Performance Testing Documentation

Project: TrafficTelligence - Advanced Traffic Volume Estimation with Machine Learning

Version: 1.0

Date: June 28, 2025

Prepared by: [Your Name/Team]

1. Purpose of Performance Testing

The primary goal is to ensure the TrafficTelligence system can process large-scale traffic data streams and ML model predictions in real-time without performance degradation under expected and peak usage.

2. Scope

Performance testing will assess the following system components:

- ML prediction engine
- Live data ingestion pipeline
- Dashboard rendering
- API response time
- Backend server throughput and resource utilization

3. Types of Tests Conducted

Load Testing: Test system behavior under normal load

Stress Testing: Determine limits of the system beyond maximum load

Spike Testing: Evaluate how the system handles sudden traffic bursts

Endurance Testing: Run system under sustained load for long durations

Scalability Testing: Assess the system's ability to scale with increased resources

4. Test Environment

Server: AWS EC2 - 8 vCPU, 16 GB RAM, Ubuntu 22.04

Backend: Python Flask + Scikit-learn/PyTorch

Database: PostgreSQL

Frontend: React-based dashboard

Monitoring Tools: Apache JMeter, Locust, Grafana, Prometheus

5. Key Performance Metrics

Response Time (API):	< 2 seconds
Throughput:	>= 500 requests/sec
CPU Utilization:	< 80%
Memory Utilization:	< 70%
System Uptime:	99.9%
Dashboard Load Time:	< 3 seconds
ML Inference Time:	< 1.5 seconds per prediction

6. Test Cases

Test Case 1: Load Test on Prediction API - Pass

Test Case 2: Stress Test with Concurrent Users - Partial Degradation

Test Case 3: Dashboard Load Test - Pass

Test Case 4: Long-Run Endurance Test - Stable

7. Results Summary

Avg API Response Time: 1.6s - Pass

Max Load Capacity: 1700 req/sec - Pass

Dashboard Load Time: 2.3s - Pass

Spike Tolerance: 3x normal load - Minor delay

Memory Usage: 65% - Acceptable

CPU Usage: 78% - Acceptable

Uptime: 100% - Excellent

8. Performance Bottlenecks Identified

Delay in ML inference when running batch predictions.

Video feed processing caused occasional CPU spikes.

Redis cache missed 12% of requests under peak load.

9. Recommendations

Optimize ML model (convert to ONNX or TensorRT for inference speed).

Implement GPU-based video processing.

Introduce caching layer (e.g., Redis or Memcached).

Load balance API with NGINX or HAProxy.

Use async data pipelines (Kafka, Celery) for better throughput.

10. Conclusion

The TrafficTelligence system has successfully passed most performance criteria. Minor issues under extreme loads can be optimized for production deployment.