

Capstone Design Project Report

Purpose of the report:

To document the design process, implementation, and outcomes of the Smart Traffic Intersection project completed for ECE 3906 and 4905.

Project Title:

Smart Traffic Intersection (99P Labs and OSU ECE)

Team # 11

Team Name: The Traffic Jammers

Team Members: Nick Sanchez, Mohamed Hambouta, Aditya Devnani

Electrical and Engineering Department

College of Engineering

Ohio State University

Course Number 3906

Date: 11/08/24

Document Change Notice

| Date | Change |
|------------|---|
| 11/08/2024 | Chapter 1 Draft Completed and Submitted |
| 11/22/2024 | Chapter 2 Draft Completed and Submitted |
| 01/28/25 | Chapter 3 test plan completed and submitted |
| 4/22/2025 | Chapters 4, 5, & 6 completed and submitted |
| | |
| | |
| | |
| | |
| | |
| | |

Executive Summary

This capstone project was completed in collaboration with 99P Labs and set out to design and build a real-time traffic monitoring system that captures and analyzes data at urban intersections. The team was tasked by the sponsor to create a low-cost, scalable solution that collects valuable traffic data to support future smart city development.

The team built a complete prototype using a Raspberry Pi, high-quality camera, Coral TPU accelerator, and computer vision tools like a YOLO-based minimalized model and DeepSORT. The system captures live traffic footage, detects and tracks vehicles and pedestrians, and displays the live data on a web application. The web application also has a page for analyzing prerecorded video with a predefined dataset. The page comes with a chatbot that can be used to generate smart SQL queries based on the users request to properly fetch requested data.

The final product meets the core goals of the project and provides a strong foundation for future upgrades. A future team should devote their time towards adding a smart storage module, exploring new sensor types like audio or thermal, and continue improving the chatbot's accuracy and usability. This system proves the concept works and gives future developers a pipeline to expand upon and deploy at a smarter traffic intersection.

Table of Contents

| | |
|---|----|
| Chapter 1: Problem Definition and Preliminary Design..... | 5 |
| 1.1 Problem Statement..... | 6 |
| 1.2 Stakeholders and End Users..... | 6 |
| 1.3 Customer Needs, Projects Constraints, and Requirements..... | 6 |
| 1.4 Use of Engineering Standards..... | 7 |
| 1.5 Social, Environmental, and/or Global Issues Impact..... | 7 |
| 1.6 Research and Market Study..... | 7 |
| 1.7 Preliminary Design..... | 8 |
| Chapter 2: Detail Design..... | 9 |
| 2.1 Development of Detail Design..... | 10 |
| 2.2 System and Subsystem Diagrams..... | 10 |
| 2.3 Software Design Process..... | 11 |
| Chapter 3: Prototype and Testing..... | 12 |
| 3.1 Prototype..... | 13 |
| 3.2 Testing..... | 13 |
| Chapter 4: Final Design..... | 15 |
| Chapter 5: User Manual..... | 16 |
| Chapter 6: Project Management, Summary, and Conclusion..... | 20 |
| 5.1 Project Management..... | 21 |
| 5.2 Summary..... | 21 |
| 5.3 Conclusion..... | 21 |

List of Figures and Tables

| | |
|---|-----------|
| Table 1: Customer Needs and Project Engineering Requirement | 1 |
| Figure 1: High level block diagram of the system | 3 |
| Figure 2: Diagram of the onboard system for the Raspberry Pi | 6 |
| Figure 3: Example of a prompt that gets sent to the API on the backend | 8 |
| Figure 4: Visual example of the live page on the frontend, the main feature when launching the application | 9 |
| Figure 5: Final Design | 14 |

Chapter 1: Problem Definition and Preliminary Design

1.1 Problem Statement

Current traffic systems across the nation provide no value or benefits when it comes to collecting data to improve traffic flow and safety. An intersection system capable of capturing data and making decisions based on the collected data provides immense value in both the business sphere and public safety sphere. With recent technological advancements, it is now possible to create a system capable of bringing live, smart data collection to a traffic intersection.

Therefore, the problem brought to the team from 99P Labs was to create a system to analyze the activity of intersections and collect data. By creating this system, future smart cities will be able to build upon the framework laid out by the project to bring never before seen applications and features to traffic intersections. By collecting this data, the team hopes to improve traffic flow and safety for all road users.

1.2 Stakeholders and End Users

The team has three stakeholders and one end user. The first stakeholder, 99P Labs, has given the team this project to develop a system for them that can help them learn more about the data at intersections and their road users, so that they can use this data later. Dr. Ziaeeferd is another stakeholder of the project. The teams' development process and product are a reflection on her course and classroom structure. The capstone team is also a stakeholder. The progress and the product reflect our commitment to the project and also our final grade for the course. The end users of this project are the future users of the Somethings Project from 99P Labs. These users will be able to use our data in their research or any future implementations.

1.3 Customer Needs, Projects Constraints, and Requirements

There were 3 main needs for the system given by 99P Labs. The system needs to be cost efficient, customizable and scalable, and capable of collecting various types of data. From these needs, the team was able to derive quantitative requirements to create the technical scope of the project as shown in **Table 1** below.

Table 1: Customer Needs and Project Engineering Requirement

| Needs | Requirement | Unit | Range (acceptable) | Ideal |
|-------|-------------|------|-----------------------|-------|
|-------|-------------|------|-----------------------|-------|

| | | | | |
|---|---------------------------------|---------|----------------------|----------------------|
| Cost Efficient, Customizable and Scalable | Open-source edge computing | \$ | Free | Free |
| Cost Efficient, Customizable and Scalable | Modular design | \$ | \$100-\$500 | \$400 |
| Diverse Data Collection | Computer vision focus | % | >60% accuracy | >90% accuracy |
| Customizable and Scalable | Embedded system | Modules | >1 module | 2 modules |
| Diverse Data Collection | Real time response | Seconds | <2 second delay | <1.5 second delay |
| Cost Efficient | Minimal storage on edge device | Gb | <5 Gb of data stored | <3 Gb of data stored |
| Customizable and Scalable | Cloud storage and visualization | Gb | >3 Gb of data stored | >5 Gb of data stored |

1.4 Use of Engineering Standards

The most important standards that the group is following are those outlined by IEEE. This is the organization that has created and published the standards for electrical engineering practices. The team has decided to implement the design using a raspberry pi which uses the IEEE 1451.2 standards for design. Using premade components has allowed the team to ensure industry standards are being followed throughout the duration of the project. Another important standard that the team is making sure is followed, is that of privacy of data collection. The standard the team is following is IEEE 1616-2004. This standard is relevant because it refers to specifically data collection of motor vehicles. The team will continue to find additional standards as the process continues to ensure the team is following the IEEE code of ethics.

1.5 Social, Environmental, and/or Global Issues Impact

The device the group is developing does collect data from intersections across the city. This could lead to a potential social impact regarding the wrongful collection of data, but that is not what the device will be doing. Since the device is a camera at a public intersection, anyone is allowed to record the traffic there for data collection. There are no environmental impacts at this time of the device the team is creating. In the future, the device should not create any disruptions in the environment.

Although there is no wrongful collection of data, the storage of large data sets still has risks if left unprotected. For example, a malicious attack may be able to inject faulty data into the large set in order to throw off metrics collected over time. The impact of this discrepancy would be a threat to public safety and company liability. To ensure that the chances of this happening are kept at a minimum, the team plans to store the data in a secure cloud server.

Since the collected data may be used to influence future applications and use cases. The team plans for the long-term scope of collected data to be for a range of 1-2 years. This allows the system to collect enough data to create strong correlations behind any possible metrics while also being short enough to be refreshed and kept up to date with the newest trends in the automobile industry.

