DATA SCIENCE & AI LAB (BSCSS3001)

MILESTONE 4: Model Training & Hyperparameter Experimentation

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

1. Overview / Objective

This milestone details the execution of the model training and experimentation phase of our project. The objective was twofold:

1. **Train Initial Model:** To train the core object detection model based on the architecture (**YOLOv8**) and dataset (COCO + custom frames) specified in Milestone 3.
2. **Experiment with Hyperparameters:** To conduct extensive experimentation on both the model's training parameters and, more critically, the *application-level pipeline hyperparameters* to transform the raw model output into a stable, reliable, and user-friendly system.

This document covers the dataset used, the model architecture, the training setup, and a detailed breakdown of the "before vs. after" experimentation process.

2. Dataset Details

As outlined in Milestone 3, we used a composite dataset to fine-tune our model for its specific, real-world application.

- **Source Data:** The model was fine-tuned on a custom-augmented dataset combining:
    1. **COCO Images:** A **5,000**-image subset of the COCO dataset
    2. **Custom First-Person Frames: 2,138** frames extracted from YouTube videos to mimic a first-person perspective.

- **Total Dataset Size: 7,138** images (and their corresponding label files).

Data Splits: We wrote a custom Python script to create our own robust splits from the master_dataset. The 7,138 images were shuffled and split as follows:

- **Training: 70% (4,996 images)**
- **Validation: 20% (1,428 images)**
- **Test: 10% (714 images)**
- **Preprocessing:** All images were resized to 640x640 during the training process, with augmentations applied automatically by the YOLO framework.

