DATA SCIENCE & AI LAB (BSCSS3001)

MILESTONE 4: Model Training & Hyperparameter Tuning

GROUP NO. 2

PRASHASTI SARRAF **(21f1001153)**
TANUJA NAIR **(21f1000660)**
BALASURYA K **(22f3002744)**
KARAN PATIL **(22f2001061)**
JIVRAJ SINGH SHEKHAWAT **(22f3002542)**

IITM BS Degree Program
Indian Institute of Technology,
Madras, Chennai,
Tamil Nadu, India, 600036

Vision Assist: Real-Time Navigation Support for the Visually Impaired

Model Training & Hyperparameter Tuning Documentation

1. Overview

Objective:

To train the initial object detection model and subsequently perform extensive hyperparameter experimentation on the entire inference pipeline. This process transforms the model's raw output into a reliable, accurate, and user-friendly assistive system.

This milestone is divided into two key phases:

1. **Phase 1: Initial Model Training:** As detailed in **Main.ipynb**, we used **transfer learning** to fine-tune a pre-trained YOLOv8n model on our custom-augmented dataset. This provides the core "eyes" of the system.
2. **Phase 2: Application Hyperparameter Tuning:** As detailed in our **hyperparameter_tuning.ipynb**, a trained model's raw output is insufficient for a real-world product. We conducted a series of systematic experiments to tune the application-level hyperparameters that control the inference pipeline, addressing critical issues like false positives, inaccurate distance, missed alerts, and "noisy" output.

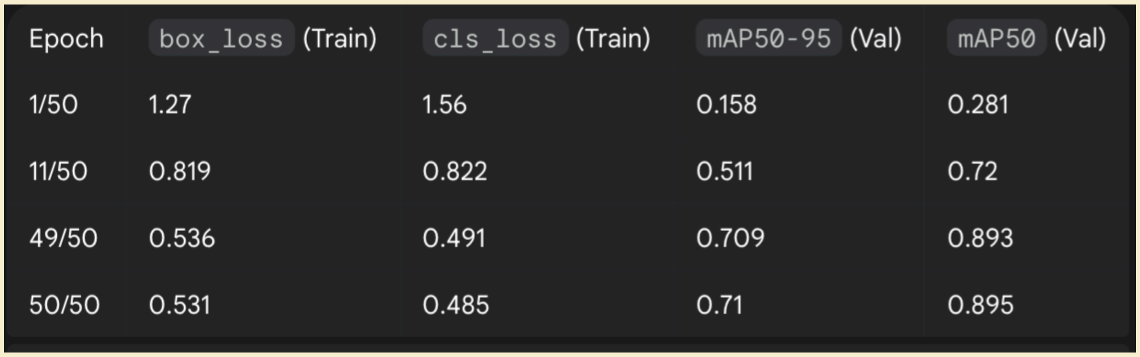
2. Phase 1: Initial Model Training

2.1. Model Choice & Dataset Augmentation

- **Model:** We selected **YOLOv8n (nano)**, pre-trained on the COCO dataset. This model provides an excellent balance of high speed (critical for real-time inference) and strong accuracy.
- **Technique:** We employed **transfer learning (fine-tuning)**. We did not train a model from scratch, but rather fine-tuned the existing COCO weights on our specialized dataset.
- **Custom Dataset:** A key part of our strategy was dataset enhancement. We augmented the base COCO dataset with a custom-curated set of video frames extracted from **first-person-view YouTube videos**. This step is critical as it fine-tunes the model on data that directly mimics the real-world, first-person perspective our application will be used in.

