Technical Report: Operational Review and Strategic Roadmap for the Python-Based Traffic Monitoring System

1. Executive Synthesis of Project Outcomes and Current Status

The Traffic Monitoring System (TMS) project successfully transitioned from an experimental prototype to a functionally stable application, fulfilling its primary goal of creating a Python-based platform for vehicle detection, persistent tracking, and velocity estimation.[1] The system leverages a robust technical stack including OpenCV for video processing, Ultralytics YOLOv8 for object detection, NumPy for numerical operations, and the SORT algorithm for tracking.[1]

Core Data Points and Analysis

The current functional status indicates a high degree of stability for the core metrics, albeit within carefully defined operational parameters. Vehicle detection is rated as fully functional, and tracking stability has been confirmed, specifically under constrained operational conditions.[1] Speed estimation is currently categorized as working with moderate accuracy, achieved through rigorous algorithmic corrections to object ID persistence and velocity calculation.[1]

A critical finding established during the development phase was the necessity of imposing a system-wide resolution constraint. The TMS achieved optimal performance and consistency by mandating operations at the 1280×720 (720p) resolution.[1] This technical requirement resolved initial challenges related to Region of Interest (ROI) misplacement and

low frame rate performance encountered when attempting to process ultra-high 4K input.[1]

Furthermore, the feature of License Plate Detection (LPD) was formally excluded from the current operational scope.[1] This exclusion was a data-driven decision, following trial implementations that revealed systemic issues across multiple technical layers, including unreliable Haar cascades, framework conflicts with Python 3.13 (YOLOv8 errors), and inconsistent Optical Character Recognition (OCR) results stemming from low plate resolution.[1] The system's finalized functional state represents a successful trade-off between architectural stability and feature breadth.

Systemic Bottlenecks and Metric Fragility

The current stability of the TMS is highly dependent upon the imposed 720p resolution constraint. Initial system designs targeting 4K resolution failed comprehensively, manifesting as errors in ROI definition, target loss during tracking, and unacceptable throughput.[1] The subsequent pivot to 720p directly addressed these issues, stabilizing both tracking performance and core metric reliability. This result suggests that the project’s existing hardware configuration and the current algorithmic pipeline (YOLOv8 paired with SORT) cannot computationally scale beyond 720p for the required real-time analysis of dense traffic. This establishes 720p as the hard performance boundary for the current iteration. Any future mandate to increase resolution, perhaps to 1080p, would require a fundamental redesign of the processing architecture, likely demanding dedicated Graphic Processing Unit (GPU) resources or a transition to significantly more computationally efficient detection algorithms.

The reliability of the core deliverable—speed estimation—remains sensitive, categorized currently as having "moderate accuracy".[1] This metric is intrinsically reliant on two factors: the continuity of tracking identification (persistent IDs) and the stability of the frame rate used for velocity calculation.[1] Moreover, the calculation currently relies on a manual conversion factor to translate distance measured in pixels per frame into a real-world velocity unit. The recommendation to automate this pixel-to-speed conversion [1] highlights a significant vulnerability. The achieved accuracy is highly contingent upon static camera calibration and continuous object ID assignment. Consequently, any environmental perturbation, such as complex lighting changes, minor camera displacement, or momentary tracking fragmentation, will immediately compromise the validity and consistency of the speed metric.

Key Table Integration

The following table summarizes the operational status and constraints of the final system configuration, optimized for 720p throughput.

Table 1: Current Traffic Monitoring System Functional Status (720p Optimized)

System Component	Current Status	Performance Metric	Key Limitation/Context
Vehicle Detection	Fully Functional	High Reliability	Stable across various traffic scenarios.[1]
Vehicle Tracking	Stable Performance	Consistent at 720p	Improved stability achieved through frame resizing and smoothing.[1]
Speed Estimation	Working	Moderate Accuracy	Contingent upon persistent tracking IDs and FPS-based velocity calculation.[1]
License Plate Detection (LPD)	Excluded	Low Reliability (Trial)	Issues encountered at detection (Haar, YOLOv8) and recognition (Tesseract OCR) stages.[1]

2. Architectural Foundation and Technical Stack Deployment

The Traffic Monitoring System was conceived as a modular, Python-centric solution aimed at real-time traffic analysis.[1] The system's success hinges on the robust integration of several

specialized libraries, forming the technical stack necessary for computer vision tasks.

