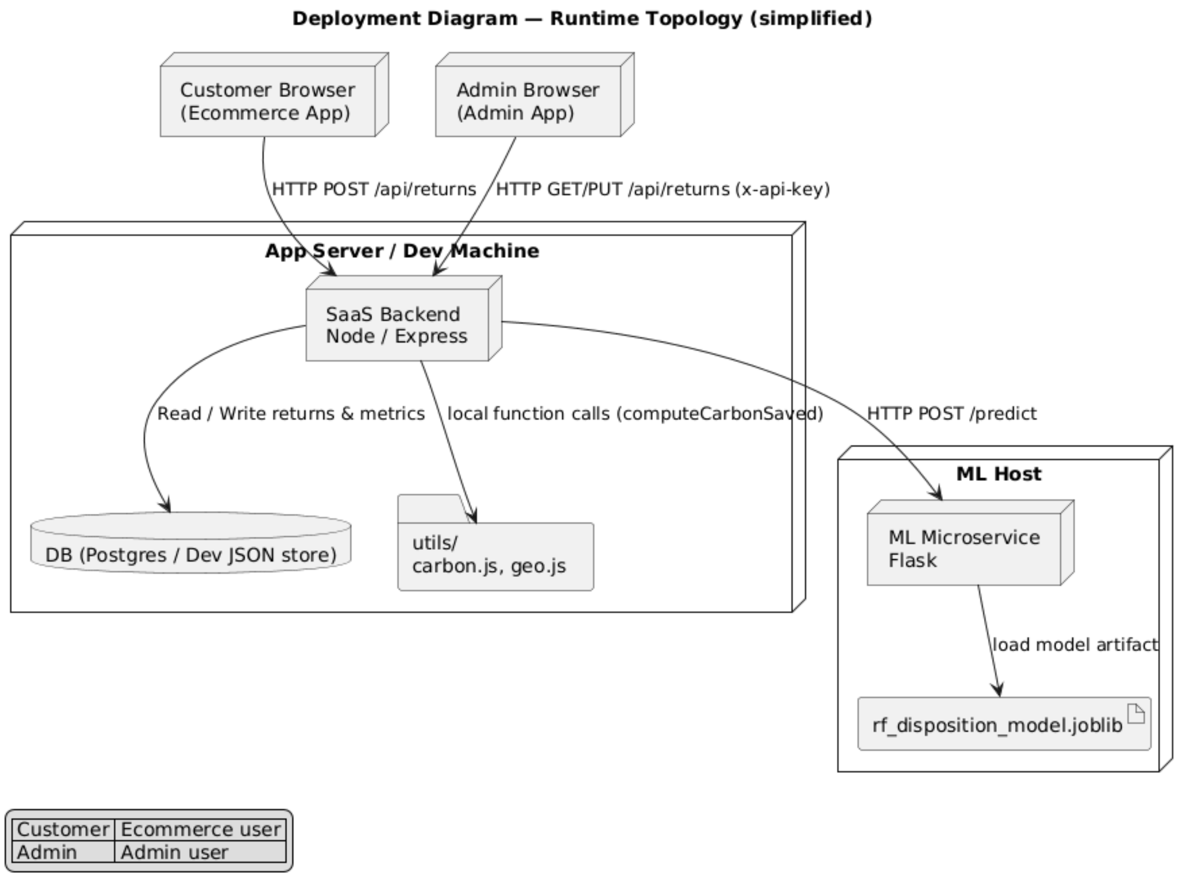

Component Diagram — Software Modules
Frontend
Ecommerce App
Admin App
POST /api/returns
GET/PUT /api/returns (x-api-key)
SaaS Backend
API Server (Express.js)
verify API key for admin routes
create/update/get returns
computeCarbonSaved(return)
Auth (API-key) Middleware
Return Service (handles CRUD)
Carbon Service computeCarbonSaved
request prediction
read/write returns
persist carbon metrics
ML Gateway (callMlForReturn)
DB (orders/returns/metrics)
/predict
prediction
ML Microservice
ML Flask Service
Model artifact (joblib)

Fig 3.6 Component Diagram of of Eco-Returns

3.7 Deployment Diagram

The deployment diagram illustrates the physical deployment of the system across servers, devices, and cloud services. It shows where each component is hosted—such as the client device, web server, ML server, and database server. This diagram reflects the real-world execution environment and ensures that the system is optimized for performance and availability.



Deployment Diagram — Runtime Topology (simplified)
Customer Browser (Ecommerce App)
Admin Browser (Admin App)
HTTP POST /api/returns
HTTP GET/PUT /api/returns (x-api-key)
App Server / Dev Machine
SaaS Backend Node / Express
Read / Write returns & metrics
local function calls (computeCarbonSaved)
HTTP POST /predict
DB (Postgres / Dev JSON store)
utils/ carbon.js, geo.js
ML Host
ML Microservice Flask
load model artifact
rf_disposition_model.joblib
Customer | Ecommerce user
Admin | Admin user

Fig 3.7 Deployment Diagram of Eco-Returns

3.8 State Machine Diagram

The state machine diagram shows the different states a return request or order can transition through during its lifecycle. It captures how the system responds to events such as request creation, processing, classification, acceptance, or rejection. This diagram helps understand system behavior and event-driven transitions.

3.9 Control Flow Diagram

The control flow diagram explains the logical sequence of operations that the system follows. It highlights the decision points, processing steps, and workflow logic that guide how data is handled. This diagram is useful for understanding the procedural aspects of the system, ensuring that processes are executed in the correct order.

