Civil Supplies Project Documentation

The Civil Supplies Project automates the monitoring and management of warehouse activities using three integrated systems:
- Facial Recognition System for identifying individuals.
- Gunny Bags Counting System for tracking loading and unloading activities.
- Number Plate Recognition System for vehicle access control.
These systems work together through a unified pipeline that processes live video feeds, enhancing operational efficiency, security.

Prerequisites & Configuration

Before running the Civil Supplies Warehouse pipeline, ensure these secrets are configured in GitHub:

Azure Configuration

- AZURE_TENANT_ID – Tenant ID for Azure authentication.
- AZURE_CLIENT_ID – Client ID of the registered Azure application.
- AZURE_CLIENT_SECRET – Client secret for the registered Azure application.
- AZURE_STORAGE_ACCOUNT_NAME – Storage account name.
- AZURE_CONTAINER_NAME – Container name for storing CCTV video chunks.

PostgreSQL Database

- PG_HOST – Host IP/hostname of the database server.
- PG_PORT – Database port.
- PG_USER – Database user.
- PG_PASSWORD – Database user password.
- PG_DATABASE – Database name.

Pipeline Breakdown:

1. Video Ingestion and Storage

- **Component:** AWS Lambda (**WarehouseAzure-S3-Transfer-Video-Function**)

- **Process:** Continuously monitors the live CCTV feed (RTSP/RTMP).
- **Action:** Segments the feed into short video clips (.ts/.mp4) and uploads them to **Azure Blob Storage** (e.g., spectradevdev).

2. Job Triggering

- **Component:** AWS Lambda (API Trigger)
- **Process:** Listens for new blob upload events in Azure.
- **Action:** Sends an HTTP POST request (/api/process) to the **FastAPI Main Service**, including the blob URL and task metadata.

3. Frame Preparation

- **Component:** FastAPI Main Service
- **Submodules:**
    - azure_downloader.py → Downloads the video clip from Azure Blob.
    - frame_extractor.py → Converts the video into sequential frames for analysis.

4. Orchestration and Model Inference

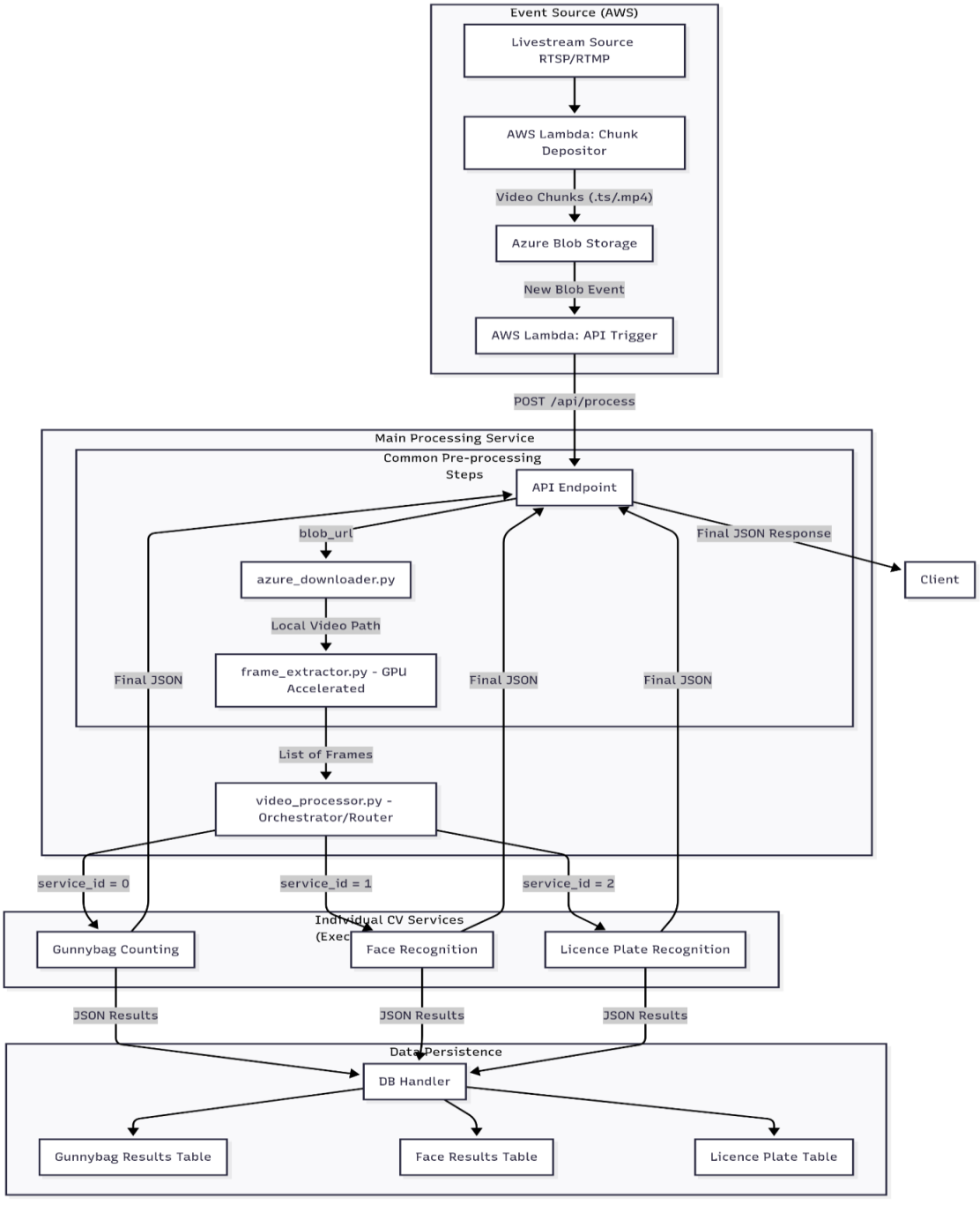
- **Component:** Task Orchestrator (videoprocessor.py)
- **Process:** Reads task type (Gunny bag, Face, NumberPlate) from the trigger payload.
- **Action:** Dispatches frames to the relevant **services**:
    - Gunnybag Counting Service
    - Facial Recognition Service
    - License Plate Recognition Service
- **Output:** Each service processes frames and produces structured results.