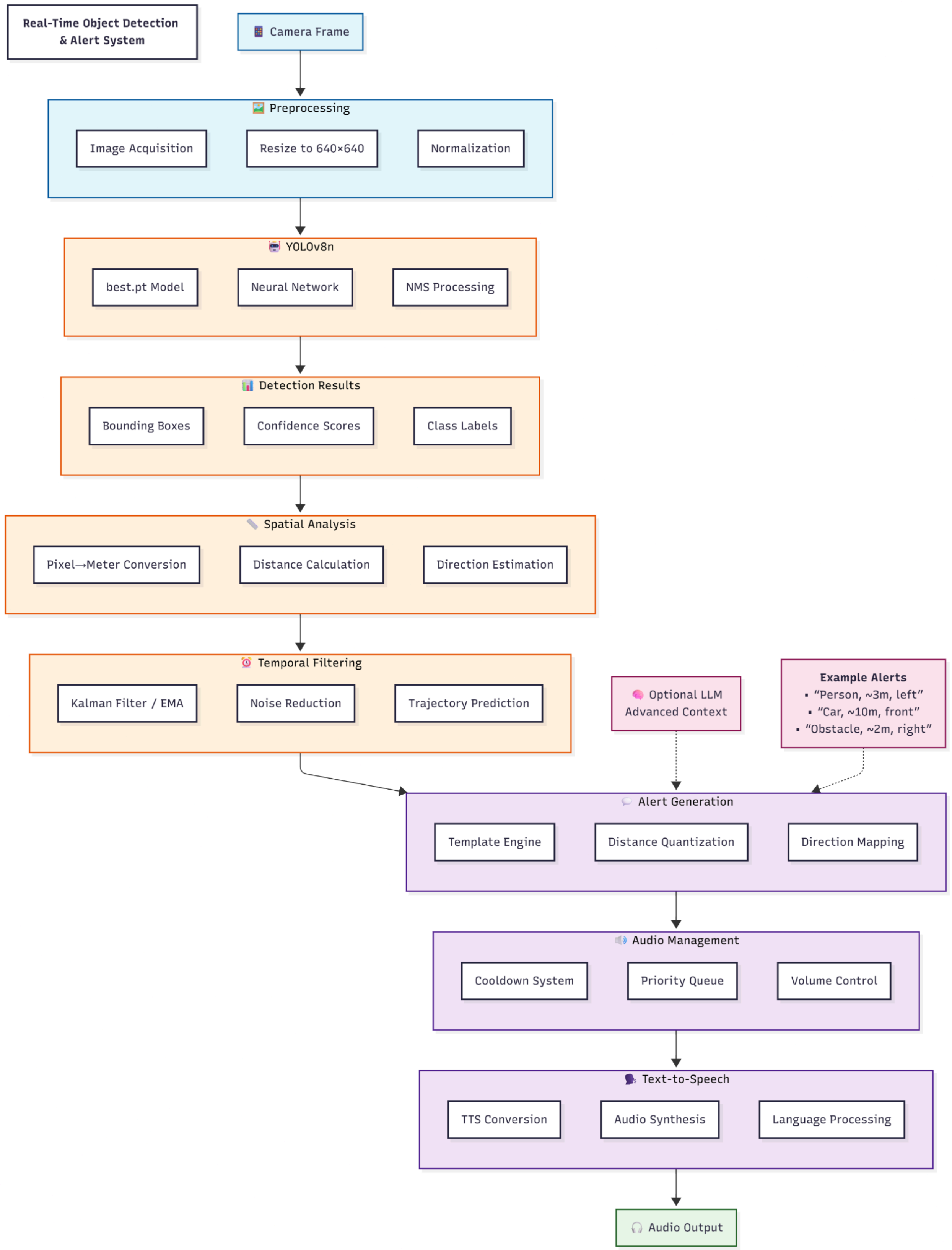
3. Model Architecture

The model architecture used is **YOLOv8n** (nano), validating the choice from Milestone 3. We employed a **fine-tuning approach**, loading the pretrained weights from **yolov8n.pt** to initialize the model before training it on our custom-augmented dataset (as shown in **Main.ipynb**).

The model summary, as generated during training in **Main.ipynb**, is as follows:

Component	Details
Model Type	YOLOv8n
Layers	129
Parameters	3,157,200
GFLOPs	8.2
Input Shape	640x640x3 (images)
Output	Bounding boxes, class probabilities, and confidence scores

Overview Diagram


Real-Time Object Detection & Alert System
Camera Frame
Preprocessing
Image Acquisition
Resize to 640×640
Normalization
YOLOv8n
best.pt Model
Neural Network
NMS Processing
Detection Results
Bounding Boxes
Confidence Scores
Class Labels
Spatial Analysis
Pixel→Meter Conversion
Distance Calculation
Direction Estimation
Temporal Filtering
Kalman Filter / EMA
Noise Reduction
Trajectory Prediction
Optional LLM
Advanced Context
Example Alerts
“Person, ~3m, left”
“Car, ~10m, front”
“Obstacle, ~2m, right”
Alert Generation
Template Engine
Distance Quantization
Direction Mapping
Audio Management
Cooldown System
Priority Queue
Volume Control
Text-to-Speech
TTS Conversion
Audio Synthesis
Language Processing
Audio Output

YOLOv8n (nano) was chosen for its favorable latency/parameter count tradeoff for real-time first-person assistive systems (low GFLOPs and 3.1M parameters) while still providing strong detection accuracy on the curated dataset. The pipeline is currently hybrid: perception is a trainable neural component (YOLOv8n), while distance, temporal smoothing and audio scheduling are implemented as rule-based modules that include **tunable** parameters whose values were set via targeted experiments rather than learning.

4. Training Setup

The following table explains the components and the parameters which are tunable in each component and expected results after fine tuning those parameters.

Component	Type	Tunable parameters (examples)	Notes / Evaluation metric
YOLOv8 detector	Trainable	learning rate, epochs, mosaic, img size	Evaluate mAP50, mAP50–95, per-class AP. (Validation: mAP50=0.836, mAP50–95=0.75).
Distance estimator	Rule-based (calibrated)	DEFAULT_KNOWN_HEIGHT, Estimated_Focal_Length_PX	Evaluate **Distance MAE (m)** against a small ground-truth set
Motion detection	Rule-based (temporal)	MOVEMENT_THRESHOLD_PIXELS, frame window size, smoothing alpha	Evaluate **False Motion Rate / Missed Motion Rate** on labelled motion validation set.
Context generator	Rule-based (templated) / optional LLM	template forms; LLM prompt design	Evaluate **comprehensibility** and **relevance** with user testing or automated BLEU/ROUGE if using LLM.