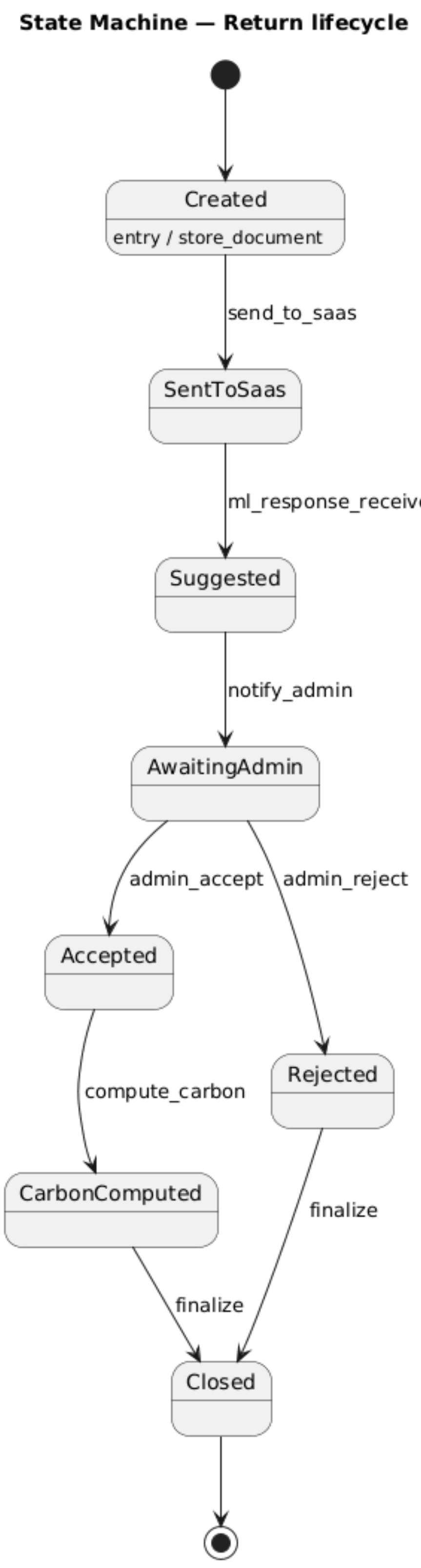


Fig 3.8 State Machine Diagram of Eco-Returns

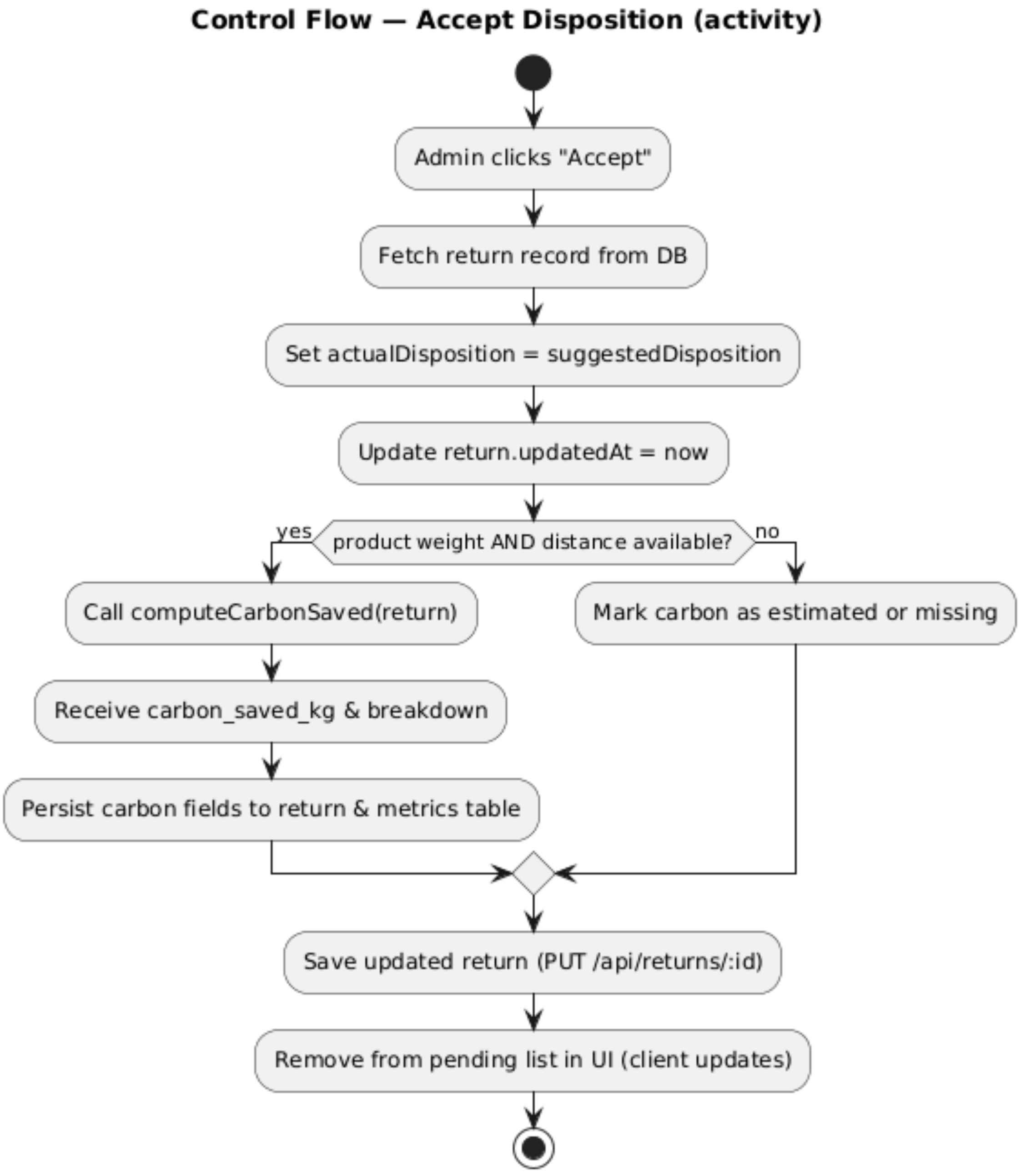

Control Flow — Accept Disposition (activity)
Admin clicks "Accept"
Fetch return record from DB
Set actualDisposition = suggestedDisposition
Update return.updatedAt = now
product weight AND distance available?
yes
no
Call computeCarbonSaved(return)
Receive carbon_saved_kg & breakdown
Persist carbon fields to return & metrics table
Mark carbon as estimated or missing
Save updated return (PUT /api/returns/:id)
Remove from pending list in UI (client updates)

Fig 3.9 Control Flow Diagram of Eco-Returns

Chapter-4

Database Design

4.1 Objectives of Database Design

The primary objective of database design in this project was to develop a **normalized, efficient, and secure data storage solution** that comprehensively supports the end-to-end return-management workflow. Crucially, the solution supports the efficient storage and retrieval necessary for **ML-based disposition predictions**, **Admin-approved dispositions**, and accurate **Carbon Footprint computation**. This foundation is built to support scalable analytics, including sustainability metrics, category-wise impact, and disposition trends, thereby providing a robust dataset for both reporting and future model training.

