



Imam Mohammad Ibn Saud Islamic University  
College of Computer and Information Sciences  
Computer Science Department

---

**Second Semester 1445/2023**

**Date: 30/4/2023**

## **Traffic Light Detection: Making the Way for Lifesavers**

Team Members	Academic Numbers
Abdulrahman Bin Moharib	441014822
Ahmad Alroqi	441017341
Khalid Aldossari	441013413

Supervisor: [Dr. Alaa Eldeen]

Submission Date: 04/02/2024



## Table of Contents

1.	Introduction: .....	7
1.1	Introduction: .....	7
1.2	Aims and objectives: .....	8
1.3	Methodology:.....	9
1.3.1	Problem Definition and Requirements Gathering: .....	9
1.3.2	Data Collection:.....	9
1.3.3	Model Selection: .....	9
1.3.4	Model Training:.....	9
1.3.5	Integration with Traffic Lights:.....	9
1.3.6	Real-time Processing:.....	9
1.3.7	Testing and Evaluation:.....	9
1.4	Team Qualifications: .....	10
1.5	Conclusion.....	10
2.	Literature Review:.....	11
2.1	Introduction: .....	11
2.2	Background: .....	12
2.3	Related Work: .....	13
2.4	Conclusion:.....	24
3.	System Analysis:.....	25
3.1	Introduction: .....	25
3.2	Dataset .....	26
3.3	Software Requirements Specification.....	26
3.4	User Characteristics .....	26
3.5	User Requirements .....	28
3.5.1	Functional Requirements:.....	28
3.5.2	Non-functional Requirements: .....	30
3.6	System Requirement.....	31
3.6.1	Use Case Diagram: .....	31
3.6.2	Sequence Diagram: .....	33
3.6.3	Activity Diagram:.....	35



3.7	Project Management Plan: .....	36
3.8	Conclusion:.....	36
4.	Design Chapter.....	37
4.1	System Architecture.....	37
4.2	Database Design.....	39
4.3	Modular Decomposition .....	40
4.4	System Organization .....	41
4.5	Algorithm .....	44
4.6	Alternative Design Methods .....	46
4.7	Graphical User Interface Design .....	48
5.	Implementantion Chapter.....	50
5.1	Implementation Requiremnts.....	50
5.1.1	Hardware Requirements.....	50
5.1.2	Software Requirements .....	50
5.1.3	Programming Language .....	51
5.1.4	Tools and Technologies.....	52
5.2	Implementation Details .....	53
5.2.1	Deployment and Installation:.....	53
5.2.2	Data Structures Description: .....	54
5.2.3	Procedures Description:.....	55
5.2.4	Graphical User Interface Description.....	58
6.	Testing Chapter.....	59
6.1	Model Results .....	59
6.2	Model Testing: .....	60
6.3	Register Page Testing:.....	62
6.4	Login Page Testing: .....	63
6.5	Password Changing Testing: .....	63
6.6	Calling Emergency Testing .....	64
6.7	Messaging Alert Testing.....	64
7.	Future Work .....	65
8.	Conclusion.....	66



9. References: .....	67
----------------------	----

#### List of Figures:

Figure 1 Ambulance detected by the camera .....	14
Figure 2 Using anchor box to detect .....	15
Figure 3 Image problems, i.e. blur .....	15
Figure 4 Result without Filtering algorithm .....	16
Figure 5 After step two.....	16
Figure 6 After step one .....	16
Figure 7 Result from camera.....	17
Figure 8 Use Case Diagram for the Model .....	31
Figure 9 Use Case Diagram for System Monitoring .....	32
Figure 10 Sequence Diagram for the Model .....	33
Figure 11 Sequence Diagram for Traffic Management .....	34
Figure 12 Activity Diagram .....	35
Figure 13 Gantt Chart for the Project Plan .....	36
Figure 14. System Architecture .....	37
Figure 15. System and Sub-Systems.....	37
Figure 16. Object Oriented Design Diagram .....	40
Figure 17. Data Flow Diagram .....	40
Figure 18. Sequence Diagram for Model and Application .....	41
Figure 19. Use Case Diagram .....	42
Figure 20. Activity Diagram for the Application.....	43
Figure 21. Profile Page Prototype .....	48
Figure 22. Login Page Prototype .....	48
Figure 23. Signup Page Prototype .....	49
Figure 24. Running the Backend Code .....	53
Figure 25. Terminal Command to Run API.....	53
Figure 26. Ngrok Interface.....	54
Figure 27. Sample from the Ambulance Dataset .....	55
Figure 28. Ultralytics Installing .....	55
Figure 29. Training the YOLOv8 Model .....	55
Figure 30. Training Result .....	56
Figure 31. Validation Code .....	56
Figure 32. Validation Result .....	57
Figure 33. Detection Code .....	57
Figure 34. Model API Code.....	57
Figure 35. Application API Code .....	57
Figure 36. Model Training Results .....	59
Figure 37. Ambulance Images Before and After the Model Prediction .....	60
Figure 38. Output of the Model without Ambulance Vehicle .....	61
Figure 39. Register Page .....	62
Figure 40. Register Page with Error .....	62



Figure 41. Login Page .....	63
Figure 42. Login Page with Error .....	63
Figure 43. Changing the Password Successfully .....	63
Figure 44. Error while Changing the Password .....	63
Figure 45. Showing the Emergency Call Button .....	64
Figure 46. Navigating to Calling 997 .....	64
Figure 47. Notification Alert.....	64

#### List of Tables:

Table 1 Team Qualifications.....	10
Table 2 Comparison Table.....	23
Table 3. Comparison Table Between Flutter and Swift.....	46
Table 4. Comparison Table Between API and WebSocket .....	47



## Abstract:

Nowadays in our crowded cities, traffic congestion has become a serious problem effecting emergency response, potentially impacting lives in critical situations. what about when it comes to arrival of emergency vehicles, such as ambulances, due to traffic congestion is a pressing concern. We can benefit from the progress of technology and enhance the efficiency of healthcare system; we will propose a solution that make a use with image processing and artificial intelligence (AI) models.

Our primary goal is to develop a system that detects approaching ambulance vehicles as they approach traffic lights, thereby prioritizing their passage and reducing response times. Integrating image processing algorithms with AI models, we aim to transform traffic management, improve emergency response times, and ultimately save lives.

This documentation serves as a comprehensive guide to our project. It offers a description about the system's architecture, methodology, and it also discusses the related work in the field. To provide a clear understanding of our approach, we present comparison table to compare between the different methods that has been used in related research.

Furthermore, it also covers user requirements, and system requirements. We present a series of diagrams that illustrate how the system components interact and work to achieve our objectives.

With this work, we are committed to making urban environments safer and more responsive during medical emergencies, showcasing the transformative power of technology to improve traffic management and emergency response.



# 1. Introduction:

## 1.1 Introduction:

This section aims to provide an overview of the project's objectives, scope, and its significance in addressing the identified challenges. The increased number of vehicles on the road has made it difficult for emergency vehicles to arrive on time. Public safety may suffer because these vehicles are unable to reach their destinations due to high traffic congestion.

This research proposes a creative approach that is used to identify emergency cars at traffic signals and giving them priority using Artificial Intelligence (AI), and Image Processing and Machine Learning techniques. This would guarantee a quicker response time and increase the efficiency of the emergency vehicles arrival time.

The proposed system would utilize camera-based recognition systems to identify emergency vehicles[1]. Once an emergency vehicle is detected, the system would communicate with the traffic light controller and give it the priority above the other traffic lights. This would allow the traffic light to turn green immediately and allow the emergency vehicle to pass through without having to wait for a long time.



## 1.2 Aims and objectives:

The primary goal of this project is to develop an advanced computer vision-based system that detects and identifies emergency vehicles, particularly ambulances, and coordinates with traffic light control systems to grant them priority passage.

