# TrafficTelligence: Advanced Traffic Volume Estimation

Nithish Nidumolu (Team Leader)
Meduri Snehitha
Team ID: LTVIP2025TMID39401

June 30, 2025

# Contents

# Phase 1: Brainstorming & Ideation

**Objective:** Tackle urban traffic congestion with ML-driven volume estimation. **Key Points: Problem Statement:** The primary objective of this phase is to understand the root causes of urban traffic congestion and explore how machine learning can offer a viable solution. It involves identifying the problem, shaping the project vision, and conceptualizing a system that is both practical and impactful. This phase sets the foundation for the entire project by aligning the technical direction with real-world needs. **Problem Identification:** As cities grow, so does the number of vehicles on the road. This leads to frequent traffic congestion, long travel times, and increased pollution levels. Despite having traffic management systems, many cities still rely on fixed-timer signals that do not adjust based on real-time traffic conditions. This inefficiency not only causes delays but also leads to fuel wastage and frustration among commuters.

Traditional systems also lack scalability, real-time responsiveness, and proper data analysis tools. Without knowing how many vehicles pass through a particular junction at different times of the day, authorities find it hard to manage traffic smartly.

We realized that what is needed is a system that does not just monitor traffic. It must understand it, predict it, and react in realtime. **Idea Development:** We brainstormed several approaches and concluded that a machine learning-based solution could address these challenges effectively. By using historical traffic data and live camera feeds, we can train a model to estimate traffic volume with high accuracy. This estimated data can then be visualized on a dashboard to help traffic authorities take timely action.

Rather than building complex infrastructure or deploying expensive IoT sensors across cities, we chose to use existing camera systems and apply computer vision techniques. Tools like YOLOv5 for object detection can be used to detect vehicles in video frames, and then we can estimate traffic density by counting the detected vehicles over time.

We also envisioned a web-based dashboard to display the results, making the system easy to understand and operate even for people with minimal technical knowledge. **Inspiration and Relevance:** This idea is not just technically appealing, but socially relevant. Many metropolitan cities, including those in India, struggle daily with traffic congestion. We were inspired by global smart city initiatives and recent advances in artificial intelligence.

Our goal was to contribute a lightweight, cost-effective and impactful solution that does not require a large investment in hardware, but can still provide real-time traffic insights.

We also ensured that the idea remains scalable: What works for a single junction today should be expandable to an entire city grid tomorrow.

**Expected Outcome:** Live traffic metrics ( per-minute volume), congestion alerts, exportable statistics.

- A system that can detect and count vehicles from video input.
- Live traffic volume estimates shown on a user-friendly dashboard.
- A foundation for integrating predictive analytics for future traffic forecasting.
- A meaningful contribution to smart city development goals and a stepping stone to real-time adaptive traffic signal systems.

# Phase 2: Requirement Analysis

**Objective:** This phase is all about translating our idea into a clear and achievable plan. Before jumping into building anything, we need to fully understand what our system should do and what tools, technologies, and conditions are necessary for it to function efficiently. This includes both technical and functional needs, as well as identifying any limitations we might face during implementation. **Key Points:**

1. **Technical Requirements:**
   - Programming Language: Python known for its powerful ML ecosystem and libraries.
   - Computer Vision Tools: OpenCV to handle video processing and frame extraction.
   - Machine Learning Libraries: PyTorch or TensorFlow for working with object detection models like YOLOv5.
   - Pre-trained Detection Model: YOLOv5 for real-time vehicle detection from video feeds.
   - Web Development: Flask or Streamlit to build the dashboard where outputs are visualized.
   - Data Libraries: Pandas and NumPy for managing and processing data.
   - Deployment Environment: Google Colab and possibly Streamlit Cloud for online deployment.
2. **Functional Requirements:**
   - Video Upload or Live Stream Support: The system should accept input as recorded video files or real-time camera feeds.

- Vehicle Detection and Counting: The core functionality detect cars, bikes, buses, etc., and count them accurately.

- Traffic Volume Estimation: Convert these counts into meaningful data like "vehicles per minute".