Audio controller	Rule-based	ALERT_COOLDOWN_GLOBAL, priority rules	Evaluate **Alert Overlap Rate** and **Latency (s)** in recorded runs.

The model was trained using the Ultralytics YOLOv8 framework on a **Tesla T4 GPU**.

- **Loss Functions:** Standard YOLOv8 losses were used:
    - **Class Loss:** Binary Cross-Entropy (BCE) (**cls_loss**)
    - **Box Loss:** CIoU (Complete Intersection over Union) (**box_loss**)
    - **DFL Loss:** Distribution Focal Loss (**dfl_loss**)
- **Evaluation Metrics:** The primary metrics tracked during training were **mAP50** (mean Average Precision at IoU 0.50) and **mAP50-95** (mean Average Precision averaged over IoU thresholds from 0.50 to 0.95).
- **Optimizer:** The framework automatically selected **AdamW** with a learning rate of **0.000119** and momentum of **0.9**.
- **Training Parameters:**
    - **Image Size:** 640x640 (**imgsz=640**)
    - **Batch Size:** 16
    - **Number of Epochs:** 50
- **Training Strategies:**
    - **Warm-up:** 3.0 epochs (**warmup_epochs=3.0**)
    - **Mosaic Augmentation:** Applied for the first 40 epochs (**close_mosaic=10**)
    - **Automatic Mixed Precision (AMP):** Enabled (**amp=True**) for faster training.

5. Hyperarameter Experiments

As noted in the "Overview," experimentation focused heavily on the **application-level pipeline hyperparameters** rather than exhaustive model training experiments. The goal was to refine the raw model output into a stable and useful assistive tool. This process is documented in **Hyperparametertuning.ipynb**.

A "TRUE BASELINE" pipeline was compared against a "TUNED" pipeline. The key parameters explored were:

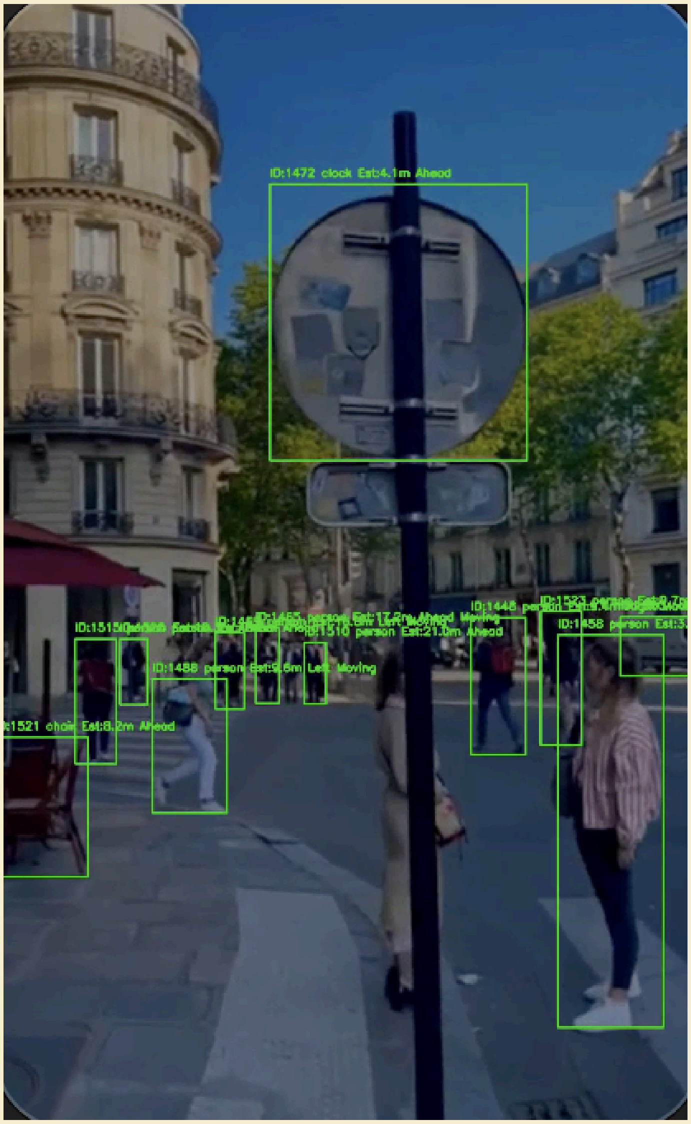
Parameter	**Baseline Value**	**Tuned Value**	**Observation / Justification**
CLASSES_TO_IGNORE	Empty list	List of 30 classes	The baseline model detected many "noisy" or irrelevant classes (e.g., cup, spoon, laptop). A deny list was created to filter these out, focusing alerts on navigation-critical objects.
DEFAULT_KNOWN_HEIGHT	1.5m	2.0m	Tuned to improve the accuracy of distance estimation for objects without a pre-set known height.
ALERT_DISTANCE_OBJECT	5.0m	12.0m	The baseline's 5m alert distance was too short for navigation. This was increased to give the user more advanced warning of objects.
ALERT_COOLDOWN_GLOBAL	0.0s	3.0s	The baseline produced "messy audio" with constant, overlapping alerts. A 3-second global cooldown ensures alerts are clean and distinct.