The specific objectives of the research are as follows:

1. Design and develop an image processing algorithm for accurate emergency vehicle detection. The algorithm should be able to identify emergency vehicles in real-time even in challenging conditions.
2. Implement an AI-based model to analyze real-time video feed from cameras installed at traffic lights. The model should be able to detect the emergency vehicles in the video feed and sends this information to the traffic light control system.
3. Develop a solid decision-making system to control traffic light timings based on the presence of emergency vehicles.
4. Evaluate the performance of the proposed system through simulations and real-world experiments. The simulations and experiments should be designed to test the performance of the system under different conditions.
5. Assess the impact of the intelligent traffic light control system on response times for the emergency vehicles. The impact of the system should be measured in terms of how it affects the response times of emergency vehicles.





## **1.3 Methodology:**

We will develop a system that uses a camera to spot ambulance cars. The system will be programmed using OpenCV, Python and machine learning techniques to train the system for detecting and recognizing emergency vehicles[2].

### **1.3.1 Problem Definition and Requirements Gathering:**

The problem is that ambulance vehicles are often caught in traffic lights, which can delay their arrival at the scene of an emergency. The specific system requirement is determining the type of emergency vehicle that we want to recognize and detect which will be the ambulance vehicles in our case.

### **1.3.2 Data Collection:**

The next step is to gather a large dataset of images or video clips that contain the Ambulance vehicles in traffic scenarios. The data should be labeled to indicate the Ambulance vehicle.

### **1.3.3 Model Selection:**

The next step is to choose a suitable computer vision model for object detection. Neural Networks (CNNs) and their variants (e.g., Faster R-CNN, YOLO, SSD) are commonly used for object detection tasks.

### **1.3.4 Model Training:**

The chosen model is then trained on the labeled data, but it is important to train the model for a sufficient amount of time to achieve good performance.

### **1.3.5 Integration with Traffic Lights:**

The final step is to integrate the object detection system with the traffic light control system. This may involve connecting the system to the traffic light controller.

### **1.3.6 Real-time Processing:**

The system should be able to operate in real-time, as emergency vehicle detection at traffic lights requires quick decision-making. This means that the object detection model should be able to process images or video frames quickly enough to generate detection results in a timely manner.

### **1.3.7 Testing and Evaluation:**

The system should be tested and evaluated to ensure that it meets the desired requirements. This can be done by testing the system on a variety of traffic scenarios and with different types of emergency vehicles. The performance will be evaluated by the accuracy of the testing samples.



## 1.4 Team Qualifications:

Table 1 Team Qualifications

Name	Qualifications
Abdulrahman Bin Moharib	Front-End developer / Intermediate in Python, Java / Knowledge in database systems (SQL) / Problem solving
Ahmad Yasser Alroqi	Programming languages (Java, Python, SQL), CyberSecurity, Front End developer, and project development
Khaled Aldossari	Data Science: Pandas, NumPy, scikit-learn / Programming Languages: Python (Intermediate) / Machine Learning Practitioner

## 1.5 Conclusion

In conclusion, what we proposed is a camera-based system that will help the community and will make the way for the emergency vehicles faster.

The proposed intelligent traffic light control system utilizing AI and machine learning for emergency vehicle detection has the potential to revolutionize the way traffic lights prioritize emergency vehicles. By accurately detecting emergency vehicles and granting them priority, the system can significantly reduce response times and improve traffic flow.

In this project we aim to develop a solid and efficient solution through a comprehensive methodology. The successful implementation of this system could have significant implications for emergency service providers and the public, leading to more effective emergency response and better traffic management.



## 2. Literature Review:

### 2.1 Introduction:

Emergency vehicles are one of the most important ways to transfer patients who are in a serious situation and save them, but one of the problems is that they always get stuck in traffic. Therefore, thanks to image detection which our system relies on, by timely identification of emergency vehicles such as ambulances we can detect ambulances and make them reach the accidents or the hospital faster.

Traffic jam takes more than 70% patients' lives in an ambulance but when the patient's condition is very serious the percentage of patient death is increased[3]. Therefore, we are creating a system that will help decrease that percentage.

Thanks to Image detection, there are a lot of systems that have a similar idea to what our system will achieve.

Advances in computer vision and image recognition techniques have opened new possibilities for automating the detection of emergency vehicles using image data.

In this chapter, our focus will be on summarizing the recent studies and existing research in the field of image detection for emergency vehicles. Understanding the challenges in the previous research papers that we will be summarizing, and understanding the methods that we will be using in our work.



## 2.2 Background:

Over the last years, image detection has become one of the important fields in artificial intelligence, which allowed people to overcome a lot of problems by creating a solution for the problem thanks to artificial intelligence. However, the effective management of emergency vehicle traffic encounters several challenges, including congestion, traffic signals, and the unpredictability of road conditions.

Traditionally, the identification of emergency vehicles has relied on sirens and horns, to alert nearby cars and people. While auditory signals are essential, they do not provide a complete solution, particularly in noisy and high traffic environments.

Visual identification of emergency vehicles through image detection technology presents a promising alternative, increasing the possibility of replacing auditory methods with better one using artificial intelligence or image processing.

Despite these promising prospects, researchers face various challenges related to image quality, lighting conditions, traffic jam, and ethical considerations. A whole understanding of the current state of research and the identification of research gaps are essential to advance the development and deployment of image detection solutions in the context of emergency vehicle manageme



## 2.3 Related Work:

In this paper [4], they talked about how it's hard for ambulances in some country who have an over-growing like India to pass through the traffic congestion. So, they proposed a method to solve this problem and in the same time it takes less time compared to other methods. Unlike other approaches that identify specific objects, this method using the surveillance cameras increases the accuracy of detecting ambulances.

In the paper they rely on YOLOv3 and CNN for some reasons including: the cost is low because it can connect 45 cameras per connection, and since its surveillance camera they use, the range is very high, which allows them to see a large amount of traffic in a single scan.

For the implementation part, they divided it to 3 parts: Training, Detection, and Tracking.

-Training: they created a small dataset containing 1000 images, and train the model.

-Pre-processing: the 1000 images were downloaded from various internet sources and given to the model to generate a trained dataset to use for the detection.

-Training: after pre-processing, they started the training using TensorFlow algorithm and Keras open-source library. This process was run 10 times and took about 15 minutes.

-Combining: after finishing the training, the data is combined and optimized using Adam optimizer "which helps to improve the accuracy", and using Fit generator the model is created.

-Detection:

1-Camera sensor captures small clips of traffic.

2-Clips are sent to processor and converted into pictures and saved in a folder.

3-Images in the folder go to YOLO program that detect if the vehicle is truck or not.

4- If it is a truck, then make a copy of the image and add bounding box and save it in another folder called "detected".

5-The image now goes to main function, and the image is processed using their model.

6-If it is an ambulance, return yes to main function.

7- If the returned value is yes, the image will be saved in "final" folder.



-Tracking: After detecting the ambulance, based on its location, the traffic light for the ambulance will turned to "Green" and all other traffic light to "Red" for 15 seconds until the next detection.

The result:



*Figure 1 Ambulance detected by the camera*

In this paper[5], they talked about using the method YOLOv3 which we are consider using for our project, which is one of the best object detection methods. Based on their proposal, their objective is to detect objectives using YOLO (You Only Look Once), which has some advantages over the other methods. So in CNN (Convolutional Neural Network) the algorithm will not look at the pictures completely, unlike YOLO. Also YOLO is faster when we compare it to the other algorithms. This algorithm is able to achieve high accuracy while running in real time.

Anchor box: detect more than one object in the same image using Bounding Boxes.