1.6 Research and Market Study

To better understand the problem and existing solutions, the team conducted research on several aspects of the project. The main technical research areas included computer vision (CV) and machine learning (ML), data storage solutions, and edge computing. Our research also included market research to identify existing products. The team explored computer vision libraries such as OpenCV and ML frameworks such as TensorFlow, to understand the capabilities of our hardware.

In terms of market research, the team investigated existing traffic management solutions and smart city initiatives. This helped us in identifying the current state of the market. Based on the research findings. The team found a camera-based system which would perform object detection, classification, tracking, and store that data locally, would be the best solution. The data would then be

The team evaluated the different concepts based on cost, scalability and data accuracy. The team initially selected the camera based, raspberry pi backed system for the ease of integration and rapid prototyping. The team would capture camera footage, perform object detection, classification and tacking and store processed data locally. Periodically, the team would completely transmit that data to the cloud for further data analysis. The next round of ideas included additional intelligence-based components, if needed, to gather a diverse data set from different camera locations.

1.7 Preliminary Design

The team decided to use a system that uses a camera to create a solution for the project. By using this system, the camera data can be fed onto a Raspberry Pi. From the Raspberry Pi, the data can then go a variety of routes to undergo processing to make real world applications and use-cases as seen in below:

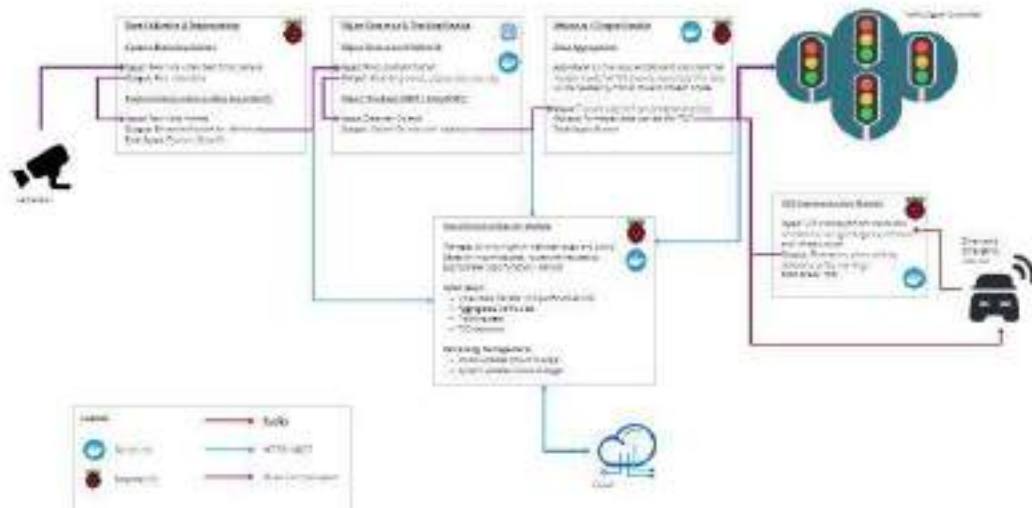


Figure 1: High level block diagram of the system

In the figure above, once the camera footage gets fed to the Raspberry Pi, it is then sent to a computer vision library on-board to extract metrics. The library is loaded on-board in order to minimize latency from API calls over the web. The extracted can be processed on board with simple scripting in Python or C++. This would be most suitable for applications that do not require high performance computing such as simple traffic light control. The metrics can also be sent to the cloud for more powerful processing such as creating heat maps and other visualizers.

To relay the outcome of these processes, the system will also be equipped with a communication module. The Raspberry Pi already comes equipped with Wi-Fi capabilities, but the team is also planning to incorporate a radio communication module in order to communicate with vehicles in a simulation environment. 99P Labs already has a test vehicle built and it is equipped with an identical module so bringing this capability to the system would allow it to fit seamlessly into what 99P Labs has already built. A quick outline of possible use cases through the use of this system architecture can be seen below.

Table 2: Overview of example use-cases

| Use-case title | Processing method | Connectivity method | Input | Output |
|-------------------------------|---------------------|---------------------|----------------|---------------------------------|
| Simple traffic signal control | Raspberry Pi script | Radio module | Camera Footage | Radio signals to control lights |

| | | | | |
|----------|-------|-------|----------------|--|
| Heat map | Cloud | Wi-Fi | Camera footage | A heat map comparing traffic volume to other metrics |
|----------|-------|-------|----------------|--|

References

- [1] "OpenCV." Open-Source Computer Vision Library. Accessed 2023-10-26. [Online]. Available: <https://opencv.org/>
- [2] "Cloud Computing." A model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources. Accessed 2023-10-26. [Online]. Available: https://en.wikipedia.org/wiki/Cloud_computing
- [3] "IEEE Standard for a Smart Transducer Interface for Sensors and Actuators - Transducer to Microprocessor Communication Protocols and Transducer Electronic Data Sheet (TEDS) Formats," IEEE Std 1451.2-1997, 1998.

Chapter 2: Detail Design

2.1 Development of Detail Design

The team was tasked with developing a system to analyze intersection activity and collect traffic data. The software the team is developing needs to perform multiple tasks, including counting vehicles, categorizing vehicle types (e.g., cars, SUVs, trucks), and identifying emergency vehicles. This data also needed to be cross-referenced with various metrics such as weather, time of day, and audio levels to analyze patterns effectively. The initial priority was to develop one test case: comparing the volume of traffic to weather conditions.

After conducting research, the team decided to incorporate software tools such as YOLOv5, Python, and SQL, along with hardware equipment like a Raspberry Pi and a Coral TPU accelerator, to implement the system. YOLOv5 was chosen for its object detection capabilities and its adaptability to various environments. Python and SQL were selected for their compatibility with YOLO, facilitating efficient data collection and storage. Python's ease of API integration also made it an ideal choice for retrieving and comparing weather data. The Raspberry Pi was selected due to its versatility, compatibility with various software and hardware, and suitability for test case implementations. The Coral TPU accelerator was included to enhance the speed and efficiency of data collection using the YOLO framework.

The team decided to collect data initially on a local edge device before migrating the system to a cloud environment. Once the system operates reliably on the local device, further research will be conducted to identify suitable cloud platforms. The ultimate goal is to enable data collection and visualization from any location by accessing the data stored in the cloud.

2.2 System and Subsystem Diagrams

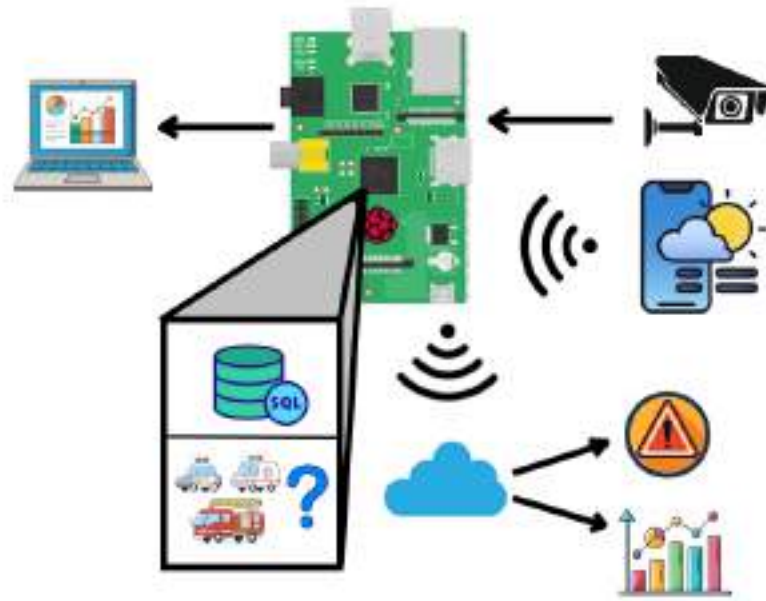


Figure 2: Diagram of the onboard system for the Raspberry Pi

The team based the overall system design on the diagram shown in section 1.7. As for the subsystem of the processes conducted on and by the Raspberry Pi, the overall flow and distribution of processes can be seen in the subsystem diagram in figure 2.