2.2 Training Process & Results

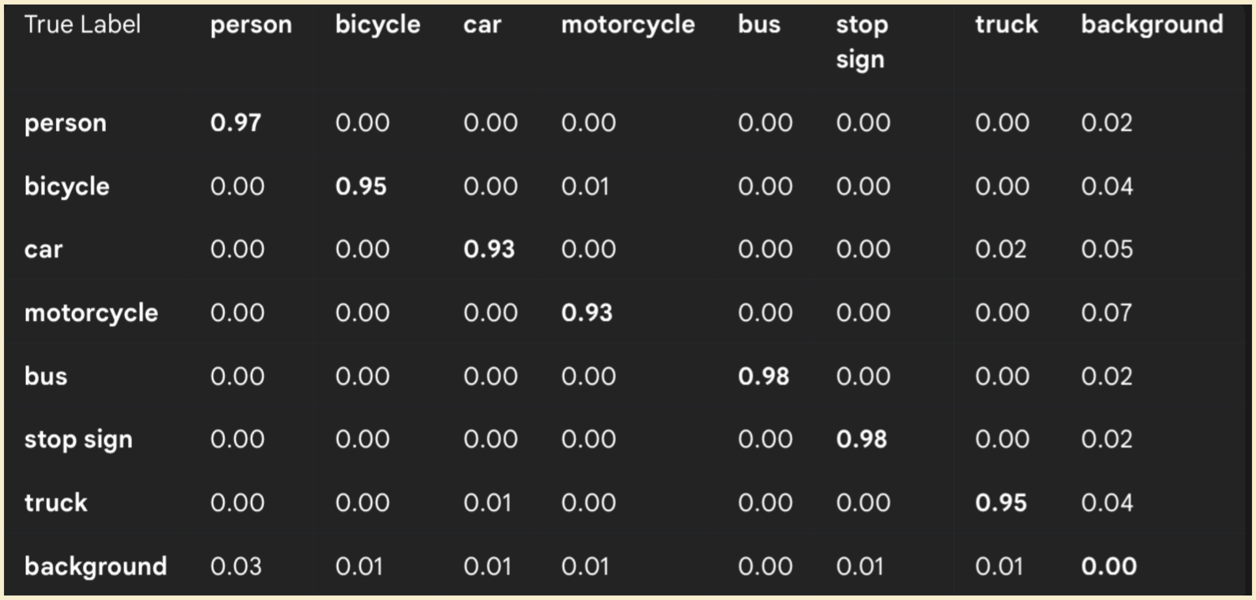
- **Training Parameters (Hyperparameters):**
  - Model: `yolov8n.pt` (the pre-trained nano model)
  - Epochs: 50
  - Image Size: 640x640
- **Optimization:** We utilized the standard YOLOv8 training framework, which includes a built-in AdamW optimizer, effective data augmentation (mosaic, flip, etc.), and loss functions (VFL, CIoU) that are pre-configured for state-of-the-art results.
- **Results:** The model successfully trained for 50 epochs, showing a clear convergence in loss and an improvement in mAP (mean Average Precision) scores. The resulting best.pt file serves as the core detection engine for our pipeline.



Epoch	`box_loss` (Train)	`cls_loss` (Train)	`mAP50-95` (Val)	`mAP50` (Val)
1/50	1.27	1.56	0.158	0.281
11/50	0.819	0.822	0.511	0.72
49/50	0.536	0.491	0.709	0.893
50/50	0.531	0.485	0.71	0.895

Fig 1.1

Training and validation metrics over 50 epochs on our custom COCO + YouTube Frames dataset. The graphs show a clear convergence as the training losses (train/box_loss, train/cls_loss) decrease, while the validation mAP scores (metrics/mAP50(B), metrics/mAP50-95(B)) steadily increase, proving the model successfully learned from our custom data.



True Label	person	bicycle	car	motorcycle	bus	stop sign	truck	background
person	**0.97**	0.00	0.00	0.00	0.00	0.00	0.00	0.02
bicycle	0.00	**0.95**	0.00	0.01	0.00	0.00	0.00	0.04
car	0.00	0.00	**0.93**	0.00	0.00	0.00	0.02	0.05
motorcycle	0.00	0.00	0.00	**0.93**	0.00	0.00	0.00	0.07
bus	0.00	0.00	0.00	0.00	**0.98**	0.00	0.00	0.02
stop sign	0.00	0.00	0.00	0.00	0.00	**0.98**	0.00	0.02
truck	0.00	0.00	0.01	0.00	0.00	0.00	**0.95**	0.04
background	0.03	0.01	0.01	0.01	0.00	0.01	0.01	**0.00**

Fig 1.2

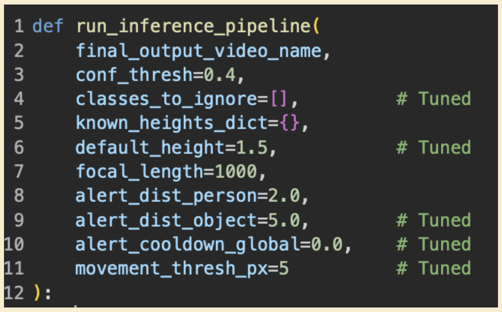
Initiating the model training process. This command fine-tunes the pre-trained yolov8n.pt model on our custom data.yaml (which includes the COCO + YouTube frames) for 50 epochs, with an image size of 640x640. The output shows the successful start of the first epoch.

