الجمهورية الجزائرية الديمقراطية الشعبية

Republique Algérienne Démocratique Populaire
وزارة التعليم العالي والبحث العلمي

Ministère de l'Enseignement Supérieur de la Recherche Scientifique

Université 20 août 1955 Skikda — **جامعة 20 أوت 1955 – سكيكدة**

Faculte des Sciences
Departement d'informatique

كلية العلوم
قسم الإعلام الآلي

الرقم: .... / ق/ إ/ أ/ ك.ع/ 2026

Autorisation de Dépôt de Mémoire de Master

Je soussigné : **Adel Lahsasna**

Certifie que l'étudiant : **Karim Aouaouda**

Spécialité : **Master Intelligence Artificielle**

Ayant soutenu le projet intitulé : **Development of a Low-Cost Edge AI Pipeline for Waste Object Characterization and Mass Estimation**

A apporté les corrections nécessaires sur son manuscrit de Master.

Signature de l'encadreur

Scanné avec CamScanner

الجمهورية الجزائرية الديمقراطية الشعبية
وزارة التعليم العالي والبحث العلمي

University of 20 August 1955 - Skikda
Faculty of Sciences
Department of Computer Science



Master Thesis

Field: Computer Science
Specialization: Artificial Intelligence

Development of a Low-Cost Edge AI Pipeline for Waste Object Characterization and Mass Estimation

Prepared by: Karim Aouaouda
Supervisor: Dr. Adel Lahsasna, Senior Lecturer Class B
University: University of 20 August 1955 - Skikda

Academic Year: 2025–2026

Dedication

This work is dedicated to my mother, sister, family, teachers, and friends whose support made the work possible. It is also dedicated to the students and engineers who believe that practical artificial intelligence can be built with limited resources when methodology, persistence, and careful engineering are combined.

Acknowledgements

I express my sincere gratitude to Dr. Adel Lahsasna for his supervision, scientific guidance, and academic support throughout this work. His direction helped transform the project from an implementation idea into a structured engineering and research contribution.

I also extend my appreciation to Dr. Soufiane Boulehouache and to the teachers of the Artificial Intelligence branch at the Department of Computer Science, University of 20 August 1955 - Skikda, for their instruction, encouragement, and academic support.

Special thanks are addressed to Abderrahmen Aoualmia for research and dataset support, to Billel Gueffal for providing a laptop that helped the implementation process, and to Walid Abdaoui for sharing practical artificial-intelligence knowledge from an industrial environment.

Finally, I thank my mother, my sister, my family, and my friends for their patience, motivation, and continuous support.

Abstract

Waste characterization is an important step in modern waste management, recycling, and environmental monitoring. The central objective of this thesis is simple: we try to estimate the mass of waste objects using nothing more than a low-cost RGB camera. This objective is difficult because mass is not directly visible in an image; it must be inferred from object category, mask geometry, depth, material behavior, density, deformation, and calibration. This thesis presents a first full version of a low-cost Edge AI pipeline for waste object characterization and mass estimation. The system uses an RGB camera and an NVIDIA Jetson Nano target platform to perform object segmentation, monocular depth estimation, metric feature extraction, volume approximation, and hybrid mass prediction.

We followed a practical machine-learning engineering workflow. A segmentation dataset was prepared by combining and adapting TACO, AquaTrash, and RealWaste. Because several sources did not provide instance masks, the work included manual annotation, detection-assisted annotation, Segment Anything based mask generation, and CVAT correction. In total, 7,648 images were annotated during the wider dataset preparation effort, and 6,460 images with 9,762 instances were retained in the final segmentation training manifest. An Ultralytics YOLO26m-seg model was fine-tuned with Optuna-assisted hyperparameter search, ZenML pipeline organization, MLflow registration, and ONNX export. The final evaluated model achieved mask mAP@50:95 of 0.5453 on validation and 0.5016 on test. For mass estimation, a hybrid residual model was trained on 745 object-level samples. The selected Extra Trees residual model reduced test MAE from 44.31 g for the physics baseline to 23.18 g, corresponding to a 47.69% improvement.

The thesis documents the mathematical foundation, system design, training pipelines, deployment-oriented architecture, and remaining limitations. The results show that the proposed pipeline is technically feasible as a low-cost Edge AI approach, while also showing that robust mass estimation requires stronger physical calibration, real measured masses, and Jetson-specific latency validation in future work.

Keywords: Edge AI, waste characterization, instance segmentation, YOLO, monocular depth estimation, mass estimation, Jetson Nano, MLOps, computer vision.

Résumé

La caractérisation des déchets constitue une étape importante pour la gestion moderne des déchets, le recyclage et la surveillance environnementale. De nombreux systèmes de vision par ordinateur réalisent la classification, la détection ou la segmentation des objets de déchets, mais peu de travaux abordent l'estimation de la masse, car celle-ci n'est pas directement observable dans une image RGB et dépend de la géométrie, du matériau, de la densité, de la déformation et de la calibration. Ce mémoire présente une première version complète d'un pipeline Edge AI à faible coût pour la caractérisation des objets de déchets et l'estimation de leur masse. Le système utilise une caméra RGB et une plateforme cible NVIDIA Jetson Nano afin d'effectuer la segmentation d'objets, l'estimation de profondeur monoculaire, l'extraction de caractéristiques métriques, l'approximation du volume et la prédiction hybride de la masse.

Le projet suit une démarche pratique d'ingénierie en apprentissage automatique. Un jeu de données de segmentation a été préparé en combinant et en adaptant TACO, AquaTrash et RealWaste. Comme plusieurs sources ne fournissaient pas de masques d'instances, le travail a inclus l'annotation manuelle, l'annotation assistée par détection, la génération de masques avec Segment Anything et la correction avec CVAT. Au total, 7 648 images ont été annotées dans l'effort global de préparation des données, et 6 460 images contenant 9 762 instances ont été retenues dans le manifeste final d'entraînement de segmentation. Un modèle Ultralytics YOLO26m-seg a été ajusté avec une recherche d'hyperparamètres assistée par Optuna, une organisation de pipeline ZenML, un enregistrement MLflow et une exportation ONNX. Le modèle évalué atteint un mask mAP@50:95 de 0.5453 en validation et de 0.5016 en test. Pour l'estimation de masse, un modèle hybride résiduel a été entraîné sur 745 échantillons objet. Le modèle résiduel Extra Trees sélectionné réduit la MAE de test de 44.31 g pour la base physique à 23.18 g, soit une amélioration de 47.69%.

Le mémoire documente les fondements mathématiques, la conception du système, les pipelines d'entraînement, l'architecture orientée déploiement et les limites restantes. Les résultats montrent la faisabilité technique de l'approche Edge AI à faible coût, tout en soulignant la nécessité d'une meilleure calibration physique, de masses réellement mesurées et d'une validation de latence spécifique au Jetson dans les travaux futurs.

Mots-clés: Edge AI, caractérisation des déchets, segmentation d'instances, YOLO, profondeur monoculaire, estimation de masse, Jetson Nano, MLOps, vision par ordinateur.

List of Abbreviations

AI	Artificial Intelligence
AP	Average Precision
API	Application Programming Interface
CVAT	Computer Vision Annotation Tool
DVC	Data Version Control
FPS	Frames Per Second
GPU	Graphics Processing Unit
IoU	Intersection over Union
MAE	Mean Absolute Error
mAP	mean Average Precision
ML	Machine Learning
MLP	Multi-Layer Perceptron
MLflow	Machine-learning lifecycle and model registry platform
MLOps	Machine Learning Operations
MQTT	Message Queuing Telemetry Transport
ONNX	Open Neural Network Exchange
RGB	Red Green Blue image format
RMSE	Root Mean Square Error
SAM	Segment Anything Model
sMAPE	Symmetric Mean Absolute Percentage Error
TACO	Trash Annotations in Context
YOLO	You Only Look Once

Contents

Dedication i

Acknowledgements ii

Abstract iii

Résumé iv

List of Abbreviations vi

1 General Introduction 1

1.1 Context 1

1.2 Problem Statement 2

1.3 Objectives 2

1.4 Research Questions 3

1.5 Contributions 3

1.6 Thesis Organization 4

2 Theoretical Background 5

2.1 Computer Vision for Waste Characterization 5

2.2 Instance Segmentation 5

2.3 Monocular Depth Estimation 6

2.4 Camera Geometry and Volume Approximation 7

2.5 Mass Estimation 7

2.6 Evaluation Metrics 8

2.7 Edge AI 8

3 State of the Art and Related Work **10**

3.1 Waste Datasets 10

3.2 Positioning Against Existing Work 11

3.3 Segmentation Models 12

3.4 Monocular Depth Models 14

3.5 Edge Hardware 14

3.6 MLOps and Deployment Tools 15

4 Proposed Methodology **16**

4.1 Overview 16

4.2 Engineering Rationale for the Main Technical Choices 16

4.3 Dataset Preparation for Segmentation 18

4.4 Segmentation Training Pipeline 19

4.5 Mass Dataset Preparation 20

4.6 Feature Engineering for Mass Estimation 21

4.7 Deployment-Oriented Methodology 21

5 System Design and Implementation **23**

5.1 Repository Structure 23

5.2 System Architecture Overview 24

5.3 Inference Pipeline Implementation 26

5.4 Geometry Module 27

5.5 Mass-Estimation Implementation 27

5.6 Edge Agent and Backend Communication 27

5.7 Release-Based Update System 30

5.8 Hardware and Software Environment 30

6 Experiments, Results and Discussion **33**

6.1 Segmentation Dataset Results 33

6.2 Segmentation Training and Evaluation 33

6.3 Segmentation Export and Registration 36

6.4 Mass Dataset Results 36

6.5 Mass Model Selection 37

6.6 Mass Evaluation 37
6.7 End-to-End Status 40
6.8 Interpretation of Segmentation Results 40
6.9 Interpretation of Mass Results 41
6.10 Error Propagation 41
6.11 Deployment Considerations 41
6.12 Threats to Validity 42

7 General Conclusion and Future Work **43**
7.1 Conclusion 43
7.2 Future Work 43

Appendix A **45**
Annotation and Tool Evidence 45
A.1 Annotation Workflow Screenshots 45
A.2 Mass Acquisition and Annotation Tool 45
A.3 Tool and Model References 46

Appendix B **47**
Artifact Traceability 47

List of Figures

4.1 Proposed edge inference cascade. Primary modules are used when available, while fallback modules preserve operation under failure or resource constraints. 16

4.2 Annotation activity flow used to transform heterogeneous waste-image sources into a segmentation-ready dataset. 18

4.3 Project annotation screenshots placed with the dataset-preparation method they support. 18

4.4 Segmentation dataset composition generated from the final dataset manifest. 19

4.5 MLOps-oriented segmentation training and publication pipeline. 20

4.6 Screenshots of tools used to organize, tune, and publish the segmentation training pipeline. 20

4.7 Mass acquisition and annotation tool screenshots supporting the object-level mass-feature protocol. 20

4.8 Mass-estimation class distribution generated from the processed feature summary. 21

5.1 Full system architecture with the main interactive components and data flows. RGB frames enter the edge cascade, MQTT carries telemetry and commands, HTTP carries results and media to the backend, and the release channel provides model and configuration updates. 25

5.2 Use-case diagram for the main system actors and operational functions, organized by actor role. Each line represents a distinct use case supported by the actor rather than a sequential pipeline. 26

5.3 Sequence-style operation map for four important edge/backend interactions: telemetry, inference, preview, and release-based update. 28

5.4 Dashboard screenshots showing the device list, device telemetry state, and inference-request entry point used by the implemented edge workflow. . . . 29

5.5 Dashboard inference execution screenshots showing the request lifecycle and the live stage sequence reported by the edge agent. 29

5.6 Dashboard inference-result screen showing the integrated output: annotated RGB segmentation overlay, returned depth preview, estimated total mass, per-object class labels, confidence values, and mass estimates. This screen demonstrates functional integration of the inference path, but it is not treated as a controlled Jetson Nano benchmark. 30

5.7 Project Jetson Nano hardware used as the target low-cost edge device. . . . 31

5.8 Dockerized service stack running the backend-supporting services used during system development and testing. 32

6.1 YOLO segmentation training curves from the recorded training artifacts. . . 34

6.2 Mask precision-recall curve from the segmentation training artifacts. 35

6.3 Normalized confusion matrix from the segmentation training artifacts. . . . 35

6.4 Mass-estimation test metric comparison generated from the test metrics artifact. 38

6.5 Mass-estimation diagnostic plots from the test evaluation artifacts. 39

6.6 Per-class and error-distribution diagnostics for the mass model. 39

A.1 CVAT-based annotation and correction screenshots from the project thesis-screenshot archive. 45

A.2 Screenshots of the mass acquisition and annotation workflow. 45

A.3 Annotation and segmentation tools referenced by the project. 46

A.4 Depth and MLOps tools used in the project workflow. 46

A.5 Deployment-supporting tools and hardware documentation screenshots. . . 46

List of Tables

1.1 Main project contributions and supporting artifacts. 4

3.1 Comparison of waste datasets and project-specific adaptation. 11
3.2 Positioning of the proposed work against typical existing approaches. . . . 12
3.3 Segmentation-model comparison for the project context. 13
3.4 Depth-estimation approaches considered by the project. 14
3.5 Edge-device comparison as of 2026-07-06. 15

4.1 Engineering rationale for the selected technologies and modeling choices. . 17
4.2 Final segmentation dataset manifest. 19

5.1 Main repository modules used by the thesis. 24

6.1 Segmentation dataset split summary. 33
6.2 YOLO segmentation evaluation metrics. 34
6.3 Segmentation model deployment artifacts. 36
6.4 Mass dataset split summary. 37
6.5 Mass-estimation metrics on validation and test sets. 38
6.6 Test MAE by class for the physics baseline and hybrid residual model. . . . 39

B.1 Traceability table for the main thesis claims. 47

Chapter 1

General Introduction

1.1 Context

The management of municipal, industrial, and informal waste streams increasingly depends on reliable information about the type, quantity, and physical characteristics of discarded objects. Conventional waste characterization relies on manual inspection, sorting, weighing, and reporting. These procedures are useful but slow, repetitive, labor-intensive, and difficult to deploy continuously in outdoor or low-resource locations. Computer vision can reduce part of this burden by detecting waste categories from images, while Edge AI can move the inference process from cloud servers to local devices placed near the source of data.