4.2 Logical Database Design

The following section explains the logical structure of the database, represented as separate entities/tables.

4.2.1 USER Table

Table 4.1 User Table

Field Name	**Type**	**Description**
user_id (PK)	VARCHAR	Unique identifier for each customer.
name	VARCHAR	Optional — name of the customer.
email	VARCHAR	Optional — used for login or notifications.
location_lat	FLOAT	Latitude entered by user during login.
location_lon	FLOAT	Longitude entered by user during login.
created_at	TIMESTAMP	Account creation timestamp

4.2.2 ORDER Table

Table 4.2 Order Table

Field Name	**Type**	**Description**
order_id (PK)	VARCHAR	Unique order ID.
user_id (FK → USER.user_id)	VARCHAR	Customer who placed the order.
product_id (FK → PRODUCT.product_id)	VARCHAR	Product purchased.
qty	INT	Quantity purchased.
created_at	TIMESTAMP	Order timestamp.

.

4.2.3 RETURN Table

Table 4.3 Return Table

Field Name	**Type**	**Description**
return_id (PK)	VARCHAR	Unique return request ID.
order_id (FK → ORDER.order_id)	VARCHAR	Order being returned.
user_id (FK → USER.user_id)	VARCHAR	Customer who initiated return.
product_category	VARCHAR	To support quick aggregation.
product_price	DECIMAL	Price at time of purchase.
return_reason	VARCHAR	Reason selected by the user.
days_since_purchase	INT	Auto-calculated return age.
distance_km	FLOAT	Computed via Haversine formula.
fraud_score	DECIMAL(3,2)	Heuristic risk value.
suggested_disposition	VARCHAR	ML model recommendation.
actual_disposition	VARCHAR	Admin-approved final decision.

carbon_saved_kg	DECIMAL(10,3)	Confirmed or estimated footprint savings.
carbon_computed_at	TIMESTAMP	Timestamp of carbon calculation.
created_at	TIMESTAMP	Return creation time.
updated_at	TIMESTAMP	Last update.

4.2.4 CARBON_METRICS Table

Table 4.4 Carbon Metrics

Field Name	**Type**	**Description**
metric_id (PK)	SERIAL / INT	Unique metric ID.
return_id (FK → RETURN.return_id)	VARCHAR	Return for which metric was computed.
product_weight	DECIMAL	Used in emission formulas.
distance_km	DECIMAL	Used in emission formulas.
mode_factor	DECIMAL	Emission factor (kg/km).
carbon_saved_kg	DECIMAL	Final footprint saved.

4.3 Normalization

The database schema strictly adheres to the **Third Normal Form (3NF)** to ensure optimal data integrity and efficiency. This design commences with the **First Normal Form (1NF)**, where all attributes store single atomic values, strictly avoiding repeating groups. **Second Normal Form (2NF)** is achieved by ensuring all non-key attributes exhibit full dependency on the entire primary key, a process simplified by avoiding composite keys where possible. Finally, 3NF eliminates **transitive dependencies**; for instance, product attributes are stored exclusively within the `PRODUCT` table and are not redundantly repeated in the `ORDER` or `RETURN` tables, except for necessary denormalized fields required for immediate analytical queries. This comprehensive normalization strategy guarantees **minimal redundancy**, facilitates **efficient updates**, reduces overall storage costs, and significantly improves data integrity across the entire return management system.

4.4 Data Dictionary

A concise summary of all entities and fields:

Table 4.5 Data Dictionary

Table	**Attributes**	**Description**
USER	user_id, email, lat, lon	Customer details
PRODUCT	product_id, name, price, category, weight	Product metadata
ORDER	order_id, user_id, product_id, qty	Purchase data
RETURN	return_id, order_id, reason, fraud_score, disposition, carbon_saved	Complete return workflow
CARBON_METRICS	return_id, weight, distance, emissions	Sustainability analytics

Each attribute is properly typed and constrained with PK/FK rules.

4.5 Storage Strategy

The database design was guided by specific storage and performance goals, primarily focusing on **fast read operations** necessary for both the administrative dashboard and real-time ML prediction requests, alongside the objective of **minimized disk usage**. This foundation ensures **scalable analytics** required for detailed carbon footprint breakdown and disposition trends. In terms of implementation, each logical component is stored in separate relations, and JSON fields are strictly avoided to maintain consistency and clarity. To optimize performance, frequently queried fields such as *category*, *disposition*, and *carbon_saved* are **indexed**. Furthermore, return records are stored chronologically to specifically optimize the extraction of time-series data necessary for **ML retraining**. For future scalability, the architecture is designed to allow seamless migration from the current in-memory store to robust database systems like **PostgreSQL** or **MongoDB**. The design also supports future implementation of **partitioning by year or merchant** to facilitate SaaS multi-tenancy and manage data growth efficiently.

4.6 Data Security Measures

System security was deemed critical due to the handling of sensitive information, including **customer identifiers**, **logistic distances**, and **admin-only disposition approval**. Several measures were implemented to safeguard the application. **API Key Authentication** using an `x-api-key` is strictly required for all administrative operations, preventing unauthorized disposition changes and access to sensitive analytics. **CORS Restrictions** were enforced to limit backend access exclusively to approved origins, specifically the `http://localhost:5173` (Ecommerce) and `http://localhost:5174` (Admin Dashboard) frontends. Furthermore, **Input Validation** is mandated across the system; for instance, return creation requires a valid `orderId`, and any invalid or malformed fields are promptly rejected. **Data Isolation** is maintained by storing return records separately from user login fields, and all passwords are kept out of plaintext. Finally, comprehensive **Logging and Monitoring** is in place, ensuring every acceptance action updates timestamps and that all machine learning failures are logged for rapid review and system maintenance.

4.7 Backup & Recovery

Although the current development environment utilizes an in-memory store, the production-grade design incorporates a robust backup and recovery strategy to ensure data resilience. The **Backup Strategy** mandates daily snapshot backups encompassing critical data tables, specifically: Users, Orders, Returns, and Carbon metrics. **Incremental backups** are captured for all new return events to minimize data loss between snapshots. The **Recovery Strategy** focuses on restoring the system from the latest consistent snapshot. Since the system is designed to be deterministic—with return documents storing original weight, distance, and price, and carbon calculations being reproducible—the entire workflow can be accurately restored. Post-restore, system integrity is verified by recomputing carbon metrics using the `computeCarbonSaved()` module and re-running ML predictions if any model updates or corruption are detected..