As can be seen in the diagram, the Raspberry Pi takes in camera footage via a wired connection (or prerecorded footage) and weather data through an API call to populate its database on the board as well as make note of any emergency vehicles. After getting these data metrics, the team can then decide to either send them over a wired connection to a laptop for testing or sending over Wi-Fi to the cloud. With cloud connectivity, the data can be visualized similarly to the laptop, and it can be used to make alerts for other devices in the system. These processes are all controlled by the main board which is the Raspberry Pi. While there are other modules like the Coral TPU, these are used purely for computing power and do not determine the flow of data and signals within the system.

2.3 Software Design Process

The team started the software design process by defining the inputs and outputs for the system. The input includes different forms of video feed frames data, and the output is periodically aggregated data saved in a structured format. The AI model architecture selected for object detection is YOLOv11. To track objects across frames, the team opted to use the DeepSORT algorithm. Figure 1 presented in the past chapter portrays some of the different software design components that would be included in the final system.

The core of the system lies in our two sub-modules for detection / tracking. Our YOLOv11 model and tracking algorithm is run inside a docker container. A docker container, in a basic sense, is a script that can be run to boot up a containerized operating system, within our edge device, that will host our AI model and install any required dependencies it needs. This would allow us to abstract our processing software components to “boxes” that expect certain inputs and output certain outputs and separate points of concerns.

For this specific module, our input includes raw camera feed data or YouTube links to live video streams. The system will use libraries like OpenCV and FFmpeg to preprocess and decode the stream into frames. Once it starts running, our module generates a series of periodically aggregated traffic data, including object types (e.g., vehicles, pedestrians), bounding box coordinates, and tracking IDs. This data will be published for later processing/storage. Additionally, metadata such as timestamps and source identifiers shall also be included. Our module also incorporates TPU acceleration to help boost AI model inference time.

This part of the system was not captured in Figure 1 as it was introduced at a later stage of our project. To enhance the system’s adaptability, including a fine-tuning module on our edge device would help improve our system’s accuracy. This module would allow the YOLOv11 model to be fine-tuned on-device using on-site collected data. This training pipeline should theoretically annotate a certain amount of image frames using unsupervised approaches such as Grounding DINO. The new model weights would then get validated and deployed without disrupting current system operations.

The fine-tuning process would rely on a training pipeline that supports incremental learning. It would use a small portion of local resources to retrain the model periodically based on newly annotated data. This approach should allow the system to adapt to specific traffic patterns at different intersections. This module is subject to change depending on the results obtained during prototyping.

The communication and API layer will be our interface between the device and the external world. Our initial approach will include a REST API with various endpoints that will allow our cloud solution to extract data from the edge device as needed.

Other areas that need to be integrated into the system include monitoring / logging and risk mitigation during failures / instability. Key metrics to be logged include frame processing times, detection accuracy rates, system uptime and health status. During prototyping and testing, the team will need to design and integrate risk redundant components or failover solutions to maintain operations in case of edge device failure. The team will also need to research and adhere to GDPR data privacy standards to ensure compliance with data protection laws.

Chapter 3: Prototype and Testing

3.1 Prototype

The prototype works by having two separate servers run locally for the front-end and back-end of the web application. The backend is used to connect to separate modules and run any processing on the Raspberry Pi like the camera, while the front-end is used to visually create and test new features to be added. The overall build of the software went without much trouble; however, the team did run into some issues with the accuracy of the chatbot. To attempt to fix these issues, the team will most likely need to spend more time tweaking the prompts that get sent via OpenAI API calls for better accuracy.

```
summary_prompt = """
You are an assistant interpreting the result of a SQL query on a traffic database.

The database contains:
- Table: traffic
- Columns: frame, id, class, and other tracking data

Instructions:
- ONLY use the result and question.
- NEVER refer to vision, images, or needing context.
- If a number is returned, explain what it represents based on the user's question.

SQL result: {result}
User question: {user_message}
Answer:
"""
```

Figure 3: Example of a prompt that gets sent to the API on the backend

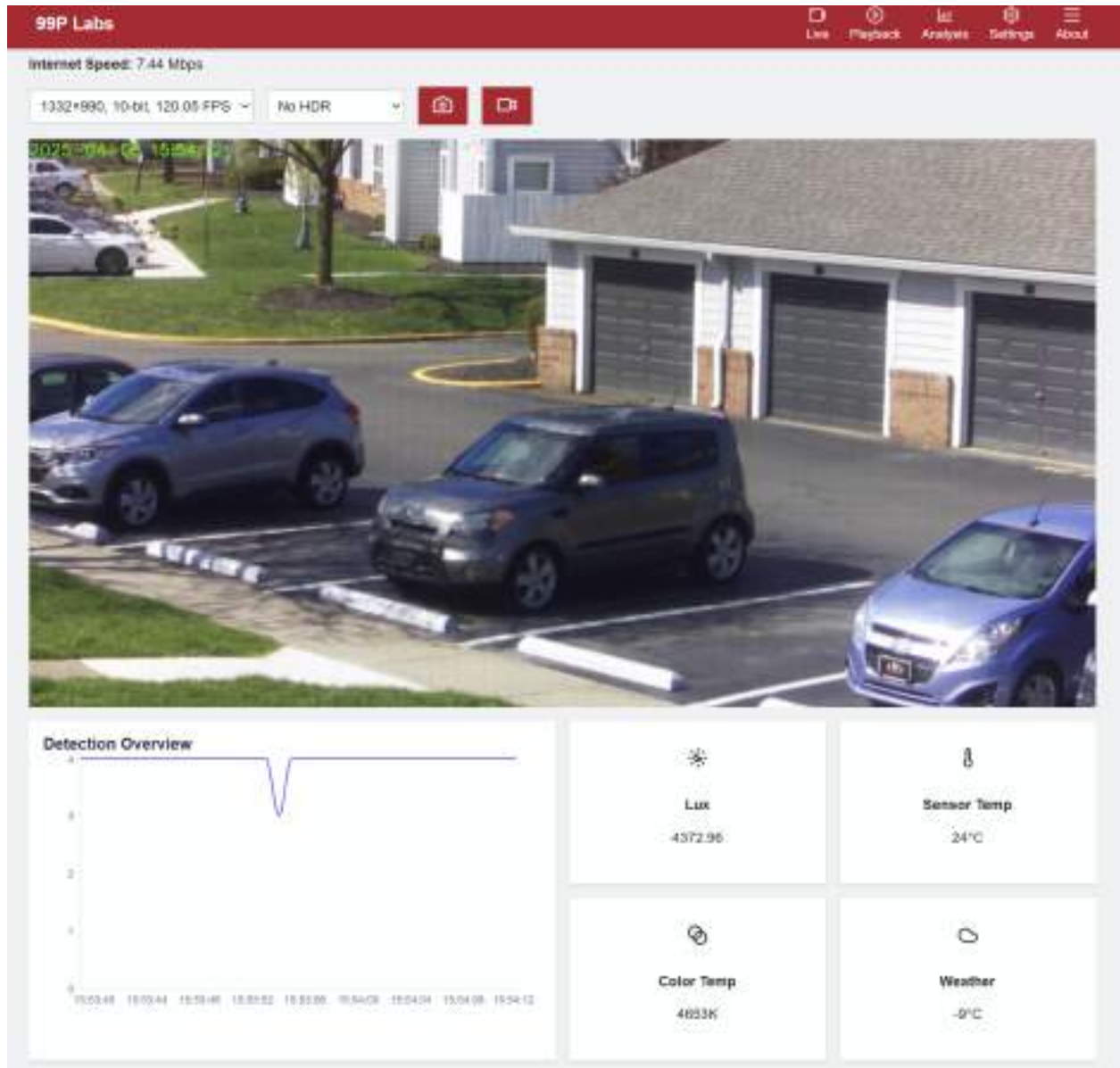


Figure 4: Visual example of the live page on the frontend, the main feature when launching the application

3.2 Testing

Software and Hardware Test Plan

This document outlines the test plan for verifying and validating the functional and performance requirements of our traffic data capture and analysis system. The testing process ensures that all subsystems (object detection, tracking, camera calibration, latency analysis, and remote connectivity) meet the expected performance and accuracy criteria.

