FLOWT Pipeline – Floating Litter Observation & Waste Tracking

Fereshteh Nayyeri

Abstract

Plastic and floating debris pose significant risks to aquatic ecosystems, marine life, and human health. Recent advances in AI and computer vision enable efficient detection, monitoring, and quantification of floating waste, generating reliable data to support clean-up efforts, policymaking, and long-term prevention. This work presents FLOWT, a modular computer vision pipeline for Floating Litter Observation & Waste Tracking. FLOWT supports video ingestion, object detection, visualization, and tracking of individual debris items across frames. A human-in-the-loop review stage ensures validated, high-quality data by allowing users to correct misclassifications before downstream use. FLOWT operates via two independent workflows: the Waste Tracking workflow, which focuses on operational monitoring and produces annotated outputs and analytical summaries without requiring machine learning expertise; and the Model Improvement workflow, which leverages validated annotations to continuously retrain and fine-tune detection models while maintaining ongoing waste-tracking operations. Additionally, FLOWT integrates AI Insight, using Large Language Models (LLMs) to transform detection outputs into actionable analyses. AI Insight interprets spatial and temporal patterns, flags anomalies, highlights recurring errors, and guides both operational decisions and model refinement. By combining these workflows and AI-driven insights, FLOWT accelerates review cycles, enhances transparency, and enables smarter monitoring and continuous improvement of floating debris detection.

1. Pipeline Objectives

FLOWT is built around two independent workflows that support both operational monitoring and continuous model development. These two independent workflows, illustrated in Figure 1, include waste tracking and model fine-tuning. The ***Waste Tracking workflow*** is designed for environmental practitioners and researchers who need to detect debris, review and correct outputs, generate annotated videos, and produce analytical summaries—without requiring any machine learning expertise. The ***Model Improvement workflow*** leverages the validated annotations produced during review to retrain or fine-tune detection models. Because the workflows operate independently, waste-tracking activities continue uninterrupted even as models are improved in parallel.

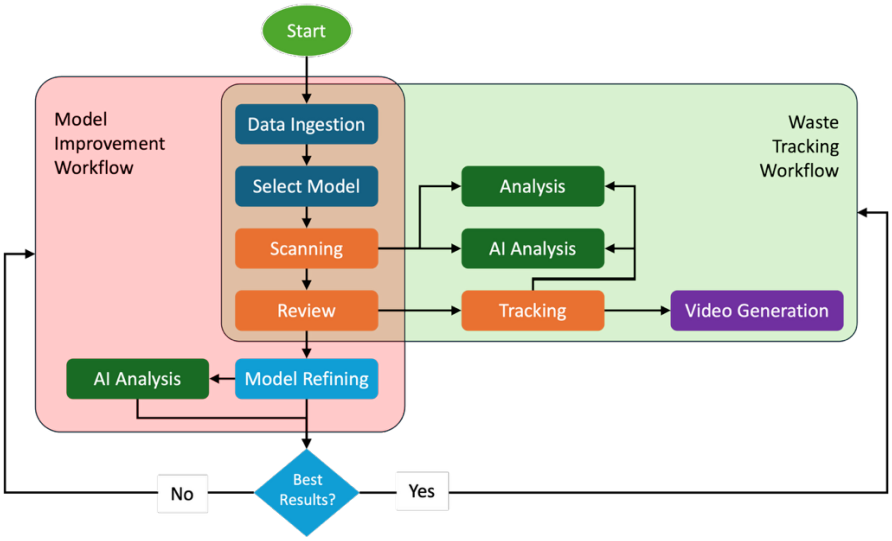

Figure1. FLOWT Pipeline including two independent workflows

FLOWT is built for scalability, maintainability, and, above all, accessibility to non-ML technical users. With a consistent and intuitive UI/UX, automated configuration handling, and guided workflows, it enables users with limited technical backgrounds to inspect detections, validate outputs, fine-tune models, and conduct analyses with confidence. By combining automated AI capabilities with essential human oversight, FLOWT offers a practical and reliable solution for environmental monitoring and marine debris management.