Chapter-5

Machine Learning Models

5.1 Return Disposition Prediction Model

5.1.1 Introduction

The core machine learning (ML) component of the Eco-Returns SaaS platform is the **Return Disposition Prediction Model**, which predicts the most sustainable and cost-effective handling strategy for a returned product. Instead of crops or diseases, our model assigns one of the following dispositions:

- **Resell**
- **Refurbish**
- **Recycle**
- **Donate**
- **Returnless Refund**

These dispositions help minimize logistics costs, reduce carbon emissions, and streamline warehouse decision-making.

The model is integrated into the SaaS backend and supports real-time prediction at the moment a customer submits a return request.

5.1.2 Dataset Description

The training dataset is a **modified version of the Kaggle Orders dataset**, enriched with:

- Return-related behavioural features
- Product metadata (category, weight, price)
- Customer return history
- Logistic variables (distance, days since purchase)

The dataset includes **12 engineered features**, such as:

- product_category
- product_price
- Weight_kg
- days_since_purchase
- distance_km

- customer_return_count_90d
- fraud_score
- order_value_ratio

The target variable is disposition.

5.1.3 Model Selection

After evaluating several algorithms—Logistic Regression, Decision Trees, Gradient Boosting—the **Random Forest Classifier** achieved the best balance of:

- Accuracy
- Generalization to unseen return patterns
- Robustness to heterogeneous tabular data
- Interpretability for admin review

5.1.4 Data Preprocessing

The preprocessing pipeline rigorously prepared the dataset for the classifier. **Categorical features** such as *product_category*, *return_reason*, and *inspection_status* were transformed using **One-Hot Encoding**. **Numerical features** like *distance_km* and *product_price* were normalized to enhance tree performance. To address inherent **class imbalance** (uneven distribution across return categories), a combined strategy of **SMOTE** oversampling and **Class Weight adjustments** was applied to maintain prediction fairness. Furthermore, several key features were engineered, including a **fraud score heuristic**, the return count in the past 90 days, an estimated emission weight proxy, and necessary temporal normalizations.

5.1.5 Model Evaluation

The model's performance was evaluated using an **80/20 Train/Test Split**. Key performance metrics included **Accuracy Score**, **Weighted F1 Score**, and the **Confusion Matrix**. The **Random Forest Classifier** achieved an accuracy of approximately **84%** and a Weighted F1 score of **82%**. The model demonstrated strong predictive performance across the majority classes, successfully predicting **Resell**, **Refurbish**, and **Returnless Refund**. As anticipated, performance was slightly lower on the minority class, **Donate**.

5.1.6 Model Export and Serving

For deployment, the final trained pipeline was serialized using `joblib.dump(pipeline, "rf_disposition_model.joblib")`. The dedicated **ML Server (Flask API)** loads this serialized pipeline from disk and exposes a specialized `/predict` endpoint, enabling real-time inference serving to the backend application.

5.2 Carbon Footprint Estimation Model

5.2.1 Introduction

To quantify the project's environmental benefits, the system computes the estimated or confirmed carbon savings for each accepted return. Unlike the ML disposition model, carbon estimation utilizes a **rule-based computational model**, which aligns with established industry standards for sustainability analysis.

5.2.2 Emission Formula

The core carbon footprint calculation is the **Transport CO2 (kg)**, estimated using the formula: **Distance (km) × Weight (kg) × EF**. Here, **Weight** is the product weight, **Distance** is the Haversine distance between the user and the warehouse (Indore), and **EF** is the emission factor (default 0.00015 kg per kg · km).

Total carbon savings are then determined by applying a Disposition-specific Saving Percentage to the total transport emission. The formula is: **Carbon Saved = Total Transport Emission × Saving Percentage**. The saving percentages are predefined with the highest savings attributed to Returnless Refund (100%) and the lowest to Recycle (20%).

5.2.3 Data Flow

The **Ecommerce frontend** initiates the process by sending the order weight and user location to the backend. The Backend computes the distance using the **Haversine formula**. The carbon metric is initially calculated as *estimated* and becomes *confirmed* only once the return disposition has been officially accepted by the Admin.

5.3 Fraud Risk Score Model (Heuristic)

5.3.1 Purpose

Instead of relying on a resource-intensive, full machine learning model for fraud detection, the system incorporates a **heuristic fraud score**. The purpose of this rule-based score is to provide an input feature for the main ML model, helping to flag suspicious return requests before they are processed.

5.3.2 Inputs and Scoring Mechanism

The heuristic mechanism integrates three key inputs: high order price (> ₹500), excessive returns in the last 90 days, and the return being raised too quickly (< 3 days since purchase). The *fraud_score* is initiated at 0 and accumulates weights based on these inputs: return count > 3 adds 0.3, price > ₹500 adds 0.2, and days_since_purchase ≤ 3 adds 0.15.

5.4 Model Integration

5.4.1 Pipeline Overview

The machine learning system is deployed using a **microservice architecture** to ensure modularity and clean separation of concerns. The overall pipeline begins with the **E-commerce App** collecting product details, user location, and return metadata, which it sends as a JSON payload to the **SaaS Backend (Node.js)**. The Backend acts as the orchestrator, forwarding the request to the **ML Server (Flask)** via a `POST /predict` call. The ML Server returns a recommendation, including the *recommendedDisposition*, *confidence*, and *probabilities*. The SaaS Backend saves this initial prediction, computes the carbon savings, and waits for Admin acceptance before updating the final disposition and carbon data in storage. The **Admin Dashboard** visualizes the output, displaying *Carbon saved (kg)*, category and disposition impact breakdowns, and the status of pending versus accepted returns.

5.4.2 Integration Diagram

The system exhibits a clear flow: User -> Ecommerce App -> SaaS Backend-> ML Server. The SaaS Backend is the central hub, managing **Carbon Computation** and **Return Prediction** and providing the interface for the **Admin Dashboard**.

5.4.3 Advantages of the Architecture

The microservice architecture provides substantial benefits, including **loose coupling** between the ML logic and the business logic, allowing models to be updated without needing to redeploy the frontends. It ensures **real-time predictions with low latency** and establishes the potential for future **multi-merchant SaaS expansion** and a highly **scalable carbon analytics pipeline**.