Purpose

The purpose of this test series is to validate the system's ability to perform real-time object detection and tracking accurately, ensure robust remote connectivity for data collection and system control, and evaluate processing efficiency and system performance under load.

Scope

The tests will cover the software, hardware, and system integration. The software tests will focus on model inference speed and tracking consistency. The hardware tests will evaluate the processing unit (raspberry pi + TPU), power consumption and connectivity performance. System integration tests will assess end-to-end latency.

The tests will be conducted using a fixed set of intersection videos and real-time footage, limiting variability in testing conditions. Moreover, network connectivity tests will be limited to available infrastructure and may not fully replicate all deployment scenarios.

Table 3: Requirement/Verification Cross-Reference Matrix

Classes of Verification: I- First Article, II-Environmental, III-Acceptance Test, IV- None

Methods of Verification: A- Analysis, T-Test, D-Demonstration, I- Inspection

| Project Requirements | Test Names | Verification Class | | | | Test Procedure Location |
|---------------------------------|----------------------------------|--------------------|----|-----|----|-------------------------|
| | | I | II | III | IV | |
| Object Detection Model Accuracy | YOLO Detection Accuracy Test | | | T | | Appendix B.1 |
| Object Tracking Stability | DeepSORT Tracking Test | | | T | | Appendix B.2 |
| Model Inference Speed | TPU Inference Speed Test | T | | | | Appendix B.3 |
| Latency (End to End) | System Response Test | | | T | | Appendix B.4 |
| FPS Benchmark | Video Processing Benchmark | | | T | | Appendix B.5 |
| Privacy & Data Security | GDPR Compliance Test | | | I | | Appendix B.6 |
| Remote SSH / Web Access | Connectivity & Access Test | | | D | | Appendix B.7 |
| System Resource Usage | System Resource Utilization Test | | | T | | Appendix B.8 |

Test Explanation:**YOLO Detection Accuracy Test:**

The objective and scope is to measure the accuracy of the detections coming in from YOLO. The procedure will use a preset video to feed the module and use a dataset to check the results against. To pass, the team expects a detection accuracy of 90%.

DeepSORT Tracking Test:

The objective of this test is to assess the consistency and reliability of object tracking. This procedure will involve a frame-by-frame analysis for accuracy. Due to hardware limitations, there is not yet a set accuracy value for the tracking, however the team has decided to pass the test based on consistency from frame to frame.

TPU Inference Speed Test:

The test will run inference on the TPU and log metrics from running YOLO to see latency. Similarly to the DeepSORT test, rather than a certain latency speed being set as pass/fail, the team has determined that if the speed results are consistent between all cases, that this test passes.

System Response Speed Test:

This test will measure the end-to-end speed of the entire system. This will be run by starting up the system and timing the difference between the action occurring in front of the camera and when it gets detected and the dashboard livestream. An acceptable value for this test is a latency of under 5 seconds

Video Processing Benchmark:

This test will measure the speed at which the system processes videos. The dashboard will be used to upload and process a preset video for analysis. This process will be timed and measured. Since the processing time is dependent on the length of the video itself, a passable time would be one that finishes processing within two videos worth of time.

GDPR Compliance Test:

This test aims to ensure that the system upholds any regulations with regards to data protection. The procedure will include validating that any data collected adheres to storage policies and that anonymity is protected. Therefore, to pass, no personally identifiable information must be stored.

Connectivity & Access Test:

This test will evaluate how the system operates remotely. The procedure will be executed by connecting to the Raspberry Pi from a distance and starting up the dashboard. As this is a demonstration test, if the application can be started from a distance, the test will pass.

System Resource Utilization Test:

This test aims to test whether the system is properly storing any uploaded and preset videos. The procedure will involve passing videos through the system and ensuring that the data collected reaches the database without any data corruption. This test will aim for a drop rate of <5%, meaning that at least 95% of data makes it through the pipeline to be stored.

Chapter 4: Final Design

After the testing was completed by the team, the team decided to keep the prototype as is. The test cases that the team determined crucial had passed and the sponsors were satisfied with the product.

The final design the team implemented had many various components with both the hardware and the software side. The final product was displayed on a dashboard that was developed in Python that had four main pages: live page, playback page, analysis page, about page and settings page. The team developed this dashboard with a Python backend, JavaScript frontend, FastAPI routes, API keys, and SQLite.

The hardware aspect of the project, the team had a Raspberry Pi, a Raspberry Pi high quality camera (HCQ) with tripod and a Coral TPU. The Raspberry Pi HCQ fed in a live stream through the raspberry pi. The Coral TPU connected to the Raspberry Pi and helped accelerate this process to make sure that processing speed of objects in from the camera was not lagging. With this high-speed connection between the camera and the raspberry pi, the team was able to display the live feed on the dashboard in its own live page. This live page had direct access to the camera feed and had important metrics reading in from the camera's sensors. Additionally, the live page had an object detection graph which constantly updated and stored the number of objects that were seen in the camera lens.



Figure 5: Live page with live feed, object tracking and sensor information.

Moving to the next page of the dashboard, the playback page. The playback page allowed the user to upload a video and ask the AI chatbot (CR-Vision) questions about it. The AI chatbot

uses an OpenAI API key, OpenWeather API and a traffic database from Vanderbilt university to answer any questions about the traffic data uploaded on the backend. The chatbot is able to answer any question regarding the database, the weather or general conversation. When accessing traffic information, the chatbot has the ability to provide you a frame if that is what you are asking for. This frame can then be accessed by typing it into the box below the video upload to go to the desired frame and find said vehicle.

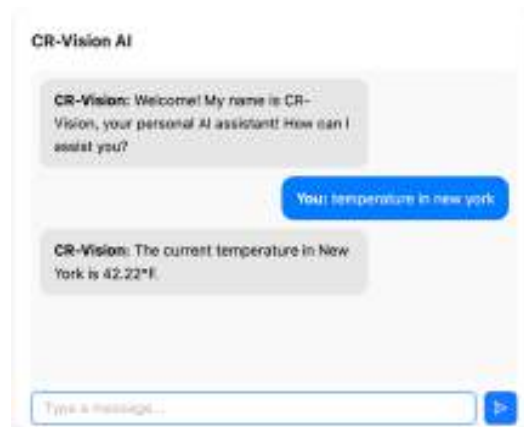


Figure 6: Asking chatbot about weather

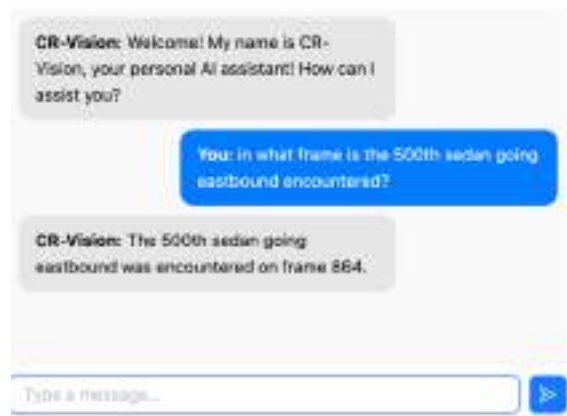


Figure 7: Chatbot providing frame for specified sedan.

The analysis page is the next page the team worked on. This page was created to create a history of the past entries the user has asked the chatbot and their success rate. Each entry has been noted of whether it was a success or failure. Below these logs, there is a graph showing retrieval times for each user input for the chatbot. This helps ensure the chatbot does not take too long to find what the user is looking for. Lastly, below the graphs with retrieval times, there is a pie chart with the success rate showing how many user entries to the chatbot have been successful.