We address this context through an edge-deployable computer-vision system for waste object characterization and mass estimation. The central engineering question is deliberately practical: can we estimate the mass of waste objects using only a cheap RGB camera? This question matters because recognition alone is not enough for operational decisions. A plastic bottle and a piece of cardboard may both be correctly classified, but collection planning, recycling yield estimation, and material-flow analysis also need approximate quantities, volume, or mass.

The system is designed around a low-cost setup composed of an RGB camera, a fixed acquisition plane, and an NVIDIA Jetson Nano device. Every technical block in the pipeline supports the same objective. Segmentation isolates each object, depth estimation gives approximate 3D structure, geometry converts pixels into physical features, density provides a physics baseline, and residual learning corrects systematic errors. The training pipeline follows a machine-learning engineering workflow with dataset preparation, annotation, hyperparameter optimization, experiment tracking, model export, and deployment-oriented packaging.

1.2 Problem Statement

Most public waste-vision datasets focus on image-level classification, object detection, or instance segmentation. Examples include TrashNet for single-object classification, TACO for in-context litter segmentation, AquaTrash for water-body waste detection, and RealWaste for landfill-image classification [1–4]. These datasets are valuable but they generally do not provide physical mass, object dimensions, calibrated depth, or volume labels. Consequently, a model trained only for detection or segmentation cannot directly infer the amount of material represented by each object.

The main research problem is therefore formulated as follows: how can a low-cost Edge AI system estimate the mass of visible waste objects using RGB imagery, segmentation masks, monocular depth, material priors, and a learned correction model, while remaining deployable on resource-limited hardware?

This problem contains three technical difficulties. First, waste objects are visually diverse and often deformed, occluded, dirty, hollow, or partially visible. Second, monocular depth is ambiguous unless the camera setup and acquisition plane are constrained. Third, mass depends not only on apparent size, but also on material density, internal emptiness, compaction, and object state. The thesis treats segmentation, geometry, and mass estimation as connected stages rather than independent tasks.

1.3 Objectives

The main objective is to build and document a complete first version of an Edge AI pipeline that estimates waste-object mass from RGB imagery. The work is organized around the following objectives:

O1. prepare a segmentation dataset for visible waste objects from heterogeneous public sources;

O2. train a deployable instance segmentation model that can provide object masks for downstream geometry;

O3. design a geometry-aware inference cascade that combines masks, monocular depth, material priors, and volume approximation;

O4. build a hybrid residual mass-estimation model that improves a physics-only density-volume baseline;

O5. export and register model artifacts so the system can move toward edge deployment;

O6. evaluate the system using traceable artifacts rather than unsupported claims.

The engineering objective is equally important. We do not only train a model in a notebook; we follow a realistic machine-learning engineering path that includes manual annotation, dataset transformation, MLOps pipeline design, experiment registration, model export, model-release planning, and edge-agent integration.

1.4 Research Questions

We organize the work around four research questions:

RQ1. How can heterogeneous public waste datasets be transformed into a unified segmentation dataset suitable for object-level training?

RQ2. Can a compact YOLO segmentation model provide masks that are useful for downstream geometry and mass estimation?

RQ3. How can monocular depth, segmentation masks, material density, and residual regression be combined to estimate mass from RGB imagery?

RQ4. What system architecture is required to move the trained pipeline from experimental artifacts toward a Jetson Nano deployment?

1.5 Contributions

We make the following data-centric and system-centric contributions:

C1. an annotation effort covering 7,648 images during the broader preparation process, including manual annotation, SAM-assisted mask generation, and CVAT correction;

C2. a final merged segmentation dataset of 6,460 retained images and 9,762 object instances;

C3. a YOLO26m-seg model with validation and test metrics, an ONNX export artifact, and MLflow registration lineage;

C4. a 745-row object-level mass dataset and a hybrid residual model that improves test MAE by 47.69% over a density-volume physics baseline;

C5. a deployable cascade architecture with primary and fallback modules, MQTT telemetry, Docker-based supporting services, and release-driven edge updates.

Table 1.1: Main project contributions and supporting artifacts.

Contribution	Evidence used in the thesis
Segmentation dataset preparation	7,648 annotated images in the broader preparation effort; final training manifest with 6,460 images and 9,762 instances under `train_output/training_outputs/metadata/dataset-manifest.json`.
Segmentation model	YOLO26m-seg training and evaluation reports under `train_output/training_outputs/reports/`; curves under `train_output/training_outputs/training/curves/`.
Model optimization and registration	Optuna best-trial artifact, MLflow registration report, and ONNX export manifest under `train_output/training_outputs/`.
Mass-estimation model	Hybrid residual model artifacts under `mass_model/artifacts/mass_estimation/mass-model-feature-residuals-notebook/`.
Edge-oriented runtime	Pipeline, agent, geometry, mass, and release-update modules under `src/edge_ai_mass/`.

1.6 Thesis Organization

The rest of the thesis is organized as follows:

- Chapter 2 presents the theoretical background: segmentation, monocular depth, volume approximation, mass regression, Edge AI, and evaluation metrics.
- Chapter 3 positions the work against related datasets, segmentation models, depth-estimation approaches, edge devices, and MLOps tools.
- Chapter 4 describes the proposed methodology, from annotation and dataset construction to hybrid residual mass estimation.
- Chapter 5 presents the system design and implementation, including runtime modules, edge-agent communication, and release-based updates.
- Chapter 6 reports the segmentation and mass-estimation experiments.
- Chapter 7 discusses interpretation, limitations, and threats to validity.
- Chapter 8 concludes the thesis and proposes future work.

Chapter 2

Theoretical Background

2.1 Computer Vision for Waste Characterization

Computer vision converts visual measurements into structured information such as object categories, locations, masks, geometric estimates, and scene-level metadata. In waste characterization, the same image may be used to identify material classes, estimate object quantities, monitor collection sites, or support sorting decisions. A classification model assigns one label to an image or crop. An object detector predicts bounding boxes and class labels. An instance segmentation model predicts one mask for each visible object, preserving object-level separation even when multiple objects belong to the same class.

For mass estimation, instance segmentation is more useful than image classification because it defines the pixel region belonging to each object. The mask supports area estimation, depth aggregation, shape descriptors, and separation of multiple objects placed on the same plane. The segmentation stage is therefore the entry point of the proposed pipeline.

2.2 Instance Segmentation

Let an RGB image be denoted by $I \in \mathbb{R}^{H \times W \times 3}$. For each object instance k, an instance segmentation model predicts a class label c_k, a confidence score s_k, a bounding box $b_k = (x_1, y_1, x_2, y_2)$, and a binary mask $M_k \in \{0, 1\}^{H \times W}$. The mask identifies pixels assigned to the object:

$$M_k(u, v) = \begin{cases} 1, & \text{if pixel } (u, v) \text{ belongs to object } k, \\ 0, & \text{otherwise.} \end{cases} \tag{2.1}$$

Segmentation quality is commonly evaluated using Intersection over Union. Given a

predicted mask $\hat{M}$ and a ground-truth mask M, IoU is:

$$\text{IoU}(\hat{M}, M) = \frac{|\hat{M} \cap M|}{|\hat{M} \cup M|}. \tag{2.2}$$

The average precision is computed by integrating precision across recall values at a given IoU threshold. The mean average precision, mAP, averages AP over classes and often over multiple IoU thresholds. The COCO-style mAP@50:95 averages AP at thresholds from 0.50 to 0.95 and is stricter than mAP@50.

The training objective of a YOLO segmentation model can be represented as a weighted multi-task loss:

$$\mathcal{L}_{seg} = \lambda_{box}\mathcal{L}_{box} + \lambda_{cls}\mathcal{L}_{cls} + \lambda_{mask}\mathcal{L}_{mask} + \lambda_{dfl}\mathcal{L}_{dfl}, \tag{2.3}$$

where the terms correspond to localization, classification, mask prediction, and distribution-based box refinement. The exact implementation depends on the model version and framework, but the important property is that the network is optimized jointly for detection and mask quality.

2.3 Monocular Depth Estimation

Depth estimation predicts the distance between the camera and visible scene points. A stereo or RGB-D camera can provide direct depth measurements, but the proposed system uses a low-cost RGB camera. Monocular depth models infer a depth map $D \in \mathbb{R}^{H \times W}$ from one image. Models such as MiDaS and Depth Anything learn depth priors from large datasets and can generalize to unseen scenes [5–7].

Monocular depth may be metric or relative. If the output is relative, a scale and offset are required to approximate metric depth:

$$D_m(u, v) = \alpha D_r(u, v) + \beta, \tag{2.4}$$

where D_r is the raw model output, D_m is metric depth, and α, β are calibration parameters. The repository runtime makes this conversion explicit through a depth-scale configuration. In the fixed-plane setup, a known background distance can also be used to transform depth differences into object heights.

2.4 Camera Geometry and Volume Approximation

In a calibrated pinhole camera, a pixel (u, v) with depth Z corresponds to a 3D point:

$$X = \frac{(u - c_x)Z}{f_x}, \qquad Y = \frac{(v - c_y)Z}{f_y}, \qquad Z = D_m(u, v), \tag{2.5}$$

where f_x, f_y are focal lengths and c_x, c_y is the principal point. The physical area represented by one pixel increases with distance. Under calibration, the approximate pixel area at depth Z is:

$$A_{px}(Z) \approx \frac{Z^2}{f_x f_y}. \tag{2.6}$$

When no full calibration is available, the implementation uses a configured pixel-to-meter factor as a controlled approximation.

For a fixed camera observing a known background plane, the height of an object pixel can be approximated by the difference between the background depth and the object depth:

$$h(u, v) = \max(0, D_{bg}(u, v) - D_{obj}(u, v)). \tag{2.7}$$

The object volume can then be estimated by integrating height over the object mask:

$$\hat{V}_k = \sum_{(u,v)} M_k(u, v) A_{px}(D_{bg}(u, v)) h(u, v). \tag{2.8}$$

This equation is the mathematical basis of the background-aware geometry estimator in the runtime. When a background depth map is unavailable, the system falls back to a constant background distance and a default thickness, which is less accurate but useful for development and smoke testing.

2.5 Mass Estimation

The simplest mass estimate multiplies volume by material density:

$$\hat{m}_{base} = \rho(c_k)\hat{V}_k, \tag{2.9}$$

where $\rho(c_k)$ is a material-specific effective density. For waste objects, effective density is not equal to solid material density. A plastic bottle, for example, contains air and may be crushed, so the effective density of the visible object is much lower than the density of solid PET. This motivates the use of empirical correction.

The proposed mass model uses a residual formulation. The physics baseline is first

computed from volume and density. The learning target is then the residual:

$$r_i = m_i - m_{base,i}. \tag{2.10}$$

A regression model $f_\theta(x_i)$ predicts the correction from object-level features x_i, and final mass is reconstructed as:

$$\hat{m}_i = m_{base,i} + f_\theta(x_i). \tag{2.11}$$

The model is trained by minimizing an error criterion over the training set:

$$\min_\theta \frac{1}{N} \sum_{i=1}^{N} w_i \left| m_i - (m_{base,i} + f_\theta(x_i)) \right|, \tag{2.12}$$

where w_i can reduce the influence of flagged outliers. The repository selected an Extra Trees residual regressor based on validation hybrid MAE.

2.6 Evaluation Metrics

Segmentation is evaluated using precision, recall, AP, and mAP. Precision and recall are defined as:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}. \tag{2.13}$$

Mass estimation is evaluated using MAE, RMSE, median absolute error, R^2, MAPE, and sMAPE:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^{N} |m_i - \hat{m}_i|, \tag{2.14}$$

$$\text{RMSE} = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (m_i - \hat{m}_i)^2}, \tag{2.15}$$

$$\text{sMAPE} = \frac{100}{N} \sum_{i=1}^{N} \frac{2|m_i - \hat{m}_i|}{|m_i| + |\hat{m}_i|}. \tag{2.16}$$

MAPE is sensitive when the true mass is very small. Since the mass dataset includes lightweight objects with masses close to a few grams, sMAPE and absolute gram errors are more stable for interpretation.

2.7 Edge AI

Edge AI refers to executing artificial-intelligence inference close to the data source rather than sending all raw data to a cloud server. In this project, Edge AI is important for offline

operation, privacy, reduced network dependency, and lower end-to-end latency. The target hardware is an NVIDIA Jetson Nano 4GB Developer Kit, a low-cost embedded AI platform with a quad-core Arm CPU, a 128-core Maxwell GPU, and 4 GB LPDDR4 memory shared by the integrated CPU and GPU according to NVIDIA's official product information [8]. The project documentation identifies the deployment environment as JetPack 4.6.6 / L4T 32.7.6.

Edge constraints shape model design. Large models may achieve higher accuracy but exceed memory, latency, or thermal limits. Therefore, the runtime is implemented as a cascade with primary models and fallback modules. The segmentation stage uses YOLO as the primary detector/segmenter, depth uses Depth Anything with MiDaS as fallback, and mass estimation uses a learned model with density fallback.

Chapter 3

State of the Art and Related Work

3.1 Waste Datasets

Public datasets for waste vision differ significantly in task, scene complexity, annotation type, and suitability for physical estimation. TrashNet contains 2,527 images across six classes on relatively simple backgrounds and has been widely used for waste classification [1]. TACO provides in-context litter images with object annotations and segmentation masks, making it more relevant for detection and segmentation [2]. AquaTrash adapts waste detection to water bodies and provides bounding boxes for a small four-class dataset [3, 9]. RealWaste contains real landfill images across material categories and is valuable for realistic appearance variation, but it is designed primarily for classification [4].

Table 3.1: Comparison of waste datasets and project-specific adaptation.

Dataset	Approx. size	Main task	Annotation type	Relevance and limitation
TrashNet	2,527 images	Classification	Image labels	Useful for material appearance, but simple background and no masks or mass labels.
TACO	1,500 images	Detection / segmentation	Boxes and masks	In-context litter images with segmentation, but dataset size is limited for deployment robustness.
AquaTrash	369 images	Detection	Boxes	Useful for glass, metal, paper, and plastic examples; no segmentation masks in the original dataset.
RealWaste	4,752 images	Classification	Image labels	Real landfill conditions and material variety; masks had to be generated and corrected.
Project segmentation dataset	6,460 retained images	Instance segmentation	YOLO masks	Unified eight-class dataset for training; no physical mass labels.
Project mass dataset	745 object rows	Mass regression	Object features and mass target	Supports hybrid mass modeling; labels include pseudo-estimation limitations.

The absence of mass labels in public datasets is the central gap addressed by the mass-estimation part of this work. We therefore separate the visual segmentation dataset from the object-level mass dataset and combine the two through geometry and feature extraction.