MOVEMENT_THRESHOLD_PIXELS	5px	25px	The baseline was "twitchy" and triggered alerts on minor camera motion. Increasing the threshold makes the system more stable, ignoring user head jitter.
YOLOv8n training epochs	-	50 epochs	Best checkpoint produced validation mAP50=0.836, mAP50–95=0.75.

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:1448 person
ID:1458 person
ID:1488 person Est:9.6m Left Moving
:1521 chair Est:8.2m Ahead

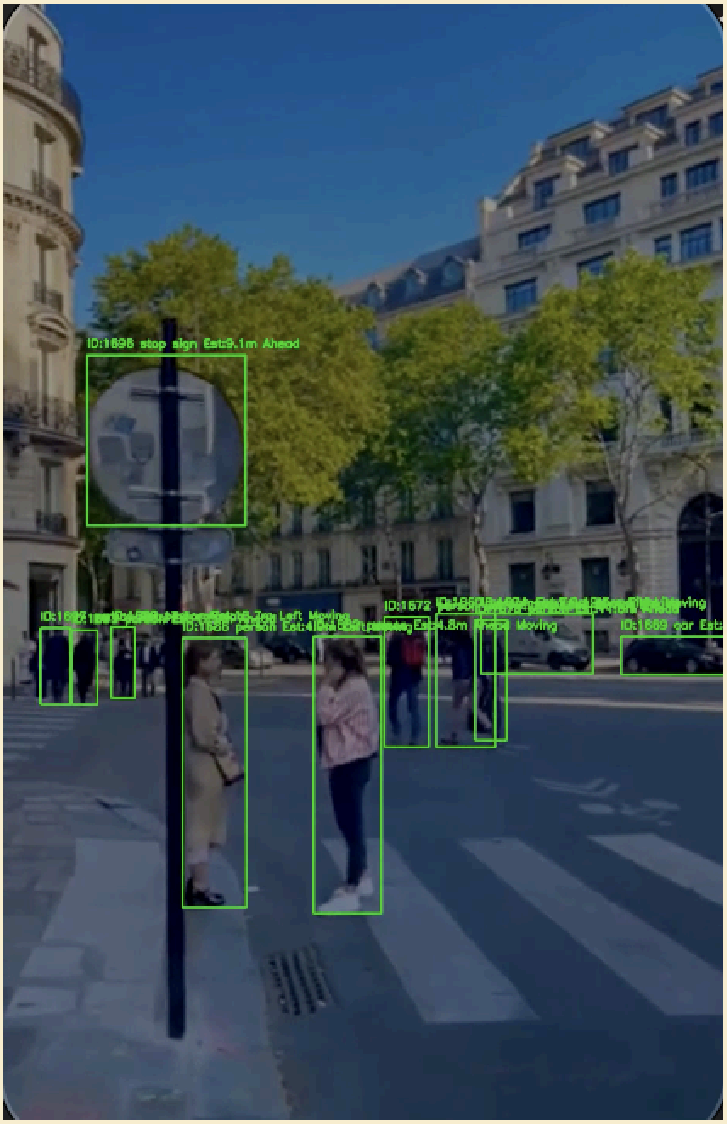


Fig 1.3

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.