Figure 8: Analysis page with user input log, retrieval times and success rates

Lastly, the team implemented an about page and a settings page. The purpose of this page was to show the purpose of our project and what the team developed in the end along with a thank you to our sponsors. The settings page had basic settings for the dashboards front end.

Chapter 5: User Manual

Product description and goal

The Smart Intersection AI Data Collection Prototype is a system designed to capture real-world traffic and environmental data from an intersection using a Raspberry Pi, camera module, Coral TPU, OpenWeatherAPI and OpenAI API. The primary goal is to collect a synchronized dataset of video streams, object detections (identifying vehicles, pedestrians, etc.), and sensor data (lux, color temp, sensor temp) to facilitate the development and analysis of AI models for smart intersection management. The document outlines the product's usage, system requirements, operational steps, and key features.

Product Usage

The prototype is used to simulate a data collection node at an intersection. It captures a live video feed, performs object detection on that feed, and publishes the detection results, alongside any connected sensor data, to a public MQTT broker. A separate web application allows users to view the live stream and configure the data input sources (stream URL and MQTT broker address). The collected data is intended for later analysis.

Requirements

Hardware:

- Raspberry Pi
- Raspberry Pi Camera Module
- Camera Lens
- Coral USB Accelerator (or other compatible Coral TPU device)

- Mount for the camera module
- Raspberry Pi Power Supply
- Micro HDMI cable and monitor (for initial setup)
- Keyboard and Mouse (for initial setup)
- MicroSD card (8GB or larger recommended)

Software:

- Raspberry Pi OS
- Python 3.8 (with a virtual environment)
- TensorFlow Lite
- PyCoral library
- Docker
- Node.js and npm (for the web application)
- Tailscale network (for remote access)
- MQTT Broker (can be hosted locally or remotely, for our prototype, the team used the public MQTT broker: broker.emqx.io)
- OpenAI API key

How to operate the system/product

Initial Raspberry Pi Setup (if already setup, move on to the next step):

- Flash Raspberry Pi OS onto the microSD card.
- Insert the microSD card into the Raspberry Pi.
- Connect the Raspberry Pi to a monitor using a micro-HDMI cable and attach a keyboard and mouse.
- Power on the Raspberry Pi.
- Follow the on-screen prompts to complete the initial OS setup, including setting up user accounts and configuring the network.
- Configure the Raspberry Pi to automatically connect to the desired Wi-Fi network at the deployment location.

Hardware Assembly (Camera):

- Carefully twist the camera sensor and lens together.
- Using the flat ribbon cable, connect the camera module to the port on the Raspberry Pi. Ensure the cable is inserted with the correct orientation. Ensure the Raspberry Pi is powered off before connecting the camera. Refer to image below.



Figure 9: Final Design

Install Dependencies and Configure Environment:

- Power on the Raspberry Pi.
- Once the OS is running and connected to the network, open a terminal.
- Navigate to the edge directory of your project code.
- Make sure the Coral TPU is connected to a USB port on the Raspberry Pi.
- Run the setup script: `bash setup.sh`
- This script will update the system, install necessary dependencies, detect hardware and OS, install Python 3.8 and set up a virtual environment, install requirements for the Edge TPU (TensorFlow Lite, PyCoral), install Docker, and verify the Coral TPU is recognized.

Set up Secure Remote Access (Tailscale):

- Install the Tailscale client on the Raspberry Pi by following the instructions on the Tailscale website for your OS (Linux).
- Install the Tailscale client on your development machine (Linux, Windows, or Mac).
- Log in to Tailscale on both devices using the same account to establish a secure, private network connection between them. Note the Tailscale IP address assigned to your Raspberry Pi.

Start the Data Collection Services:

- Start the Live Stream and Sensor Data Service:

- Open a terminal on the Raspberry Pi (you can now do this remotely via SSH using the Tailscale IP).
- Navigate to the edge directory.
- Activate the Python virtual environment created by setup.sh.
- Run the main script: `python main.py`
- This script initiates the camera live stream (accessible via HTTP on port 8000), exposes web server endpoints, and starts sending sensor data (if configured in the script) via MQTT. The live stream will be available at `http://<TAILSCALE_IP>:8000/stream.mjpg`.
- Start the Object Detection Service:
 - Open a new terminal session on the Raspberry Pi.
 - Navigate to the edge/detection directory.
 - Activate the Python virtual environment for detections.
 - Run the detection script: `python main.py`
 - This script intercepts frames from the live stream, performs object detection using the Coral TPU, and publishes the detection results as MQTT messages. Make sure to change the live stream like in the main file to match the tailscale IP address, as the team did in the previous step.

Run the Web Application (Frontend):

- On your development machine, ensure Node.js is installed.
- Navigate to the front-end directory of your project code.
- If this is the first time running, install dependencies: `npm install`
- Start the development server: `npm run dev`
- Open a web browser and navigate to the address provided by npm (usually `http://localhost:5173` or similar).
- In the web application, you should be able to configure the live stream URL (using the Tailscale IP of the Raspberry Pi and port 8000, e.g., `http://<TAILSCALE_IP>:8000/stream.mjpg`) and the address of your MQTT broker.
- Make sure to set up a `.env` file that has all the required API keys, which include OpenAI and OpenWeather API.

Run the Web Application (Backend):

- On your development machine, ensure any necessary dependencies are installed with pip install.
- Navigate to the back-end directory of your project code.
- Start the backend server: `python -m uvicorn main:app --reload`

Maintenace

Software Updates: Periodically update the Raspberry Pi OS and installed libraries (including TensorFlow Lite and PyCoral) to ensure you have the latest features and security patches. This can typically be done via the terminal using `sudo apt update` and `sudo apt upgrade`.

Script Restarts: While the scripts are designed to run continuously, occasional restarts of the `edge/main.py` and `edge/detection/main.py` scripts may be necessary to resolve unforeseen issues or after software updates.

Configuration Verification: If the MQTT broker or live stream source is changed, verify the configuration in both the web application and the relevant Python scripts on the Raspberry Pi is correct.

Engineering Troubleshooting report

Problem 1

The chatbot defaults to the fallback response instead of generating a desired response. This problem is most likely caused by the first classification prompt returning the correct classification but with improper formatting like spaces or quotation marks. Therefore, to solve the problem, any extra characters should be stripped before branching into further logic. This can done with a simple python line like the one below:

```
classification = classification.strip().lower().strip('"')
```

Chapter 6: Project Management, Summary, and Conclusion

5.1 Project Management

The team kept flexibility at the center of its approach and actively addressed challenges as they came up. When issues with attendance and uneven work distribution occurred, members communicated openly, stayed respectful, and worked through disagreements directly. For project management, the team used Miro to plan tasks, Excel to track progress, and Discord as the main communication platform.

At the start of the project, the team struggled with a lack of clear direction. To solve this, members met with the sponsor in a more informal setting and clarified the project's goals. Another problem emerged when the team tried to include too many unnecessary features. A meeting with the professor helped narrow the focus and prioritize only what was essential to complete the project.

For the next steps, the team recommends the following improvements:

- Add a smart storage module to store live traffic data and allow this live data to be interacted with by the chatbot queries from the playback page.
- Explore new types of data collection, such as audio or thermal data, to enhance the system's analysis capabilities.

These steps would strengthen the project's core functionality and expand its potential for future use.

5.2 Summary

The project reached several key milestones that marked steady progress toward the final solution. First, the team successfully set up a computer vision library, which served as the foundation for processing traffic footage. Next, the team integrated this library with a Raspberry Pi and camera, enabling live video analysis at the edge. After confirming reliable data collection, the team developed both a frontend and backend to support a functional web application. Finally, the team created a chatbot capable of generating SQL queries, allowing users to interact with the traffic database in a natural, conversational way.

