AI Field Application Engineer (FAE) - Technical Assessment Report

Project: Edge AI Object Detection System

Author: Selahattin Sina Bekmezci - Date: 12 December 2025

1. Executive Summary

This report summarizes the development and optimization of a high-performance Edge AI Video Analytics System. The objective was to engineer an end-to-end pipeline covering model training, hardware-specific optimization, and deployment on edge devices.

The project successfully leveraged **NVIDIA TensorRT (INT8 Quantization)** to optimize a YOLOv8 model, achieving a throughput of **147.6 FPS** on an NVIDIA RTX 3060 Laptop GPU. **Despite using the constrained COCO128 dataset (128 images) for rapid prototyping**, the model demonstrated strong convergence and detection capability, proving the efficacy of the implemented training and augmentation pipeline.

2. Model Training Performance

The model was trained on the **COCO128** dataset, a small subset of COCO used here to validate the training-to-deployment pipeline. A custom training strategy was employed to maximize feature learning from this limited data.

- **Architecture:** YOLOv8 Nano (v8n)
- **Dataset:** COCO128 (128 Images - Subset)
- **Training Strategy:** 10 Epochs with Cosine Learning Rate Scheduler and Automatic Mixed Precision (AMP).
- **Data Augmentation:** Strong augmentations were applied to prevent overfitting, which is a critical risk with small datasets:
    - **Mosaic:** 1.0 (Enabled)
    - **MixUp:** 0.15
    - **HSV Jitter:** Hue(0.015), Saturation(0.7), Value(0.4)

Final Key Metrics (Best Epoch):

Metric	Value	Interpretation
mAP@0.5	0.717	**Excellent convergence given the limited dataset size.** Proves effective learning.
mAP@0.5:0.95	0.441	Strong precision for a rapid prototype model.
Precision	0.675	Indicates reliable positive predictions within the trained subset.
Recall	0.625	Successful object retrieval rate for the validation set.

3. Optimization & Benchmarking Results

The core objective was to optimize the PyTorch model for edge deployment using NVIDIA TensorRT. The tables below present the comparative analysis of different inference backends running on an NVIDIA GeForce RTX 3060 Laptop GPU.

Table 1: Inference Speed Comparison:

Inference Backend	Precision	Batch Size	Avg Latency (ms)	Throughput (FPS)	Speedup vs Baseline
ONNX Runtime	FP32	1	9.88 ms	98.16 FPS	1.00x (Baseline)
TensorRT	FP16	1	7.01 ms	137.55 FPS	1.40x (+40%)
TensorRT	**INT8**	**1**	**6.55 ms**	**147.60 FPS**	**1.50x (+50%)**

Performance Analysis:

1. INT8 Quantization delivered the highest throughput (147.6 FPS), exceeding the real-time requirement (30 FPS) by nearly 5x.

2. Latency: The average latency was reduced to 6.55 ms with INT8, ensuring minimal delay for critical video analytics tasks.

3. Efficiency: Despite the performance boost, GPU Memory usage remained efficient (~740 MB), leaving room for concurrent processes.

4. Real-Time System Architecture

To ensure the system can handle high-FPS video streams without bottlenecks, a Multi-Threaded Hybrid Tracking architecture was implemented.

- Problem: Running the Object Detector on every single frame consumes excessive GPU resources.
- Solution:
    - **Detector (YOLOv8):** Runs once every **30 frames** to correct drift and identify new objects.
    - **Tracker (KCF):** Tracks objects in intermediate frames with very low computational cost.
    - **Fusion Engine:** Calculates **IoU (Intersection over Union)** between Detector and Tracker outputs. If **IoU < 0.5** ,the system detects "Drift" and re-initializes the tracker using the detector's coordinates.

Outcome: This architecture allows the system to maintain high tracking accuracy while significantly reducing the average GPU load.

5. Deployment & API

The final solution is deployed as a RESTful Microservice using FastAPI and Docker.

- **Endpoint `/detect`:** Accepts image payloads and returns standardized JSON responses (Bounding Box, Class, Confidence).
- **Endpoint `/health`:** Provides readiness probes for container orchestration.
- **Endpoint `/metrics`:** Exposes real-time telemetry (FPS, P95 Latency, GPU Usage) for monitoring dashboards.

6. Quality Assurance (Unit Tests)

To ensure system reliability and prevent regressions, a comprehensive Unit Test suite was implemented using `pytest`. The tests cover critical components of the pipeline:

- **ONNX Validation:** Verifies that the exported model supports dynamic shapes (Batch 1 vs. Batch 4).
- **Logic Verification:** Tests the Fusion Engine's IoU calculation and Drift decision logic to prevent tracking errors.
- **System Check:** Validates TensorRT engine loading and Warm-up routines to ensure the inference engine is initialized correctly.

Test Result: All 7 unit tests passed successfully **(7 Passed in 3.91s)**, confirming the mathematical correctness and stability of the pipeline.

7. Conclusion

The technical assessment requirements were successfully met. The project demonstrates a complete MLOps pipeline, transforming a standard PyTorch model into a highly optimized TensorRT engine capable of **147 FPS**. The implementation is modular, testable, and ready for production deployment.