2.1 Waste Tracking Workflow

The *waste tracking workflow* manages the full process of floating litter observation, from data ingestion and automated scanning to human-in-the-loop review, object tracking, analysis, and annotated video generation. During tracking, FLOWT adapts to different video types: in standard videos with smooth frame-to-frame motion, **IoU-based tracking** is used to reliably associate objects across frames; however, in timelapse footage, where large temporal gaps cause objects to shift significantly between frames and result in zero-intersection IoU values, the system switches to **template matching**, enabling robust re-identification of debris items despite abrupt positional changes. This workflow empowers users to manually correct detection results, track unique objects, and generate detailed analytical outputs.

- **Goal:** Monitor and analyse floating litter.
- **Process:**
  Data Ingestion → Scanning → Review → Tracking → Analysis → Video Generation
- **Key Features:**
  - Manual review and correction of detection results.
  - Tracking of unique litter objects using IoU and template matching.
  - In-depth analysis and annotated video generation.

2.2 Model Improvement Workflow

Complementing this, the *model improvement workflow* focuses on enhancing detection performance by transforming validated annotations into high-quality, curated training datasets. As the fine-tuning dataset grows over time - sourced from individual frames extracted during user-reviewed video processing - FLOWT ensures that no redundant images are added when videos are reprocessed or revalidated. To maintain a clean and efficient training set, the system uses a **hash-based deduplication mechanism**, comparing the hash signatures of new frames against previously stored samples. This prevents duplicate images from entering the dataset and preserves the integrity and diversity of the curated training data used for model refinement. This workflow also supports version-controlled fine-tuning and ensures that refined models can be seamlessly reintegrated into the inference pipeline without interrupting ongoing waste-tracking tasks. Together, these workflows deliver a flexible, reliable, and user-friendly system for environmental monitoring and model evolution.

- **Goal:** Iteratively improve detection model performance.
- **Process:**
  Data Ingestion → Scanning → Review → Model Refinement
- **Key Features:**
  - Creation of high-quality, curated training datasets.
  - Version-controlled fine-tuning of models.
  - Seamless integration of improved models into the inference workflow.

A dedicated analysis module provides two complementary capabilities: *detection-based analytics*, offering performance summaries through metrics such as accuracy, precision, recall, class distributions, confidence scores, and tag frequency patterns to identify potential bias or drift. *AI-driven insights*, powered by OpenAI Large Language Models (LLMs), which interpret detection trends, highlight anomalies, and produce human-readable recommendations for improving data quality and operational workflows. These generative AI insights act as an analytical assistant, helping users recognise non-obvious patterns and make informed decisions.

3. Pipeline Architecture Overview

FLOWT's pipeline is built on a modular technical architecture designed to deliver a seamless and intuitive user experience across all workflow stages. The Navigation System ensures smooth movement between components by maintaining state persistence—for example, preserving selected videos, chosen models, and user inputs across pages—while a unified video display standardises all uploaded formats into .mp4. Complementing this, FLOWT's centralised configuration management uses a YAML-based system to automatically store and restore user preferences, with clearly structured sections dedicated to video generation, tracking, and fine-tuning. Building on these foundations, the pipeline stages work together to handle ingestion, automated scanning, human-in-the-loop

review, tracking, analysis, model improvement, and system orchestration. The following table maps each pipeline stage to its associated modules, key functions, and operational notes.

Navigation System Features:

- State Persistence: Maintains video and model selections across pages.
- Consistent UI: Uniform .mp4 video display, regardless of original format.
- Intuitive Navigation: Previous/Next buttons across all pages.

Configuration Management Features:

- Centralised YAML-based configuration (config/config.yaml).
- User preferences are automatically saved and restored.
- Modular sections for video generation, tracking, and fine-tuning.