3.2 Positioning Against Existing Work

Existing waste-vision systems usually stop at classification, detection, or segmentation. These outputs are useful for recognizing material categories, but they do not directly answer the operational question of approximate quantity. Our contribution is positioned as an engineering bridge between visual recognition and physical estimation: we use RGB imagery to produce masks, infer depth, approximate geometry, and estimate mass through a

physics-plus-learning model.

Table 3.2: Positioning of the proposed work against typical existing approaches.

Approach	Typical output	Main limitation for mass estimation	Position of this work
Image classification systems	One material class per image or crop.	No object mask, no object separation, no geometric information.	We use classification implicitly through object classes, but add masks, depth, and object-level features.
Object detection systems	Bounding boxes and class labels.	Box area is too coarse for irregular or hollow waste objects.	We use instance masks because pixel-level object support is more suitable for geometry and volume.
Instance segmentation systems	Object masks and classes.	Usually stop at recognition and do not estimate physical quantity.	We use segmentation as the first stage of a mass-estimation pipeline.
Depth-estimation studies	Relative or metric depth maps.	Depth alone does not identify material, object boundaries, or density.	We combine depth with masks, material priors, and background-aware volume approximation.
Cloud-based waste analytics	Centralized detection or classification reports.	Network dependency and privacy concerns in low-resource environments.	We target an offline-capable Jetson Nano edge setup with deployable artifacts.
Proposed approach	Class, mask, depth-assisted volume, and mass estimate.	First version still needs stronger physical calibration and measured mass labels.	The novelty is the RGB-only, low-cost bridge from waste recognition to approximate mass.

3.3 Segmentation Models

Classical semantic segmentation architectures such as U-Net and DeepLabv3+ assign class labels to pixels but do not naturally separate individual objects [10, 11]. Mask R-CNN is a

strong instance segmentation baseline that predicts masks for object proposals [12]. However, its two-stage design is often heavier than single-stage models for edge deployment. SAM is a promptable foundation model for segmentation and is useful for annotation acceleration, but it is not designed as a compact edge inference model in this project setting [13]. Ultralytics YOLO segmentation models provide a practical balance between speed, deployment tooling, and instance mask prediction [14].

Table 3.3: Segmentation-model comparison for the project context.

Model family	Output type	Strength	Limitation for this project
U-Net	Semantic mask	Strong pixel localization and data-efficient training in many domains.	Does not directly separate multiple instances of the same class.
DeepLabv3+	Semantic mask	Multi-scale context and refined boundaries.	Semantic output is less convenient for object-level mass estimation.
Mask R-CNN	Instance masks	High-quality instance masks and strong research baseline.	Heavier two-stage inference for low-cost edge hardware.
SAM	Promptable masks	Excellent annotation assistance and zero-shot mask proposal.	Requires prompts and is too heavy for the intended Jetson Nano runtime.
YOLO26m-seg	Instance masks	Single-stage inference, practical tooling, export support, and real-time orientation.	Compactness can reduce performance on visually ambiguous classes.

We use SAM during dataset preparation and YOLO26m-seg during runtime-oriented training. This separation is important: SAM acts as an annotation accelerator, while YOLO is the deployable model.

3.4 Monocular Depth Models

Monocular depth estimation has advanced through large-scale training and transformer backbones. MiDaS demonstrated robust cross-dataset relative depth by mixing multiple datasets [5]. Depth Anything expanded the role of large-scale unlabeled data and pseudo-labeling for generalizable depth [6]. Depth Anything V2 improves fine details and robustness with stronger teacher-student training and synthetic data practices [7]. These models are attractive because they avoid the cost of depth cameras, but they also introduce scale ambiguity and require calibration for physical measurements.

Table 3.4: Depth-estimation approaches considered by the project.

Approach	Role	Suitability
MiDaS small	Runtime fallback	Lightweight and robust relative depth, but requires scale calibration.
Depth Anything	Primary depth family	Strong generalization and practical model availability.
Depth Anything V2 Small	Preferred primary model	Better detail and robustness than earlier versions, with a smaller model variant suitable for edge exploration.
RGB-D sensor	Alternative hardware	More direct metric depth, but increases cost and changes the low-cost camera assumption.

3.5 Edge Hardware

The hardware choice determines the feasible model size and latency. Raspberry Pi 5 is a strong general single-board computer, but it does not include a CUDA-capable GPU. The Raspberry Pi AI Kit adds a Hailo-8L accelerator with 13 TOPS for supported workloads [15, 16]. The Jetson Nano provides CUDA-capable GPU inference at low cost, but has limited memory and older software constraints. Jetson Orin Nano Super is much stronger, with NVIDIA reporting up to 67 TOPS after software updates, but it is beyond the low-cost legacy target used in this project [17].

Table 3.5: Edge-device comparison as of 2026-07-06.

Device	Compute profile	AI acceleration	Project interpretation
Raspberry Pi 5	Quad-core Arm Cortex-A76 up to 2.4 GHz; RAM variants up to 16 GB.	CPU only unless accelerator is added.	Good general SBC, but less direct for CUDA/PyTorch deployment.
Raspberry Pi 5 + AI Kit	Raspberry Pi 5 plus Hailo-8L module.	13 TOPS neural accelerator.	Efficient for supported models, but requires Hailo conversion workflow.
Jetson Nano 4GB	Quad-core Arm A57 and 128-core Maxwell GPU; 4 GB shared LPDDR4 memory.	CUDA GPU, 472 GFLOPS reported by NVIDIA.	Selected low-cost edge target; shared memory and JetPack version constrain model choice.
Jetson Orin Nano Super	6-core Arm CPU and Ampere GPU; 8 GB class platform.	Up to 67 TOPS reported by NVIDIA.	Strong future upgrade path for heavier models and higher FPS.

3.6 MLOps and Deployment Tools

We use a practical MLOps stack. ZenML structures training workflows into reproducible steps [18]. Optuna automates hyperparameter search through studies and trials [19]. MLflow records model lineage and supports model registration [20]. DVC supports data and pipeline versioning [21]. The Hugging Face Hub provides versioned repositories for model and dataset artifacts [22]. Docker provides containerized services [23], while Mosquitto implements an MQTT broker suitable for lightweight telemetry [24]. GitHub Releases provide a release-asset mechanism used by the edge updater [25]. ONNX and TensorRT are relevant for inference portability and NVIDIA GPU optimization [26, 27].

The value of this toolchain is not merely technical variety. It supports a real engineering path: data preparation is traceable, experiments are registered, models can be exported, deployment assets can be versioned, and edge devices can be updated without manually copying files.

Chapter 4

Proposed Methodology

4.1 Overview

The methodology is organized as a cascade from image acquisition to mass output. A camera captures an RGB frame from a fixed setup. The segmentation model identifies waste objects and generates masks. The depth model estimates a depth map. The geometry module combines masks, depth, calibration, and background information to estimate object dimensions and volume. The material mapping assigns effective density from the predicted class. Finally, the mass module estimates mass using either a learned residual model or a density fallback.

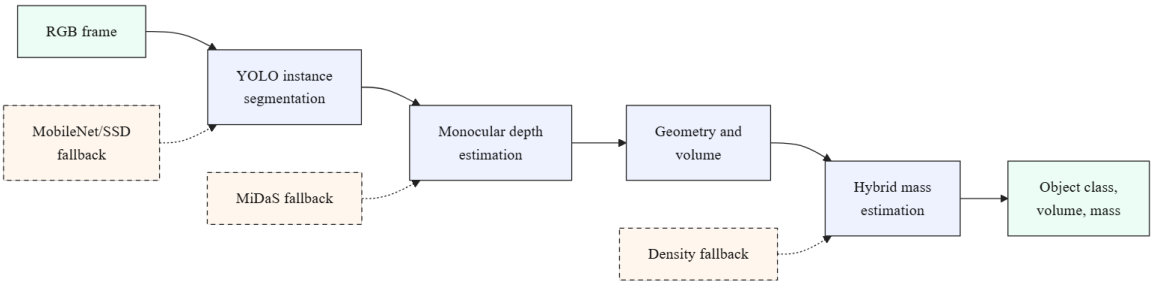


Figure 4.1: Proposed edge inference cascade. Primary modules are used when available, while fallback modules preserve operation under failure or resource constraints.

4.2 Engineering Rationale for the Main Technical Choices

The design choices were driven by the central constraint of the thesis: estimating waste-object mass from a cheap RGB camera on limited edge hardware. Table 4.1 summarizes the main decisions and the practical problem each one solves.

Table 4.1: Engineering rationale for the selected technologies and modeling choices.

Choice	Problem solved	Rationale
YOLO26m-seg	Need object-level masks with deployable inference.	A single-stage segmentation model offers a practical trade-off between mask quality, speed-oriented deployment, and export support. Mask R-CNN and SAM are useful references, but they are less suitable as the main Jetson Nano runtime model.
Segment Anything during annotation	Need masks for datasets that did not provide segmentation labels.	SAM accelerates mask generation when boxes or object regions are available; CVAT correction then restores human control over annotation quality.
Depth Anything V2 with MiDaS fallback	Need approximate 3D information without buying an RGB-D camera.	Monocular depth keeps the hardware low cost, while the fallback protects the pipeline when the primary model is too heavy or unavailable.
Density-volume baseline	Need an interpretable first physical estimate.	Multiplying estimated volume by effective material density gives a transparent baseline that can be inspected and corrected.
Residual mass model	Need to correct systematic physics errors.	Waste objects are hollow, crushed, and material-dependent; learning the residual lets the model improve the baseline without discarding physical structure.
Extra Trees residual regressor	Need robust tabular learning on a small feature dataset.	Extra Trees handles nonlinear feature interactions, mixed feature scales after preprocessing, and small datasets better than a high-capacity neural model in this first version.
Jetson Nano	Need low-cost offline edge deployment.	The device supports CUDA-based AI experimentation at low cost, but its memory limits force compact models, fallbacks, and export-oriented design.
ZenML, Optuna, MLflow, ONNX	Need reproducible engineering evidence.	These tools make the work traceable: the pipeline is organized, tuning is recorded, models are registered, and deployment artifacts are exportable.

4.3 Dataset Preparation for Segmentation

The segmentation dataset was built from heterogeneous sources. TACO already contains segmentation annotations, but AquaTrash provides bounding boxes, and RealWaste provides classification images. Therefore, the preparation process transformed classification and detection data into instance segmentation data. AquaTrash masks were manually annotated. RealWaste boxes were first generated using a detection model and then used as prompts for Segment Anything. SAM masks were corrected manually in CVAT. TrashNet was also annotated during the broader effort but was not retained in the final segmentation training set.

This workflow reflects a data-centric ML engineering process. The key challenge was not only choosing a model, but creating a dataset format that the model could learn from. The wider annotation effort covered 7,648 images: 369 AquaTrash images, 4,752 RealWaste images, and 2,527 TrashNet images. The final training manifest retained 6,460 images and 9,762 instances from TACO, AquaTrash, and RealWaste after filtering and conversion. The classes were organic waste, plastic bottle, plastic bag, rigid plastic, metal can, glass, paper/cardboard, and mixed waste. Before rendering the distribution figure, the class totals were recalculated from the train, validation, and test split dictionaries in the final manifest. The validated totals are 727 organic waste instances, 690 plastic bottle instances, 918 plastic bag instances, 1,581 rigid plastic instances, 1,189 metal can instances, 713 glass instances, 1,515 paper/cardboard instances, and 2,429 mixed waste instances; these values sum to the manifest total of 9,762 instances.

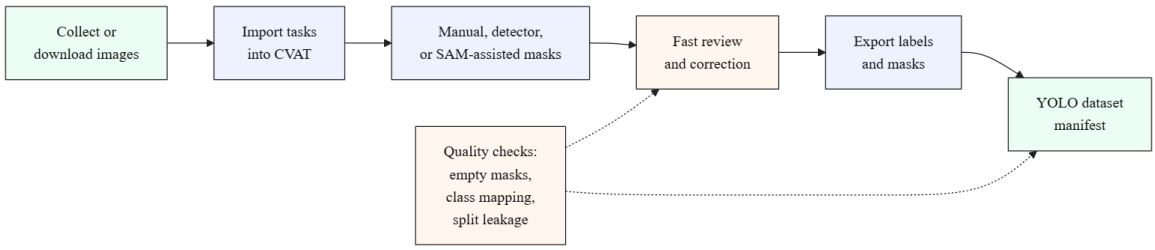
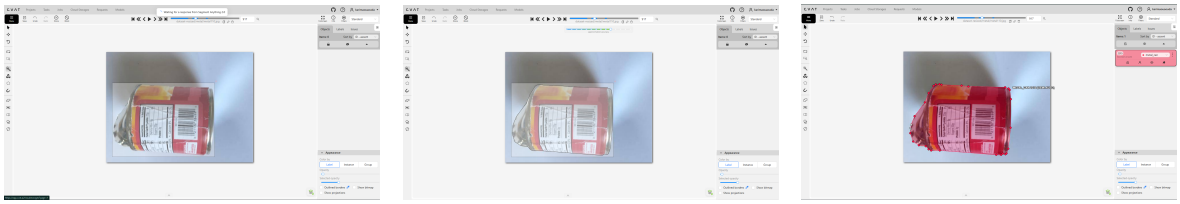
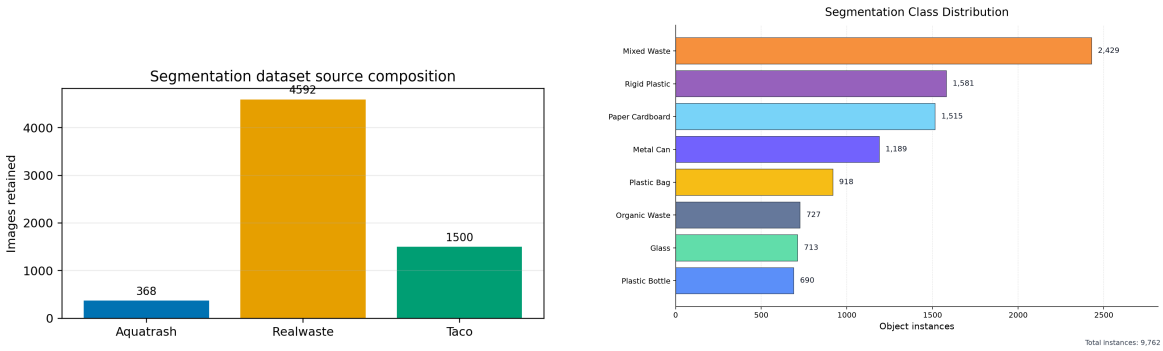


Figure 4.2: Annotation activity flow used to transform heterogeneous waste-image sources into a segmentation-ready dataset.