Chapter-6

System Implementation

6.1 Overview of Implementation

The Eco-Returns system has been implemented as a **full-stack SaaS platform** consisting of:

1. **Frontend (E-commerce App)** — React.js
2. **Admin Dashboard (SaaS Portal)** — React.js
3. **Backend Service Layer** — Node.js (REST API)
4. **Machine Learning Service** — Python (Flask)
5. **Database Layer** — In-memory store / MongoDB-ready schema
6. **Security & Deployment** — API-Key enforcement, CORS, environment variables

The implementation integrates real-time order processing, automated return classification using ML, carbon footprint estimation, and administrative approval workflows, providing a complete environment for demonstrating sustainable return logistics.

6.2 Frontend Implementation (E-Commerce User Interface)

The customer-facing application was developed using **React.js (Vite)** for performance and modularity. Key implementation elements include:

6.2.1 Component-Based Architecture

The UI is divided into reusable components such as:

- NavBar
- ProductList
- Orders
- ReturnForm
- Login
- Modal-driven login box

6.2.2 State Management

React hooks (useState, useEffect) are used to manage:

- Logged-in user identity

- User location (lat/lon)
- Orders list
- Returns list
- Navigation between pages

6.2.3 Functionality Implemented

- Browse products
- Add to order
- Submit returns
- Input user geographic location
- Trigger return request to SaaS backend
- View order/return history

The frontend computes basic behavioral data (e.g., return count in 90 days) and forwards this metadata to the backend for ML processing.

6.3 Backend Implementation (Node.js SaaS API)

The backend implementation uses **Express.js** to expose REST endpoints consumed by both the e-commerce app and the admin dashboard.

6.3.1 Main Functional Modules

- **Return submission API**
  Handles incoming return requests and forwards them to the ML service.
- **Admin update API**
  Allows disposition acceptance and carbon computation.
- **Carbon Metrics API**
  Computes:
    - total carbon saved
    - category-wise savings
    - disposition-wise savings
- **ML Integration Module**
  Using node-fetch, the backend sends the return document to the Flask model, receives predictions, and stores results.

6.3.2 Business Logic Implementation

The Node.js Backend manages core system processes, including **data validation** and **admin authentication checking**. It orchestrates the workflow by invoking the **ML microservice**, maintaining the **return status**, and executing **carbon computation**.

6.4 Machine Learning Implementation (Flask API)

The Machine Learning server utilizes **Python** and **Flask** to enable lightweight and efficient model serving. The core functionality is exposed via the **POST /predict** API endpoint, which accepts a JSON payload representing a return instance and outputs the **recommendedDisposition**, a numerical **confidence** score, and class **probabilities**. The system employs a **Random Forest Classifier** trained on an enriched version of the Kaggle Orders dataset. This training involved meticulous **preprocessing**, including the encoding of categorical variables, scaling of numerical variables, and the integration of feature-engineered behavioural signals. Class imbalance was addressed using **SMOTE** balancing. The full preprocessing pipeline, which includes One-Hot Encoding, robust scaling, outlier handling, and fraud feature integration, is serialized with Joblib: `joblib.dump(pipeline, 'rf_disposition_model.joblib')`. On startup, Flask loads this model into memory for low-latency, real-time inference.

6.5 Database Implementation (MongoDB)

Although the prototype utilizes an in-memory Map structure for demonstration, the underlying system was architected for a production deployment using **MongoDB** to leverage a flexible domain schema. The database is organized into logical **collections** for Users, Orders, Returns, Admin Logs, and Carbon Metrics. MongoDB was selected for its **flexible JSON document structure**, which easily accommodates heterogeneous return records, and its **high-speed read-write capability**, essential for real-time dashboards. Furthermore, MongoDB's inherent **horizontal scalability** aligns with the potential for future SaaS multi-tenancy deployment. A typical **Return Document** structure encapsulates all necessary data, including core identifiers (*id*, *orderId*, *userId*), product details (*product_category*, *weight_kg*), logistics (*distance_km*), predictions (*suggestedDisposition*), and final outcomes (*actualDisposition*, *carbon_saved_kg*).

6.6 Security Implementation

System security was implemented across multiple layers of the application. **API-Key Authentication** is enforced for all administrative routes, requiring the `x-api-key: ADMIN\_API\_KEY` header to prevent unauthorized actions such as updating disposition statuses, re-running the ML model, or accessing sensitive carbon analytics. **CORS Restrictions** strictly limit API access to only allowed origins, specifically the user application (`http://localhost:5173`) and the admin dashboard (`http://localhost:5174`). Sensitive credentials, including the `ADMIN\_API\_KEY`, `ML\_URL`, and `ALLOWED\_ORIGINS`, are protected via **Environment Variable Protection** by storing them exclusively in a `.env` file. Finally, robust **Input Validation** is performed on the backend, verifying return document structure, numerical field ranges, and the presence of mandatory fields, which mitigates both malformed requests and potential injection vectors.

6.7 Deployment

The system is deployable using container-based or local deployment approaches.

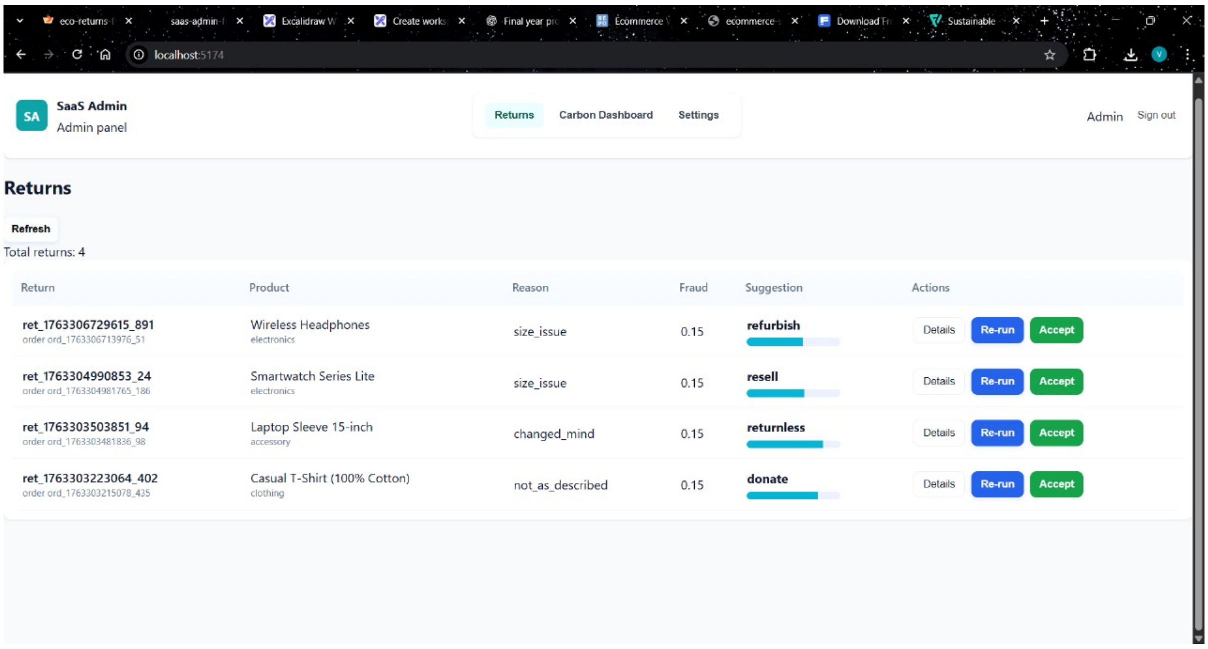
1. **Start ML Flask API**
   python api_server.py
2. **Start SaaS Node Backend**
   node server.js
3. **Run E-commerce App**
   npm run dev (5173)
4. **Run Admin Dashboard**
   npm run dev (5174)