5. Persistence and Response

- **Component:** DB Handler + FastAPI Response

- **Process:**
    - DB Handler writes structured results into corresponding database tables (gunny bags, faces, numberplates).
    - FastAPI endpoint sends a **JSON response** back to the client confirming job completion.

System Architecture:



Services

1. Facial Recognition System

The Facial Recognition System identifies individuals captured in warehouse CCTV feeds. It aims to automate identity verification, improve security, and maintain accurate logs.

1.1 Workflow

Frame Input

↓

Person Detection (YOLOv8l)

↓

Tracking (BoT-SORT)

↓

Face Cropping

↓

Face Recognition (Buffalo_L)

↓

Result Aggregation

↓

Final Output

1.2 Models Used

- **YOLOv8l**
    - Task: Person detection in frames.
    - mAP (mean Average Precision): **52.9**.
- **Buffalo_L (InsightFace)**
    - Task: Face embedding extraction for recognition.
    - Accuracy: **93.16%** on standard benchmarks.

1.3 Key Technical Components

- YOLOv8l: Detects persons in video frames.
- BoT-SORT Tracker: Maintains consistent IDs across frames.
- Buffalo_L Model: Extracts embeddings for recognition.
- FAISS: Performs fast similarity searches on embeddings.
- Database (face_database.pkl): Stores known embeddings.
- Asyncio → Offloads CPU-heavy tasks like embedding extraction and FAISS search to

1.4 Integration Points

This module integrates with the pipeline's frame extractor and orchestrator to process faces from video clips stored in the cloud.

1.5 Implementation Details

The pipeline resets trackers per clip, processes every 5th frame for recognition, and aggregates results to produce consistent identity outputs.

2. Gunny Bags Counting System

The Gunny Bags Counting System tracks and counts gunny bags being loaded or unloaded across a defined line. Its goal is to automate inventory updates and reduce manual counting errors.

2.1 Workflow

Input Video

↓

YOLO Detection (Gunny Bags)

↓

Non-Max Suppression

↓

Tracking (Kalman Filter + Hungarian Algorithm)

↓

Crossing Detection

↓

Count Updates

↓

Output Statistics

2.2 Models Used

- **YOLO (Ultralytics)**
    - Task: Detect gunny bags in each video frame.
    - Confidence threshold: 0.35.
- **Kalman Filter (OpenCV)**
    - Task: Smooth object trajectories and predict positions when detections are missing.
- **Hungarian Algorithm (scipy.optimize.linear_sum_assignment)**
    - Task: Assign detections to existing tracks with minimum cost based on IoU and distance.

2.3 Key Technical Components

- YOLO (Ultralytics): Detects gunny bags.
- Kalman Filter: Smooths trajectories and predicts missing detections.
- Hungarian Algorithm: Matches detections to tracks.
- Crossing Detection Logic: Determines loading/unloading events.