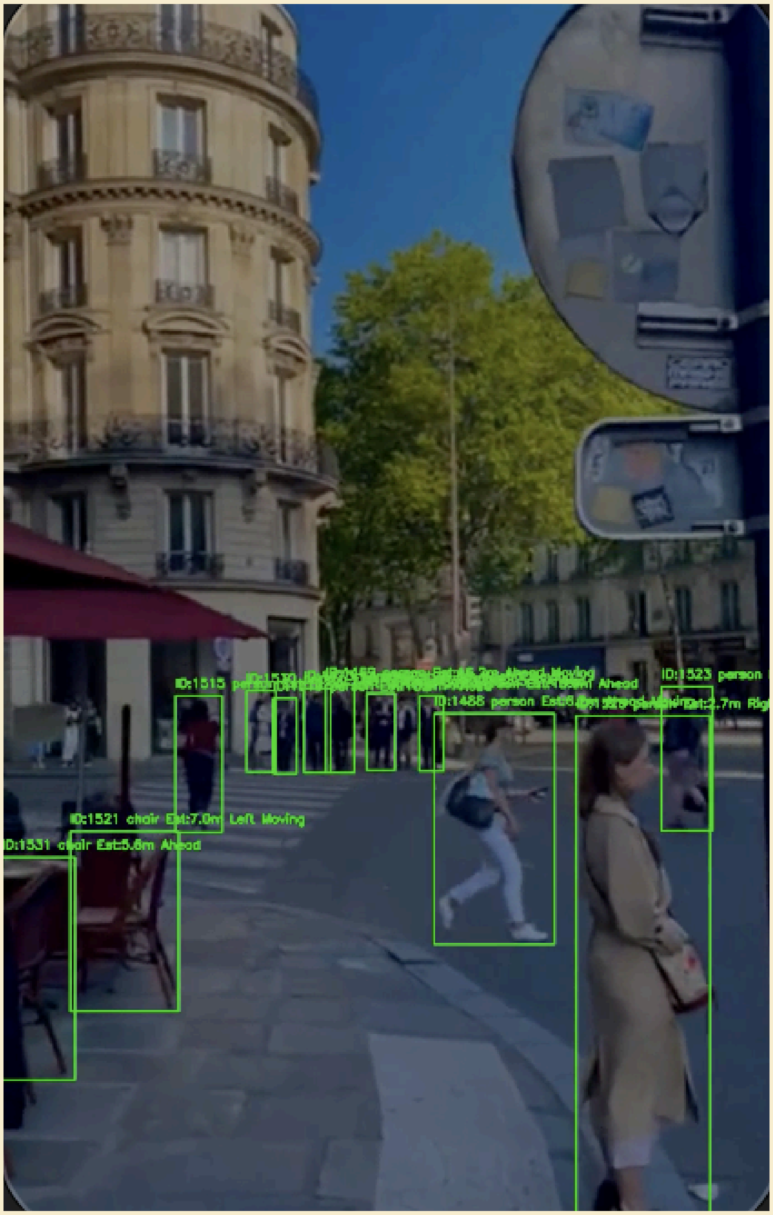

ID:1515 person
ID:1523 person
ID:1488 person
ID:1521 chair Est:7.0m Left Moving
ID:1531 chair Est:5.6m Ahead

Fig 1.4

Baseline run incorrectly labels a static chair "Moving" due to camera motion.