*Figure 2 Using anchor box to detect*

Training:

The training sets includes: (ImageNet, COCO, VOC) but some images were not perfectly tested for some reasons including:

- Instability of the camera, some images are blurred
- Some images are blocked by other objects
- Some images may have a low resolution, or bad quality due to the bad weather



*Figure 3 Image problems, i.e. blur*

They used Google Colab for the training, which is an excellent solution to get a better GPU and faster results, and they were able to use Tesla K80 GPU.

### Performance of Algorithms:

mAP (Mean Average Precision): IoU threshold of 0.5 i.e. 50%, the mAP is 98.14%

IoU (Intersection Over Union): confidence threshold 0.25, the average IoU is 83.19%

F1 Score (calculated from Confusion Matrix which gives us prediction results, number of correct and incorrect predictions): For confidence threshold 0.25, the f1-score is 0.94

Now after putting the boxes to the algorithm, this is the result without filtering the boxes:



*Figure 4 Result without Filtering algorithm*

In this image now, each cell has 5 anchor boxes, so the total predictions for the model is  $19 \times 19 \times 5 = 1805$  boxes. So here the filtering algorithm plays an important rule. By using the filter algorithm, we are expecting a smaller number of object detected and there are two steps:

- Remove boxes that has a low score because they are most likely not objects
- Choose only one box from the overlapping boxes that detect the same object



*Figure 5 After step two*



*Figure 6 After step one*





And finally here are some results:



*Figure 7 Result from camera*



In this paper [6], the authors aimed to enhance the detection of ambulance vehicles using a straight forward camera-based system.

#### 1. Camera-Based System:

- The idea of their project lies in the using of a basic camera, like those found in smartphones, as the primary hardware. This camera was set up to capture real-time footage of vehicles on the road, and this footage was the foundation of their vehicle detection system.

#### 2. Data Preprocessing:

- The project involved the conversion of video data into individual frames. Each frame represents a single image capturing a moment in the video. This step is critical because it simplifies the process of analyzing and identifying vehicles within the video.

#### 3. YOLO Algorithm:

- To detect vehicles within these frames, the authors used the YOLO (You Only Look Once) algorithm, which is known for its ability to quickly and accurately identify objects within images or frames. YOLO breaks down the image into a grid and determines the likelihood of objects existing within each grid cell, making it efficient for real-time applications.

#### 4. Convolutional Neural Network (CNN):

- Once YOLO pinpoints a vehicle in a frame, the project's Convolutional Neural Network (CNN) comes in to categorize it. The CNN has been specifically trained to determine if the identified vehicle is indeed an ambulance. CNNs are good in image recognition tasks and have been widely used for various applications, including object classification.

#### 5. Training Phase:

- The team needed to "train" the system to distinguish between different types of vehicles, a process important for its effectiveness. They used a dataset comprising about a thousand images of various vehicles, including ambulances.
- Before training, these images were preprocessed to standardize and enhance their quality, ensuring the system could learn effectively.
- TensorFlow and Keras, which are popular deep learning frameworks, played an essential role in training the CNN. These tools facilitate the process of teaching the computer system to recognize patterns and features that distinguish ambulances from other vehicles.



#### 6. Detection Phase:

- The detection phase involves several sequential steps:
- First, the video footage was divided into individual frames to simplify analysis.
- YOLO was then utilized to identify vehicles within each frame, outlining their positions.
- Finally, the CNN was used to classify these vehicles, determining whether they were ambulances or not.

#### 7. Accuracy:

- One of the achievements of the project is its high level of accuracy. Their system successfully identified ambulance vehicles with an accuracy rate of 84%. This means that it can reliably recognize ambulances in most instances, making it a valuable tool for quick response and prioritization during emergencies.

This project's simplicity, effectiveness, and accuracy underscore its potential to significantly improve road safety and emergency response. By integrating readily available technology like cameras with advanced machine learning algorithms, these researchers have showed the practical applications of computer vision in addressing real-world challenges.



In this paper [7], they try to reduce the problems by introducing an OCR algorithm that will detect the offending vehicle number plate and fine the drivers. The sensors were calibrated together to achieve spatial and temporal alignment and reduce sampling errors, and a statistical filtering algorithm was added to remove point cloud outliers and interference.

The main purpose of the system is to detect the vehicles through image instead of sensors. There will be a camera installed in front of the ambulance that will record the vehicle's license plate number. The captured footage is then converted into images and the YOLO algorithm is used to extract the vehicles. The extracted image is then processed by an OCR algorithm to detect and store vehicle details.

There are 5 modules.

#### 1-Videos processing module

In this module, the real time camera is used to get input image, which is installed in front of the ambulance, the camera will capture live video streaming and convert the video into image frames by extracting the image from video.

#### 2-Image conversion module

Image conversion is the conversion of the true color image RGB to the grayscale image. The image frame that is extracted from the video input will be in the RGB image. These RGB images will be converted to grayscale by eliminating the background images and saturation information by white and black color conversion.

#### 3-Vehicle detection module

The module is used to detect the vehicle and the type of the vehicle.

#### 4-Number plate recognition

In this module, the detected vehicle using Yolo algorithm is segmented and recognize the number plate using OCR (Optical Character Recognition) algorithm and the recognized number plate will be stored in a file for the further use.

#### 5-Database module

A database is an organized collection of image and train dataset information, or data, typically stored computer system.

In the previous paper, the vehicles which block the way for the emergency vehicles like ambulance and fire fighting vehicles, those vehicle's number plate will be identified if the vehicle doesn't give way to the emergency vehicle for more than 60 seconds. The identified number plate will be stored in the database so that the corresponding vehicles will be charged fine.



In this paper[8], the authors address the problem of lane detection and tracking in challenging conditions. Lane detection and tracking are important tasks in autonomous driving systems, advanced driver-assistance systems (ADAS), and computer vision applications related to intelligent transportation systems.

The authors develop an algorithm that uses image processing techniques and machine learning algorithms to enhance the accuracy and robustness of lane detection and tracking systems. The following are the methods used for this problem:

1. **Preprocessing:** The input image frames are preprocessed to enhance the lane markings and remove noise. Techniques such as grayscale conversion, contrast enhancement, Gaussian filtering, and morphological operations are used to improve the quality of the input images.
2. **Lane Detection and Feature Extraction:** The preprocessed image frames are then segmented to extract lane markings. The authors use a combination of techniques such as edge detection, Hough transform, and region of interest (ROI) extraction to identify lanes in the image.
3. **Lane Tracking:** The authors employ a tracking algorithm based on the Kalman filter. The Kalman filter predicts the lane positions in each frame and adjusts the predictions based on the actual lane detections, improving the accuracy and robustness of lane tracking.

The authors have tested the presented system on various condition based on the detection rate, and they have gained this result: in clear daytime average lane detection rate is 98.07%. In the rainy day, the average lane detection rate is 96.85%. In the snowy day, the average lane detection rate is 95.29%.



In this paper[9], the main challenge addressed is the detection of traffic signs under complex and challenging conditions, particularly during nighttime and bad weather. It is making detection models that may struggle in these scenarios due to poor visibility and lighting conditions.

The algorithm is based on the YOLOv5s-A2 architecture, which is an enhancement of the YOLO (You Only Look Once) model. Path Aggregation Module (PA) enhances multi-scale feature representation, improving detection accuracy. Attention Detection Head Module (ADH), to decrease aliasing effects in cross-scale fusion, ADH attention to limit prediction information, aiming to enhance feature representation. The strategy for data augmentation to increase the number of category instances is also designed to improve the model performance without sacrificing inference speed.

The model achieves high precision and fast detection of traffic signs, even with a minimal increase in the number of parameters. The proposed algorithm demonstrates improved performance, particularly in challenging scenarios, making it a more effective solution for real-world traffic sign detection.