(a) CVAT task view. **(b)** Mask correction. **(c)** Review context.

Figure 4.3: Project annotation screenshots placed with the dataset-preparation method they support.



(a) Source composition.

(b) Class distribution.

Figure 4.4: Segmentation dataset composition generated from the final dataset manifest.

Table 4.2: Final segmentation dataset manifest.

Source	Declared images	Retained images	Instances	Notes
AquaTrash	368	368	450	Detection data converted to masks.
RealWaste	4,752	4,592	4,623	Classification data converted and filtered.
TACO	1,500	1,500	4,689	Native segmentation source.
Total	6,620	6,460	9,762	Final YOLO segmentation manifest.

4.4 Segmentation Training Pipeline

The segmentation training pipeline follows a governed sequence documented in the repository. The data are converted to YOLO segmentation format, validated, split, fingerprinted, and used for hyperparameter tuning. Optuna searches training hyperparameters. The selected configuration is then trained with an Ultralytics YOLO26m-seg checkpoint pretrained on general visual data. The final model is evaluated on validation and test splits, exported to ONNX, and registered through MLflow.

The final dataset split contained 4,522 training images, 969 validation images, and 969 test images. The training configuration used image size 640 and batch size 32 in the recorded Optuna best trial. The best Optuna objective was the mask mAP@50:95 metric, with 10 recorded trials, of which 5 completed, 3 failed, and 2 were pruned. This reflects a realistic tuning process under limited compute resources rather than an idealized exhaustive search.

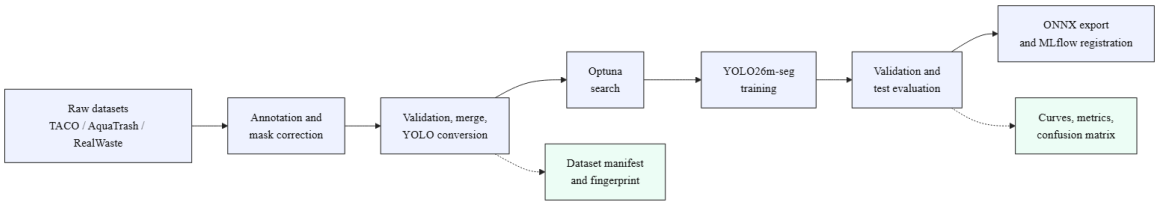
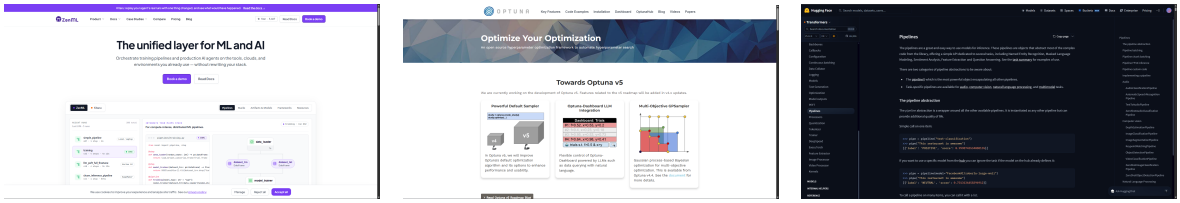


Figure 4.5: MLOps-oriented segmentation training and publication pipeline.



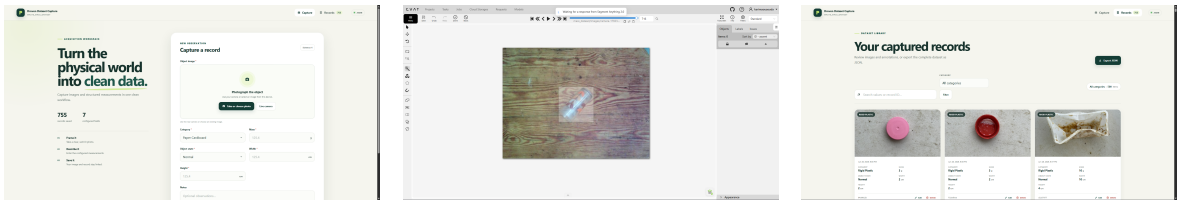
(a) ZenML pipeline tool. **(b)** Optuna tuning tool. **(c)** Artifact publication context.

Figure 4.6: Screenshots of tools used to organize, tune, and publish the segmentation training pipeline.

4.5 Mass Dataset Preparation

The mass-estimation component requires object-level features and mass targets. Public waste datasets do not generally include object mass or measured dimensions. We therefore designed a custom dataset protocol in which each row represents an object instance with class, material, mask geometry, volume-related features, density priors, depth features, and a real or pseudo mass target. A FastAPI-based tool was developed to support acquisition and annotation of mass and dimension information. Due to project time constraints, the first version of the mass dataset uses a pseudo-labeled approach derived from RealWaste images and estimated mass metadata. This is treated as a limitation, not as equivalent to scale-measured physical mass.

The processed mass feature table contains 745 object rows and 52 features. The train/validation/test split contains 521, 112, and 112 rows respectively. Splitting was group-aware by object identifier to prevent overlap between train and evaluation samples.



(a) Acquisition form. **(b)** Annotation screen. **(c)** Recorded rows.

Figure 4.7: Mass acquisition and annotation tool screenshots supporting the object-level mass-feature protocol.

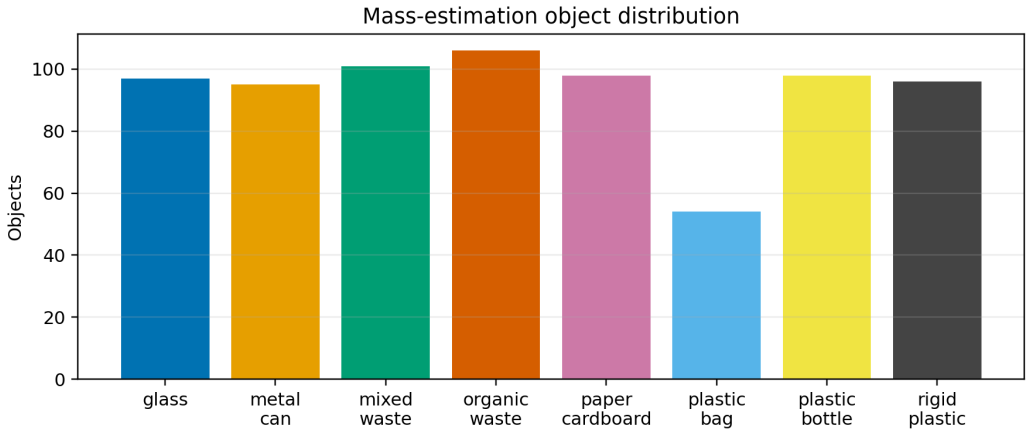


Figure 4.8: Mass-estimation class distribution generated from the processed feature summary.

4.6 Feature Engineering for Mass Estimation

The mass feature set includes categorical features, geometry features, depth features, and physics-derived baseline features. The repository explicitly excludes leakage columns such as the target mass, residuals, predictions, and derived error values from the training feature columns. This is important because the hybrid residual model must learn from observable or deployment-computable inputs rather than from target-derived quantities.

The baseline mass is computed from selected volume and effective density:

$$m_{base,g} = V_{cm^3} \rho_{g/cm^3}. \tag{4.1}$$

The residual target is:

$$correction_g = real_mass_g - mass_base_g. \tag{4.2}$$

The selected model predicts this correction and adds it back to the base estimate. This decomposition allows the model to preserve a physically meaningful baseline while learning systematic deviations caused by hollow objects, material variability, and geometric approximation errors.

4.7 Deployment-Oriented Methodology

Deployment is treated as part of the methodology rather than an afterthought. The runtime uses a YAML-configured cascade. Each stage has a primary module, optional fallback, and latency budget. The edge agent handles telemetry, inference requests, preview events, model-deployment commands, firmware-update commands, and offline outbox storage. GitHub

Release manifests can install model, configuration, and program assets with checksum verification, backup, rollback, and health checks. The dashboard layer is designed around Dockerized services, PostgreSQL relational storage, and Mosquitto MQTT messaging. PostgreSQL is suitable for this system because device telemetry, inference sessions, media metadata, and result review all require structured records, joins, filtering, and indexed queries rather than simple file storage.

The current repository contains the runtime and orchestration foundations. However, we distinguish deployment readiness from completed hardware benchmarking. We have ONNX export and edge-agent architecture, but the repository evidence does not yet contain a final Jetson Nano FPS, latency, thermal, or TensorRT benchmark report. Therefore, hardware performance is presented as future validation rather than as a measured result.

Chapter 5

System Design and Implementation

5.1 Repository Structure

The repository is organized into training, inference, mass estimation, orchestration, configuration, documentation, and artifact folders. The main runtime package is located under `src/edge_ai_mass/`. Segmentation training logic is located under `src/edge_ai_mass/training/`. Mass-estimation pipelines are under `src/edge_ai_mass/mass_estimation/`. Runtime inference modules are under `src/edge_ai_mass/modules/`, and the cascade is assembled under `src/edge_ai_mass/pipeline/`. Edge-agent and deployment mechanisms are under `src/edge_ai_mass/agent/` and `src/edge_ai_mass/orchestration/`.

Table 5.1: Main repository modules used by the thesis.

Path	Role
`src/edge_ai_mass/training/`	YOLO dataset preprocessing, training pipeline, tracking, publication, and evaluation support.
`src/edge_ai_mass/mass_estimation/`	Mass preprocessing, feature building, model training, evaluation, and residual model wrapper.
`src/edge_ai_mass/pipeline/`	Runtime cascade construction and stage orchestration.
`src/edge_ai_mass/modules/`	Detection, depth, geometry, material, and mass-estimation modules.
`src/edge_ai_mass/agent/`	Backend-facing device agent, telemetry, MQTT, inference runner, preview, and model deployment handling.
`src/edge_ai_mass/orchestration/`	Release-driven updater, manifest handling, backups, rollbacks, and install rules.
`configs/`	Pipeline, training, mass-estimation, orchestration, and agent configuration files.

5.2 System Architecture Overview

The implemented system is organized around three interacting environments: the machine-learning engineering environment where datasets and models are prepared, the backend host where services and dashboard functions are coordinated, and the edge device where camera frames are converted into object-level mass estimates. Figure 5.1 shows the main entities and data flows.

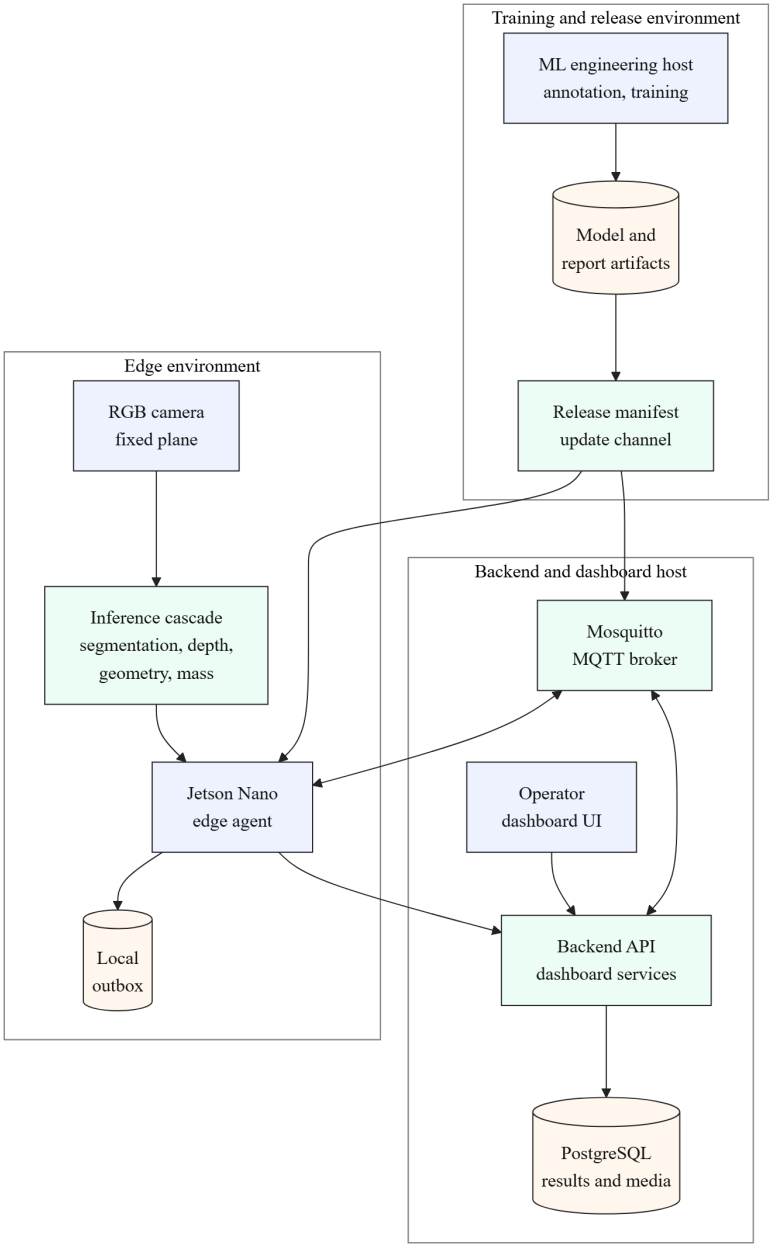


Figure 5.1: Full system architecture with the main interactive components and data flows. RGB frames enter the edge cascade, MQTT carries telemetry and commands, HTTP carries results and media to the backend, and the release channel provides model and configuration updates.

The operational use cases are shared between the ML engineer, the dashboard operator, the backend administrator, and the edge device. Figure 5.2 summarizes the main use cases that are implemented or directly supported by the repository and supporting tools.