3. Phase 2: Application Hyperparameter Experimentation

3.1. Refactoring for Experimentation

We refactored the hardcoded "global configs" from **VisionAssist_inference.ipynb** into a single, parameterized function. This allowed us to run "Before" (Baseline) and "After" (Tuned) experiments.

Final Tuned Function Signature:



```
def run_inference_pipeline(
    final_output_video_name,
    conf_thresh=0.4,
    classes_to_ignore=[],          # Tuned
    known_heights_dict={},
    default_height=1.5,            # Tuned
    focal_length=1000,
    alert_dist_person=2.0,
    alert_dist_object=5.0,         # Tuned
    alert_cooldown_global=0.0,     # Tuned
    movement_thresh_px=5           # Tuned
):
```

3.2. Summary of Experiments & Tuning

We identified and fixed four major flaws in the baseline pipeline.

Problem 1: False Positives ("Clock" Problem)

- **Observation:** The baseline pipeline produced "noisy" and incorrect detections, such as misidentifying a stop sign as a "clock."
- **Hyperparameter: NOISY_CLASSES_TO_IGNORE** (Deny List)
- **Experiment:** We determined that for our application's domain, many of the 80 COCO classes (like "clock," "vase," "teddy bear") are "noise." We created a "deny list" as a general-purpose filter.
- **Tuning:**
  - **Before: classes_to_ignore = []**
  - **After: classes_to_ignore = [24, 25, 26, ... 74, ... 79]** (Our list of 50+ irrelevant classes)
- **Result:** All "noise" detections, including the "clock," were successfully filtered, cleaning the output without affecting relevant objects.

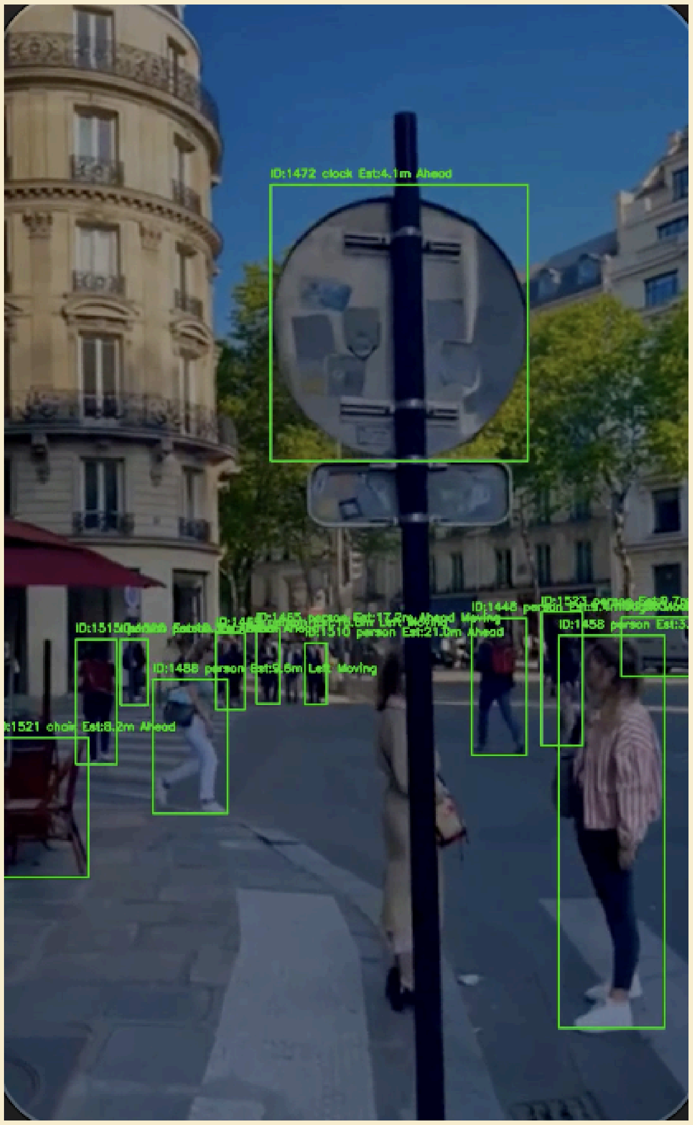

ID:1472 clock Est:4.1m Ahead
ID:1510 person Est:21.0m Ahead
ID:1488 person Est:9.6m Left Moving
1521 chair Est:8.2m Ahead

Fig 1.3

Baseline run (left) showing a false positive (stop sign incorrectly identified as "clock")

