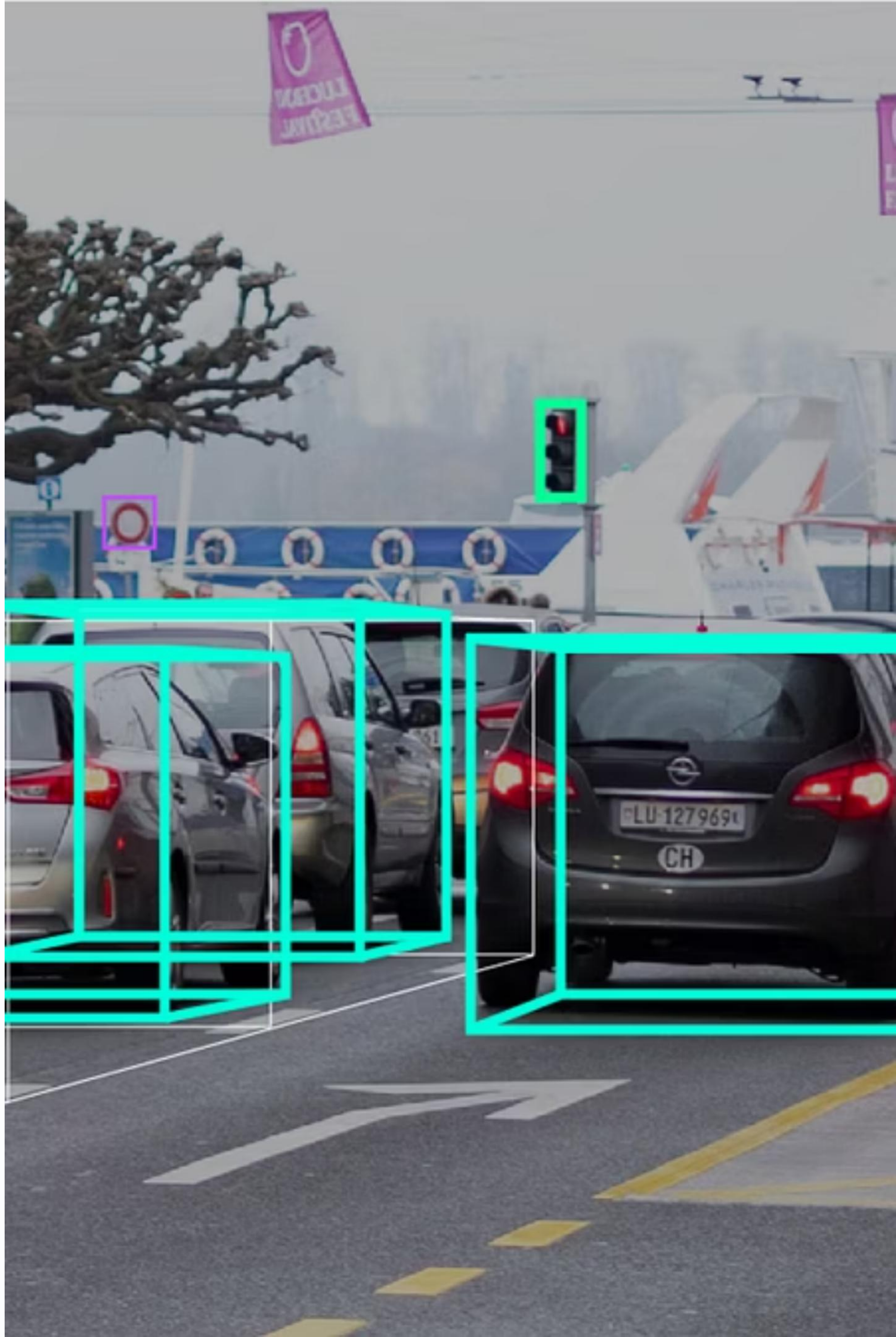
Real-time vehicle and pedestrian detection

Developing a **robust, real-time detection system** to improve road safety and reduce pedestrian-vehicle collisions in dense urban environments. System targets **live alerts, accurate localization**, and lightweight deployment for in-vehicle and roadside units.