Table 2 Comparison Table

Paper title	Objective	Method	Contributions	Limitations
Paper [4]	Using the surveillance cameras to detect ambulances	YOLO/CNN	Proposed a method to detect the ambulance using YOLO and CNN	Lack of information on system's real-world testing
Paper [5]	Real time object detection	YOLOv3	Achieved high accuracy with YOLOv3 in real-time object detection	Limited information on specific use cases or scenarios
Paper [6]	Detection of Ambulance Using Computer Vision	YOLO/CNN	Successfully detected ambulances with 84% accuracy	Limited insight into the system's scalability or adaptability
Paper [7]	Ambulance Blocking Vehicles Identification By Artificial Intelligence	Not specified	Introduced an OCR algorithm for identifying vehicles blocking ambulances	No details on the OCR algorithm or its accuracy
Paper [8]	Vision-Based Robust Lane Detection and Tracking in Challenging Conditions	Kalman Filter	Enhanced accuracy and robustness in lane tracking using the Kalman filter	Limited information on the model's performance in diverse road conditions
Paper [9]	Faster Light Detection Algorithm of Traffic Signs	YOLOv5s-A2	Improved traffic sign detection in challenging conditions with YOLOv5s-A2	Limited information on scenarios where the model might struggle



Table 2 shows the main objective of each research paper. These papers collectively explore diverse objectives utilizing YOLO, YOLOv3, YOLOv5s-A2, CNN, and Kalman Filter methods, and it shows the contributions and limitations of the papers, it also shows how to dedicate the advanced systems in traffic management.

## 2.4 Conclusion:

For this chapter, we talked about how important AI in our lives nowadays and how it will change a lot of things including the way to identify the emergency vehicles.

We also summarized some papers regarding our project and our idea and we discussed the methods they used in their researches, and finally we compared the papers that were relative to our idea and the methods that was used.

In conclusion, we found that using YOLO is the best approach. The next chapter, we will talk about the methodology, datasets, and how we can build and train our model.





## 3. System Analysis:

### 3.1 Introduction:

In the previous chapters, at the beginning we discussed and showed the main problem with the emergency vehicles in traffic jams, outlining aims, challenges, and objectives emphasizing the use of AI and image processing to detect emergency vehicles.

In the second part Literature review, we discussed the methodology, outlining the steps, including the problem definition, data collection, model selection, training, integration with traffic lights, real-time processing, and testing and evaluation.

We also showed that image detection in artificial intelligence has become essential for providing solutions for the challenges in traffic management. However, unlike traditional reliance on auditory signals like sirens, visual identification through image detection technology offers a better solution and promising alternative. Then we made a comparison table that summarizes objectives, methods, contributions, and limitations of related research papers using YOLO, YOLOv3, YOLOv5s-A2, CNN, and Kalman Filter.

In this chapter, we will specify the Software Requirements Specification (SRS) including the user and system requirement and the functional and non-functional requirements for each of them.



## 3.2 Dataset

The dataset that will be used for this model contains nearly 4700 images that will be split into training and testing sets, these images are collected from various dataset websites including "Ambulance last generate"[10] , " annotated Ambulance images"[11], and "Emergency vs Non-Emergency Vehicle Classification"[12], these images are companied between an ambulance image, and normal vehicle images.

## 3.3 Software Requirements Specification

This section will specify the steps of the model and the tasks it will operate. Software Requirements Specification (SRS) for “Traffic Light Detection Making the Way for Lifesavers” outlining the functional and nonfunctional requirements for the development of the system. This document serves as a reference to understand the project's scope and features.

## 3.4 User Characteristics

In this section, we will provide a general overview of the user characteristics of the system. Specify the unique characteristics that influence their interactions with the system. The primary categories of intended users and their associated characteristics are as follows.

### General Public

- **Educational Level:** Users within this group may have a wide range of educational backgrounds, from high school graduates to individuals with higher education degrees.
- **Experience:** Most users in this category are unlikely to have professional experience in emergency medical services or dispatching.
- **Technical Expertise:** The level of technical proficiency varies, with users typically possessing basic to moderate skills in using smartphone applications.



#### App Administrators:

- **Educational Level:** Educational backgrounds among app administrators can vary, but they often relate to IT or system administration.
- **Experience:** App administrators have experience in app administration and user support, ensuring the proper functioning of the application.
- **Technical Expertise:** They possess advanced technical proficiency, enabling them to maintain and troubleshoot the application's infrastructure. operates effectively including General Department of Traffic.

#### Operators, City or Traffic Authorities:

- **Educational Level:** Individuals in this category typically have educational backgrounds in urban planning, traffic management, or city administration.
- **Experience:** They have experience in policymaking and city planning to enhance traffic management.
- **Technical Expertise:** They are experts in understanding traffic policies and rules, contributing to informed decisions in traffic management.



## 3.5 User Requirements

### 3.5.1 Functional Requirements:

1. Ambulance detection: The system should be able to detect emergency vehicles, upon detection, the system shall identify the approaching ambulance.
2. Vehicle Counting: The system shall continuously count the number of vehicles passing through designated the traffic lights.
3. Vehicle Recognition: The system shall be capable of identifying and differentiate between vehicle and emergency vehicles, such as ambulances, within the counted vehicles.
4. Real-Time Processing: The system should be able to get the live video from the traffic light and process it in real time.
5. Integrating with traffic lights: The system should consistently integrate with existing traffic light control infrastructure to avoid disruptions in traffic management.
6. Estimated Time for Arrival: The system should estimate the arrival time of the approaching emergency vehicle.
7. Recognition of Ambulance Departure: The system shall recognize when the ambulance has safely cleared the monitored point.
8. Override Capability: In cases of technical failure or system malfunctions, there should be a manual override option for traffic lights to ensure traffic management continuity.
9. Integration with Emergency Dispatch Centers: Enable communication between the system and emergency dispatch centers to provide real-time updates on the vehicle's status and route.



10. **Compatibility:** The system should be compatible with a variety of emergency vehicles, regardless of make or model
11. **Reliability:** The system must be highly reliable, ensuring that it correctly identifies emergency vehicles and grants them priority without false positives or negatives.
12. **Traffic Monitoring and Reporting:** The system should offer tools for monitoring traffic conditions and generating reports on the performance of the traffic light control system.
13. **Emergency Service Coordination:** The system should have the capability to communicate with local emergency service centers, enabling them to request traffic light priority when responding to incidents.



### 3.5.2 Non-functional Requirements:

#### Performance

- Improve performance by using computers with high RAM and high processors.
- The system must support the possibility of being done periodically and automatically.

#### Security

- The system should be secure and not able to modify by outsiders.

#### Usability

- The ability to support Arabic and English in the system interface.

#### Availability

- The system should be available all the time.

#### Correctness

- The system should have correct data and information.

#### Confidentiality

- Preserve information access control and disclosure restrictions.

#### Integrity

- Avoid improper modification or destruction.

#### Availability

- The information must be available to access and use all the time.

#### Portability

- The software shall be deployed on any machine.

#### Quality Control

- Another crucial necessity is system quality control.
- All users should be served quickly and efficiently by the system.

## 3.6 System Requirement

In this part, we will show some diagrams and describe the system requirements and show how the model will interact with the other component including the traffic light.