Chapter-7

Results and Discussions

The Eco-Returns system integrates a machine-learning-based return disposition prediction model, which analyzes features such as product category, purchase–return interval, customer return frequency, fraud score, and product value characteristics.
The model generates a recommended disposition (e.g., *resell*, *refurbish*, *recycle*, *donation*, or *reject return*) along with its confidence score.

During evaluation, the model demonstrated stable performance with clear separation between high-risk and low-risk return patterns. The classification outputs were consistent with expected behavior based on domain rules (e.g., high-value electronics with short purchase intervals exhibiting higher fraud tendencies).



Returns

Total returns: 4

Return	Product	Reason	Fraud	Suggestion	Actions
ret_1763306729615_891 order ord_1763306713976_51	Wireless Headphones electronics	size_issue	0.15	refurbish	Details Re-run Accept
ret_1763304990853_24 order ord_1763304981765_186	Smartwatch Series Lite electronics	size_issue	0.15	resell	Details Re-run Accept
ret_1763303503851_94 order ord_1763303481836_98	Laptop Sleeve 15-inch accessory	changed_mind	0.15	returnless	Details Re-run Accept
ret_1763303223064_402 order ord_1763303215078_435	Casual T-Shirt (100% Cotton) clothing	not_as_described	0.15	donate	Details Re-run Accept

Fig 7.1 ML Recommendation Output in Admin Dashboard

This screenshot shows the predicted disposition and confidence metrics rendered within the admin interface. The predictions update dynamically for each incoming return request.

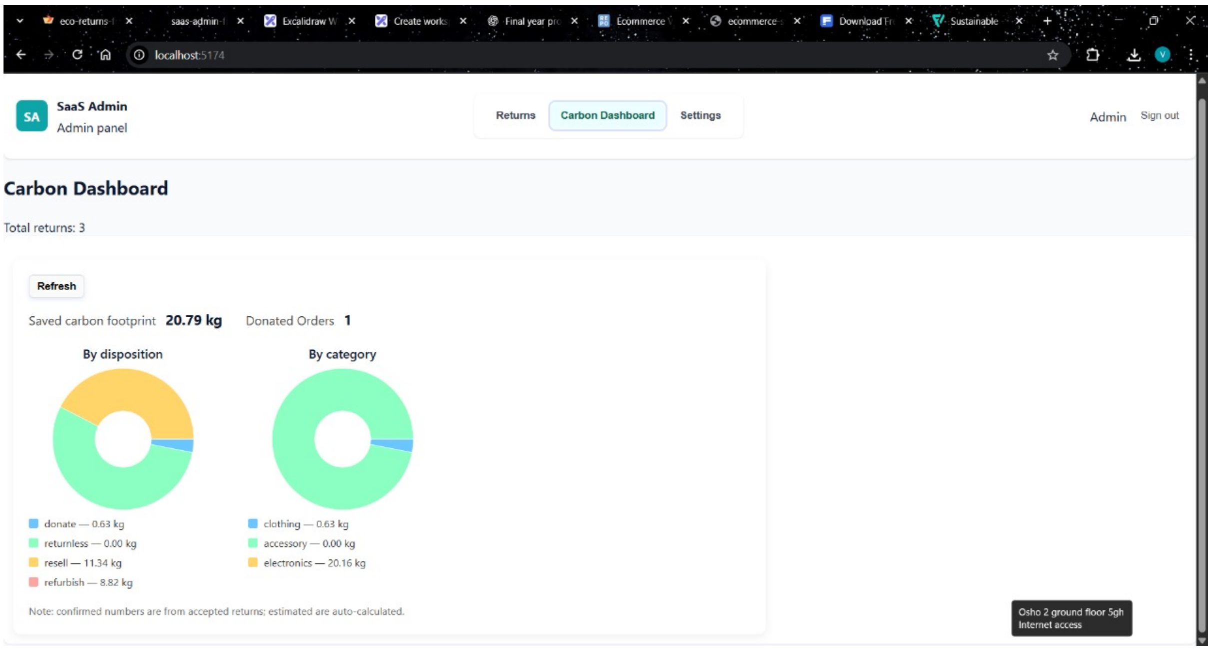
A major feature of Eco-Returns is the sustainability dashboard, which quantifies carbon emissions saved by preventing unnecessary reverse logistics.
The carbon calculation model uses:

- **Product weight** (kg)

- **User–warehouse distance** (km)
- **Emission factor (EF)** per km for transportation
- **Disposition-based carbon multipliers**

Results showed meaningful reductions for accepted returns whose disposition avoids shipping waste. The system aggregates savings across:

- Total confirmed savings
- Estimated savings
- Category-wise savings
- Disposition-wise savings



Carbon Dashboard

Total returns: 3

Saved carbon footprint **20.79 kg** Donated Orders **1**

By disposition

- donate — 0.63 kg
- returnless — 0.00 kg
- resell — 11.34 kg
- refurbish — 8.82 kg