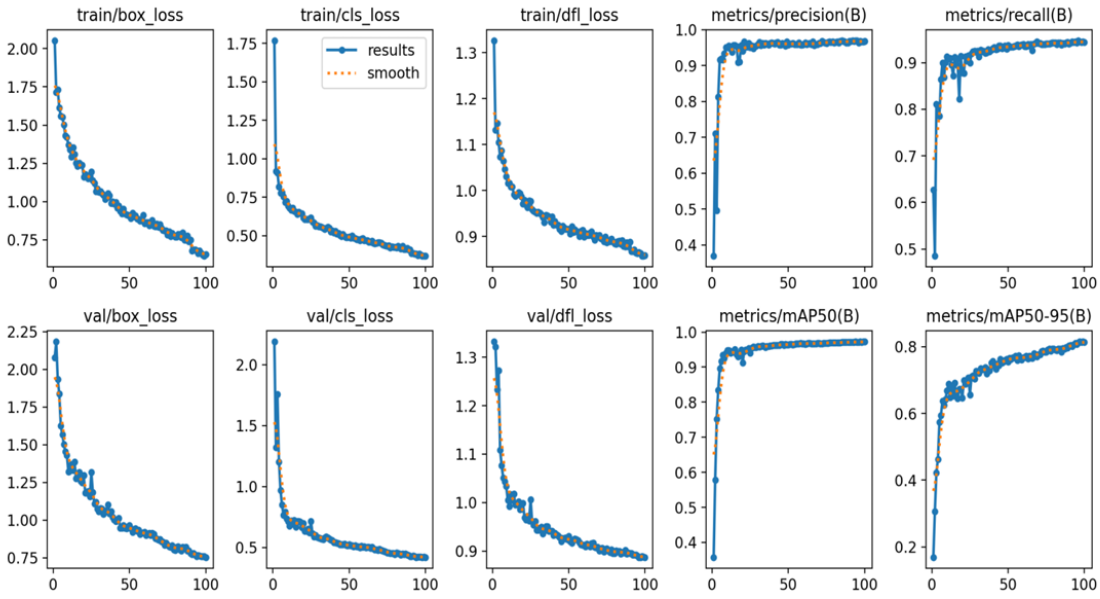
2.4 Integration Points

This module integrates with the pipeline to process warehouse footage and provide real-time loading/unloading statistics.

2.5 Implementation Details

Tracks have defined states (tentative → confirmed). Events are detected based on line crossing, with logs produced at regular frame intervals.

2.6 Model Related Graphs:



3. Number Plate Recognition System

The Number Plate Recognition System identifies and verifies vehicle license plates entering or exiting the warehouse. It ensures secure access control by matching plates against an authorized database.

3.1 Workflow

Video Input

↓

YOLO Detection (License Plates)

↓

Tracking (BoT-SORT)

↓

OCR (PaddleOCR)

↓

Cleaning & Validation

↓

Fuzzy Matching with Database

↓

Result Aggregation

↓

Final Output

3.2 Models Used

- **YOLO (License Plate Detection):**
  - Task: Detect license plates in frames.
  - mAP@0.5: 0.9759
  - AP@0.5:0.95: 0.8997
  - Precision: 0.9849
  - Recall: 0.9457
- **PaddleOCR:**
  - Task: Extract alphanumeric text from cropped license plate regions.
  - Character Matching Accuracy: ~96% – 98%
  - BLEU Score: ~0.93% – 0.96%
  - BoT-SORT Tracker: Maintain consistent tracking IDs across frames for each vehicle/plate.

3.3 Key Technical Components

- YOLO: Detects license plates.
- PaddleOCR: Extracts alphanumeric text.
- BoT-SORT Tracker: Maintains persistent IDs.
- Regex Validators: Validate plate formats.
- Fuzzy Matching: Compares detected plates with database.
- PostgreSQL Database: Stores valid plates and access permissions.

3.4 Integration Points

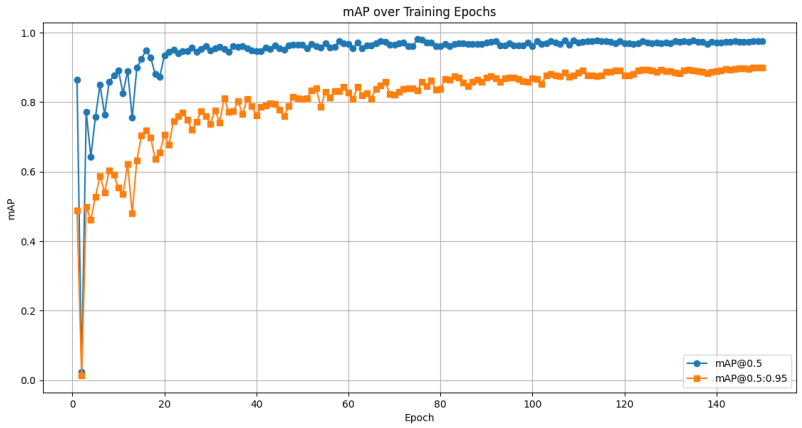
This module integrates with the pipeline orchestrator to process vehicle footage and update access logs.

3.5 Implementation Details

Processes every nth frame (fps/3), aggregates OCR results, and locks final matches once validated with the database.

3.6 YOLO model metric graphs:

3.6.1. mAP over Training Epochs



3.6.2. Training vs Validation Loss