Vehicle Detection System – Technical Report

Model: YOLOv10 (Base)

Executive Summary

We developed a real-time vehicle detection and classification system leveraging the **YOLOv10 (Base)** model, managed through a user-friendly Streamlit GUI. The system is optimized for high-performance deployment on dual NVIDIA T4 GPUs, achieving a detection rate of **≈93 FPS** across 11 vehicle classes with high accuracy (**mAP@50 ≈0.752**). This makes it an ideal, production-ready solution for high-efficiency traffic monitoring and intelligent transportation applications.

YOLOv10 was selected for its superior balance of speed and accuracy, essential for real-time applications:

- **Real-time Performance:** Its single-stage architecture ensures low latency (approximately 7.7 ms/image on dual T4).
- **High Accuracy:** The model features improved feature extraction over its predecessor (YOLOv8), making it highly effective for challenging targets like small vehicles (motorcycle) and complex shapes (truck).
- **Efficiency:** A streamlined, optimized backbone and a reduced parameter count (≈11.1M) enable faster inference without sacrificing performance.
- **Multi-scale Detection:** The architecture effectively handles a wide range of vehicle sizes, from large buses to small motorcycles in dense traffic.

2.2 Chosen Variant Details

- **Model:** YOLOv10 (base / medium-sized variant)
- **Parameters:** ≈11.1M
- **Input Size:** 640×640 (Can be configured to 416×416 for maximum speed.)
- **Architecture:** Optimized CSP backbone, enhanced PANet neck, and an anchor-free detection head.

2.3 Alternatives Considered

Model	Pros	Cons	Decision
YOLOv8	Stable, good car detection	Slower, less accurate on small vehicles	Not chosen
Faster R-CNN	High accuracy (two-stage)	High latency, non-real-time	Rejected
EfficientDet	Good efficiency	Complex pipeline, harder to tune	Rejected

RetinaNet	Strong on rare classes	Slower than YOLO variants	Rejected

3. Tracking Algorithm (Optional)

Vehicle tracking is implemented optionally to maintain object identity across frames.

- **Recommendation: DeepSORT:** We recommend this appearance-based tracker for its robustness in re-identifying vehicles after temporary occlusion. It works seamlessly with YOLO bounding boxes and maintains stable IDs with only a minimal FPS drop (≈5–10%).
- **Alternative: ByteTrack:** A lightweight, motion-only tracker best suited for extreme high-FPS scenarios, though less reliable under long occlusions.

DeepSORT Implementation Snippet:

```
from deep_sort_realtime.deepsort_tracker import DeepSort

tracker = DeepSort(max_age=30, n_init=3, nn_budget=100)
tracks = tracker.update_tracks(detections, frame=frame)
```

4. Optimization Summary

The system's high performance is due to several key optimizations:

- **Dataset Expansion:** The dataset was significantly increased from 370 to 1,110 images using augmentations (motion blur, cutout) to improve robustness, particularly against blur and occlusion.
- **Model Architecture:** The chosen YOLOv10 variant (≈11.1M parameters) and its multi-scale feature extraction effectively balance accuracy and speed across all vehicle sizes.
- **Training:** Optimal training was achieved at **100 epochs** (best mAP with no overfitting). The deployed model is named `yolov10.pt`.
- **Hardware/Computation:** Inference on dual NVIDIA T4 GPUs delivers a total speed of **≈93 FPS**, calculated from a per-image processing time of ≈10.7 ms (≈7.7 ms for YOLOv10 + ≈3 ms for pre/post-processing).

5. Performance Analysis5.1 Speed and FPS

- **Total Processing Time:** ≈10.7 ms per image.
- **Frame Rate: ≈93 FPS**.
- **Conclusion:** This capability is fully real-time and ideal for live traffic monitoring.

5.2 Detection Accuracy (Test Set)

Metric	Value
Precision	0.732
Recall	0.667
mAP@50	**0.752**
mAP@50–95	0.552

- **Highlights:** We observed high mAP@50–95 for large vehicles (Bus: ≈0.615 / Truck: ≈0.603) and excellent precision for small objects (Motorcycle: ≈0.858).

5.3 Resource Efficiency

- **Model Size:** ≈11.13M parameters, ≈28.5 GFLOPs.
- **GPU Usage:** The system demonstrates efficient VRAM utilization on dual T4s, suitable for a moderate-resource real-time deployment.

6. Setup Instructions6.1 Core Dependencies
ultralytics>=8.0.0
opencv-python>=4.8.0
pillow>=10.0.0
torch>=2.0.0
torchvision>=0.15.0
numpy>=1.24.0
streamlit>=1.28.0
Deep-sort-realtime

6.2 Installation Steps

1. **Clone & Navigate:** `git clone https://github.com/arnab9961/vehicle_detection.git`; `cd vehicle_detection`.
2. **Environment:** Create and activate a virtual environment: `python -m venv venv`; `source venv/bin/activate`.
3. **Install:** Install all required dependencies: `pip install -r requirements.txt`.
4. **Model Weights:** Ensure `yolov10.pt` (or current weights file) is placed in the project root.
5. **Run:** Launch the application: `streamlit run app.py`.

7. Application Features (Streamlit GUI)

The user interface provides comprehensive control and visualization:

- **Image Detection:** Drag-and-drop image upload, real-time bounding box display, and downloadable results.
- **Video Detection:** Custom video upload, frame-by-frame updates, annotated video output, and aggregated statistics.
- **Configuration:** An adjustable panel for confidence and IoU thresholds (0.0–1.0), and model path management.
- **Statistics Dashboard:** Displays total detections, class-wise counts, and performance metrics.

8. Model Training Details8.2 Training Configuration (`data.yaml`)

```
train: ./dataset/images/train
val:   ./dataset/images/val
test:  ./dataset/images/test
nc: 11
names: [
  'Auto Rickshaw', 'Cycle Rickshaw', 'CNG / Tempo', 'Bus',
  'Jeep / SUV', 'Microbus', 'Minibus', 'Motorcycle',
  'Truck', 'Private Sedan Car', 'Trailer'
]
```

8.3 Training Command

```
from ultralytics import YOLO
model = YOLO('yolov10m.pt')
results = model.train(
    data='data.yaml',
    epochs=100,
    imgsz=640,
    batch=16,
    device=0, # GPU
    patience=20,
    project='vehicle_detection',
    name='yolov10_vehicles'
)
```

8.4 Key Hyperparameters

Parameter	Value
Learning rate	0.01
Momentum	0.937
Weight decay	0.0005

Batch size	16
Image size	640
Epochs	100

9. Future Enhancements

Our plan for further development includes:

- **Tracking Integration:** Implementing the full DeepSORT/ByteTrack pipeline with trajectory visualization.
- **Advanced Analytics:** Adding speed estimation, traffic flow analysis, and heatmaps.
- **Multi-camera Support:** Developing cross-camera tracking capabilities and a centralized dashboard.
- **Edge Deployment:** Optimization and documentation for low-resource platforms like NVIDIA Jetson and Raspberry Pi.

10. Conclusion

The final YOLOv10-based vehicle detection system successfully delivers on the goals of real-time performance (≈93 FPS) and strong accuracy (mAP@50 ≈0.752). Its robust performance and efficient design confirm its status as a production-ready solution for complex intelligent transportation challenges.