### 3.6.1 Use Case Diagram:

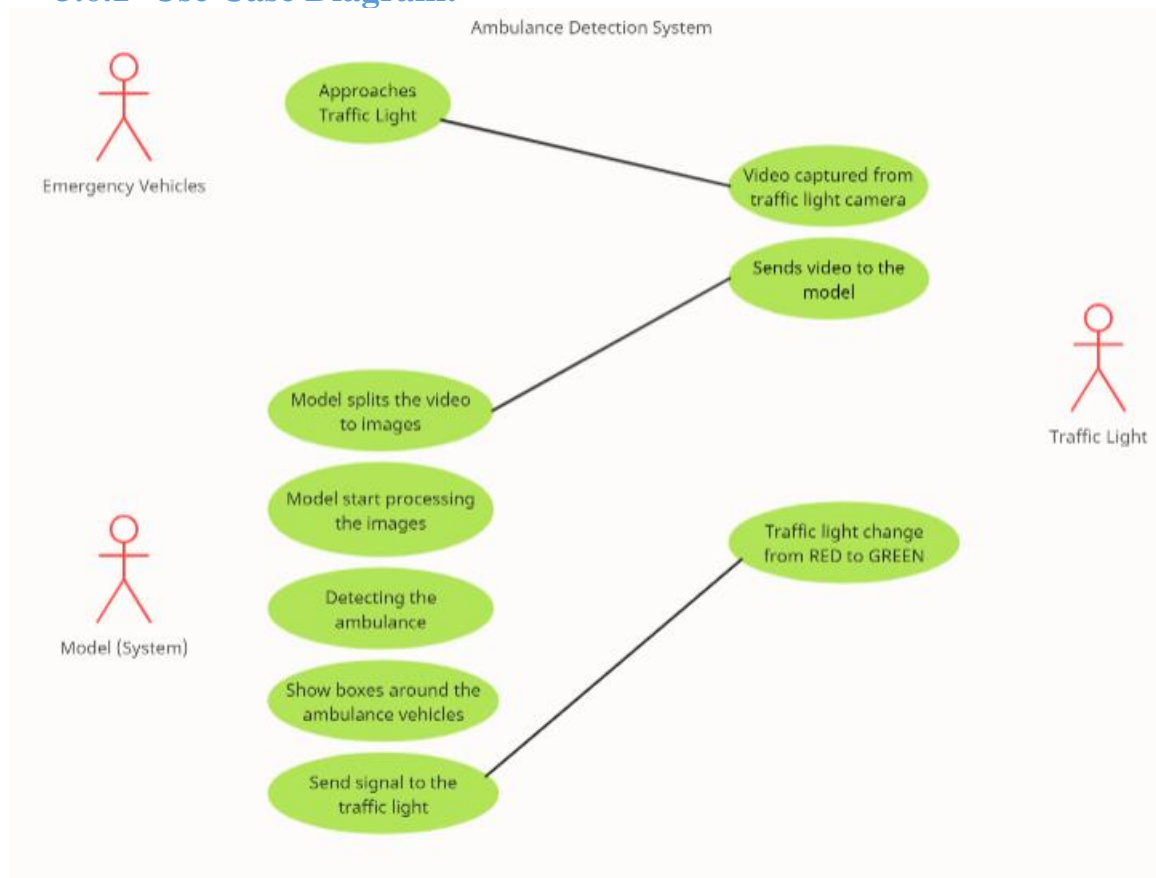
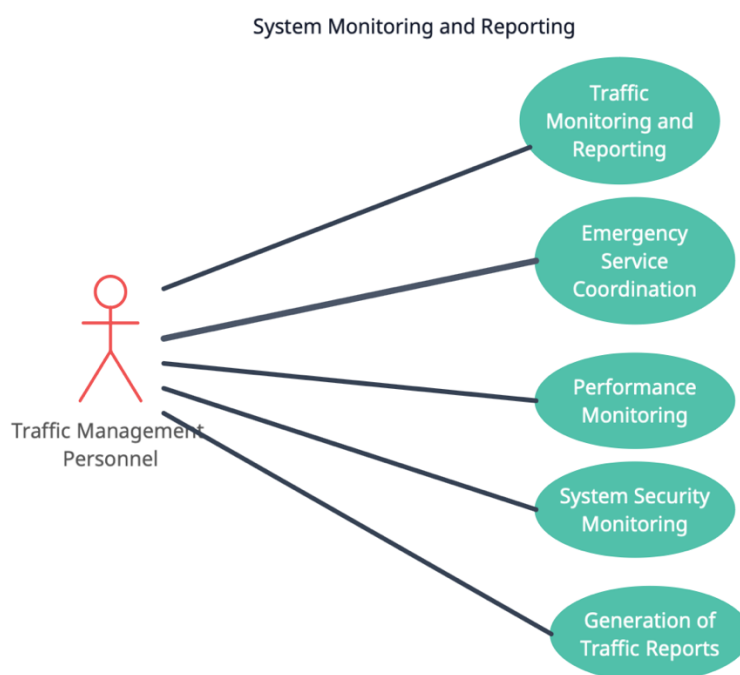


Figure 8 Use Case Diagram for the Model

The use case diagram [Figure 8] shows the main actors in the system are the ambulance driver, the ambulance detection system, and the traffic light system. It will show the following case, the connection between the traffic light and the system and how they interact with each other after the emergency vehicle approached the traffic light. Firstly, the video that has been taken from the surveillance camera will be sent to the model and split into images, which then will be processed by the model and when it detects the ambulance it will send back a signal to notify the traffic light to turn GREEN.



*Figure 9 Use Case Diagram for System Monitoring*

The use case diagram [Figure 9] shows how different actor can collaborate with the system to monitor traffic conditions, manage emergency responses, and maintain effective traffic management, ensuring that ambulances and other emergency vehicles can reach their destinations quickly and safely.