These milestones directly addressed the project's main goal: making traffic data easier to access and understand. The final solution lets users review footage, pull traffic data, and communicate with the system through the chatbot. Although the team didn't implement features like long-term data storage or advanced sensor input, the current system meets the core objectives and sets a strong foundation for future development. The solution proves the concept works and gives the sponsor and stakeholders a clear path forward.

5.3 Conclusion

The final solution establishes a working pipeline and framework for analyzing traffic data, with clear potential for future expansion. Future work could focus on adding long-term data storage, integrating additional sensor types such as audio or thermal, and enhancing the chatbot's capabilities to support more advanced queries. The current system provides a solid foundation, and future teams can build on this work to develop a fully scalable and real-time traffic analysis platform.

Nick: I contributed to the project as the team lead and worked on building the chatbot and the analysis page of the final solution. Throughout the year, I learned a great deal about project management, especially in areas I had never explored before, such as creating Gantt charts and writing detailed test plans. My biggest weakness going into the project was my ability to quickly adapt and pivot when things didn't go as planned. This project helped me improve that skill significantly. There were several moments when the team started moving in the wrong direction, and I had to step in, reassess the situation, and guide the team toward a better path. Taking on this responsibility helped me grow not just as a developer but also as a leader.

Adi: I contributed to the project by working on the backend, frontend, and the chatbot on the playback page. This project made an everlasting impact on my perspective about group projects. I learned new technical skills as long as soft skills that I did not have much experience with before. Within technical skills, I learned how to model your own LLM in Python and multiple aspects of full stack development such as API and SQL implementations along with UI features. I learned about the importance of sending emails on time, scheduling meetings on time and being prepared for meetings so that they are beneficial to all of the members of the team. I believe the year started off confusing and a little lost for all of us, but the team was able to constantly figure out what was needed and expected of us. Being able to accommodate situations that were out of our control or open ended throughout the year helped me learn how to deal with these experiences if they came up again. Lastly, I believe the group that I worked with is amazing. I believe it is the reason all of these issues were able to be fixed. The three of us worked together and made sure to tackle any problem either one of us had together, which was crucial to our success.

Mohamed: I contributed to the project by working on the data collection pipeline involving the Raspberry pi, camera module and Coral TPU. I focused on building the data collection process. The main lesson learned was the importance of effective communication and strong team building. While technical skills are undeniable valuable, and can be acquired and learned through practice, the ability to collaborate and communicate effectively, and understanding different perspectives is a far more challenging skill to develop. Over the past year, I've encountered situations where clearer communication could have helped. I recognize that this is an area I need to work on. Moving forward, I am committed to actively improving my strategies and focusing on contributing positively to team collaboration.

Appendix A



Semester 2 Gantt Chart

[illegible]

Appendix B

| | |
|---|-------------------------|
| B.1 Yolo Detection Accuracy Test | |
| Method of Verification: Test | |
| Type of Test: Acceptance | |
| Date(s)/ Total Times: 3/18 | Location: Online |
| Test Equipment: Laptop, Raspberry Pi | |
| Test Engineer: Mohamed Hambouta | |
| Witness: Nick Sanchez | |
| Criteria for Success: <input checked="" type="checkbox"/> Passes – The condition once achieved the test passes i.e. verified the requirement. Must be measurable and objective. Be careful with setting multi conditional phrases. <input type="checkbox"/> Fails – The condition once achieved the test fails i.e. verified the requirement; must be measurable and objective. You can check the box to show the test result | |
| Test Procedure: <ol style="list-style-type: none">1. Deploy YOLO model in its docker container.2. Run detection on a preset video3. Compare data recorded from video to predefined dataset and record accuracy4. Compare recorded accuracy to 90% benchmark | |
| Test Results: The precision metric was calculated by only comparing the number of detected objects in each frame on the test dataset. The out-of-the box YOLO model was able to achieve a ~92% precision metric when detecting the number of objects. However, it struggled with classification. | |

```

EXPLORER
├── TESTS [SSH: 192.168.1.234]
│   ├── .venv
│   ├── videos
│   ├── accuracy.py
│   ├── detection_comparison...
│   ├── final_accuracy_metric...
│   ├── main.py
│   ├── PIC1_dataset.xlsx
│   ├── scene1_hg.json
│   ├── tracking_results_24_f...
│   └── yolov11n.pt
├── PROBLEMS
├── OUTPUT
├── DEBUG CONSOLE
├── TERMINAL
├── PORTS
└── POSTMAN CONSOLE

(.venv) capstone@raspberrypi:~/TrafficJammers/tests $ python accuracy.py
Frame-by-frame comparison saved to detection_comparison_results.csv

==== YOLO Detection Accuracy Metrics ====
Average Precision: 0.9249
Average Recall: 1.0749
Average F1-score: 0.8582
Final accuracy metrics saved to final_accuracy_metrics.txt
(.venv) capstone@raspberrypi:~/TrafficJammers/tests $

```

Action Items:

None, the object detection model can just be swapped by a better one.

Test Engineer:
MH, 03/23/2025

Witness:
NS, 03/23/2025

| B.2 DeepSORT Tracking Test | |
|---|-------------------------|
| Method of Verification: Test | |
| Type of Test: Acceptance | |
| Date(s)/ Total Times: 3/18 | Location: Online |
| Test Equipment: Laptop, Raspberry Pi | |
| Test Engineer: Mohamed Hambouta | |
| Witness: Nick Sanchez | |
| Criteria for Success: <div> <input type="checkbox"/> Passes – The condition once achieved the test passes i.e. verified the requirement. Must be measurable and objective. Be careful with setting multi conditional phrases. </div> <div> <input checked="" type="checkbox"/> Fails – The condition once achieved the test fails i.e. verified the requirement; must be measurable and objective. </div> | |
| You can check the box to show the test result | |

| | |
|---|----------------------------|
| Test Procedure: | |
| <ol style="list-style-type: none"> 1. Load video frames into the Docker container containing the DeepSORT algorithm. 2. Have DeepSORT run on consecutive frames. 3. Measure the tracking accuracy over frames. 4. Determine whether the accuracy is able to stay consistent without any fluctuations. | |
| Test Results: | |
| Failed. The tracking algorithm provided by Ultralytics (YOLO creator) was not able to accurately keep track of the objects between frames. | |
| Action Items: | |
| None, the team needs to use better tracking algorithms or implement our own. | |
| Test Engineer: MH, 03/23/2025 | Witness: NS, 03/23/2025 |

| | |
|--|-------------------------|
| B.3 TPU Inference Speed Test | |
| Method of Verification: Test | |
| Type of Test: First Article | |
| Date(s)/ Total Times: 3/18 | Location: Online |
| Test Equipment: Laptop, Raspberry Pi | |
| Test Engineer: Mohamed Hambouta | |
| Witness: Nick Sanchez | |
| Criteria for Success: <input checked="" type="checkbox"/> Passes – The condition once achieved the test passes i.e. verified the requirement. Must be measurable and objective. Be careful with setting multi conditional phrases. <input type="checkbox"/> Fails – The condition once achieved the test fails i.e. verified the requirement; must be measurable and objective. You can check the box to show the test result | |
| Test Procedure: | |
| <ol style="list-style-type: none"> 1. Deploy and launch YOLO model in its Docker container. | |

2. Process test images or videos and log the inference time in a log file
3. Using the log file, compute the average time.
4. Compare the range of the values to the computed average and look for outliers.
5. If results are consistently within 1 standard deviation of the average, test passes.