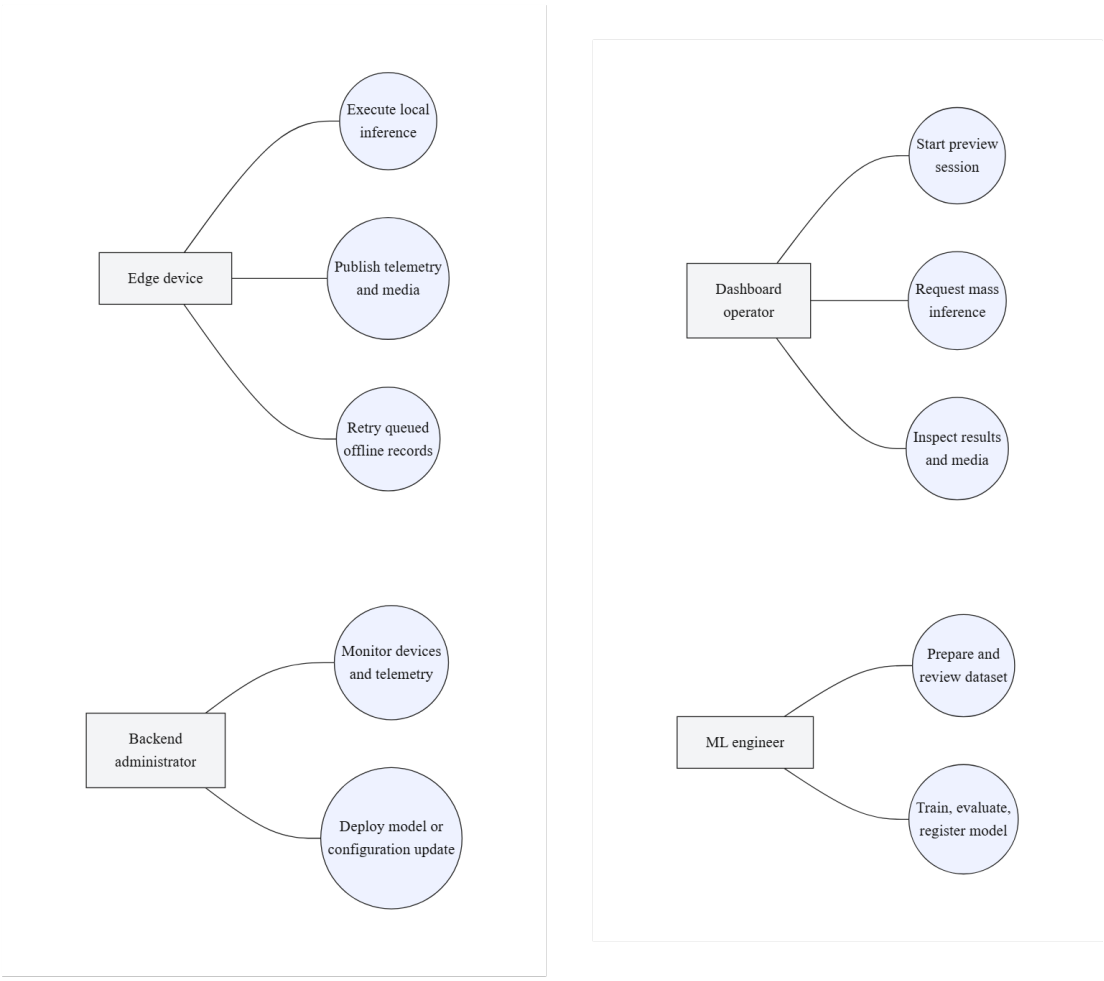


Figure 5.2: Use-case diagram for the main system actors and operational functions, organized by actor role. Each line represents a distinct use case supported by the actor rather than a sequential pipeline.

5.3 Inference Pipeline Implementation

The inference pipeline is implemented as a sequence of stages. Each stage wraps a primary model and an optional fallback model. The detection stage returns object detections with bounding boxes, class labels, confidence values, and masks. The depth stage returns a depth map. The geometry estimator converts raw depth to metric depth, extracts object depth statistics, estimates physical area and height, and integrates volume over the mask. The mass stage receives a feature dictionary and returns mass, material, volume, and warnings.

The implementation is designed to keep stage outputs explicit. Each object estimate stores detection data, depth statistics, geometry, volume, material, mass, method names, calibration identifiers, background identifiers, and warnings. This is useful for debugging because a poor mass estimate can originate from detection, depth, calibration, geometry, density, or regression.

5.4 Geometry Module

The geometry module estimates metric geometry from masks, depth, and background information. It supports three operating modes. With a camera calibration object, it uses calibrated pixel area and depth-dependent physical dimensions. With a background depth map, it estimates object height from the difference between background and object depth. Without a background depth file, it can use a configured constant background distance and default thickness. This hierarchy is suitable for a fixed acquisition plane, where a saved empty-plane depth map can later improve volume estimates.

The geometry module also records warnings such as missing background depth, invalid object depth, empty mask, or default-thickness fallback. These warnings are important for deployment because they allow the dashboard or backend to identify low-confidence predictions.

5.5 Mass-Estimation Implementation

The runtime contains two mass-estimation mechanisms. The density estimator is deterministic and computes mass from volume and effective density. The regression estimator is a neural MLP head in the runtime path, while the trained mass-model artifact in the mass-model pipeline is a residual tabular model. The artifact selected in the completed mass pipeline is an Extra Trees residual model saved as a joblib object. The thesis reports the completed artifact metrics rather than claiming that the MLP runtime checkpoint has the same performance.

This distinction is important. The repository shows that the mass-model pipeline has reached a completed artifact state, while the default runtime configuration still expects a checkpoint path for a regression estimator. Integrating the saved residual joblib artifact into the final runtime module is a natural next engineering step.

5.6 Edge Agent and Backend Communication

The edge device agent coordinates backend HTTP, MQTT, telemetry, inference, media output, preview handling, firmware update commands, and model deployment commands. It starts by preloading the inference runtime when the device capability enables inference. It publishes telemetry with status, active models, and device state. When an inference request is received, it creates stage reports, runs the pipeline, renders requested media, uploads or queues media, and publishes the final inference result.

Offline reliability is addressed through a file outbox. If telemetry, MQTT events, or media uploads fail, records are stored locally and retried later. This design is suitable for

edge deployments where network connectivity may be intermittent.

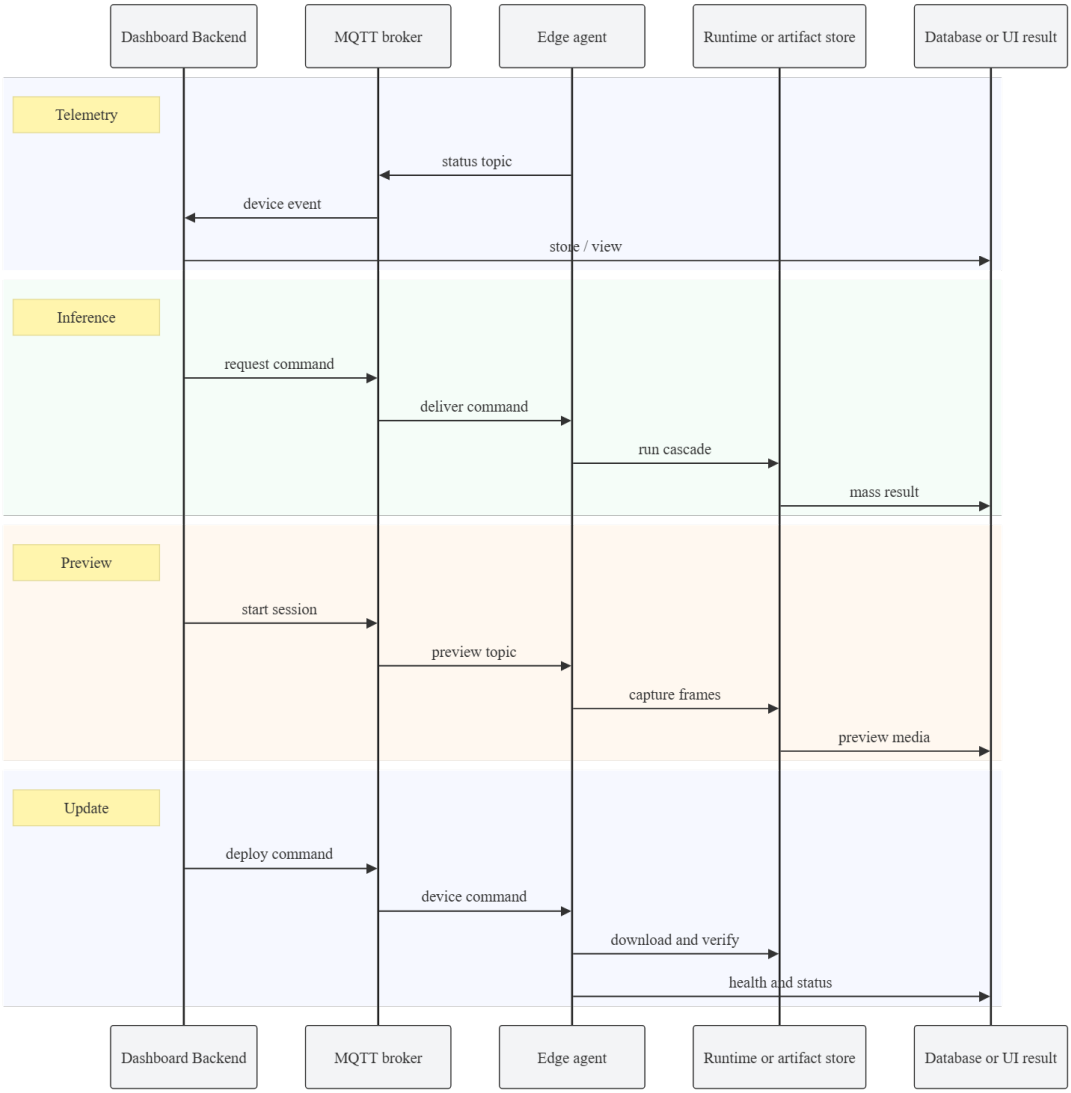
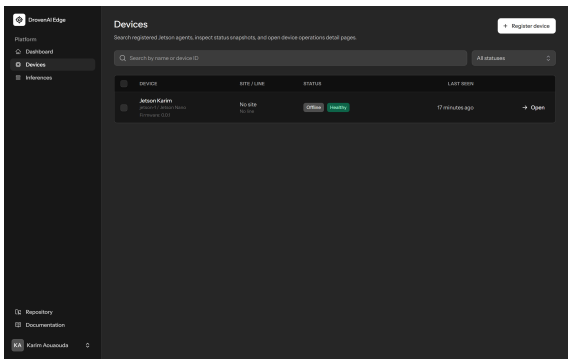
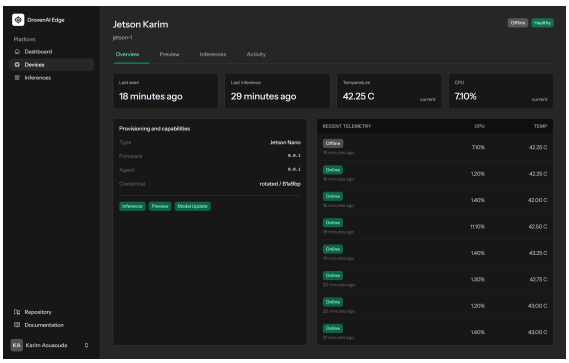


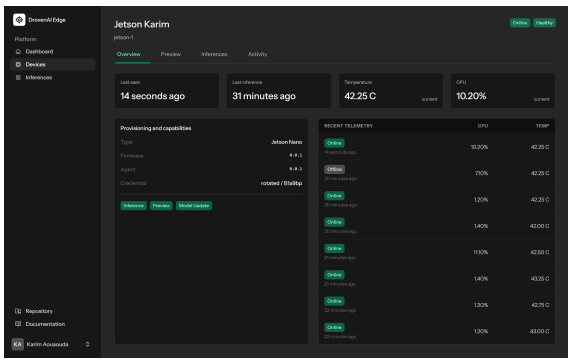
Figure 5.3: Sequence-style operation map for four important edge/backend interactions: telemetry, inference, preview, and release-based update.



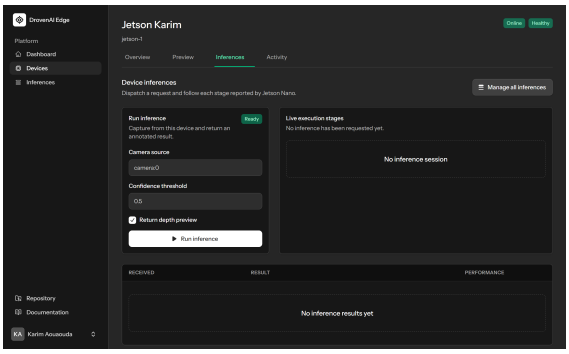
(a) Registered devices list.



(b) Offline device overview.

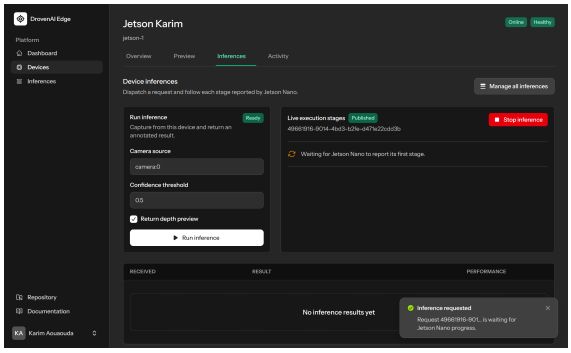


(c) Online telemetry overview.

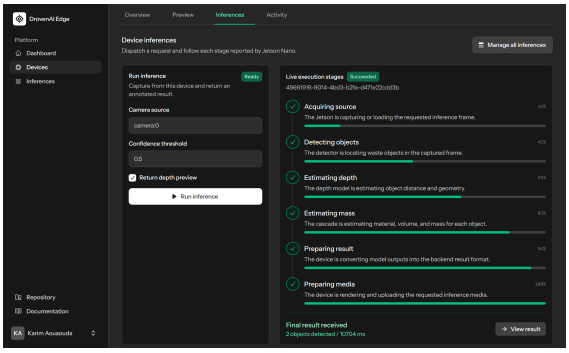


(d) Inference request form.

Figure 5.4: Dashboard screenshots showing the device list, device telemetry state, and inference-request entry point used by the implemented edge workflow.



(a) Requested inference session.



(b) Completed execution stages.

Figure 5.5: Dashboard inference execution screenshots showing the request lifecycle and the live stage sequence reported by the edge agent.