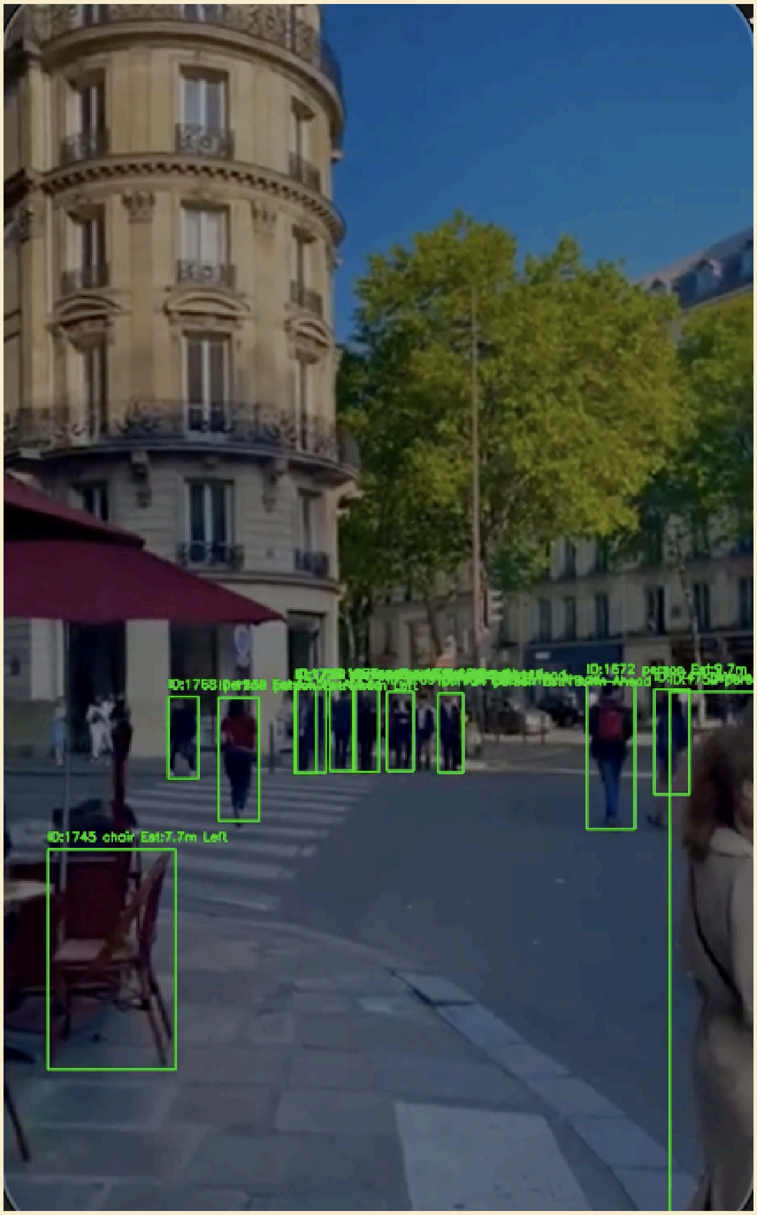


Fig 1.5

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.

Although our main experiments focused on pipeline behavior, we also tested a **training-level improvement** using Optuna-driven tuning. The tuned model improved **mAP50 from 0.836 → 0.89**, validating that our dataset benefits from targeted augmentation and optimizer adjustments.

However — due to time constraints and overall system goals — we retained the **previous 50-epoch detector** as our primary deployable model for pipeline experimentation.

We will integrate this *higher-accuracy tuned checkpoint* into the Motion & Context pipeline evaluation in **Milestone-5**.

6. Regularization & Optimization Techniques

The training process in **Main.ipynb** utilized the built-in capabilities of the YOLOv8 framework.

- **Data Augmentation:** **augment=True** applies a suite of augmentations including mosaic, horizontal flip, scaling, and color space (HSV) adjustments. This helps the model generalize better to varied real-world lighting and object orientations.
- **Normalization:** The architecture heavily utilizes **Batch Normalization** layers after convolutional layers to stabilize training and speed up convergence.
- **Weight Decay:** A weight decay of **0.0005** was applied during optimization as a regularization technique to prevent overfitting.