Test Results:

Test passed as it met sufficient metrics for processing time.

```

EXPLORER
TESTS [SSH: 192.168.1.234]
  .venv
  videos
  accuracy.py
  detection_comparison.py
  final_accuracy_metrics.py
  main.py
  PIC1_dataset.txt
  scene1_hg.json
  tracking_results_04_1_
  yolov11n.pt

TERMINAL
(.venv) capstone@raspberrypi:~/TrafficJammers/tests $ python accuracy.py
Frame-by-frame comparison saved to detection_comparison_results.csv

==== YOLO Detection Accuracy Metrics ====
Average Precision: 0.9249
Average Recall: 1.0749
Average F1-score: 0.8582
Final accuracy metrics saved to final_accuracy_metrics.txt
(.venv) capstone@raspberrypi:~/TrafficJammers/tests $
  
```

Action Items:

None.

Test Engineer:
MH, 03/23/2025

Witness:
NS, 03/23/2025

| B.4 System Response Speed Test | |
|---|---------------|
| Method of Verification: Test | |
| Type of Test: Acceptance | |
| Date(s)/ Total Times: 3/20 | Location: Lab |
| Test Equipment: Laptop, Raspberry Pi, Raspberry Pi Camera | |
| Test Engineer: Nick Sanchez | |
| Witness: Mohamed Hambouta | |
| Criteria for Success: | |

☒ Passes – The condition once achieved the test passes i.e. verified the requirement. **Must be measurable and objective. Be careful with setting multi conditional phrases.**

☐ Fails – The condition once achieved the test fails i.e. verified the requirement; must be measurable and objective.

You can check the box to show the test result


Test Procedure:

1. Start up the pipeline on a laptop.
2. Measure the time from when an object is shown in front of the camera to when it shows on the dashboard with metrics
3. Repeat 3-5 times and compute average.
4. Log internet traffic for each trial.
5. Compare average latency to benchmark of 5 seconds.

Test Results:





| | |
|---|------------------------------------|
|  | |
| <p>The system consistently responded in under 5 seconds over several trials, causing this test to pass.</p> | |
| <p>Action Items: No action items.</p> | |
| <p>Test Engineer: NS, 03/26/2025 Must write name or initials</p> | <p>Witness: MH, 03/26/2025</p> |

| B.5 Video Processing Benchmark | |
|---|------------------|
| Method of Verification: Test | |
| Type of Test: Acceptance | |
| Date(s)/ Total Times: 3/20 | Location: Online |
| Test Equipment: Laptop, Raspberry Pi, Raspberry Pi Camera | |

Test Engineer: Nick Sanchez

Witness: Adi Devnani

Criteria for Success:

☒ **Passes** – The condition once achieved the test passes i.e. verified the requirement. **Must be measurable and objective. Be careful with setting multi conditional phrases.**

☐ **Fails** – The condition once achieved the test fails i.e. verified the requirement; must be measurable and objective.

You can check the box to show the test result

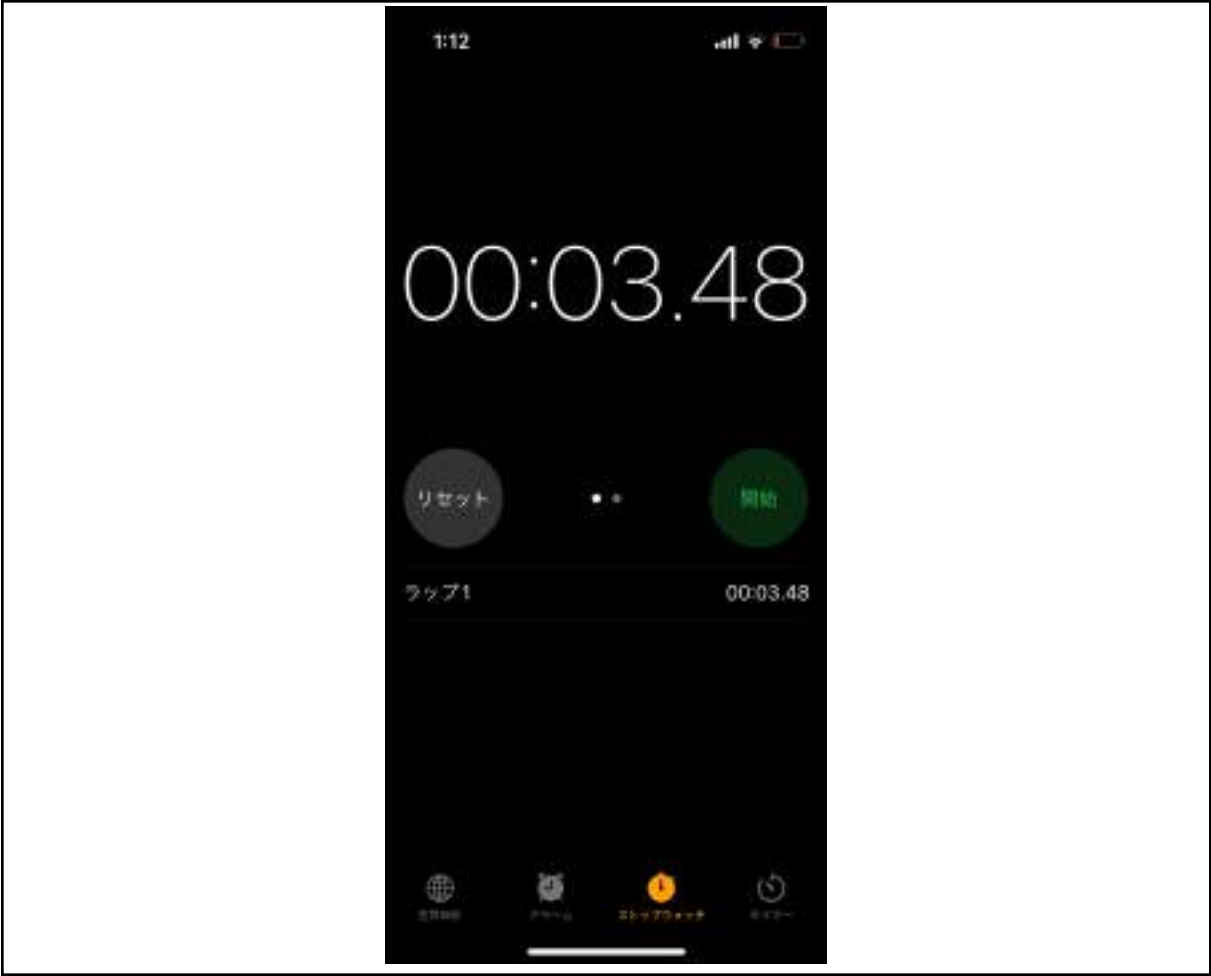
Test Procedure:


1. Start up the pipeline on a laptop.
2. Measure the time from when a video is uploaded to when it is able to be analyzed
3. Repeat 3-5 times and compute average.
4. Compare average processing time to benchmark of 2 times the video length.

Test Results:


Times recorded from uploading a 5 minute video for analysis:





| | |
|--|--|
|  | |
| Action Items: The benchmark should not have been set so high for a video that is processed locally. May need to redo test with a more accurate timing method and stricter benchmark. | |
| Test Engineer: NS, 03/26/2025 | Witness: AD, 03/26/2025 |

| |
|---|
| B.6 GDPR Compliance Test |
| Method of Verification: Inspection |
| Type of Test: Acceptance |

| | | |
|--|--|----------------------------|
| Date(s)/ Total Times: 3/25 | | Location: Online |
| Test Equipment: Laptop, Raspberry Pi | | |
| Test Engineer: Nick Sanchez | | |
| Witness: Adi Devnani | | |
| Criteria for Success: <ul style="list-style-type: none"> <input checked="" type="checkbox"/> Passes – The condition once achieved the test passes i.e. verified the requirement. Must be measurable and objective. Be careful with setting multi conditional phrases. <input type="checkbox"/> Fails – The condition once achieved the test fails i.e. verified the requirement; must be measurable and objective. | | |
| You can check the box to show the test result | | |
| Test Procedure: <ol style="list-style-type: none"> 1. Check the latest data storage policies. 2. Sweep through database parameters to ensure no private data is being stored. 3. Verify that any anonymity of captured targets is maintained. | | |
| Test Results:  <p>The data used for the project was found to have been anonymized by its creators. This confirms that data being used for the project is compliant with any privacy guidelines.</p> | | |
| Action Items: No action items. | | |
| Test Engineer: NS, 03/23/2025 | | Witness: AD, 03/23/2025 |

| | |
|--|-------------------------|
| B.7 Connectivity and Access Test | |
| Method of Verification: Demonstration | |
| Type of Test: Acceptance | |
| Date(s)/ Total Times: 3/20 | Location: Online |
| Test Equipment: Laptop, Raspberry Pi, Pi Camera | |