### 3.6.2 Sequence Diagram:

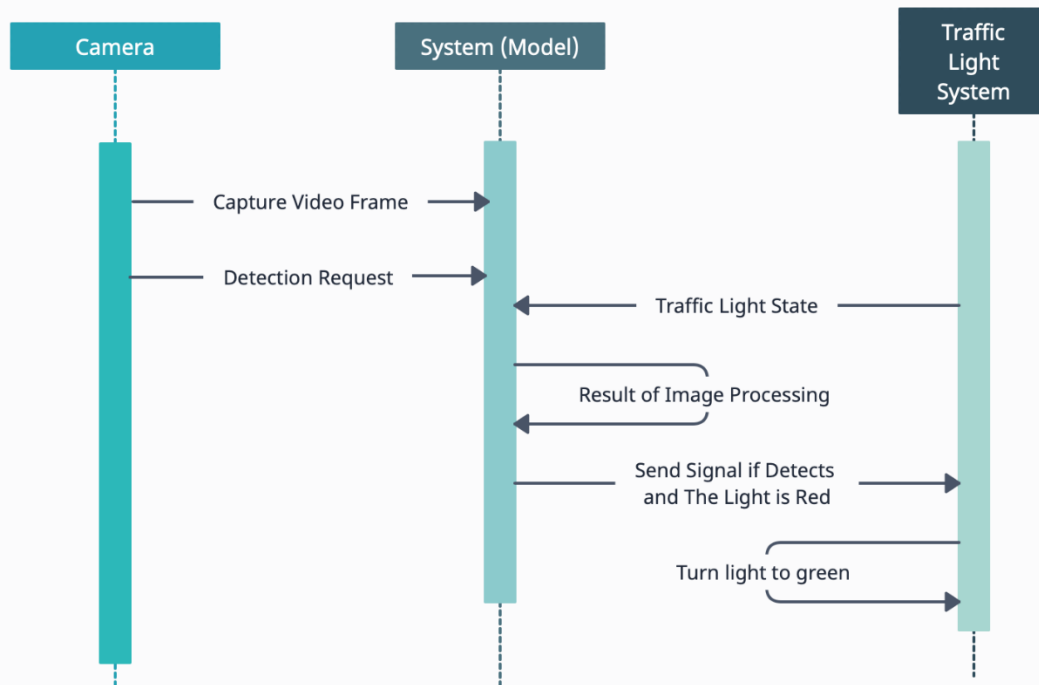
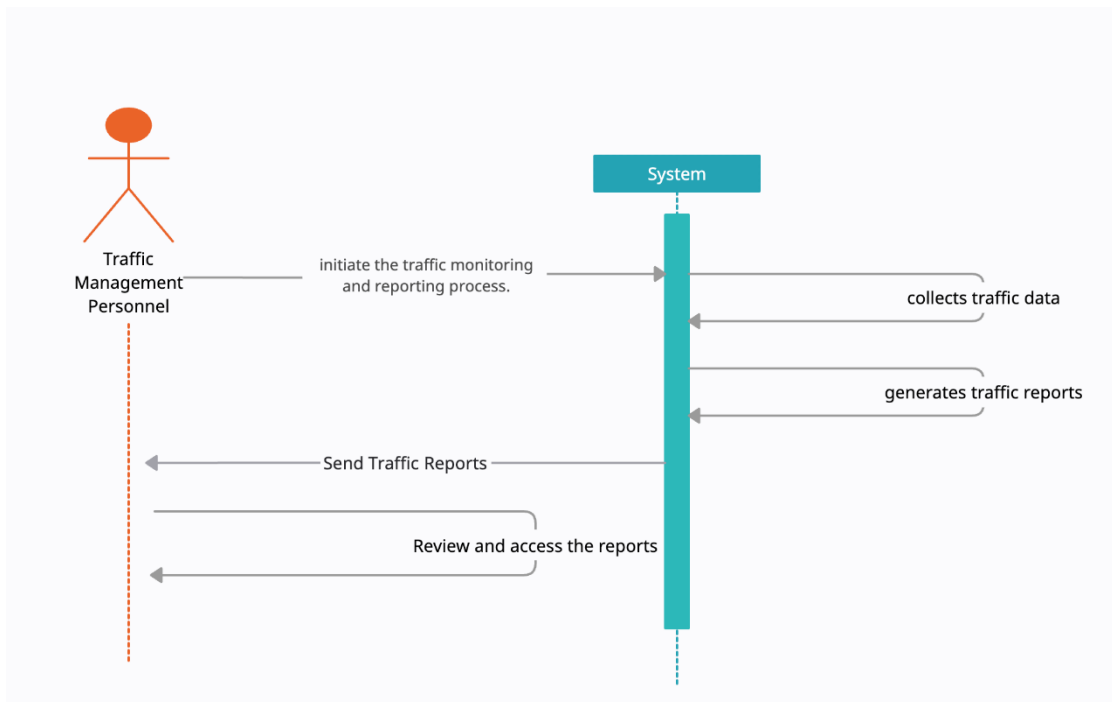


Figure 10 Sequence Diagram for the Model

The sequence [Figure 10] begins with the camera capturing a video frame. The camera then sends a detection request to the traffic light system. After that, the traffic light system then uses image processing to analyze the video frame and identify any vehicles that are present. If the traffic light system detects any vehicles, it will send a signal to the traffic light to turn the light green.



*Figure 11 Sequence Diagram for Traffic Management*

The sequence diagram [Figure 11] shows the process of traffic monitoring and reporting. Lustrating the sequence of interactions between the System management control and the Traffic Light System for monitoring the system's status and generating reports. It allows the administrator to stay informed about the system's performance and traffic conditions, which is crucial for effective traffic management and responding to incidents involving emergency vehicles.

### 3.6.3 Activity Diagram:

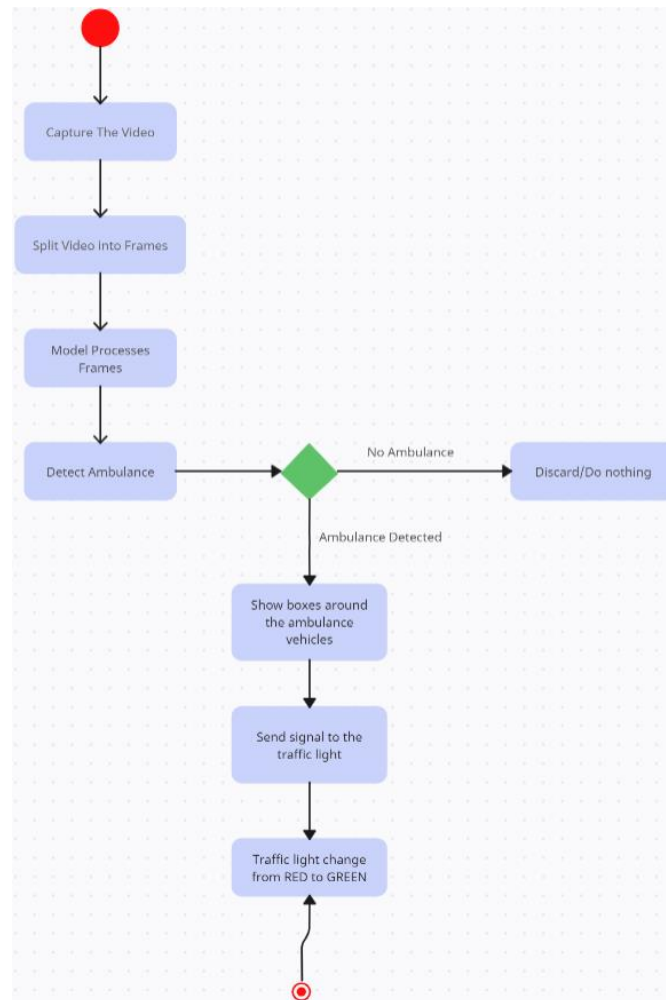


Figure 12 Activity Diagram

This activity diagram [Figure 12] shows the steps of the system starting from capturing the video and splitting in into frames so that the model can start the process of the detection. Secondly, there will be two outputs, either the model will detect an ambulance or not. If detected, it will put boxes to show the ambulance vehicle and send a signal to the traffic light so it can give the priority for the traffic light that has the ambulance to go green before the other traffic lights, and if it didn't detect it will do nothing.



### 3.7 Project Management Plan:

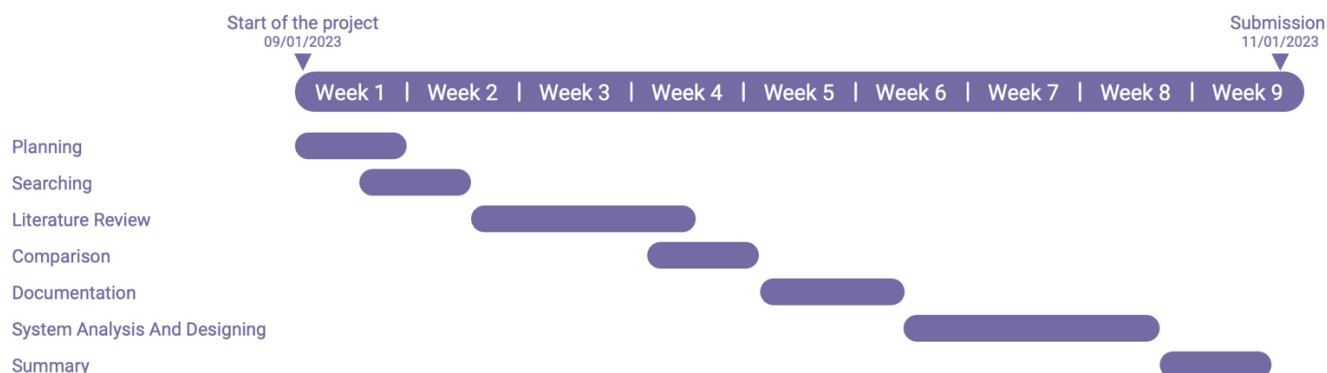


Figure 13 Gantt Chart for the Project Plan

### 3.8 Conclusion:

In conclusion, in this chapter we began by reviewing what we documented so far, following by specifying the dataset and where we will collect them from. Then, we discussed the Software Requirements Specification, beginning with providing an overview of the user characteristics by showing the targeted users and how they can benefit from it, also showing administrators the operators and how they can keep the system effective. After that, we showed the Functional and Non-Functional requirements and gave a list for each one of them with a brief explanation for each requirement. Also, we created a diagram to show how the system will be working and how it can connect with the model and get the information from it. Finally, we showed our progress in this project by providing a Gantt Chart with the dates of our progress.