The established architecture relies heavily on OpenCV for foundational tasks, including frame manipulation and subsequent video output handling (via VideoWriter as recommended for results saving).[1] Object detection is executed by Ultralytics YOLOv8, recognized for its speed and accuracy in modern object recognition tasks.[1] The critical task of maintaining object identity across frames—tracking—is handled by the SORT (Simple Online and Realtime Tracking) algorithm.[1] Auxiliary libraries include NumPy for efficient array and numerical computation, crucial for velocity calculation, and TQDM for visualization of processing progress.[1]

It must be noted that from the outset, the project acknowledged a limitation in scope: the high complexity of reliable license plate detection (LPD).[1] Although preliminary trials were conducted, LPD was explicitly recognized as a high-risk feature, paving the way for its eventual exclusion to prioritize core system stability.

3. System Initialization and Environment Hardening Failures

The initial deployment phase encountered significant friction, transforming environmental setup from a standard procedural task into a prolonged troubleshooting phase. These early challenges highlight critical infrastructure risks related to dependency management and component integration that necessitated substantial ad-hoc corrective action.

Addressing Dependency Management Deficits

A primary hurdle was the incomplete provisioning of the execution environment. The analysis confirms that necessary dependencies, including specific modules such as 'skimage' and supporting components for the tracking framework, were not preinstalled.[1] This required the manual installation of all identified dependencies.[1] The necessity of manual installation procedures, rather than relying on a standardized package manager environment list (e.g., requirements.txt or a virtual environment definition), suggests that a robust, centrally version-controlled or containerized environment was not initially enforced, increasing the initial labor expenditure on environment hardening rather than core algorithmic development.

Resolution of Core Component (SORT) Integration Failures

Integrating the SORT tracker presented severe non-standard installation barriers. Initial attempts to install the 'sort' package via the standard Python package manager (pip) failed because the package was not available on PyPI.[1] This forced an immediate deviation from standard deployment protocol.

Furthermore, efforts to compile the component resulted in a CMake Build Error, specifically sighting a missing OpenCVConfig.cmake file.[1] This error typically signals a failure in resolving required build paths for external compiled libraries. To circumvent the compilation complexities, the team made a strategic pivot: abandoning the attempt at a compiled, optimized tracking solution and switching to a local implementation. The final resolution involved using a local 'sort' folder containing the necessary Python modules, effectively utilizing a Python-only implementation of the SORT framework.[1]

Python Versioning and Compatibility Mitigation

The selection of Python 3.13 for the project foundation introduced instability across the stack. The newer Python release generated numerous warnings and minor incompatibilities when interacting with existing library versions.[1] This instability required a system-wide remediation effort where all core libraries were explicitly updated to their latest versions compatible with Python 3.13.[1]

Technical Debt Analysis

The confluence of deployment failures—the inability to use standard installation procedures for SORT, the CMake compilation error, and the compatibility conflicts introduced by Python 3.13—provides strong evidence that the chosen technology stack was either highly experimental or insufficiently validated prior to deployment. Substantial developer time was diverted to resolving infrastructure and integration problems rather than focusing on the primary algorithmic challenges of tracking and speed calculation. The reliance on manual dependency installation and, critically, the use of a local, non-standard 'sort' module folder creates technical debt. This non-standard deployment method significantly increases the

overhead required for system maintenance, replication, and scaling across different developer or production environments.

Key Table Integration

The early developmental hurdles are systematically documented in the diagnostic review below, formalizing the friction points encountered during system initialization.