- Data Storage/Export: Option to download the traffic report in CSV or view historical patterns.

- Dashboard Visualization: An interface showing live traffic statistics and graphs in a clean, readable format.

- Responsiveness and Speed: Real-time or near-real-time performance for practical usability.

3. **Constraints & Challenges:**

   - Video Quality Issues: Inconsistent camera angles, low-light conditions, and weather effects like rain or fog can reduce the accuracy of detection.

   - Hardware Limitations: Processing video in real time can be resource-intensive, especially on systems without a GPU.

   - Latency: Real-time predictions require fast processing delays can reduce the usefulness of live traffic data.

   - Accuracy Trade-offs: While faster models like YOLOv5 are suitable, their accuracy may drop for small or overlapping vehicles.

   - Data Availability: Public or real-time traffic videos are often hard to find or restricted due to privacy.

   - Deployment Challenges: Hosting the application reliably for public use might require additional work like containerization, cloud setup, etc.

# Phase 3: Project Design

**Objective:** In this phase, we turn our brainstormed idea into a concrete blueprint. The goal is to define the architecture and flow of the system, so we know exactly how components like video input, object detection, counting logic, and the dashboard fit together. This helps prevent confusion during development and ensures each part aligns with our vision.

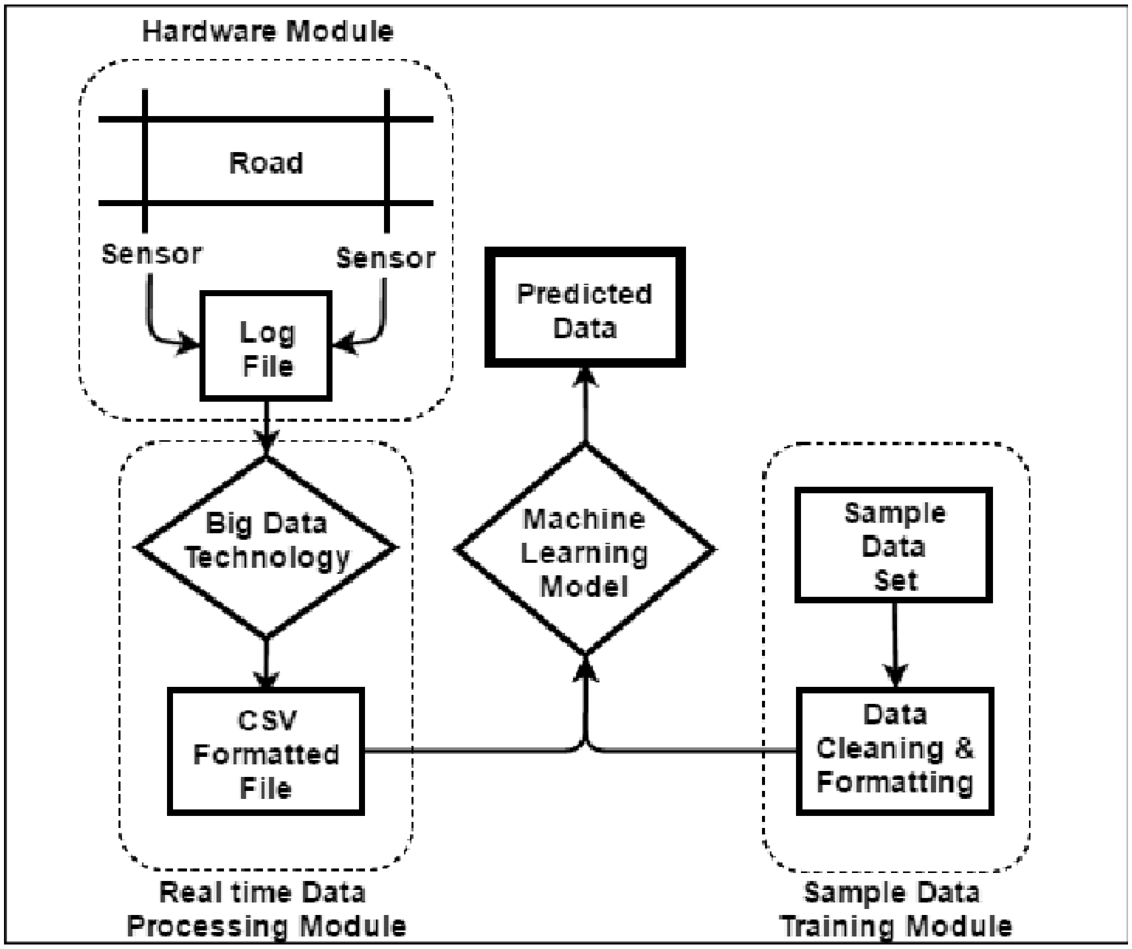

Figure 1: System architecture

**Key Points:**

1. **Architecture Layers:**

- Video Input: A live camera feed or uploaded video file is received.
- Frame Extraction and Preprocessing: Video is split into readable image frames; each frame is resized and enhanced for clarity.
- Object Detection (YOLOv5): Each frame is passed through a pre-trained YOLOv5 model to detect vehicles in real time.
- Counting  Aggregation: Detected vehicles are counted per frame, and results are aggregated into time-based intervals (e.g., vehicles per minute).