To provide a clear understanding of the pipeline's structure and functionality, we organise the workflow into distinct stages, each responsible for a specific set of tasks. The pipeline progresses from initial data ingestion and automated scanning to review, tracking, and video generation, followed by detailed analysis, AI-driven insights, and iterative model improvement. Finally, orchestration ensures smooth integration and execution of all components.

3.1 Pipeline Stage Mapping Table

The following table maps each stage to its corresponding files or modules, highlights the key functions and classes involved, and provides additional notes to clarify their roles within the overall system.

Stage	Related Files/Modules	Key Functions/Classes	Notes
Data Ingestion	`pages/1_Data_Ingestion.py`	load_image(), load_video(), extract_metadata()	Converts uploaded videos to .mp4, extracts metadata
Scanning	`pages/2_Scanning.py`	select_model(model_architecture) • load_model() • run_inference()	Supports base/fine-tuned models
Review	`pages/3_Review.py`	load_detections(), edit_classification(), add_tag(), validate_curations(), get_curation_stats(), bulk_mark_tp(), bulk_mark_tp(), save_curation(), save_changes(), export_curated_data()	Bulk operations scoped to page
Tracking	`pages/4_Tracking.py`	track_objects(), calculate_iou(), match_template(), get_tracking_stats()	Timelapse support; adjustable thresholds
Video Generation	`pages/5_Video_Generation.py`	draw_options(), generate_video()	MP4/AVI/MOV; quality options
Trash Analysis	`pages/6_Trash_Analysis.py`	generate_analysis_report() • generate_detection_report() ○ calculate_metrics ○ analyse_confidence_distribution ○ analyse_tags • generate_tracking_report(), ○ analyse_trackings	Detection analysis, tracking analysis, metadata analysis, visualisations
AI Insight	`pages/7_AI_Insight.py`	get_provider_info() get_token_info() short_prompts() long_prompts()	Analysis using the Open AI agents
Model Improvement	`pages/8_Model_Refining.py`	create_curated_dataset prepare_ft_dataset get_system_info get_training_config run_fine_tuning display_model_versions display_ai_analysis	Version-controlled model fine-tuning

Orchestration	`Flowt_pipeline.py`	main()	Launches pipeline workflow

3.2. Source Code Structure

FLOWT's source code is organised into modular stages, each encapsulated in dedicated files to promote clarity, maintainability, and extensibility. This structured approach ensures each module has a clearly defined responsibility, enhancing reproducibility, scalability, and ease of future enhancements

- **Data Ingestion** stage `(pages/1_Data_Ingestion.py)` handles video upload, conversion to `.mp4`, and metadata extraction, providing foundational inputs for downstream tasks. Sample input data for this stage can be downloaded from the CSIRO data repository. (https://data.csiro.au/collection/csiro:72792)

- **Scanning** stage `(pages/2_Inference.py)` performs object detection[1] using either baseline or refined models, supporting configurable confidence thresholds.

- **Review** stage `(pages/3_Review.py)` enables manual curation of detections through users, scoped for bulk operations per page.

- **Tracking** stage `(pages/4_Tracking.py)` maintains temporal consistency of detected objects across frames, leveraging two separate algorithms: IoU calculations and template matching supported for timelapse videos.

- **Video Generation** stage `(pages/5_Video_Generation.py)` generates annotated videos, accommodating multiple formats and quality options.

```
FLOWT/
├── flowt/
│   ├── pages/
│   │   ├── 1_Data_Ingestion.py
│   │   ├── 2_Scanning.py
│   │   ├── 3_Review.py
│   │   ├── 4_Tracking.py
│   │   ├── 5_Video_Generation.py
│   │   ├── 6_Trash_Analysis.py
│   │   ├── 7_AI_Insight.py
│   │   ├── 8_Model_Refining.py
│   │   ├── 9_Edge_Deployment.py
│   │   └── Documentation.py
│   ├── data/
│   ├── Flowt pipeline.py
│   ├── src/
│   ├── config/
│   │   ├── config.yaml
│   │   └─floating_litter_classes.json
│   ├── requirements.txt
│   ├── Dockerfile
│   └── run.sh
├── models/
└── models_ft/
```

Figure2. Source Code Structure

- **Analysis** stages `(pages/6_Analysis.py)` provides statistical analysis of detected and tracked objects.