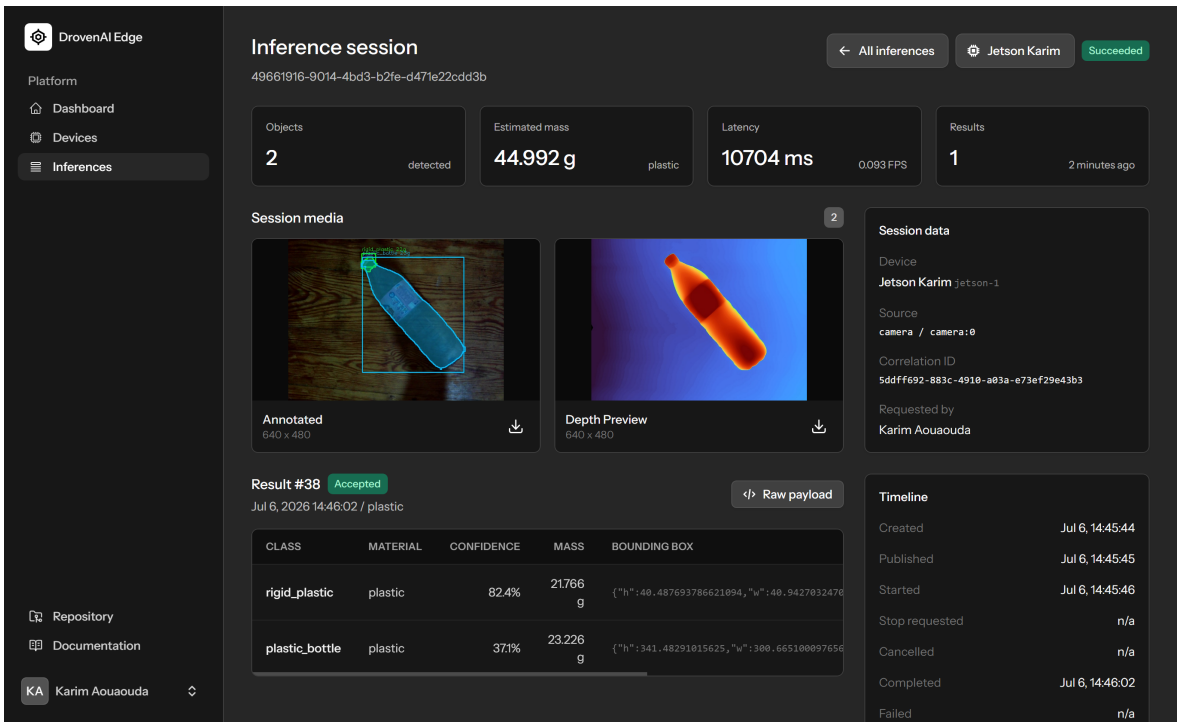


Figure 5.6: Dashboard inference-result screen showing the integrated output: annotated RGB segmentation overlay, returned depth preview, estimated total mass, per-object class labels, confidence values, and mass estimates. This screen demonstrates functional integration of the inference path, but it is not treated as a controlled Jetson Nano benchmark.

5.7 Release-Based Update System

The orchestration updater applies model, configuration, engine, and application assets from GitHub Releases or local install plans. It can download a manifest, match the configured release channel, verify SHA256 checksums, install files atomically, create backups, run health checks, restart the application, and roll back if necessary. This mechanism supports remote maintenance of deployed devices without manual file copying.

The update process is compatible with our deployment plan: trained YOLO weights or ONNX exports, mass-model artifacts, configuration files, and application wheels can be published as release assets. GitHub Releases are appropriate for this first version because they provide versioned downloadable artifacts and asset metadata [25].

5.8 Hardware and Software Environment

The target edge device is an NVIDIA Jetson Nano 4GB Developer Kit with JetPack 4.6.6 / L4T 32.7.6 as specified by the project context. The 128-core Maxwell GPU shares the 4 GB LPDDR4 memory with the rest of the embedded platform, which makes memory budgeting

important during model selection and inference packaging. The image-acquisition device used in the project setup is a low-cost 1080p webcam rather than an industrial camera. This choice matches the low-cost objective, but it can affect repeatability because exposure control, lens distortion, focus stability, mounting rigidity, and lighting robustness are weaker than in industrial vision hardware. The thesis also considers Raspberry Pi and Jetson Orin Nano alternatives in the literature review, but the implementation target remains the Jetson Nano. The broader software stack includes Python, Ultralytics, PyTorch, OpenCV, Depth Anything, MiDaS, scikit-learn style tabular modeling, ZenML, Optuna, MLflow, DVC, Docker, PostgreSQL, Mosquitto, and a Laravel dashboard.



Figure 5.7: Project Jetson Nano hardware used as the target low-cost edge device.

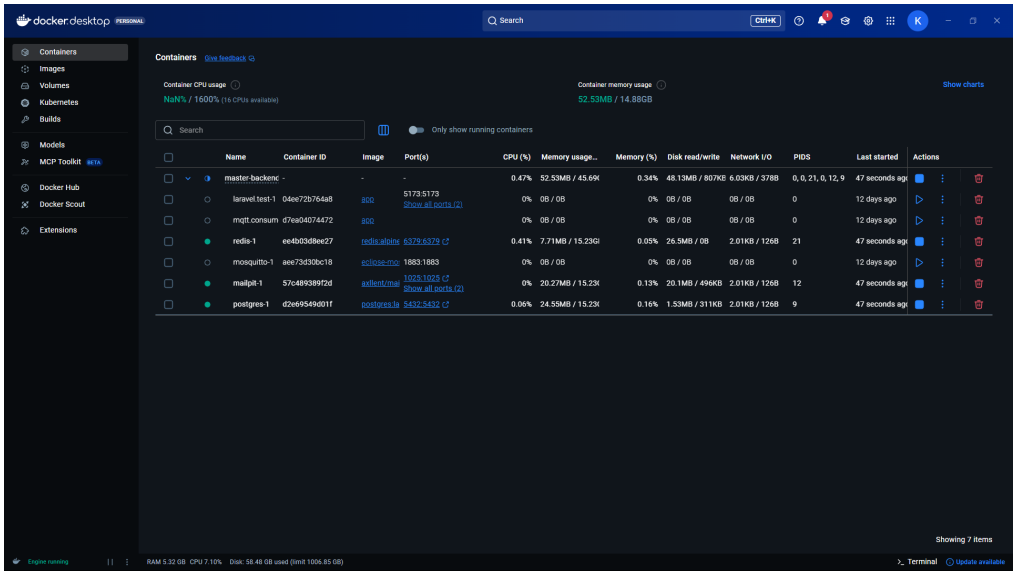


Figure 5.8: Dockerized service stack running the backend-supporting services used during system development and testing.

Chapter 6

Experiments, Results and Discussion

6.1 Segmentation Dataset Results

The final segmentation manifest contains 6,460 retained images and 9,762 object instances. The data were split into 4,522 training images, 969 validation images, and 969 test images. Table 6.1 summarizes the split-level evidence.

Table 6.1: Segmentation dataset split summary.

Split	Images	Instances	Share of images
Train	4,522	6,645	70.0%
Validation	969	1,526	15.0%
Test	969	1,591	15.0%
Total	6,460	9,762	100.0%

The dataset distribution is not perfectly balanced. Mixed waste, rigid plastic, paper/cardboard, and plastic bag have relatively high instance counts, while organic waste, glass, plastic bottle, and metal can are lower in some splits. This imbalance is important when interpreting class-level performance. Broad classes such as mixed waste are also intrinsically harder because they combine visually heterogeneous objects.

6.2 Segmentation Training and Evaluation

The segmentation model was trained from the YOLO26m-seg checkpoint with an Ultralytics segmentation task. The run is treated as a completed 60-epoch training run because the final evaluation artifacts correspond to the completed planned training output. Some checkpoint

metadata stores lower epoch-index fields due to checkpoint bookkeeping, but the reported validation and test metrics are the final evaluation results for the completed run. Evaluation was performed with batch size 1 and CPU fallback because the evaluation process recorded CUDA out-of-memory fallback. This means the reported accuracy metrics are valid for model quality, but the evaluation artifact is not a Jetson speed benchmark.

Table 6.2: YOLO segmentation evaluation metrics.

Split	Box P	Box R	Box mAP50	Box mAP50:95	Mask mAP50	Mask mAP50:95
Validation	0.8076	0.5647	0.6276	0.5804	0.6217	0.5453
Test	0.7322	0.5583	0.5853	0.5307	0.5765	0.5016

The validation mask mAP@50:95 of 0.5453 and test mask mAP@50:95 of 0.5016 indicate that the model learned a useful segmentation representation across the merged waste dataset. The result is significant because the model was not trained on one controlled visual distribution; it had to learn from landfill images, in-context litter, water-body waste examples, and masks generated through different annotation procedures. The gap between mAP@50 and mAP@50:95 shows that loose localization is easier than high-IoU mask quality. This is expected for irregular waste objects because their boundaries are often transparent, crushed, dirty, partially occluded, or visually similar to the background.

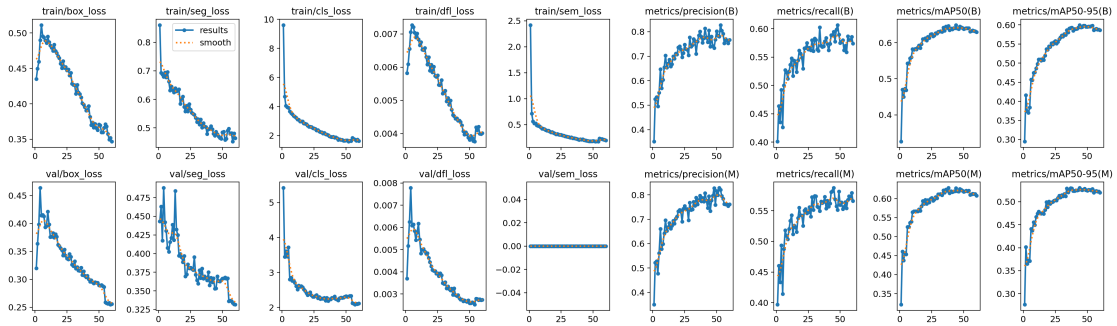


Figure 6.1: YOLO segmentation training curves from the recorded training artifacts.

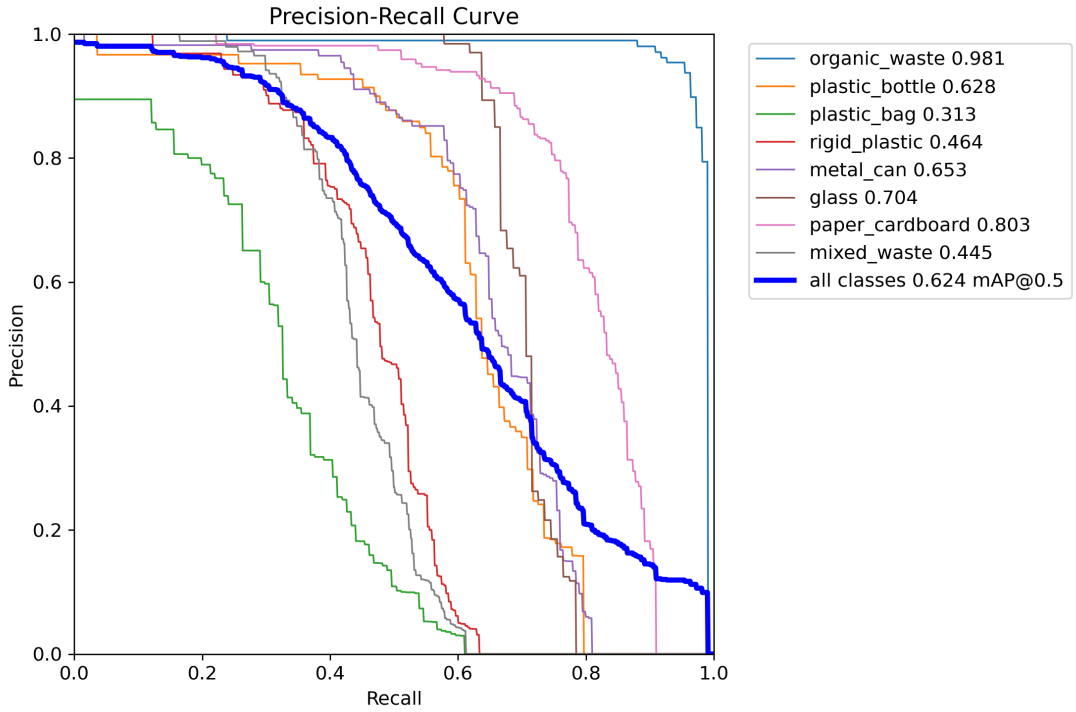


Figure 6.2: Mask precision-recall curve from the segmentation training artifacts.

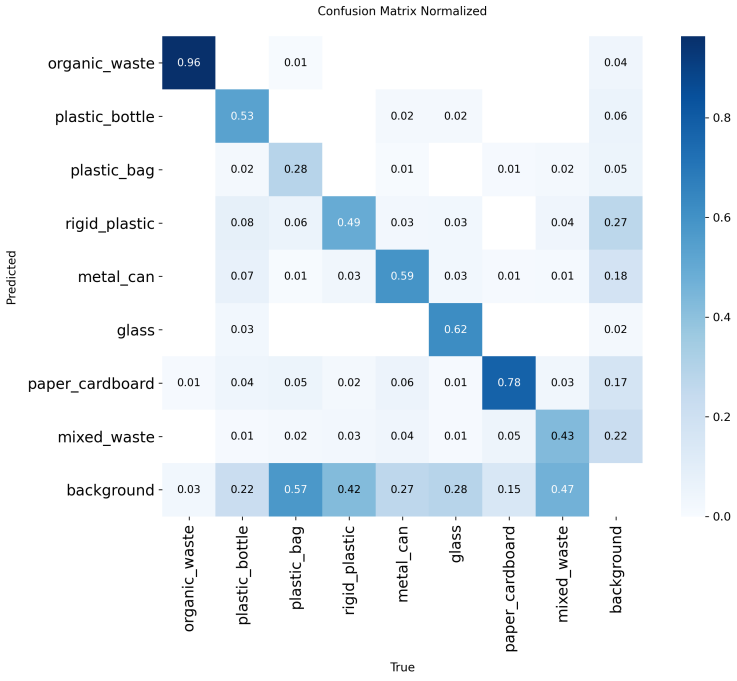


Figure 6.3: Normalized confusion matrix from the segmentation training artifacts.