## 4. Design Chapter

This chapter provides a brief overview of the design methodologies, laying the groundwork for detailed discussions on each of the model and the application. We will explain with diagrams and algorithm of project showing the pseudocode.

### 4.1 System Architecture

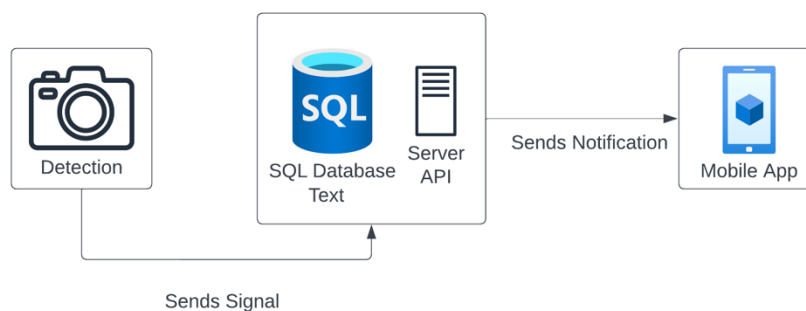


Figure 14. System Architecture

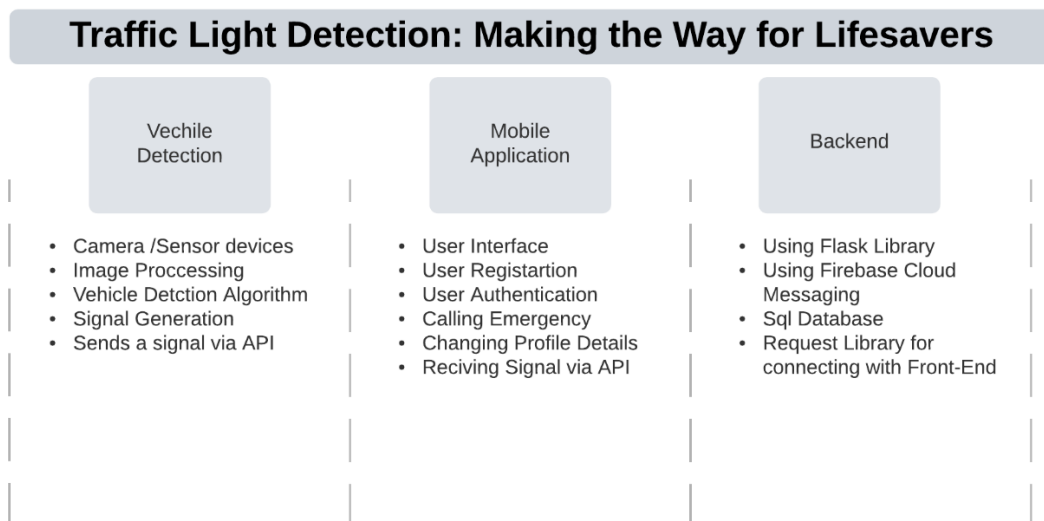


Figure 15. System and Sub-Systems



### **Vehicle Detection**

- Components:
  - Camera/sensor devices
  - Image processing module
  - Vehicle detection algorithm
- Functionality:
  - Captures images or data from cameras or sensors.
  - Processes the data using image processing techniques to identify vehicles.
  - Utilizes a detection algorithm to recognize and locate vehicles.

### **Mobile Application Sub-System:**

- Components:
  - User Interface (UI)
  - Signal reception module
  - User registration/authentication
- Functionality:
  - Displays information about detected vehicles through a user-friendly interface.
  - Receives signals or notifications from the detection system.
  - Manages user registration and authentication for secure access.

### **Back End:**

- Components:
  - Python Code using Flask
  - Sql database
  - Firebase Cloud Messaging
- Functionality:
  - Connecting with the the application (Front End)
  - Getting login and signup credentials

This breakdown covers the key components and functionalities of the system for vehicle detection and signal transmission through a mobile application. Depending on specific requirements, additional features or sub-systems may be necessary.



## 4.2 Database Design

The database comprises a "User" entity with two key attributes: "Username" and "Password." The "Username" attribute, stored as a string, serves as a unique identifier chosen by the user. Meanwhile, the "Password" attribute, also a string, stores the hashed representation of the user's password to ensure data security. The hashed passwords are stored, employing secure hashing algorithms. In this simplified representation, the "Username" is designated as the primary key. This database structure is commonly employed in basic user authentication systems, prioritizing the secure storage of user credentials. While this description encapsulates the fundamental elements of a user authentication database, practical implementations may include additional fields, such as email addresses or creation dates, to enhance user management functionalities.

## 4.3 Modular Decomposition

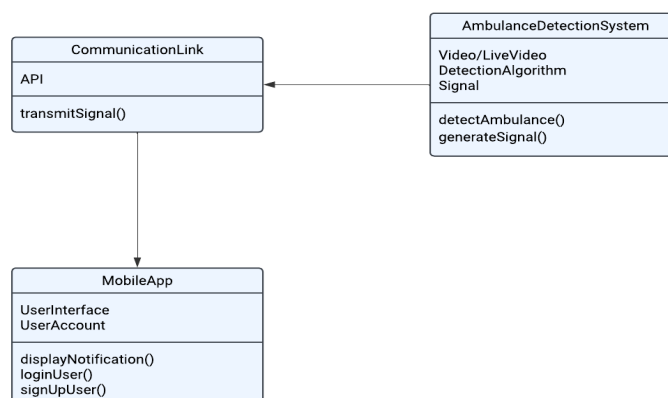


Figure 16. Object Oriented Design Diagram

[Figure 16] is an OOD diagram shows the main component which starts from the model detection to the back-end code and lastly to the front-end (Notify Me) Application.

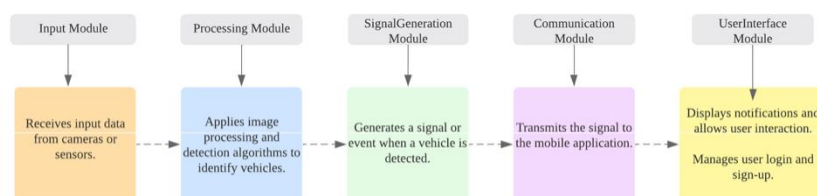


Figure 17. Data Flow Diagram

[Figure 17] is a Data Flow Diagram which shows the flow of the system starting from the input Module all the way to the output Module which is the User Interface, it start from receiving the video (Input Module) from the surveillance camera and passes it to the model to start detecting (Processing Module), in case it detects an ambulance, it will transmit a signal (SignalGeneration Module) to the back-end of the application (Communication Module), and finally the back-end will send the signal to the front-end (User Interface Module) which will then pop up the notification.