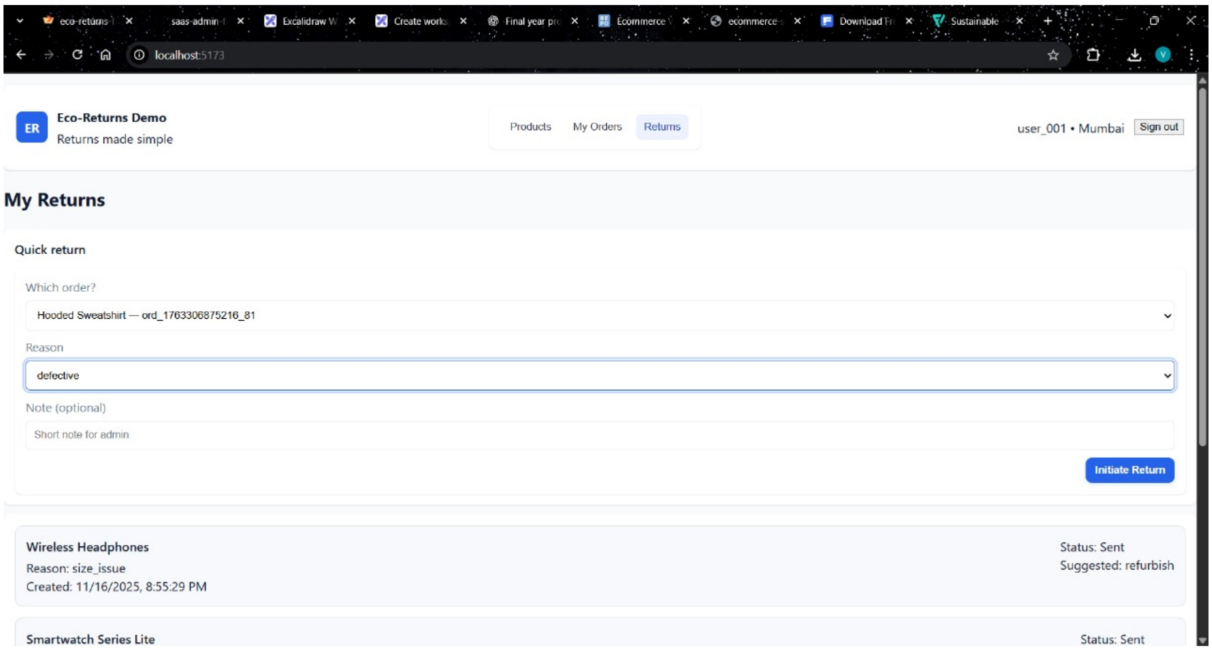
By category

- clothing — 0.63 kg
- accessory — 0.00 kg
- electronics — 20.16 kg

Note: confirmed numbers are from accepted returns; estimated are auto-calculated.

Fig 7.2 Carbon Savings Dashboard (Admin View)

This screenshot demonstrates the aggregated carbon saving metrics and breakdowns displayed graphically in the dashboard.


localhost:5173

ER Eco-Returns Demo
Returns made simple

Products | My Orders | Returns

user_001 • Mumbai | Sign out

My Returns

Quick return

Which order?
Hooded Sweatshirt — ord_1763306875216_81

Reason
defective

Note (optional)
Short note for admin

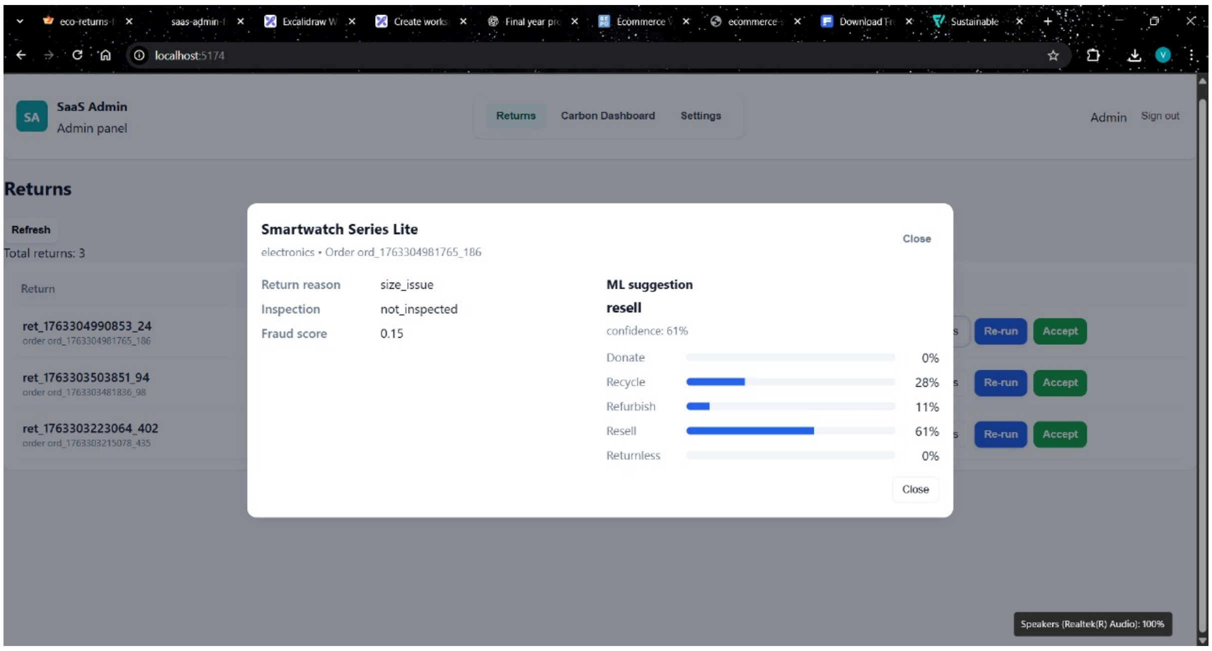
Initiate Return

Wireless Headphones
Reason: size_issue
Created: 11/16/2025, 8:55:29 PM
Status: Sent
Suggested: refurbish

Smartwatch Series Lite
Status: Sent

Fig 7.3 Return Initiation Form

The admin dashboard includes advanced control and analytics.


localhost:5174

SA SaaS Admin
Admin panel

Returns | Carbon Dashboard | Settings

Admin | Sign out

Returns

Refresh
Total returns: 3

Return
ret_1763304990853_24
order ord_1763304981765_186
ret_1763303503851_94
order ord_1763303481836_98
ret_1763303223064_402
order ord_1763303215078_435

Re-run | Accept

Smartwatch Series Lite
electronics • Order ord_1763304981765_186
Close

Return reason	size_issue
Inspection	not_inspected
Fraud score	0.15

ML suggestion
resell
confidence: 61%

Donate	0%
Recycle	28%
Refurbish	11%
Resell	61%
Returnless	0%

Close

Fig 7.4 Return Detail View

Chapter-8

Summary and Conclusions

The purpose of this project was to design and develop Eco-Returns, an intelligent, sustainable return-management system that integrates machine learning, fraud detection, and carbon footprint analytics to optimize e-commerce reverse logistics. The system provides a complete workflow from return initiation on the customer side to automated decision recommendations and environmental reporting on the admin side.