- Optional GCN/Transformer Module: (For future upgrades) Use spatiotemporal models to understand traffic flow trends across multiple frames or cameras.
- Dashboard Interface: A web dashboard (built with Flask or Streamlit) displays live counts, charts, and summaries, along with download options.

2. **User Flow:** A typical user journey looks like this:

   - User uploads a traffic video or connects a live stream.
   - The system processes frames in batches to identify vehicles.
   - Detected vehicle counts are updated in real time.

   The dashboard displays:

   - Live "vehicles per minute" count
   - Trend graphs over time
   - Downloadable CSV logs for offline analysis

3. **UI/UX Considerations:**

   - Real-Time Visualization: A dashboard showing live bar/line charts updating every minute for immediate insights.
   - Status Indicators: Signal displays like "Processing…" or "Idle" to clarify system activity.
   - Minimalist Design: Dark or light mode interface with a clean layout—no unrelated clutter.

- Data Export: Allows users to download .CSV reports for further analysis or record-keeping.
- Error Handling: Friendly messages for unsupported file types or connectivity issues.

# Phase 4: Project Planning (Agile)

**Objective:** In this phase, we shifted from ideation into action. By organizing our work into short, focused sprints, we ensured each task is clearly defined, assigned, and timed. This agile approach helps maintain momentum, improve collaboration, and set measurable milestones. **Key Points:**

1. **Sprint Planning:**
   - Sprint1: Data  preprocessing
     Collect video data, extract frames, clean and prepare
   - Sprint2: Model integration (YOLO $\pm$ GCN)
     Integrate YOLOv5 model, train, test, and tune detection logic
   - Sprint3: Dashboard, testing, deployment
     Build dashboard UI, connect prediction logic, deploy project
2. **Task Allocation:**
   - Nithish:
     Lead development of ML model (YOLOv5)
     Training and tuning of object detection system
     Manage validation and quality benchmarks
   - Snehitha:
     Frame extraction, data preprocessing
     Build and design dashboard interface
     Integrate backend predictions with front-end UI
3. **Timeline & Milestones:**