- **AI Insight** stages `(pages/7_Analysis_AI.py)` provides AI-driven interpretations of detected classes and refined models.

- **Model Improvement** stage `(pages/8_Model_Refining.py)` facilitates dataset creation, model refining, and deployment with version control to prevent duplication.

- **Edge Deployment** stage `(pages/9_Edge_Deployment.py) -` Future work will focus on compressing and optimising the best-performing refined model for efficient execution on edge devices such as embedded GPUs or portable AI accelerators, enabling real-time, on-site inference in field environments.

- **Orchestration** layer `(Flowt_pipeline.py)` serves as the main entry point, seamlessly coordinating the execution of all the above stages.

[1] The object detection model is trained on a predefined taxonomy of litter and debris classes. A full description of the class hierarchy and labels is provided in Appendix A to help users assess suitability for their application.

4. Deployment and Performance Considerations

This section is intended for IT staff and engineers who wish to deploy and run the pipeline. The pipeline has been designed to be executed without requiring any machine learning expertise, using pre-trained models and default configurations.

4.1 Hardware Requirements

The pipeline can be run on a range of systems depending on data volume and processing speed requirements.

- **Minimum requirements (CPU-based execution):**
  - **Processor:** Modern multi-core CPU
  - **RAM:** ≥ 8 GB
  - **GPU:** Not required
  - **Suitable for:**
    - Small datasets
    - Testing and development
    - Laptop or standard desktop environments

- **Recommended requirements (GPU-accelerated execution):**
  - **GPU:** NVIDIA GPU with ≥ 8 GB VRAM
  - **RAM:** ≥ 16 GB
  - **Suitable for:**
    - Large video collections
    - Faster processing times
    - Desktop workstations or shared servers

- **High-performance environments (optional):**
  - **HPC or cloud instances** with GPU support can be used for large-scale or batch processing workloads.

Note: The pipeline automatically uses GPU acceleration when available; otherwise, it falls back to CPU execution without requiring user intervention.

5. AI Insight

The AI Insight component of the FLOWT pipeline leverages OpenAI's Large Language Models (LLMs) to convert raw detection outputs into interpretable and actionable insights. These models analyse temporal detection patterns, assess spatial consistency, and identify anomalies that may indicate issues such as systematic false positives, environmental variability, or data degradation. By translating complex inference outputs into human-readable narratives, the LLM acts as an analytical assistant that supports users in recognising subtle trends, diagnosing errors, and making informed decisions across the workflow.

5.1 Integration with workflows

AI insight is implemented in two complementary modes depending on the workflow.

- **Waste Tracking workflow** AI Insight functions as a dedicated stage within the pipeline and is presented as a separate entry as 7th stage in the main navigation menu. This design emphasises interpretability of trash analysis (6th stage) by integrating system-generated summaries derived from scanning, tracking, and historical performance logs to highlight correlations that traditional statistical methods may overlook. For instance, the system can detect recurring misclassification patterns across specific lighting conditions, recommend parameter adjustments in the tracking configuration, or propose targeted model refinement strategies based on observed drift in model performance. These generative insights help guide dataset curation, model retraining priorities, and operational decision-making, contributing directly to continuous improvement of the detection system.