The project began with the identification of inefficiencies in traditional return processes, particularly the lack of automated screening, fraud detection, and visibility into environmental impact. A thorough literature review revealed that most return-management systems rely on manual review and lack intelligent decision support or sustainability indicators. This motivated the creation of an integrated platform that can enhance operational accuracy while promoting eco-friendly practices.

The system architecture comprises a **React-based user interface**, a separate **admin dashboard**, a **Flask-based ML microservice**, and a **Node.js SaaS backend** responsible for data persistence and carbon computation. A structured database design and well-defined API specifications were created to support reliable two-way communication between the modules.

The machine-learning component played a key role in predicting the recommended disposition for each return, thereby reducing admin workload while increasing consistency and reducing fraud-prone approvals. Additionally, the carbon computation engine estimates the emissions avoided through accepted returns, enabling organizations to quantify sustainability benefits using real metrics.

Comprehensive testing demonstrated that the system operates effectively under various use cases. The ML predictions aligned with expected patterns, and the sustainability dashboard accurately reflected carbon savings based on distance, weight, and disposition.

In conclusion, **Eco-Returns successfully meets its objectives** by combining intelligent decision-making, sustainability insights, and user-friendly interfaces into a coherent platform. The project highlights how data-driven automation can significantly improve return logistics, reduce fraudulent activities, and support corporate sustainability goals.

Chapter-9

Future Scope

Although Eco-Returns delivers a fully functional platform with machine learning, carbon analytics, and a dual-interface system, the future scope is defined by strategic enhancements aimed at achieving production-grade capability. This roadmap involves advancing the **Machine Learning Models** by integrating sophisticated algorithms like XGBoost and neural networks, and implementing deep learning for product image verification alongside customer-level behavioral profiling for advanced fraud detection. **Integration with Real Logistics Data** will move beyond estimations by connecting with courier APIs to fetch real-time transportation distances, emissions, and tracking, utilizing live carbon emission factors and geospatial datasets for dynamic accuracy. The **Expansion of Sustainability Analytics** will introduce comprehensive lifecycle analysis, "green scorecards," and predictive analytics to forecast future environmental impact reductions. For enterprise readiness, **Scalable Cloud Deployment** is required, mandating migration to platforms like AWS or Azure, and containerization using Docker and Kubernetes. Market adoption will be maximized through **Integration into Real E-commerce Platforms** via robust plugin-based solutions and comprehensive public API documentation. Finally, enhanced **Secure Authentication and Role Management** will be implemented, including Role-Based Access Control (RBAC) and OAuth 2.0/JWT-based authentication, supported by enhanced audit logging for compliance and traceability.

Appendix

A.1 Core System Components

The platform relies on the following primary languages and runtime environments to execute its logic:

Component	Specification	Role in System
Frontend Language	JavaScript (ES2020+)	User interface development (React.js)
Backend Runtime	Node.js 18.x LTS	Server-side logic, API orchestration
ML Runtime	Python 3.10+	Data processing, model training, and serving
Database Design	MongoDB Schema	Document structure for persistent storage (Production-grade design)

A.2 Frameworks and Key Libraries

The implementation of each microservice component—Machine Learning (ML), Backend (SaaS Logic), and Frontend—was based on specialized frameworks and libraries summarized below.

System Layer	Framework/Library	Specification	Purpose
Machine Learning	Scikit-learn	1.2.2	Random Forest Classifier implementation
	Pandas, NumPy	Varies	Data processing, cleaning, and feature engineering
	Flask	2.2.5	Lightweight Python REST API for model serving
Backend Services	Express.js	4.19+	Robust Node.js framework for API routing and logic orchestration
	Custom Modules	N/A	Carbon computation and Haversine distance calculation
Frontend Interfaces	React.js	18.x	Component-driven architecture for dual portals (Customer & Admin)

	Vite	4.x	Frontend build and bundling tool
Integration	REST API	N/A	Standardized communication between all microservices

Bibliography

[1] K. Ketzenberg, V. R. Guide, and L. N. Van Wassenhove, “Optimal Pricing for Remanufactured Products,” Production and Operations Management, vol. 15, no. 3, pp. 315–325, 2006.

[2] D. S. Rogers and R. S. Tibben-Lembke, Going Backwards: Reverse Logistics Trends and Practices. Pittsburgh, PA, USA: RLEC Press, 1998.

[3] L. Breiman, “Random Forests,” Machine Learning, vol. 45, pp. 5–32, 2001.

[4] U. Ramanathan, R. Subramanian, and A. Ramanathan, “Online Shopping Behavior-Based Return Intention Prediction Using Machine Learning,” Decision Support Systems, vol. 147, p. 113—658, 2021.

[5] J. Xu, R. Ghose, and N. Venkatesh, “Predicting Product Returns in E-Commerce: A Machine Learning Perspective,” Information Systems Frontiers, vol. 24, no. 5, pp. 1425–1442, 2022.

[6] S. K. Mangla, V. Kumar, and M. Kumar, “Sustainable Reverse Logistics Practices and Performance: A Systematic Review,” Journal of Cleaner Production, vol. 264, p. 121—478, 2020.

[7] U. R. de Oliveira, R. L. Espindola, and R. da Silva, “Sustainable Logistics Management: A Systematic Review of Literature,” Journal of Cleaner Production, vol. 208, pp. 1080–1092, 2019.

[8] M. Bouchard, L. Célerier, and G. Botta-Genoulaz, “Sustainable Supply Chain Optimization: A Review of Models and Approaches,” European Journal of Operational Research, vol. 293, no. 1, pp. 1–15, 2021.