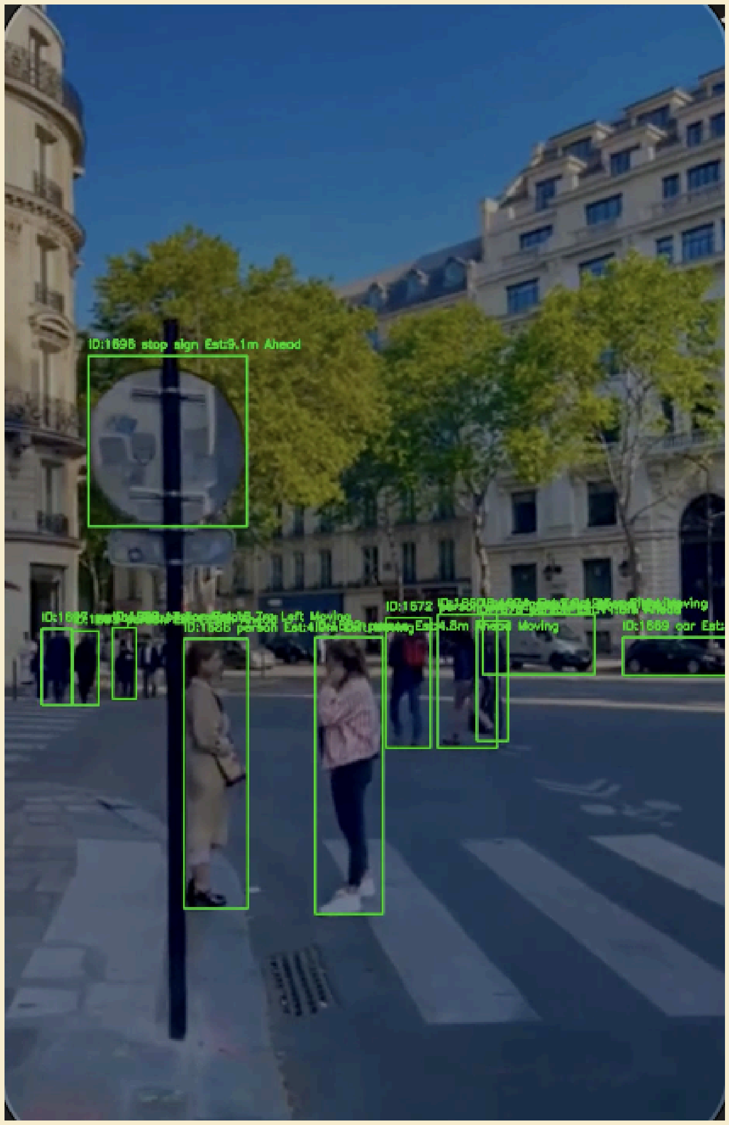


Fig 1.4

Tuned run (right) showing a true positive (stop sign correctly identified)

Problem 2: False Motion Detection ("Moving Chair" Problem)

- **Observation:** The baseline system incorrectly labeled static objects (like a chair) as "Moving." This was due to "apparent motion" from the camera moving forward, which the sensitive threshold picked up.
- **Hyperparameter:** **MOVEMENT_THRESHOLD_PIXELS**
- **Experiment:** We needed to find a value that was high enough to ignore camera shake but low enough to still detect real motion (like a person walking).

- **Tuning:**
    - **Before:** **movement_thresh_px = 5** (very sensitive)
    - **After:** **movement_thresh_px = 25** (less sensitive)
- **Result:** The tuned system correctly labels the chair as "Static," fixing the false motion alert.