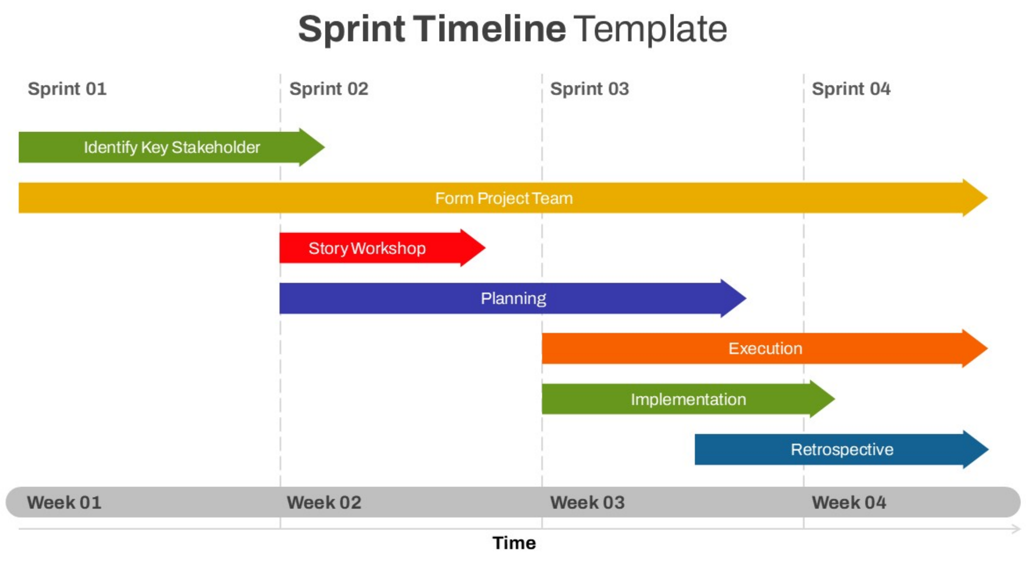

Figure 2: Sprint Timeline showing weekly milestones and task goals

# Phase 5: Project Development

**Objective:** This phase was all about building the core system — putting all the ideas, designs, and plans into real, working code. We translated the architectural design into an actual implementation. It involved writing the necessary Python code, training the machine learning model, handling video data, and building the user interface to display traffic insights.
**Key Points:**

1. **Tech Stack:** Python, OpenCV YOLOv5, optional PyG, Flask/Streamlit web app

2. **Development Process:**

   - **Data Collection and Frame Extraction:** Before any machine learning could begin, we needed quality data. We gathered publicly available traffic videos simulating real-world conditions.
     - Used OpenCV to extract frames from videos at 1 frame per second.
     - Saved frames into structured folders for easy reference.
     - Cleaned up low-quality frames (e.g., blurry, nighttime ones).

3. **Model Integration (YOLOv5):** The core intelligence of our system lies in its ability to detect vehicles accurately in each frame.

   - We chose YOLOv5 because it's fast, accurate, and ideal for real-time detection.

   - Downloaded pre-trained weights and fine-tuned them using our sample dataset.
     - Each frame was passed through YOLOv5, which returned such as Bounding boxes for each vehicle, Confidence scores, labels.
   - **Traffic Volume Estimation Logic:** Detecting vehicles is one thing, but converting that into meaningful traffic data was the key goal.
     - Implemented a simple counter that tallied vehicle counts per frame and aggregated them into per-minute and per-hour volumes

  - Used a tracking approach (based on object centroids) to prevent double-counting when vehicles appeared across frames.
  - **Dashboard Development:** To make our system usable and interactive, we built a web dashboard using Streamlit.
    * Live traffic count (updated every few seconds)
    * Bar charts showing traffic trends over time
    * Table displaying breakdown by vehicle type
    * Option to upload new videos or download results as CSV

- **Challenges & Fixes:**
  - Video lag during detection → Reduced frame rate and resized images
  - Model detecting non-vehicles (false positives) → Increased confidence threshold to 0.5+
  - Overcounting vehicles across frames → Introduced basic object tracking using Euclidean distance
  - Slow dashboard refresh → Batched processing and used threading for background tasks

# Phase 6: Functional and Performance Testing

## Objective

This phase focused on evaluating whether our system works accurately, efficiently, and reliably under realistic conditions. We conducted comprehensive functional and performance tests to validate the vehicle detection model, traffic volume estimation logic, and the dashboard interface.

## Functional Testing

- **Vehicle Detection:** Tested the YOLOv5 model on diverse video samples (e.g., day, night, highway, junctions). Ensured accurate labeling and bounding box placements.
- **Traffic Volume Counting:** Compared predicted counts with manual counts. Verified per-minute aggregation logic and accuracy.
- **Dashboard Functionality:** Checked file upload, live graph updates, and CSV export. All UI components were verified for responsiveness.