Witness: Mohamed Hambouta

Criteria for Success:

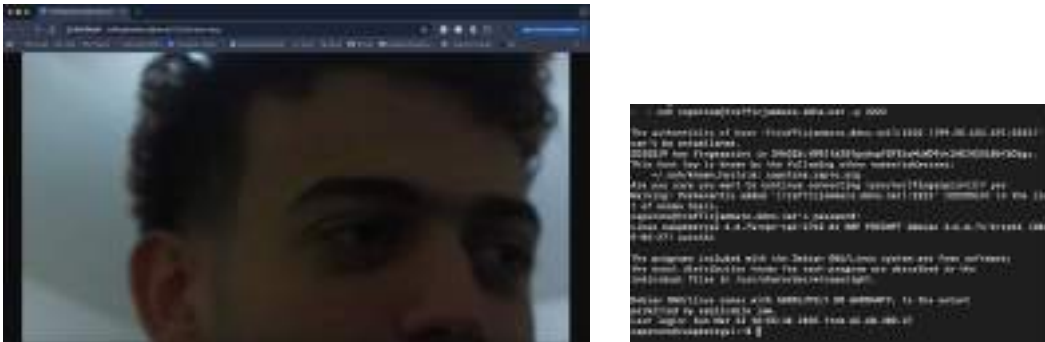
- ☒ **Passes** – The condition once achieved the test passes i.e. verified the requirement. **Must be measurable and objective. Be careful with setting multi conditional phrases.**
- ☐ **Fails** – The condition once achieved the test fails i.e. verified the requirement; **must be measurable and objective.**

You can check the box to show the test result

Test Procedure:

1. Boot up laptop from a location remote from Raspberry Pi
2. SSH into Raspberry Pi and launch dashboard
3. Ensure that dashboard functions as planned from remote location

Test Results:



Able to access the ssh remotely from my computer. Accessing live video feed on the left with terminal commands on the right

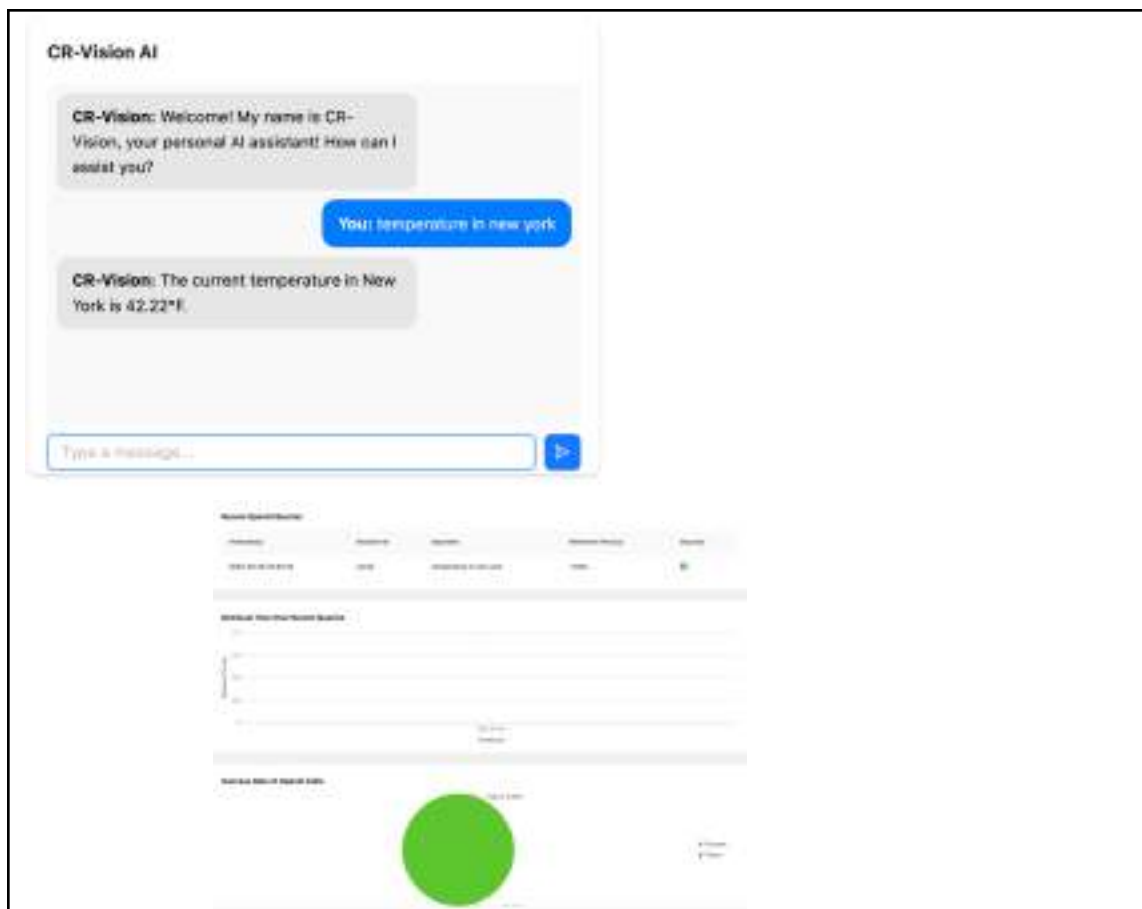
Action Items:

No action needed

Test Engineer:
AD, 03/23/2025

Witness:
MH, 03/23/2025

| | |
|---|-------------------------|
| B.8 System Resource Utilization Test | |
| Method of Verification: Test | |
| Type of Test: Acceptance | |
| Date(s)/ Total Times: 3/20 | Location: Remote |
| Test Equipment: Laptop, Raspberry Pi, Pi Camera | |
| Test Engineer: Adi Devnani | |
| Witness: Niesk Sanchez | |
| Criteria for Success: <input checked="" type="checkbox"/> Passes – The condition once achieved the test passes i.e. verified the requirement. Must be measurable and objective. Be careful with setting multi conditional phrases. <input type="checkbox"/> Fails – The condition once achieved the test fails i.e. verified the requirement; must be measurable and objective. You can check the box to show the test result | |
| Test Procedure: <ol style="list-style-type: none"> 1. Run system under load 2. Check data storage against predefined data set 3. Compare drop rate to 5% benchmark | |
| Test Results: Temperature in New York chatbot query: | |



Analytics page with more queries linked to analytics page:



One query for traffic database:

CR-Vision: Welcome! My name is CR-Vision, your personal AI assistant! How can I assist you?

You: how many trucks are there in the set of data?

CR-Vision: The dataset contains 1,260 trucks.



Asking for multiple queries in chatbot:

CR-Vision: Welcome! My name is CR-Vision, your personal AI assistant! How can I assist you?

You: in what frame is the 500th sedan going eastbound encountered?

CR-Vision: The 500th sedan going eastbound was encountered on frame 864.

Type a message...



Action Items:

The system was able to answer questions appropriately; the weather questions used the API while the traffic questions accessed the database. The chat entries were stored in the analytics page and were recorded as successes as they all worked.

Test Engineer:

Witness:

| | |
|---|----------------|
| AD, 03/26/2025 Must write name or initials | NS, 03/26/2025 |
|---|----------------|