The precision-recall curve and confusion matrix support a qualitative interpretation. Classes with distinctive shape or material appearance perform better, while broad or visually overlapping categories are more difficult. Mixed waste and plastic-related categories are expected to produce more confusion because the class boundaries are semantic rather than

purely visual. For the downstream mass objective, this matters because segmentation errors do not remain local: a missed mask removes an object from the mass estimate, an imprecise mask changes area and volume, and a class error changes the effective material density used by the baseline.

6.3 Segmentation Export and Registration

The model was exported to ONNX with opset 17, image size 640, batch size 1, dynamic shape disabled, and simplification enabled. The exported ONNX file size is approximately 94.4 MB, and the export manifest records a SHA256 digest. The registration report identifies an MLflow model named `edge-ai-mass-yolo-segmentation`, version 1, with alias `candidate`. These artifacts support deployment preparation and model lineage.

Table 6.3: Segmentation model deployment artifacts.

Artifact	Value
Best weights	`train_output/training_outputs/models/best.pt`
ONNX export	`train_output/training_outputs/exports/onnx/best.onnx`
ONNX size	94,394,598 bytes
ONNX options	Batch 1, image size 640, opset 17, simplify true, half false, dynamic false.
MLflow model	`edge-ai-mass-yolo-segmentation`, version 1, alias candidate.

6.4 Mass Dataset Results

The mass-estimation feature table contains 745 rows and 52 features. The real-mass target has a mean of 59.79 g, median of 25 g, and maximum of 700 g. This distribution is skewed, with many light objects and a smaller number of heavy objects. The train, validation, and test sets contain 521, 112, and 112 rows respectively.

Table 6.4: Mass dataset split summary.

Split	Rows	Share
Train	521	70.0%
Validation	112	15.0%
Test	112	15.0%
Total	745	100.0%

Outlier handling was enabled with class-wise IQR flagging and downweighting. The preprocessing report records 61 flagged rows and an outlier weight of 0.35. This choice is appropriate for a first mass model because the pseudo-labeled mass target can contain unusual or inconsistent values, and the model should not be dominated by extreme examples.

6.5 Mass Model Selection

The selected model is an Extra Trees residual regressor. This choice is appropriate for the first mass model because the dataset is tabular, relatively small, nonlinear, and composed of mixed numerical and categorical-derived features. A deep neural network would add capacity without enough measured mass data to justify it, while a purely linear model would struggle with interactions between object class, volume, density, mask geometry, and depth reliability. Extra Trees provides a robust ensemble compromise: it can learn nonlinear corrections, reduce variance through many randomized trees, and remain lightweight for deployment packaging. The model was chosen by validation hybrid MAE. Its training summary records 400 estimators, maximum depth 10, and minimum samples per leaf 2. The hybrid formula used in the artifact is:

$$mass_pred_g = mass_base_g + model(features). \tag{6.1}$$

The selected model therefore learns a correction rather than a direct mass from scratch. This decision is central to the method. The physics baseline gives the model a first estimate grounded in volume and material density, while the residual learner focuses on the systematic errors that the baseline cannot represent, such as hollow containers, crushed objects, class-dependent effective density, and unreliable depth-derived volume.

6.6 Mass Evaluation

The hybrid residual model substantially improves the physics-only baseline on absolute error metrics. On the test set, MAE decreases from 44.31 g to 23.18 g and RMSE decreases from

88.69 g to 50.23 g. The median absolute error decreases from 15.67 g to 9.18 g. The R^2 score improves from -0.0281 to 0.6702. sMAPE decreases from 85.04% to 49.04%. MAPE increases because very small masses make percentage errors unstable, so sMAPE and gram-level errors are emphasized.

The improvement occurs because the baseline is physically meaningful but too rigid. A density-volume estimate assumes that the visible volume and effective density are already reliable. In practice, waste objects violate this assumption. Bottles and cans are hollow, plastic bags have large visible area but very small mass, glass objects can be overestimated by density priors, and organic waste varies strongly by moisture and compaction. The residual model learns these repeated deviations from the feature table. It does not replace physics; it learns when and how the physics approximation should be corrected.

Table 6.5: Mass-estimation metrics on validation and test sets.

Split	Method	MAE (g)	RMSE (g)	MedAE (g)	R^2	MAPE (%)	sMAPE (%)
Validation	Physics baseline	63.97	133.17	18.40	-0.035	108.46	89.52
Validation	Hybrid residual	32.95	59.26	11.71	0.795	150.12	59.78
Test	Physics baseline	44.31	88.69	15.67	-0.028	78.08	85.04
Test	Hybrid residual	23.18	50.23	9.18	0.670	83.01	49.04

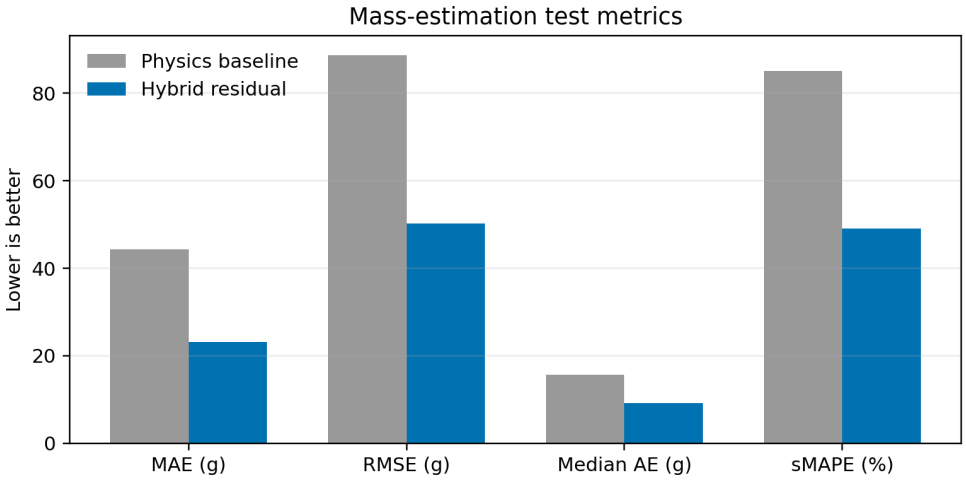
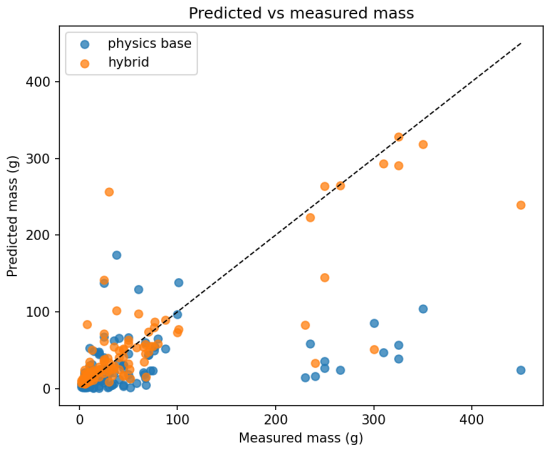
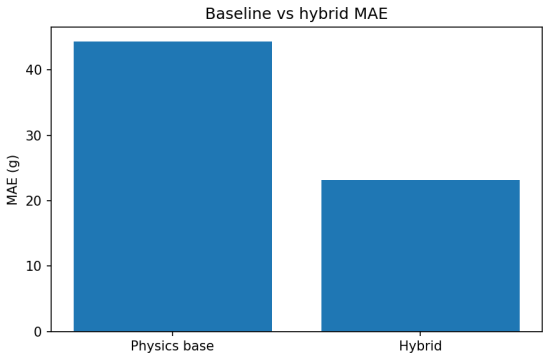


Figure 6.4: Mass-estimation test metric comparison generated from the test metrics artifact.

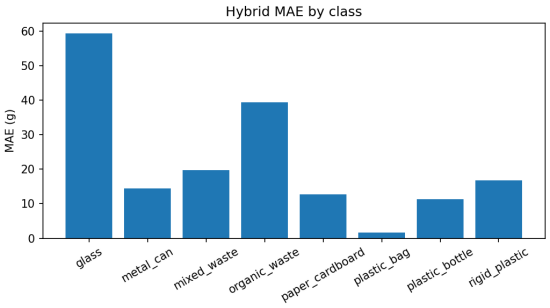


(a) Predicted versus real mass.

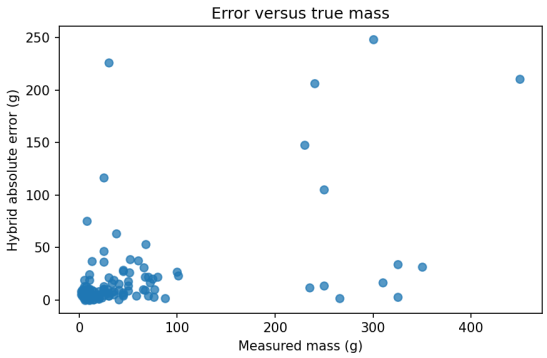


(b) Baseline versus hybrid errors.

Figure 6.5: Mass-estimation diagnostic plots from the test evaluation artifacts.



(a) Per-class MAE.



(b) Error versus true mass.

Figure 6.6: Per-class and error-distribution diagnostics for the mass model.

Table 6.6: Test MAE by class for the physics baseline and hybrid residual model.

Class	Baseline MAE (g)	Hybrid MAE (g)	Relative MAE reduction
Glass	150.20	59.33	60.5%
Metal can	31.10	14.38	53.8%
Mixed waste	26.06	19.71	24.4%
Organic waste	39.59	39.30	0.7%
Paper/cardboard	20.15	12.65	37.2%
Plastic bag	2.28	1.71	25.1%
Plastic bottle	23.40	11.36	51.5%
Rigid plastic	22.12	16.71	24.5%

The strongest improvements occur for glass, metal can, and plastic bottle. These

classes have recognizable object forms and systematic differences between apparent volume and effective mass. A glass container, for example, can be assigned a high baseline mass when the density prior dominates, while the residual model can learn from examples that visible container volume does not equal solid material volume. Metal cans and plastic bottles benefit for a similar reason: hollow container geometry creates predictable baseline error patterns. Organic waste shows almost no MAE improvement in the test set, which suggests that the baseline density-volume approximation and the available features do not capture its variability well. Organic objects can vary by moisture, compaction, and irregular shape, so this class should be prioritized for better measured data in the next version.

6.7 End-to-End Status

The segmentation model and mass model have completed first-version training and evaluation artifacts. The runtime pipeline, geometry estimator, density fallback, edge agent, MQTT telemetry, dashboard inference workflow, and release updater are implemented in the repository. The project has an ONNX segmentation export and a deployable architecture. The dashboard result screenshot in Figure 5.6 demonstrates a functional inference path that returns an annotated segmentation overlay, depth preview, and mass table. However, a full controlled end-to-end benchmark on the Jetson Nano is not yet present in the evidence artifacts. Therefore, the thesis reports model-level results and deployment design, while reserving final edge FPS, latency, memory, and TensorRT performance for future hardware validation.

6.8 Interpretation of Segmentation Results

The segmentation model achieves a useful first-version performance level on a difficult merged waste dataset. A test mask mAP@50:95 of 0.5016 is meaningful because the dataset combines different sources, image conditions, annotation origins, and waste appearances. The model is not only learning from one clean background distribution. It must generalize across in-context litter, water-related waste examples, and landfill-style material images transformed into segmentation annotations.

The lower recall compared with precision suggests that some objects remain undetected or under-segmented. For mass estimation, missed objects are more damaging than minor class confusion because a missed object contributes no estimated mass. Future training should therefore consider class-balanced sampling, targeted augmentation for weak classes, additional real images from the target camera setup, and lower-confidence post-processing strategies that preserve candidates for downstream filtering.

6.9 Interpretation of Mass Results

The mass model results support the hybrid residual strategy. A pure physics baseline is interpretable but too simple for irregular waste. The residual model learns systematic corrections from class, material, geometry, depth, and baseline features. The large improvement in MAE and RMSE indicates that the learned correction captures useful structure.

At the same time, the mass results must be interpreted conservatively. The first mass dataset includes pseudo-estimated labels rather than a fully scale-measured dataset. Therefore, the model demonstrates pipeline feasibility and modeling behavior, but it does not yet establish final physical accuracy in the real world. A future dataset with controlled camera capture, measured dimensions, scale-measured mass, repeated examples, and material-state labels would provide stronger evidence.

6.10 Error Propagation

The pipeline is sequential, so errors can propagate. A segmentation error changes mask area and class. A class error changes material density. A depth error changes object height and volume. A volume error changes the physics baseline and therefore the residual model input. The final mass estimate is therefore a compound output:

$$\hat{m} = g_\theta(I, \hat{M}, \hat{c}, \hat{D}, \hat{V}, \hat{\rho}). \tag{6.2}$$

The practical implication is that mass evaluation should eventually be performed end-to-end using captured frames, not only by evaluating the tabular model on precomputed features. The current thesis separates component-level results because those are the traceable artifacts currently available.

6.11 Deployment Considerations

The Jetson Nano is suitable for a low-cost proof of concept but imposes strict memory and performance constraints because its integrated GPU shares the 4 GB LPDDR4 memory with the rest of the platform. The recorded evaluation already shows a CUDA out-of-memory fallback during one evaluation run, though this occurred in the evaluation environment rather than necessarily on the final Jetson deployment. The ONNX export is a necessary step for optimized inference, but TensorRT conversion and benchmarking remain future work. A final deployment should measure cold-start time, per-stage latency, FPS, memory usage, power draw, temperature, and stability over long runtime sessions. The 1080p webcam used for

acquisition is also a practical limitation: it supports the low-cost objective, but an industrial camera would provide more reliable optics, exposure behavior, mounting stability, and repeatable lighting response.

The primary/fallback architecture is appropriate for this environment. If the segmentation model fails to load, a lighter detector can still preserve limited functionality. If the depth model is too slow, MiDaS small can be used as a fallback. If learned mass inference is unavailable, density-based estimation can still provide an interpretable result with warnings.

6.12 Threats to Validity

Internal validity is affected by pseudo-labeled mass targets, missing full hardware benchmarks, and the gap between the tabular residual artifact and the default runtime MLP checkpoint configuration. Measurement validity is affected by the use of a low-cost 1080p webcam rather than an industrial camera, because image noise, lens distortion, focus, lighting, and camera mounting can influence segmentation, depth, and geometry estimates. External validity is affected by the use of public datasets whose distributions may differ from the target deployment environment. Construct validity is affected by the difficulty of defining effective density for hollow or crushed waste objects. Reproducibility is strengthened by recorded artifacts, manifests, checksums, and pipeline documentation, but final reproducibility would benefit from a complete tagged release containing all trained weights, mass artifacts, configs, and benchmark scripts.