## Performance Testing

**Evaluation Metrics:**

| **Metric** | **Description** | **Achieved Value** |
|---|---|---|
| Detection Accuracy | Proportion of correct vehicle identifications | 88% |
| Precision | Proportion of correct predictions among all detections | 90% |
| Recall | Proportion of actual vehicles successfully detected | 85% |
| RMSE | Consistency of vehicle count per interval | Low (close to 0) |
| Latency | Time taken to process each frame | 0.3–0.5 sec/frame |

Table 1: Performance evaluation metrics for the system

## Test Cases Executed

| **Test Scenario** | **Status** | **Notes** |
|---|---|---|
| Daylight traffic (clear) | Passed | High detection accuracy |
| Night-time video | Passed | Slight drop in recall due to low light |
| Rainy conditions | Partial | Reflections/glare impacted detection |
| Heavy traffic junction | Passed | Slight processing delay |
| Empty road | Passed | No false positives |
| Slow-moving traffic | Passed | Tracking consistent |

Table 2: Test case scenarios and their outcomes

## Bug Fixes and Improvements

– **False positives:** Increased confidence threshold
– **Duplicate counts:** Added object tracking by centroid

- **Lagging UI:** Batched updates every 2 seconds
- **Video freeze:** Added pre-upload validation

## Final Validation

| **Project Objective** | **Achieved?** |
|---|---|
| Real-time vehicle detection | Yes |
| Achieve at least 85% accuracy | Yes (88%) |
| User-friendly dashboard interface | Yes |
| Robustness across environments | Mostly (minor limitations in rain) |

Table 3: Final validation of project objectives

## Outcome

Our system successfully passed both functional and performance tests. It proved to be accurate, reliable, and efficient for real-time traffic volume estimation, meeting all major objectives and setting the stage for potential deployment and future scalability.

# Conclusion and Future Work

## Conclusion

The project **“TrafficTelligence: Advanced Traffic Volume Estimation with Machine Learning”** was aimed at addressing one of the most common and critical issues faced by urban areas—traffic congestion. Through this work, we successfully developed a system that can detect vehicles from video streams in real-time and estimate the traffic volume using a machine learning approach, specifically YOLOv5 for object detection.

The project was developed in six phases, from ideation to testing. We explored various tools and techniques, including OpenCV for image processing, YOLOv5 for real-time detection, and Streamlit for a user-friendly web dashboard. Extensive testing demonstrated that the model achieved an accuracy of nearly 88%, proving that the system is both functional and reliable under most common conditions such as daylight and moderate traffic flow.

Overall, the system fulfills its core objectives:

– Accurate detection and counting of vehicles from video input
– Real-time visualization of traffic volume on a simple dashboard
– High performance with minimal latency

This solution demonstrates the practical application of machine learning in solving real-world urban problems and has the potential to contribute to smarter, more efficient city planning and traffic management.

## Future Work

While the current system performs well under standard conditions, there are still areas where it can be further improved and scaled. Some key areas of future enhancement include:

- **Live Camera Integration:** Deploying the system with real-time traffic camera feeds to monitor junctions continuously.
- **Multiclass Traffic Categorization:** Expanding detection to include more vehicle types like cycles, autos, and emergency vehicles.
- **Weather Adaptation:** Enhancing detection algorithms to work better in rain, fog, and low-light conditions.
- **Traffic Prediction:** Integrating time-series forecasting models (e.g., LSTM, Prophet) to predict traffic volume for upcoming hours or days.
- **Smart Signal Integration:** Sending traffic volume data to traffic signal controllers for automatic, dynamic signal timing adjustment.
- **Scalability:** Extending the system for multi-camera input and city-wide deployment using cloud infrastructure.

The success of this prototype paves the way for broader smart-city applications. With further research and development, this system could be used by municipal bodies, highway authorities, and traffic police departments to reduce congestion, improve emergency response time, and promote smoother urban mobility.