Table 2: System Initialization Diagnostic Summary

Problem Description	Observed Cause	Systemic Impact	Resolution Implemented
Missing Dependencies (e.g., 'skimage')	Modules were not preinstalled or managed centrally.	Development blocked until manual resolution.	Manual installation of required dependencies.[1]
SORT Package Installation Failure	Package not available on PyPI.	Forced deviation from standard deployment protocol.	Used local 'sort' folder with required modules.[1]
CMake Build Error (OpenCVConfig.cmake)	Missing required build file; required Python-only implementation switch.	Abandonment of compiled/optimized tracking component.	Switched to using Python-only SORT.[1]
Python 3.13 Versioning Issues	Warnings and minor incompatibilities with the new Python release.	Potential for unexpected runtime behavior or instability.	Updated all core libraries to latest compatible versions.[1]

4. Core Video Processing and Performance

Optimization

The most significant architectural decision of the project involved the fundamental change in video input handling, a pivot directly motivated by performance and stability requirements. The analysis confirms that the initial attempt to process ultra-high-resolution video inputs was the central barrier to system functionality.

Frame Resolution Strategy and ROI Calibration

The system was initially designed to handle 4K video input. This high resolution immediately caused a critical failure in display and localization: the Region of Interest (ROI) rectangle, which defines the monitored traffic area, was displayed at an incorrect spatial position.[1] This misalignment rendered the collected data unusable. The corrective action required simultaneously reducing the frame resolution and recalibrating the detection region. Video frames were downscaled to 1280×720, and the ROI center was adjusted relative to the new, smaller frame dimensions.[1] This strategic downscaling proved essential for accurate scene registration.

Mitigation of Data Access Errors

During the implementation of the video reading pipeline, an UnboundLocal Error was observed when the system attempted to access frame dimensions before the initial frame had been successfully read.[1] Although the specific code modification is not detailed in the report, this class of error is resolved by implementing proper sequential initialization: ensuring that variables defining frame geometry (width, height) are only accessed or utilized after a successful frame reading operation, thereby preventing premature variable reference.

Frame Rate Stability and Throughput Improvement

The high computational load imposed by the initial 4K resolution led to unacceptable performance, manifested as a low Frame Per Second (FPS) rate.[1] This low FPS was detrimental

not only to user experience but, critically, to algorithmic stability. Performance improved significantly after the transition to 720p resolution.[1] This improvement solidified 720p as the operational standard, leading to the formal recommendation that this resolution must be maintained for consistent stability and throughput going forward.[1]

Interoperability between Computational Load and Tracking Robustness

The performance optimization provided a direct solution to a major tracking failure. Initial analysis linked temporary target loss in the SORT tracking component to a combination of "low frame skip and high object density".[1] When the system operated at a low FPS (a consequence of the 4K processing bottleneck), the physical distance vehicles traveled between sequential frames increased dramatically. This increased temporal displacement complicated the prediction and data association tasks performed by the SORT algorithm's underlying Kalman filter, leading to tracking fragmentation. The switch to 720p boosted the processing rate, effectively reducing the inter-frame time delta and thus minimizing the displacement, which restored the SORT tracker's ability to maintain target continuity.[1] This demonstrates that frame rate stability is not merely a cosmetic performance metric but a fundamental prerequisite for achieving robust and reliable tracking metrics.

5. Object Persistence and Velocity Calculation Accuracy

The transition from a simple detection system to a functional monitoring system required sophisticated refinements in object identification and metric derivation to ensure that vehicle data could be consistently calculated and attributed.

Tracking Robustness and Target Loss Prevention

As previously noted, initial operational trials suffered from temporary target loss in the SORT tracker, particularly under conditions of high object density.[1] This issue was resolved by

reinforcing the system's stability through frame resizing to 720p and implementing improved smoothing techniques within the tracking pipeline.[1] The successful implementation of these measures resulted in tracking capability being certified as "Stable performance at 720p".[1]

Stabilizing Object Identification for Metric Calculation

A critical system instability was identified in the speed estimation component, where estimated velocities fluctuated erratically.[1] The root cause was the frequent resetting of object IDs.[1] This object ID fragmentation occurs when the tracking algorithm fails to match a known vehicle from frame N to frame $N+1$, forcing the creation of a new track ID and disrupting the historical trajectory data. Since the velocity is calculated based on differential distance ($\Delta Distance$) measured across the track's lifetime (Δt), an ID reset causes the calculation to restart from zero, leading to an immediate speed reading drop followed by an anomalous spike. This volatility rendered the speed metric unreliable. The resolution was the introduction of persistent tracking IDs, which likely involved integrating a secondary identity management layer to smooth over momentary tracking failures, coupled with a stable FPS-based velocity calculation methodology.[1]