Chapter 7

General Conclusion and Future Work

7.1 Conclusion

We presented a full first version of a low-cost Edge AI pipeline for waste object characterization and mass estimation. The project combined dataset engineering, instance segmentation, monocular depth, geometric volume approximation, residual mass modeling, MLOps practices, and edge deployment architecture. The work demonstrates that a practical pipeline can be built from available public datasets and project-specific annotations, while also showing that mass estimation requires more than classification or detection.

The segmentation dataset preparation produced a final manifest of 6,460 images and 9,762 instances, supported by a broader annotation effort of 7,648 images. The trained YOLO26m-seg model reached test mask mAP@50:95 of 0.5016 and was exported to ONNX. The mass-estimation pipeline produced a 745-row object-level dataset and selected an Extra Trees residual model that reduced test MAE from 44.31 g to 23.18 g compared with the density-volume baseline. The runtime implementation provides a staged cascade with fallback modules, geometry-aware volume estimation, telemetry, release-based updates, and dashboard-oriented communication.

The main scientific and engineering result is the demonstration that visual segmentation, monocular depth, material priors, and residual learning can be combined into a coherent mass-estimation pipeline for waste objects. The current version is suitable as a foundation for deeper physical calibration and real deployment validation.

7.2 Future Work

Future work should first build a fully measured mass dataset. Each object should be photographed under the target camera setup, annotated with an instance mask, weighed with a

scale, measured physically, and recorded with material state such as empty, full, crushed, wet, or dirty. This would replace pseudo labels with stronger ground truth and allow more reliable error analysis.

Second, camera calibration and background depth capture should be completed for the fixed acquisition plane. A saved empty-plane depth map, checkerboard calibration, and known plane dimensions would improve volume estimation. Third, the saved residual joblib model should be integrated into the runtime mass module so that deployed inference uses the same model family evaluated in the mass pipeline. Fourth, ONNX and TensorRT benchmarks should be executed on the Jetson Nano, including latency, FPS, memory, temperature, and power measurements. Fifth, more target-domain images should be collected from the actual deployment environment to reduce dataset shift.

The segmentation training strategy should also be extended into a staged robustness curriculum. A useful next version would pretrain on noisier and more variable sources such as AquaTrash and TACO, then fine-tune on more regular environment datasets such as RealWaste and TrashNet while applying lighting augmentation and noise augmentation. A final fine-tuning stage should use a custom dataset captured with the same camera, lighting, background, and object placement used in the experiment environment. This sequence would expose the model first to broad variation, then to cleaner object appearances, and finally to the target deployment distribution.

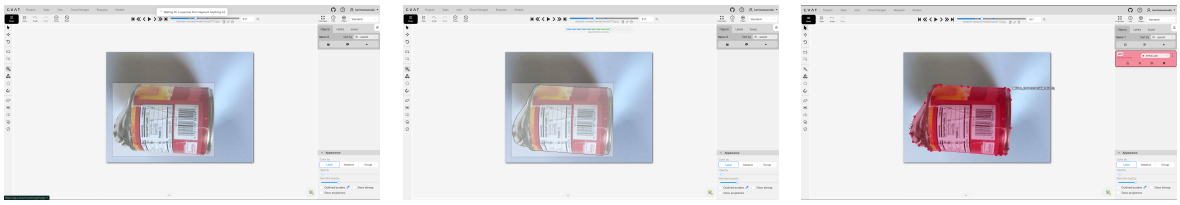
Future hardware experiments should evaluate an RGB-D or dedicated depth camera as an alternative acquisition setup. Direct depth sensing would reduce the scale ambiguity of monocular depth, improve metric volume estimation, and provide a stronger basis for mass prediction. This would increase hardware cost and change the low-cost assumption, but it would provide an important comparison point for judging how much accuracy is lost when using only a 1080p RGB webcam.

Finally, the dashboard can be extended from local Docker deployment toward cloud-backed operation using AWS, Azure, or Google Cloud when remote monitoring is required. The system should maintain offline-first operation at the edge, with cloud services used for fleet monitoring, model updates, and long-term analytics rather than mandatory real-time inference.

Appendix A
Annotation and Tool Evidence

A.1 Annotation Workflow Screenshots

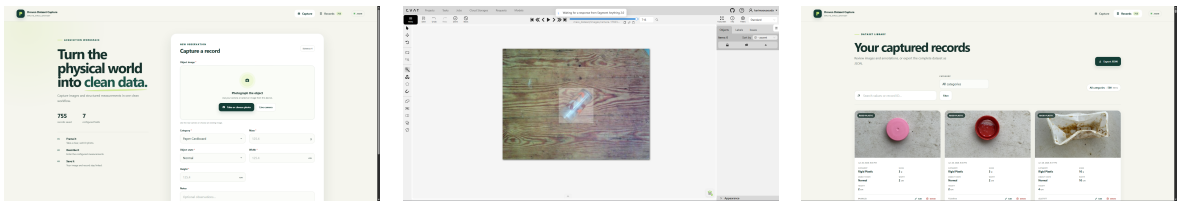
The following screenshots document the practical annotation workflow used to create and correct segmentation data. They are included as project evidence rather than as benchmark figures.



(a) Annotation view 1. **(b)** Annotation view 2. **(c)** Annotation view 3.

Figure A.1: CVAT-based annotation and correction screenshots from the project thesis-screenshot archive.

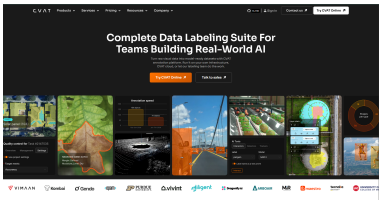
A.2 Mass Acquisition and Annotation Tool



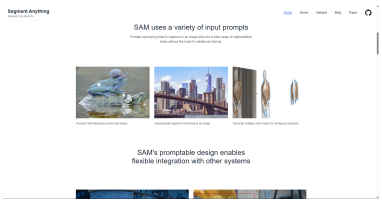
(a) Acquisition interface. **(b)** Mass annotation. **(c)** Records view.

Figure A.2: Screenshots of the mass acquisition and annotation workflow.

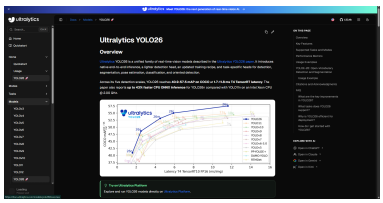
A.3 Tool and Model References



(a) CVAT.



(b) Segment Anything.

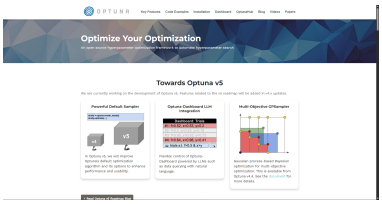


(c) YOLO26.

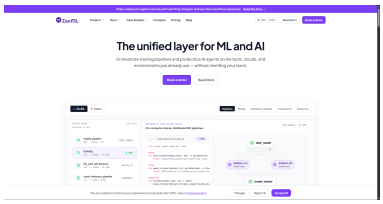
Figure A.3: Annotation and segmentation tools referenced by the project.



(a) Depth Anything.

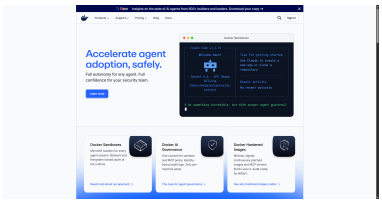


(b) Optuna.

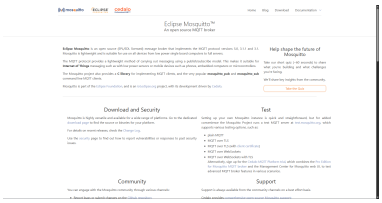


(c) ZenML.

Figure A.4: Depth and MLOps tools used in the project workflow.



(a) Docker.



(b) Mosquitto.



(c) Jetson Nano.

Figure A.5: Deployment-supporting tools and hardware documentation screenshots.

Appendix B
Artifact Traceability

Table B.1: Traceability table for the main thesis claims.

Claim	Artifact path
Final segmentation dataset contains 6,460 images and 9,762 instances.	`train_output/training_outputs/metadata/dataset-manifest.json`
Segmentation model evaluated on validation and test splits.	`train_output/training_outputs/reports/evaluation.json`
Training curves and PR/confusion figures are available.	`train_output/training_outputs/training/curves/`
ONNX export was produced.	`train_output/training_outputs/exports/exports_manifest.json`
Optuna tuning was executed.	`train_output/training_outputs/optimization/optuna_best.json`
MLflow registration exists.	`train_output/training_outputs/reports/registration.json`
Mass feature table contains 745 rows and 52 features.	`mass_model/data/processed/mass_estimation/feature_summary.json`
Mass split is 521/112/112 rows.	`mass_model/data/processed/mass_estimation/split_summary.json`
Best mass model is Extra Trees residual.	`mass_model/artifacts/mass_estimation/mass-model-feature-residuals-notebook/models/best_model_metadata.json`

Claim	Artifact path
Mass test metrics show 47.69% MAE improvement.	`mass_model/artifacts/mass_estimation/mass-model-feature-residuals-notebook/evaluation/test/metrics.json`
Runtime cascade exists.	`src/edge_ai_mass/pipeline/pipeline.py` and `src/edge_ai_mass/pipeline/factory.py`
Geometry-aware volume estimation exists.	`src/edge_ai_mass/modules/geometry/volume_estimator.py`
Edge agent and MQTT handling exist.	`src/edge_ai_mass/agent/runner.py` and related agent modules.
Release-based update mechanism exists.	`src/edge_ai_mass/orchestration/updater.py`

Bibliography

[1] Gary Thung and Mindy Yang. Trashnet: Dataset of images of trash. https://github.com/garythung/trashnet, 2016. Accessed 2026-07-02.

[2] Pedro F. Proenca and Pedro Simoes. Taco: Trash annotations in context for litter detection. *arXiv preprint arXiv:2003.06975*, 2020. URL https://arxiv.org/abs/2003.06975.

[3] Harsh Panwar, P. K. Gupta, Mohammad Khubeb Siddiqui, Ruben Morales-Menendez, Prakhar Bhardwaj, Sudhanshu Sharma, and Iqbal H. Sarker. Aquavision: Automating the detection of waste in water bodies using deep transfer learning. *Case Studies in Chemical and Environmental Engineering*, 2020. doi: 10.1016/j.cscee.2020.100026. URL https://www.sciencedirect.com/science/article/pii/S2666016420300244.

[4] Sam Single, Saeid Iranmanesh, and Raad Raad. Realwaste: A novel real-life data set for landfill waste classification using deep learning. *Information*, 14(12):633, 2023. doi: 10.3390/info14120633. URL https://www.mdpi.com/2078-2489/14/12/633.

[5] Rene Ranftl, Katrin Lasinger, David Hafner, Konrad Schindler, and Vladlen Koltun. Towards robust monocular depth estimation: Mixing datasets for zero-shot cross-dataset transfer. *arXiv preprint arXiv:1907.01341*, 2019. URL https://arxiv.org/abs/1907.01341.

[6] Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. *arXiv preprint arXiv:2401.10891*, 2024. URL https://arxiv.org/abs/2401.10891.

[7] Lihe Yang, Bingyi Kang, Zilong Huang, Zhen Zhao, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything v2. *arXiv preprint arXiv:2406.09414*, 2024. URL https://arxiv.org/abs/2406.09414.

[8] NVIDIA. Jetson nano brings the power of modern ai to edge devices. https://www.nvidia.com/en-us/autonomous-machines/

embedded-systems/jetson-nano/product-development/, 2026. Accessed 2026-07-06.

[9] Harsh Panwar. Aquatrash dataset. https://www.kaggle.com/datasets/harshpanwar/aquatrash, 2020. Accessed 2026-07-02.

[10] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *arXiv preprint arXiv:1505.04597*, 2015. URL https://arxiv.org/abs/1505.04597.

[11] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. *arXiv preprint arXiv:1802.02611*, 2018. URL https://arxiv.org/abs/1802.02611.

[12] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017. URL https://arxiv.org/abs/1703.06870.

[13] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollar, and Ross Girshick. Segment anything. *arXiv preprint arXiv:2304.02643*, 2023. URL https://arxiv.org/abs/2304.02643.

[14] Ultralytics. Instance segmentation with ultralytics yolo. https://docs.ultralytics.com/tasks/segment/, 2026. Accessed 2026-07-02.

[15] Raspberry Pi Ltd. Raspberry pi 5. https://www.raspberrypi.com/products/raspberry-pi-5/, 2026. Accessed 2026-07-02.

[16] Raspberry Pi Ltd. Raspberry pi ai kit documentation. https://www.raspberrypi.com/documentation/accessories/ai-kit.html, 2026. Accessed 2026-07-02.

[17] NVIDIA. Jetson orin nano super developer kit. https://www.nvidia.com/en-us/autonomous-machines/embedded-systems/jetson-orin/nano-super-developer-kit/, 2026. Accessed 2026-07-02.

[18] ZenML GmbH. Zenml documentation. https://docs.zenml.io/, 2026. Accessed 2026-07-02.

[19] Optuna Project. Optuna documentation. https://optuna.readthedocs.io/, 2026. Accessed 2026-07-02.

[20] MLflow Project. Mlflow model registry. https://mlflow.org/docs/latest/ml/model-registry/, 2026. Accessed 2026-07-02.

[21] DVC Project. Data version control user guide. https://doc.dvc.org/user-guide, 2026. Accessed 2026-07-02.

[22] Hugging Face. Hugging face hub documentation. https://huggingface.co/docs/hub/en/index, 2026. Accessed 2026-07-02.

[23] Docker Inc. Docker overview. https://docs.docker.com/get-started/docker-overview/, 2026. Accessed 2026-07-02.

[24] Eclipse Foundation. Eclipse mosquitto documentation. https://mosquitto.org/documentation/, 2026. Accessed 2026-07-02.

[25] GitHub. About releases. https://docs.github.com/en/repositories/releasing-projects-on-github/about-releases, 2026. Accessed 2026-07-02.

[26] ONNX Project. Open neural network exchange. https://onnx.ai/, 2026. Accessed 2026-07-02.

[27] NVIDIA. Nvidia tensorrt documentation. https://docs.nvidia.com/deeplearning/tensorrt/latest/index.html, 2026. Accessed 2026-07-02.