In addition to the architecture decision previously explained, we performed a **latency benchmark comparison** between YOLOv8n and YOLOv5s on both CPU and GPU devices. The results clearly supported the use of YOLOv8n for our real-time navigation setting.

(refer *scripts/Combined_Hyperparameter_Tuning_and_Feedback.ipynb*)

Model	CPU Inference Time (avg)	GPU (T4) Inference Time (avg)	Result
YOLOv8n	**0.8519 sec/frame**	**0.0723 sec/frame**	Fastest
YOLOv5s	1.9033 sec/frame	0.0816 sec/frame	Slower, higher resource use

YOLOv8n delivers **2.2× faster inference on CPU** and **~12% faster on T4 GPU** while maintaining excellent accuracy (mAP50=0.836, mAP50-95=0.75).

On top of the default YOLOv8 optimization pipeline, we conducted **hyperparameter tuning** using **Optuna search** for learning rate, weight decay, momentum, and core augmentation settings. The search space included:

lr0: 1e-4 → 5e-3 (log scale)
weight_decay: 1e-5 → 5e-4
momentum: 0.90 → 0.96
degrees: 0 → 15
scale: 0.6 → 0.9

The **best trial** converged with **early stopping (patience=3)** based on mAP50, using the following configuration (also stored in `fine_tuned_best_args.yaml`)

Parameter	Best Value	Parameter	Best Value
lr0	0.000119022	translate	0.1349
lrf	0.547097	scale	0.7781
momentum	0.916268	fliplr	0.3881
weight_decay	0.0002035	hsv_s	0.4367
optimizer	SGD	hsv_v	0.5262
degrees	1.25		

This configuration achieved **mAP50 = 0.89**, the highest among our tuning trials.

It confirms improvement over the baseline fine-tuned checkpoint (mAP50=0.836).

Along with the decisions on the training the models and tuning the model parameters we have update the design and architecture of our modules such that it will result in better optimization over compute and inference times to follow the trade offs we followed the below observation and update the design as per the observations to achieve better output

Why Not MiDaS?

- Creates depth maps showing relative depth but not exact distances
- Slow for real-time processing
- Not suitable for precise distance measurements needed for audio alerts

Our Solution: Camera Geometry

- Far objects appear small (low pixel height)
- Close objects appear large (high pixel height)
- Distance formula: distance = (real_object_height × focal_length) / object_height_in_pixels
- One-time calibration: Set focal_length and default heights (e.g., 1.5m for person)
- Result: Fast, numerical distances in meters for clear audio alerts

The Problem with ByteTrack

- Overly sensitive - tracks minor pixel movements
- Reports static object motion from camera shake
- Creates unnecessary alerts for stationary objects

7. Initial Training Results

The model training detailed in **Main.ipynb** completed successfully for 50 epochs.The training log shows the model converged well, with validation losses decreasing steadily.

- **Validation Metrics:** The final validation of the `best.pt` checkpoint achieved excellent performance on the 1,427 validation images:
    - **mAP50: 0.836**
    - **mAP50-95: 0.75**
- **Convergence:** The training and validation loss curves (**box_loss**, **cls_loss**) show successful convergence over the 50 epochs. The validation metrics (mAP50, mAP50-95) peaked at epoch 1, dropped, and then steadily climbed back up, finishing at a very strong 0.75 mAP50-95.
- **Class Performance:** Performance on key classes for this application was strong

Class	**mAP50 (Validation)**	**mAP50-95 (Validation)**
person	0.959	0.871
car	0.910	0.810
bus	0.745	0.702
stop sign	0.840	0.805

- **Qualitative Results:** The **VisionAssist_inference.ipynb** and **Hyperparametertuning.ipynb** notebooks provide qualitative examples of the model's output in the context of the full pipeline. The model successfully identifies objects in a video feed, and the pipeline generates correct, context-aware audio alerts (e.g., "Caution: bicycle, about 4 meters, Ahead."). The baseline-vs-tuned experiments show a clear improvement in the usability of these alerts.