Achieving Moderate Accuracy in Speed Estimation

The velocity determination process relies on converting the displacement measured in pixels per frame into a real-world velocity measure.[1] Following the implementation of ID persistence, the system achieved a status of working with "moderate accuracy".[1]

The Algorithmic Necessity of ID Persistence

The severe speed fluctuation observed prior to intervention confirms that the velocity computation is an explicit differential measurement tied to the continuity of the vehicle's identified track. The corrective action—introducing persistent tracking IDs—validates the speed metric by ensuring that trajectory history is maintained even if the base SORT algorithm momentarily loses confidence in the association. This strategic modification

ensures that the system reports stable and validated speed data, rather than reporting noise generated by tracking discontinuity.

The Calibration Gap for Operational Readiness

While accuracy is currently moderate, the system's utility is constrained by its reliance on a manually derived, static conversion factor for translating abstract pixel movement into actionable metrics like kilometers per hour or miles per hour.[1] The report explicitly recommends adding automatic calibration for pixel-to-speed conversion as a future necessity.[1] Since this feature is a recommendation, it is currently absent. This dependency on static calibration represents the greatest threat to the system's deployment viability and scalability. Any physical alteration to the camera setup—including subtle changes in pitch, yaw, height, or lens zoom—would render the fundamental conversion factor invalid, necessitating a manual, error-prone recalibration process. Without the implementation of automated calibration, the TMS cannot be considered fully resilient or scalable for operational use.

6. Analysis of Scoped Feature Exclusion: License Plate Recognition Trial

The decision to formally exclude License Plate Detection (LPD) was not arbitrary but was the result of a rigorous failure mode analysis across three distinct technical stages, providing substantial justification for the reduction in scope.[1]

Inconsistent Performance of Traditional Detection Methods

The initial trial utilized traditional computer vision methods, specifically Haar cascades, for the detection stage of the license plate.[1] This approach yielded unreliable and "flickering" results.[1] Flickering indicates high instability in the detection output, suggesting that Haar cascades exhibited an unacceptable combination of high false-positive rates and poor temporal consistency, making the localized region inconsistent for subsequent processing steps, such

as OCR.

Framework Integration Barriers

A critical, non-algorithmic failure occurred during the integration of modern deep learning detection frameworks. Attempts to leverage the YOLOv8 hub model references resulted in a FileNotFoundError specifically when operating under the constraints of Python 3.13.[1] This finding directly links the planned LPD enhancement to the general instability and versioning conflicts identified during the environment hardening phase (Section 3). The chosen Python distribution proved incompatible with the necessary state-of-the-art detection framework, creating an insurmountable technological barrier for reliable LPD implementation.

OCR Reliability Issues Under Practical Constraints

Even when detection was momentarily successful, the Optical Character Recognition (OCR) stage, utilizing Tesseract, produced inconsistent results.[1] The root cause of the recognition failure was traced directly back to the low resolution of the license plates within the cropped image.[1] This is a fundamental limitation of the captured input quality relative to the OCR engine's requirements.

The Interdependency Conflict (720p vs. LPD)

The failure of LPD highlights a crucial architectural trade-off. The core system optimization required the downscaling of video input to 720p to achieve stable FPS and functional tracking.[1] This optimization, necessary for system success, simultaneously introduced an input constraint—low-resolution license plates—that caused the failure of the OCR component.[1] Therefore, the performance optimization required to make the primary features (detection, tracking, speed estimation) functional created an intractable technical barrier for the tertiary feature (LPD). Achieving both stable traffic monitoring and reliable license plate identification would necessitate either significant capital investment in hardware capable of processing 4K or higher resolutions in real-time, or the deployment of extremely advanced, low-resolution OCR algorithms.