Problem background

The system has four core goals:

- **Enhance driver awareness** by detecting pedestrians and nearby vehicles in real time.
- **Reduce accident risk** via **distance-based warnings** and prioritized alerts.
- Deliver a lightweight, **deployable solution** using deep learning optimized for real-time inference.
- Provide a **foundation for ITS research and autonomous driving assistance modules.**

Baseline approach

Design and implement a real-time detection pipeline combining:

Detector

YOLOv9 for high-speed object detection with class and bbox outputs.

Distance Estimation

Monocular pinhole model for per-object distance approximations.

Alerts

Distance thresholds generate prioritized driver warnings and logging.

Baseline model: YOLOv9 — do we really need to fine-tune it?

Domain adaptation

Learn road geometry, reflections, and low-light artifacts present in dashcam footage.

Class specialization

Focus on relevant road classes (person, car, bus, truck, bike, motor) to reduce confusion.

Fine-tuning benefits — continued

Localization gains

Improved bounding-box regression for small, distant, or partially occluded pedestrians — better distance inputs.

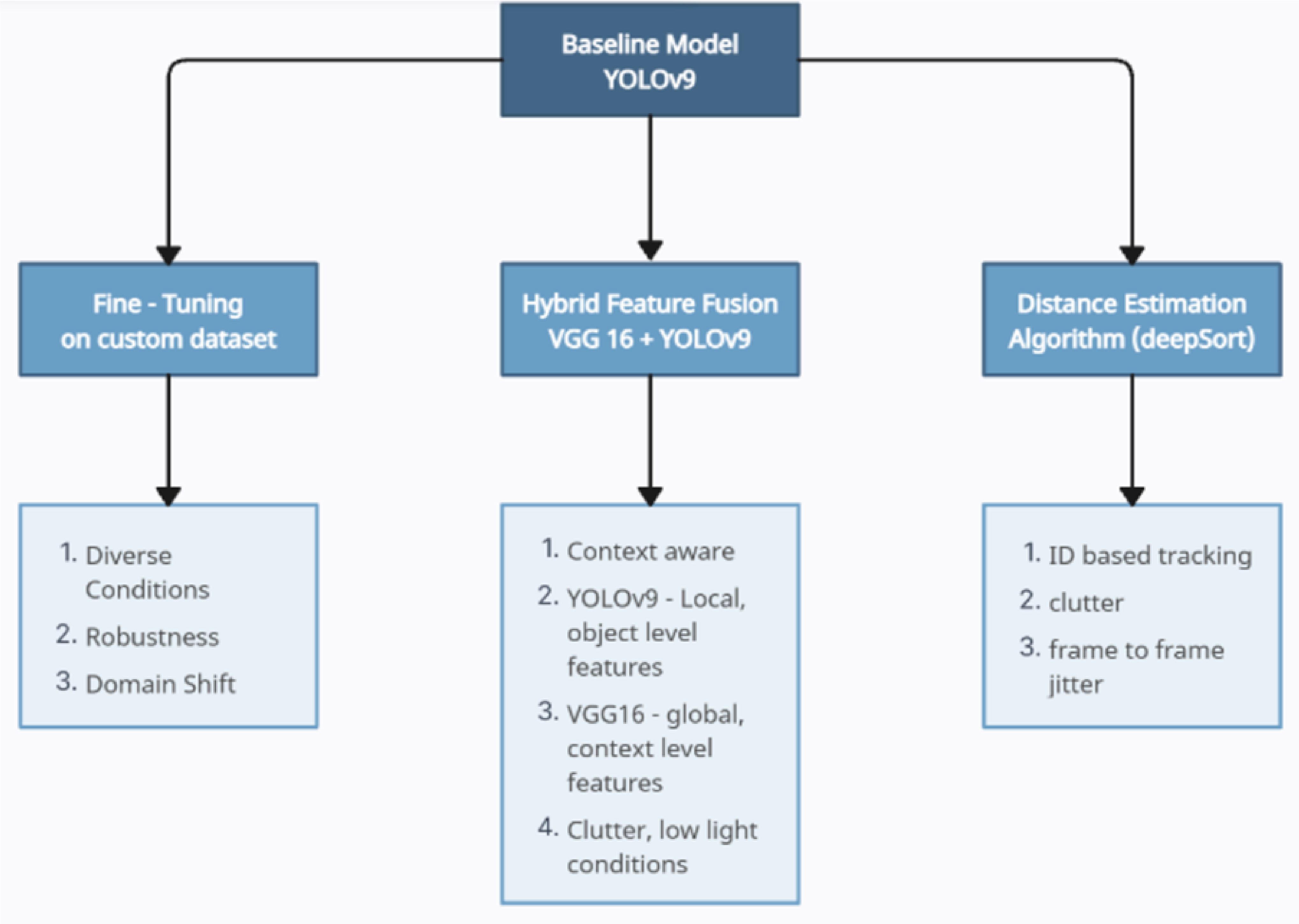
Reduced domain shift

Models tuned on BDD data generalize more consistently across weather, time-of-day, and city scenes.