Within this stage, the LLM provides a suite of analytical capabilities, including:

- **Data Summary & Insights**, offering a high-level interpretation of object detections and spatio-temporal trends.
- **Performance Analysis**, highlighting detection stability, potential false positives, and temporal fluctuations.
- **Class Distribution Analysis**, characterising the composition of identified waste categories across the dataset.
- **Quality Assessment**, identifying inconsistencies, degradation points, and potential issues in the underlying imagery or annotations.
- **Custom Query**, enabling users to pose domain-specific questions for targeted, contextualised explanations.

- **Model Improvement workflow** AI Insight is embedded directly inside the *Model Refining* stage. This tighter integration aligns the LLM outputs with model selection and evaluation steps, supporting a more iterative fine-tuning process. For each refined model, the system produces detailed, model-specific insights such as:

  - **Training Performance Analysis**, assessing convergence behaviour, loss dynamics, accuracy patterns, and potential overfitting.
  - **Parameter Optimisation**, where the LLM interprets training metadata to suggest adjustments to hyperparameters, augmentation choices, or configuration settings.
  - **Custom Query**, allowing users to explore the behaviour of individual models and request explanations tailored to their improvement goals.

By differentiating how AI Insight is embedded in each workflow, FLOWT supports both operational decision-making during waste tracking and strategic refinement during model development. This dual-mode design enables the LLM to operate not only as a reporting tool but also as an active reasoning component that enhances situational awareness throughout the system.

In a nutshell, these two separate structured integrations ensures that generative AI augments user understanding at the appropriate stage, either by interpreting real-world detection outputs or by guiding model optimisation, thereby enhancing transparency, accelerating review cycles, and promoting continuous improvement across the pipeline.

5.2 Limitations and Safeguards

While the LLM-driven insight module provides substantial analytical value by interpreting detection patterns, identifying anomalies, and guiding model refinement, its use also requires careful consideration to ensure reliability and responsible operation. As with any generative AI system, there are inherent constraints that must be acknowledged, along with procedural measures designed to uphold the accuracy, transparency, and integrity of the insights produced. The following sections outline the key limitations of the approach and the safeguards implemented to mitigate potential risks.

5.2.1 Limitations

- **Risk of Hallucination:** LLMs may generate interpretations or recommendations that extend beyond the evidence provided in the system summaries.

- **Dependence on Input Quality:** Insight accuracy is constrained by the completeness and correctness of the detection and tracking data supplied to the model.

- **Limited Statistical Validation:** LLM-generated conclusions are not a substitute for quantitative evaluation and must be cross-checked against performance metrics.

- **Contextual Ambiguity:** The model may misinterpret results when frame-level or environmental context is insufficiently represented in the metadata.

- **Non-deterministic Outputs:** Generative responses may vary across sessions, requiring users to verify key findings when consistency is critical.

5.2.2. Safeguards

- **Grounded Prompting:** Insights are generated only from structured detection summaries and validated metadata to minimise unsupported reasoning.
- **Human-in-the-Loop Oversight:** All recommendations are subject to user review, ensuring expert judgement remains central to operational decisions.
- **Restricted Data Exposure:** Only non-sensitive, aggregated information is sent to the LLM, adhering to data governance and privacy requirements.
- **Cross-Validation Workflow:** Users are encouraged to verify AI-generated insights using statistical metrics and system logs before acting on them.
- **Operational Warnings:** The interface displays notifications when insights require cautious interpretation, such as during incomplete scans or early model training cycles.

6. Installation and Deployment

6.1 Prerequisites

The pipeline requires the following software and hardware components to ensure optimal performance:

- Python 3.9 or later
- Docker for containerised deployment
- A GPU-enabled environment for efficient fine-tuning and inference

6.2 Local Installation

The pipeline can be installed locally using the following steps:

```
git clone https://github.com/FNayyeri/FLOWT.git
cd FLOWT
./run.sh
```

6.3 Docker Deployment

For reproducible, portable deployment, the system provides full Docker support:

```
docker build -t flowt-pipeline .
docker run -p 8501:8501 flowt-pipeline
```

7. Technology Stack

FLOWT is implemented using a lightweight but robust technology stack:

- **Frontend:** Streamlit
- **Backend:** Python
- **AI Services:** OpenAI APIs
- **Containerisation:** Docker
- **Data Storage:** Local filesystem (YAML, JSON, CSV)

8. Future Enhancements

Planned extensions aim to broaden the system's scalability, efficiency, and real-world applicability across both cloud-based and edge-based environments:

- **Cloud Integration:** Integration with cloud-based storage platforms such as AWS S3 and Azure Data Lake to support scalable data storage, efficient data management, and multi-user access across distributed teams and deployments.
- **Edge Deployment and Optimisation:** Future development will focus on preparing the pipeline for deployment on edge devices by compressing and optimising the best-performing refined models to run efficiently on embedded GPUs and low-power AI accelerators. This enables real-time, on-site inference while significantly reducing latency and reliance on server-based processing.
- **Real-Time Field Operation:** Edge-based inference allows the system to operate effectively in remote or bandwidth-constrained environments, ensuring continuous functionality without dependency on high-bandwidth network connectivity.
- **Efficiency and Responsiveness:** By enabling near-instantaneous inference results at the point of data capture, edge deployment improves system responsiveness and supports time-critical environmental monitoring tasks.
- **Scalability and Practical Applicability:** Preparing the pipeline for both cloud and edge execution expands its usability across diverse operational contexts, supporting scalable, field-based automated debris detection while preserving the high accuracy and robustness achieved through fine-tuning.

9. Licence License and Third-Party Dependencies

9.1 License for This Pipeline

The original code and documentation developed as part of this pipeline are released under the **Creative Commons Attribution 4.0 International (CC BY 4.0)** license. This license permits users to share, adapt, and reuse the work, including for commercial purposes, provided that appropriate attribution is given to the original author.

9.2 Use of YOLO and License Constraints

At its current stage, this pipeline relies on **YOLO framework (Ultralytics[2])** for object detection. YOLO is released under the **GNU Affero General Public License (AGPL-3.0)**, which introduces specific obligations and restrictions, particularly with respect to commercial use and deployment. Users intending to apply this pipeline in commercial or production environments must ensure compliance with the YOLO license terms or obtain a commercial license from Ultralytics.

Although YOLO is used as the default object detection framework in the present implementation, the pipeline has been designed in a **modular and extensible manner**. With appropriate modifications, the YOLO-based detection component can be replaced by alternative object detection models or frameworks that use different licensing schemes. This flexibility allows future users to adapt the pipeline to meet specific technical, operational, or licensing requirements, including scenarios where more permissive commercial use is desired.

[2] Ultralytics. *YOLO: Real-Time Object Detection Models*. Ultralytics. Available at: https://github.com/ultralytics/ultralytics