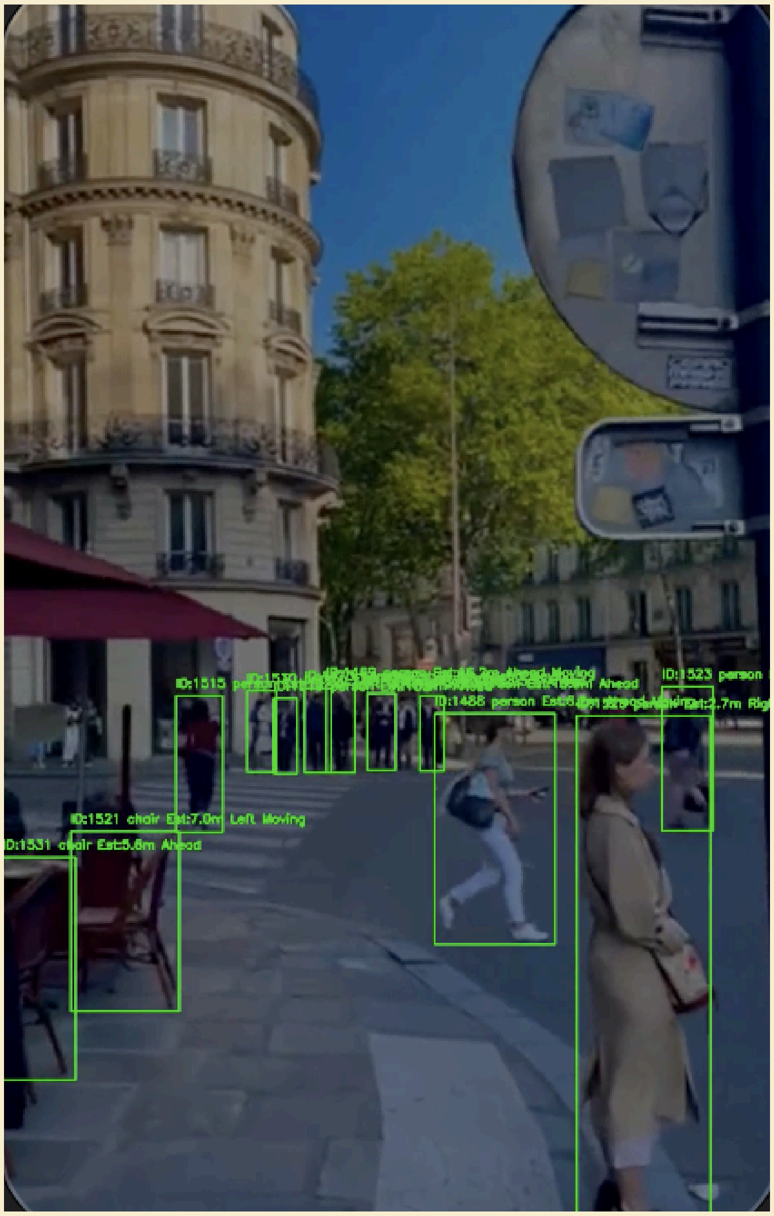

ID:1515 person
ID:1523 person
ID:1488 person
ID:1521 chair Est:7.0m Left Moving
ID:1531 chair Est:5.8m Ahead

Fig 1.4
Baseline run (left) incorrectly labels a static chair "Moving" due to camera motion.