8. Project Repository Structure

```
Group-2-DS-and-AI-Lab-Project/
├── annotations/
│   ├── data.yaml
│   ├── instances_test.json
│   ├── instances_train.json
│   └── instances_val.json
├── docs/
│   ├── Milestone_1.pdf
│   ├── Milestone_2.pdf
│   ├── Milestone_3.pdf
│   └── Milestone_4.pdf
├── results/
│   └── eda/
│       ├── aspect_ratios.png
│       ├── bbox_areas.png
│       ├── class_distribution.png
│       ├── object_aspect_ratios.png
│       ├── object_locations_heatmap.png
│       └── objects_per_image.png
├── scripts/
│   ├── data_loading/
│   │   ├── .gitignore
│   │   ├── Custom Data Collection Script.ipynb
│   │   └── dataset_sample_collection_annotation.py
│   ├── training/
│   │   └── Main.ipynb
│   ├── DSAI_eval.ipynb
│   ├── EDA_MS_COCO.ipynb
│   └── Hyperparametertuning.ipynb
├── DATA_GOVERNANCE.md
└── README.md
```

9. Model Artifacts

- **Model Checkpoint:** The best trained model weights are saved as **yolov8n_custom_coco_best.pt** and stored in Google Drive (**/content/drive/My Drive/VisionAssist-Models/**)

- **Training Scripts:** The **Main.ipynb** notebook contains the complete script used for data preparation, splitting (70/20/10), and model training.
- **Inference & Experiment Notebooks:** **VisionAssist_inference.ipynb** contains the final inference pipeline code. **Hyperparametertuning.ipynb** contains the code used for pipeline experimentation and comparison.
- **Logs:** Full training and validation logs, including metrics per epoch and per class, are available in the output cells of the **Main.ipynb** notebook.

10. Observations / Notes for Next Milestone

- **Key Observation:** The initial training results are very promising. The YOLOv8n model provides strong object detection performance (0.75 mAP50-95) on our custom dataset. The most critical finding from this milestone is that the **raw model output is not directly usable** for an assistive application. The **pipeline hyperparameter tuning** (cooldowns, motion thresholds, class filtering) documented in Section 5 was essential to create a stable and non-overwhelming user experience.
- **Issues / Next Steps:** While the current tuned pipeline is highly effective, it may still require some final small changes or fine-tuning to its parameters before a formal evaluation.
- **Plan for Milestone 5 (Model Evaluation & Analysis):** The next milestone is dedicated to formally evaluating the system. The plan is to:
    1. Run the tuned inference pipeline on the **10% unseen test set** (714 images), which was created in **Main.ipynb** but not used for training or validation. This will provide unbiased performance metrics.
    2. Provide a detailed **error analysis** on these test results to identify specific limitations (e.g., common objects it misses, distance estimation errors, or situations where the pipeline logic fails).
    3. Discuss the system's overall **limitations and possible improvements** in preparation for real-world user testing.
    4. Perform the motion and distance calibration experiments described above and populate the parameter metrics table.
    5. Prepare a small user trial with 3-4 visually impaired volunteers for subjective evaluation of alert clarity, response time and usability. This will

enable combining automated metrics and real-user feedback to prioritize changes before deployment.

Closing Notes:

We have extended Milestone-4 to clarify system architecture, distinguish trainable vs rule-based components, and (critically) to provide **reproducible and empirical tuning procedures** for pipeline parameters(in progress). The YOLOv8n detector training details and validation metrics are unchanged and are cited above; however, the document now explicitly demonstrates how the non-neural modules will be systematically optimized and evaluated in Milestone-5 — including dataset collection, grid search strategies, and metric definitions — so that every tuned parameter can be justified quantitatively rather than by observation.

The document includes design and implementation changes and exclusion of some of the already planned and prospective technologies in the previous milestones and we have following justifications over the decisions made for the same

Declaration of Authorship & Review

We hereby declare that this submission is the original work of the project team. We have personally reviewed and approved the document for submission.

Declaration of Authorship & Review

Member	Status
Tanuja Nair	Approved
JIVRAJ SINGH SHEKHAWAT	Approved
BALASURYA K	Approved
PRASHASTI SARRAF	Approved
Karan Patil	Approved