Appendix A: Object Class Taxonomy

	Class Name	Material	Description / Examples
1	Packaging	Soft Plastic / Hard Plastic / Cardboard / Paper / Thin Film Bag	Colorful packaging materials with printed designs, brand names, or text. Examples include snack wrappers, chip bags, and printed paper food wrappers. Often lightweight and easily carried by water currents, they tend to float and spread widely in waterways.
2	Other_packaging	Aluminium / Soft Plastic / Hard Plastic / Cardboard / Thin Film Bag	Unbranded or plain packaging with a smooth or shiny surface, often silver, white, or metallic. Examples include foil from chocolate bars, unmarked packaging film, or industrial wrapping. Typically lacks writing or colorful patterns.
3	S_bubblewrap	Soft Plastic	Soft, transparent or translucent sheets with raised air pockets, commonly used for protecting fragile goods during shipping. Pieces may appear partially deflated and tangled in other debris.
4	S_label	Soft Plastic	Thin, flat plastic labels from bottles or packages. They are often colorful, waterproof, and contain printed brand information. They can detach easily from containers and float separately.
5	S_squeeze	Soft Plastic	Soft, flexible plastic bottles designed to dispense liquids like toothpaste, sauces, or lotions. They may be partially crushed and have caps still attached.
6	S_straw	Soft Plastic	Thin cylindrical drinking straws, often single-use and lightweight. They can be whole or broken into smaller fragments, commonly found in waterways.
7	PS_string	Soft Plastic	Includes fishing lines, synthetic ropes, or string. Often entangled with seaweed or other debris. Commonly used in fishing, boating, or packaging.
8	P_cardboard	Cardboard	Crumpled, ripped, or soaked cardboard pieces from boxes or packaging. If colorful and branded, classify under Packaging. When plain or worn, stays in this category.
9	P_foodcontainer	Hard Plastic / Paper	Rigid or semi-rigid containers used for takeaway meals or pre-packaged food. Includes lunch boxes, clamshell containers, or paper food trays.
10	PH_cup	Hard Plastic	Hard plastic cups used for beverages, often stackable. Can be found whole or broken into sharp fragments.
11	H_packaging	Hard Plastic	Plastic drink bottles with smooth, rounded edges. Includes water bottles, soda bottles, and sports drink containers.
12	H_otherbottle	Hard Plastic	Plastic bottles that are not for beverages, such as cleaning product containers, spray bottles, or buckets.
13	H_plate/bowl	Hard Plastic	Large, round dishes made of hard plastic, such as picnic plates or mixing bowls.
14	H_utensil	Hard Plastic	Plastic eating utensils like forks, spoons, and knives. Typically, lightweight and found near food packaging debris.
15	DH_lid	Hard Plastic	Small, round plastic lids from bottles, jars, or containers. These are among the most common littered items due to their small size.
16	D_polystyrene	Polystyrene	Lightweight foam material used in food packaging, cups, and protective packaging. Breaks easily into small fragments that float extensively.
17	M_beveragecan	Metal	Aluminium cans used for drinks such as soda or beer. Can be intact or crushed.
18	M_foodcan/tin	Metal	Metal tins or cans used for food storage, like soup cans or tuna tins. May show signs of rusting when exposed to water.
19	M_aerosol	Metal	Pressurized spray cans used for deodorants, insecticides, or cleaning sprays. Usually cylindrical with a plastic cap.
20	R_ball/balloon	Rubber	Deflated or torn balloons, rubber balls, or similar rubber-based recreational items. May still have strings or ribbons attached.
21	G_beveragebottle	Glass	Glass bottles used for beverages like beer or soda. Can be whole or broken into sharp fragments.
22	F_facemask	Fabric	Disposable or reusable face masks made from fabric or mixed materials. Commonly found since the COVID-19 pandemic.
23	T_wood/timber	Timber	Wooden fragments such as sticks, construction offcuts, or driftwood that originate from man-made sources.
24	Other	Any	Any man-made object not specifically listed above. Examples include toys, electronics, or unusual items like shoes or clothing.

Appendix B: Glossary of Terms

	Term	Description
1	Bounding Box	A rectangle drawn around an object in an image to indicate its location.
2	GPU (Graphics Processing Unit)	A processor designed to handle complex calculations quickly, often used for training AI models and processing images or videos.
3	HPC (High-Performance Computing)	Powerful computer systems used to run very large or complex calculations quickly.
4	IoU (Intersection over Union)	Shows how much a predicted box matches the actual object, from 0 (no match) to 1 (perfect match).
5	LLM (Large Language Model)	An AI model trained on vast amounts of text to understand and generate human-like language.
6	Precision	The fraction of correctly identified objects out of all objects the model predicted.
7	Recall	The fraction of correctly identified objects out of all actual objects in the image.
8	VRAM (Video RAM)	Memory on a GPU that stores data for processing images, videos, or AI computations.
9	YAML	A human-readable file format often used to store configuration settings for programs.
10	YOLO	A popular object detection model that predicts objects in images in real-time.