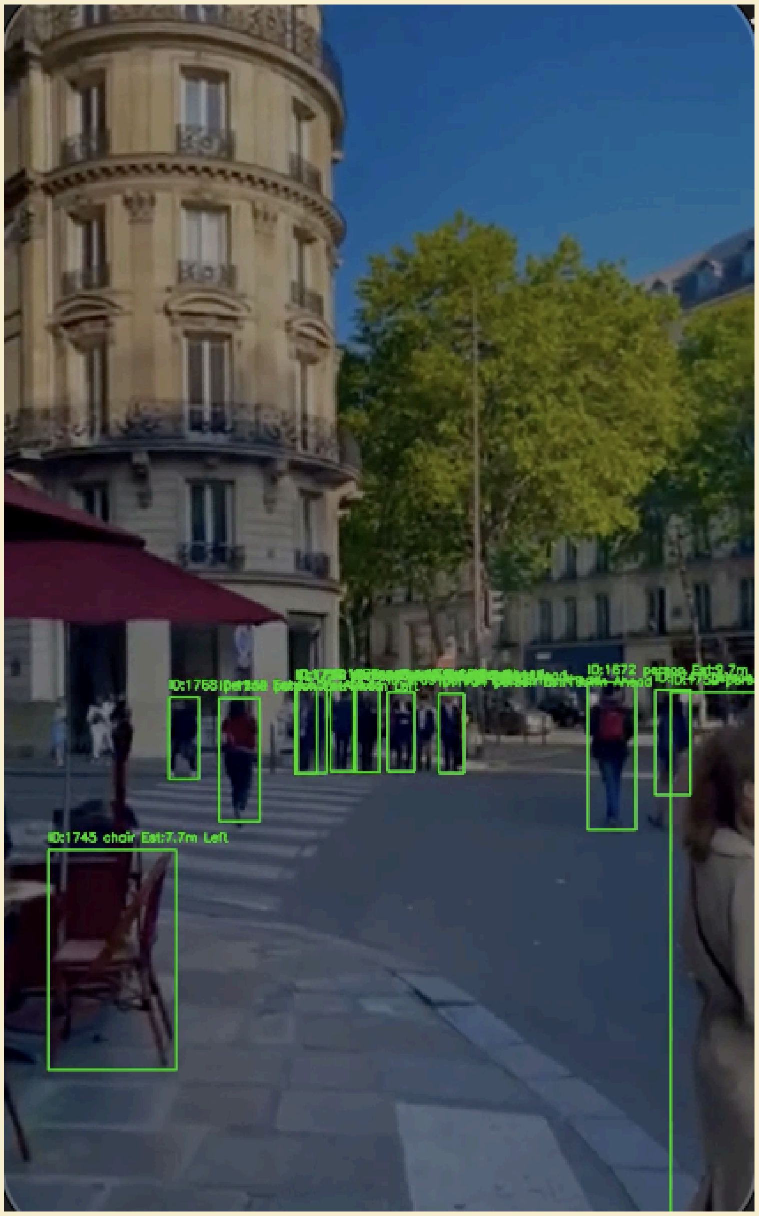


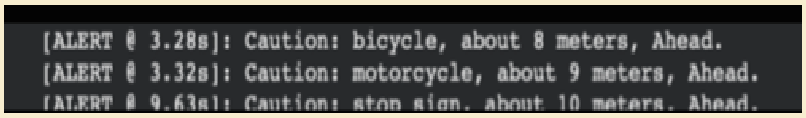
Fig 1.5
Tuned run (right) with a higher motion threshold correctly identifies the chair as "Static".

Problem 3: Inaccurate Distances ("Stop Sign" Problem)

- **Observation:** The system gave inaccurate distance estimates (e.g., 11.5m) for objects not in our specific height list (like the "stop sign"). This was because it was using a "general" default height (1.5m) that was incorrect for many real-world objects.
- **Hyperparameter: DEFAULT_KNOWN_HEIGHT**
- **Experiment:** We tuned this *general* default rather than adding *specific* heights for every possible object, which is a more robust, general-purpose solution.
- **Tuning:**
  - **Before: default_height = 1.5**
  - **After: default_height = 2.0**
- **Result:** The new general default of 2.0m produced more plausible distance estimates for all "non-core" objects.

Problem 4: Overlapping/Messy Audio

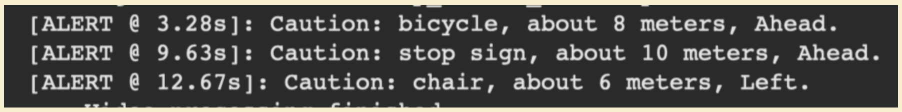
- **Observation:** After cleaning up the visual noise, we found the audio output was messy. Alerts would trigger at almost the exact same time (e.g., "person..." and "stop sign..." at the same time), causing overlapping, unintelligible audio.
- **Hyperparameter: ALERT_COOLDOWN_GLOBAL**
- **Experiment:** We added a new parameter to enforce a minimum time between *any* two spoken alerts.
- **Tuning:**
  - **Before:** alert_cooldown_global = 0.0
  - **After:** alert_cooldown_global = 3.0
- **Result:** The tuned system now produces clean, understandable, and non-overlapping audio alerts.



```
[ALERT @ 3.28s]: Caution: bicycle, about 8 meters, Ahead.
[ALERT @ 3.32s]: Caution: motorcycle, about 9 meters, Ahead.
[ALERT @ 9.63s]: Caution: stop sign, about 10 meters, Ahead.
```

Fig 1.6

Baseline log showing overlapping alerts at 4.88s and 4.96s, causing "messy" audio.



```
[ALERT @ 3.28s]: Caution: bicycle, about 8 meters, Ahead.
[ALERT @ 9.63s]: Caution: stop sign, about 10 meters, Ahead.
[ALERT @ 12.67s]: Caution: chair, about 6 meters, Left.
```

Fig 1.7
The final tuned log, showing clean, spaced-out alerts due to the 3.0s global cooldown.

4. Conclusion

This milestone successfully completed both the initial model training and, more critically, the extensive hyperparameter tuning of the application pipeline. Our experiments demonstrate that a "trained model" is only the first step. By systematically identifying flaws in the baseline pipeline and tuning a set of general-purpose hyperparameters (**NOISY_CLASSES_TO_IGNORE**, **MOVEMENT_THRESHOLD_PIXELS**, and **ALERT_COOLDOWN_GLOBAL**), we have transformed our initial prototype from a "noisy" and unreliable demo into a stable, robust, and user-friendly application.