Strategic Pivot to EasyOCR

The assessment of the Tesseract failure led to a specific strategic recommendation for future LPD trials: the consideration of EasyOCR.[1] EasyOCR often employs advanced deep learning models (such as Convolutional Recurrent Neural Networks, or CRNNs) which are generally more robust and effective than Tesseract when faced with degraded, low-resolution, or non-standard text. This recommendation is an acknowledgment of the inherent low-resolution constraint caused by the 720p operation and proposes an algorithmically superior solution designed to operate more effectively within the existing operational envelope.

Key Table Integration

The systemic reasons for the exclusion of LPD are formally justified below, detailing failure points at each stage of the recognition pipeline.

Table 3: Multi-Stage Failure Analysis of License Plate Detection (LPD) Trial

LPD Stage	Problem Description	Observed Cause	Impact on Project
Plate Detection (Traditional)	Haar Cascades were unreliable and flickering.	Low spatial stability/high noise sensitivity of traditional methods.	Inability to localize plates consistently.[1]
Plate Detection (Modern)	YOLOv8 hub model references failed.	Incompatibility between YOLOv8 and Python 3.13.	Feature development blocked by fundamental library conflict.[1]
Character Recognition (OCR)	Tesseract produced	Low resolution of cropped license	Inaccurate metric output, rendering

	inconsistent results.	plates.	feature unusable.[1]
Systemic Status	LPD Excluded	Optimized system resolution (720p) degraded LPD input quality.	Required temporary scope reduction to focus on core stability.[1]

7. Conclusions and Strategic Recommendations

The Traffic Monitoring System has achieved core functional stability through strategic constraint implementation, confirming vehicle detection, tracking, and speed estimation are operational.[1] The stabilization was directly contingent upon resolving severe environment integration failures and performance bottlenecks inherent in high-resolution video processing. The conclusions lead directly to several non-negotiable architectural mandates and a structured, prioritized development roadmap.

Operational Performance Constraints and Maintenance

The 720p resolution must be codified as a non-negotiable architectural constraint for the current system iteration. The formal mandate is to maintain this 1280×720 resolution to guarantee consistent FPS and stability, as proven by the system's performance history.[1] Furthermore, for accountability and data auditability, the process of saving all operational results to video output must be formally implemented using OpenCV's VideoWriter functionality.[1]

Path Forward for Automated Calibration Implementation

The current "moderate accuracy" of speed estimation is fundamentally fragile due to the reliance on static, manual pixel-to-speed conversion. To transition the TMS from a prototype to an operationally resilient system, the highest priority development item must be the "Add automatic calibration for pixel-to-speed conversion".[1] This requires a dedicated research and implementation phase focusing on perspective transformation mapping (homography), which

dynamically calculates the pixel-to-meter ratio based on known scene geometry or reference objects. Implementing this feature will decouple accuracy from physical camera setup, addressing the primary threat to metric validity and ensuring scalability.

Evaluation of Advanced OCR Solutions

The exclusion of License Plate Detection remains justified based on the low reliability demonstrated by Tesseract under the necessary $720p constraints. Future LPD integration attempts must pivot away from legacy OCR methods. The project should prioritize the evaluation of EasyOCR for its comparative robustness in handling low-resolution input.[1] The next LPD trial should be contingent upon successful benchmarking of EasyOCR's performance under the current $720p operational envelope.

Addressing System Scalability and Resiliency

The severe environment hardening friction encountered during initialization (Python 3.13 incompatibilities, manual dependencies, non-standard component deployment) introduced technical debt and compromised deployment predictability. While the current system is stable, its replicability is low. Future efforts must focus on architectural resilience to prevent recurrence of such systemic setup failures. A strategic recommendation is the immediate adoption of containerization technologies (e.g., Docker or Singularity) to encapsulate the environment. This measure will ensure that the specific, working combination of Python 3.13, updated libraries, and the locally integrated SORT tracker is packaged and deployed uniformly, eliminating future environmental versioning conflicts and significantly hardening the system against subsequent maintenance issues.