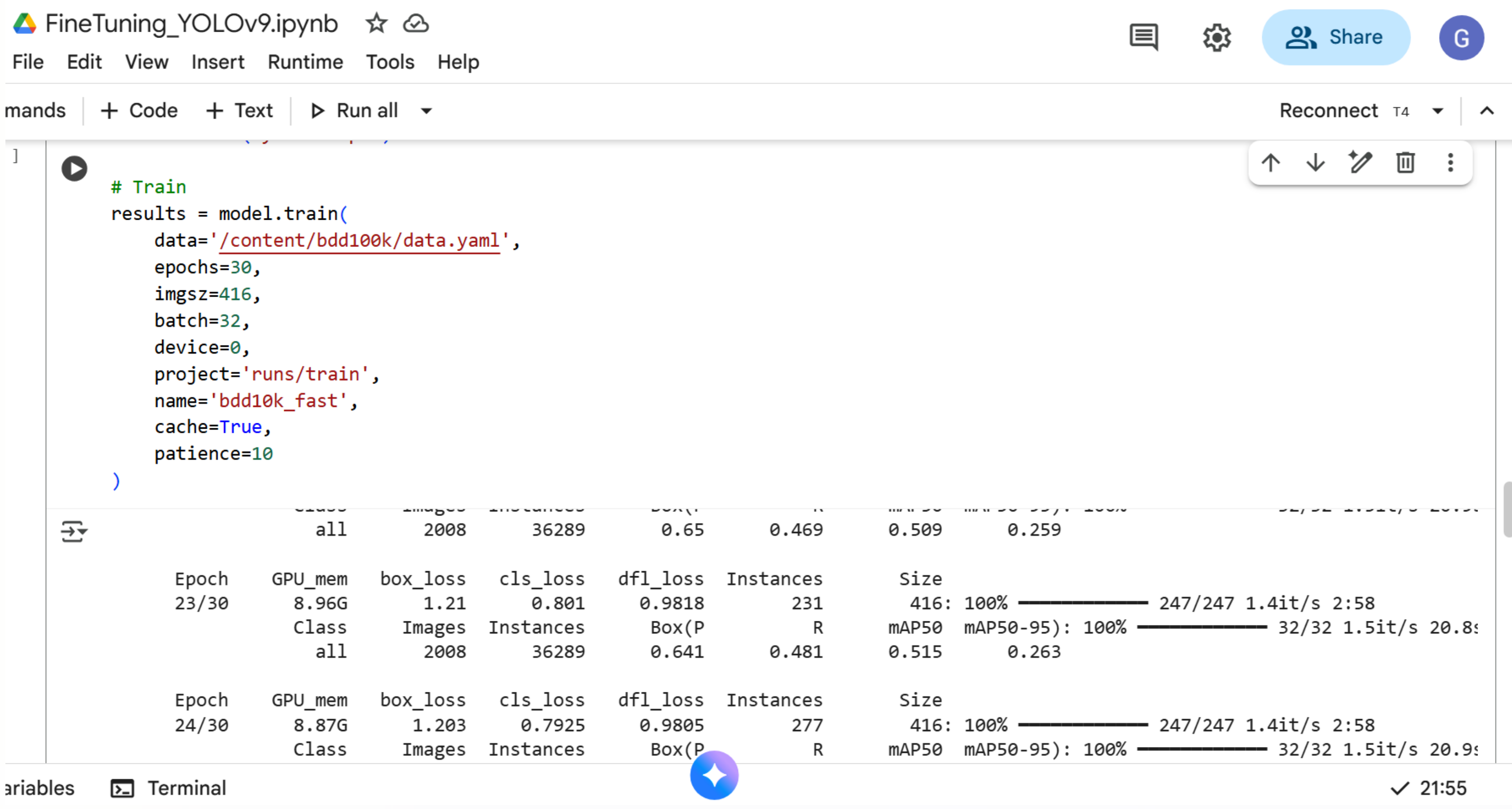
Faster adaptation

Road-aware weights speed up transfer learning for lane detection, signage, and tracking tasks.

3 – Phase Approach for improving the Baseline Yolov9 model



Fine tuning Yolov9 using the BDD10k Dataset




```
# Train
results = model.train(
    data='/content/bdd100k/data.yaml',
    epochs=30,
    imgsz=416,
    batch=32,
    device=0,
    project='runs/train',
    name='bdd10k_fast',
    cache=True,
    patience=10
)
```

```
                 all       2008      36289       0.65      0.469      0.509      0.259

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      23/30     8.96G       1.21      0.801     0.9818        231        416: 100% ━━━━━━━━━━━━ 247/247 1.4it/s 2:58
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 32/32 1.5it/s 20.8s
                 all       2008      36289      0.641      0.481      0.515      0.263

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
      24/30     8.87G      1.203     0.7925     0.9805        277        416: 100% ━━━━━━━━━━━━ 247/247 1.4it/s 2:58
               Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 32/32 1.5it/s 20.9s
```

```
Validating /content/yolov9/runs/train/bdd10k_fast/weights/best.pt...
Ultralytics 8.3.214 🚀 Python-3.12.12 torch-2.8.0+cu126 CUDA:0 (Tesla T4, 15095MiB)
YOLOv9c summary (fused): 156 layers, 25,322,332 parameters, 0 gradients, 102.3 GFLOPs
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100% ━━━━━━━━━━━━ 32/32 1.3it/s 24
                   all       2008      36289      0.672      0.491      0.537      0.276
                   car       1987      20806      0.739      0.672      0.725       0.46
            pedestrian        663       2880      0.592      0.432      0.462      0.214
         traffic light       1130       5480       0.67      0.373      0.422      0.152
          traffic sign       1643       7123      0.686      0.486      0.539      0.278
Speed: 0.1ms preprocess, 6.7ms inference, 0.0ms loss, 1.2ms postprocess per image
Results saved to /content/yolov9/runs/train/bdd10k_fast
```

Fine tuned Model Performance:
YOLOv9: mAP50 = **53.7%** on BDD10K dataset
Baseline YOLOv9: mAP50 = **~67%** on COCO

Important Context:
Lower mAP50 compared to COCO baseline is **expected and normal** because:

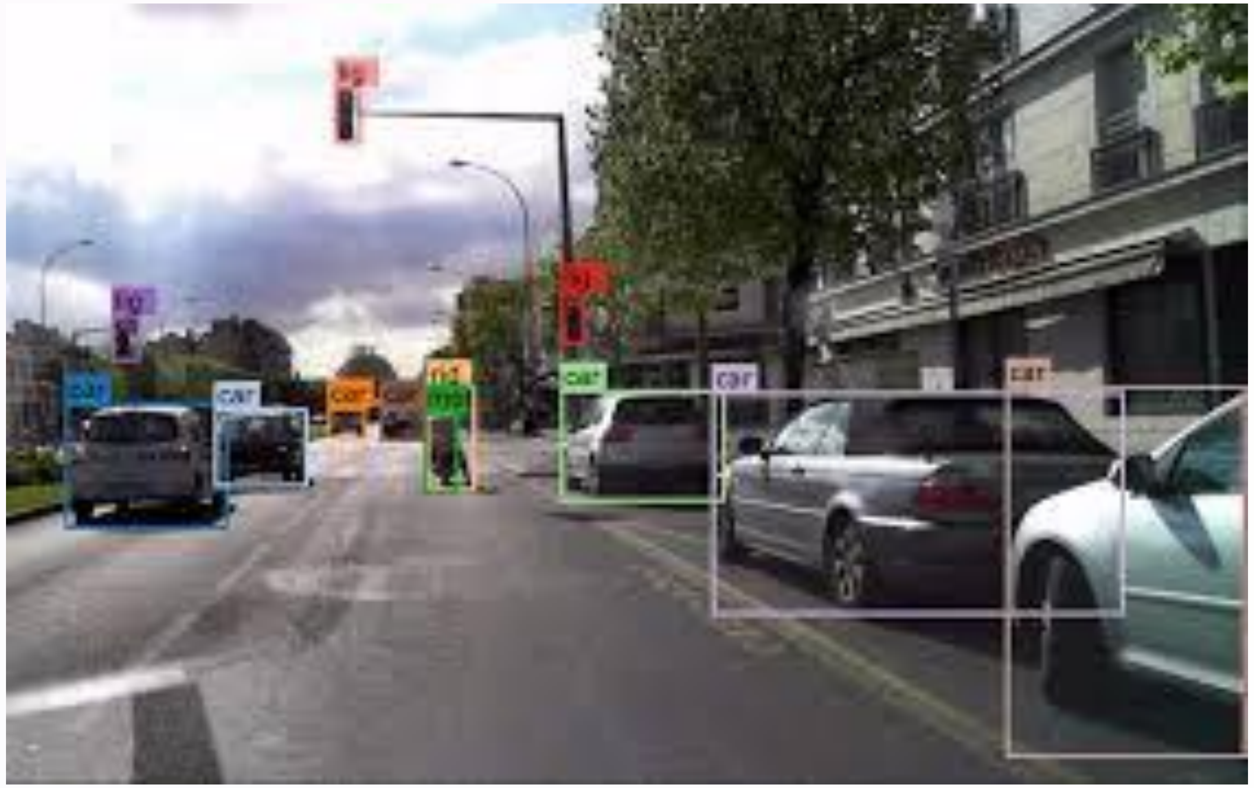
1. **BDD10K is harder** - real-world driving scenarios with occlusions, weather conditions, varying distances
2. **Different dataset** - COCO has cleaner, more varied training data
3. **Specific domain** - Traffic detection is more challenging than general object detection
4. **BDD10k's 5 classes** vs COCO's 80 classes

Dataset: BDD100K & BDD10K

Berkely DeepDrive datasets — curated for realistic driving scenarios and multi-task perception.

BDD100K

100K dashcam videos across cities, times, and weather; supports detection, segmentation, and tracking.



BDD10K

10K-image subset for practical fine-tuning and validation with consistent annotation schema.

Distance estimation algorithms

Current method: monocular pinhole camera model. Assumptions and tradeoffs:

- **KNOWN WIDTH: real-world width of the object (meters).**
- **FOCAL LENGTH: camera focal length in pixels.**
- **BOX WIDTH: width of the detected object's bounding box in pixels.**

$$\text{Distance} = \frac{\text{Known Width} \times \text{Focal Length}}{\text{Apparent Width in Pixels}}$$

1. **Assumptions:** known real-world width, object roughly perpendicular to camera, lens distortion is already corrected.
2. **Pros:** simple, fast, easy to implement on edge hardware.
3. **Cons:** sensitive to pose, occlusion, and inter-subject width variation (pedestrian postures).

Algorithm — deepSORT tracker

Why deepSORT is a pragmatic choice for ITS:

Consistent tracking

Maintains identity across frames enabling stable distance and speed estimates.

Occlusion handling

Re-identifies objects after short losses using motion + appearance features.

Real-time ready

Efficient Kalman filter + lightweight CNN embeddings suitable for live monitoring.

Enables analytics

Supports trajectory, speed profiling, safety-zone violation detection, and per-object smoothing.

Next steps — Improvements

1. Integrate **front-end UI and back-end pipeline** to stabilize frame ingestion and buffer handling, **improving the speed and efficiency**
2. Benchmark **alternate distance estimators** like deepSORT and quantify error vs range.
3. Iterate **3 - phase plan** for **improving the model accuracy** over and above the baseline model