## 4.4 System Organization

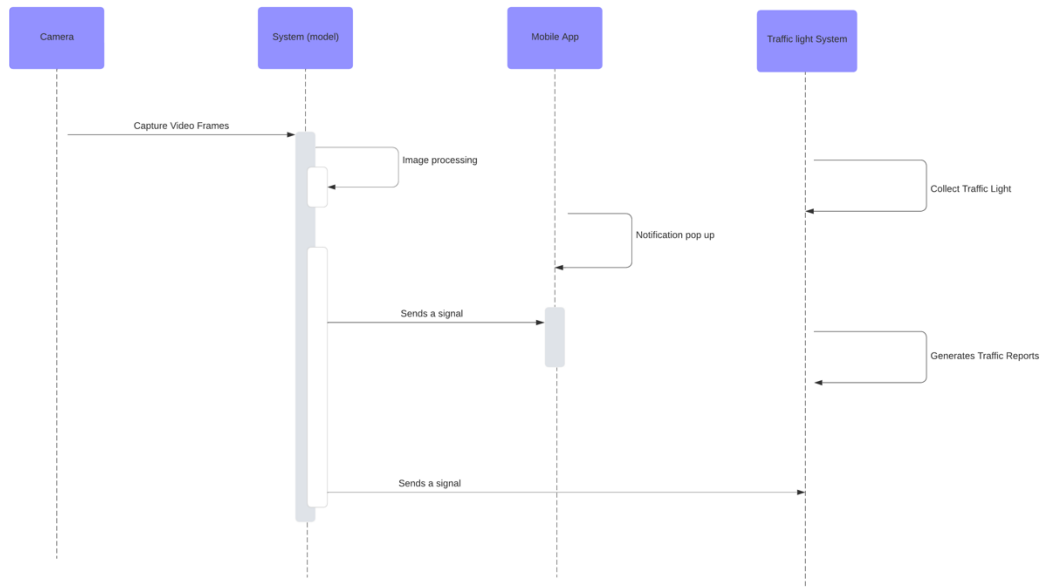


Figure 18. Sequence Diagram for Model and Application

The sequence diagram [Figure 18] shows the process of traffic monitoring and reporting. Lustrating the sequence of interactions between the Camera and the System (Model) and the mobile application. It allows the application to get the signal and pop up the notification, while the traffic light system also receive a signal to let the traffic light go GREEN.

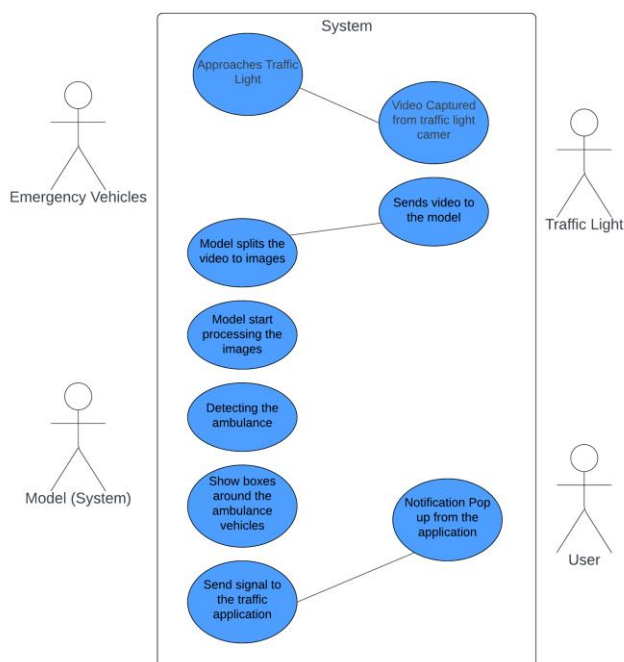


Figure 19. Use Case Diagram

The use case diagram [Figure 19] shows the main actors in the system are the user of the application, the ambulance detection system, and the traffic light system. It will show the following case, the connection between the traffic light and the application to receive the signal the system and how they interact with each other after the emergency vehicle approached the traffic light. Firstly, the video that has been taken from the surveillance camera will be sent to the model and split into images, which then will be processed by the model and when it detects the ambulance it will send back a signal to the application and notify the user.

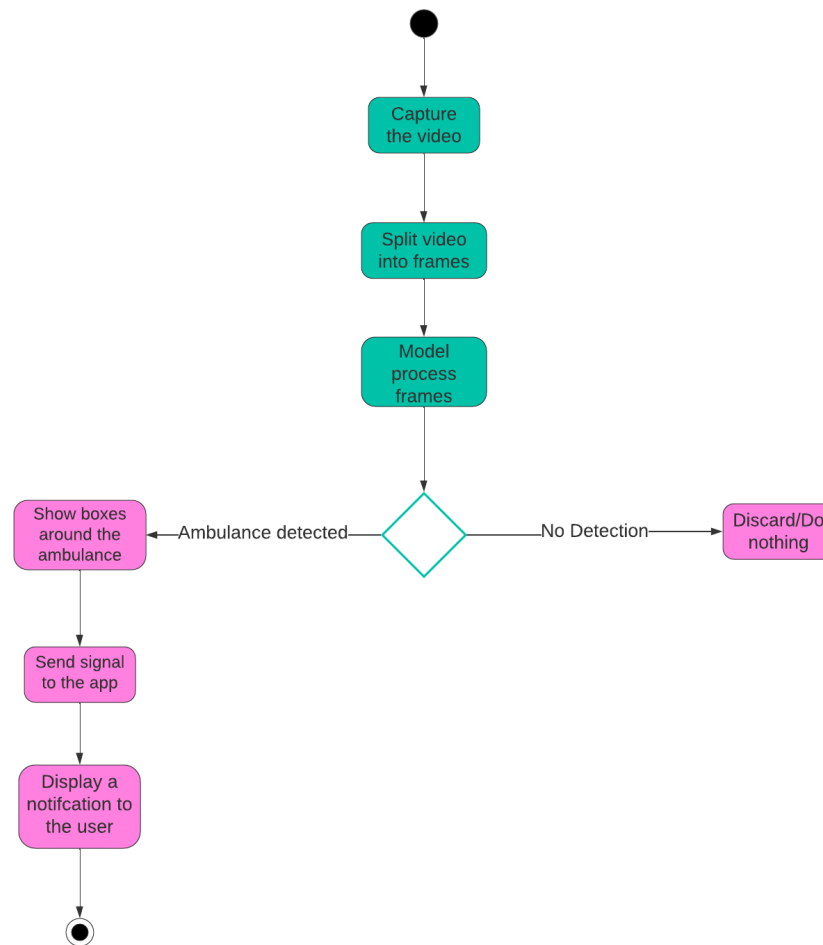


Figure 20. Activity Diagram for the Application

This activity diagram [Figure 20] shows the steps of the system starting from capturing the video and splitting in into frames so that the model can start the process of the detection. Secondly, there will be two outputs, either the model will detect an ambulance or not. If detected, it will put boxes to show the ambulance vehicle and send a signal to the app so it can notify the users that there an incoming ambulance vehicle, and if it didn't detect it will do nothing.



## 4.5 Algorithm

### Pseudocode:

Input: Video stream containing traffic

Output: Processed video stream with ambulance detection

1. Set up video input and output paths:
  - a. Define the directory for videos.
  - b. Specify the input video file path.
  - c. Define the output video file path.
2. Initialize video capture and output writer:
  - a. Open the video file using OpenCV's VideoCapture.
  - b. Read the first frame to get its dimensions (H, W).
  - c. Initialize an output video writer using OpenCV's VideoWriter.
3. Set up YOLO model:
  - a. Define the path to the YOLO model.
  - b. Load the YOLO model using the Ultralytics library.
4. Set the confidence threshold for object detection.
5. Process each frame in the video:
  - a. Retrieve the next frame from the video capture.
  - b. Perform object detection using the YOLO model on the current frame.
  - c. Extract results, including bounding boxes and confidence scores.
6. Process detected objects:
  - a. Find the object with the highest confidence score.
  - b. If the confidence score is above the threshold:
    - i. Draw a bounding box around the detected object.
    - ii. Display the class name above the bounding box.
    - iii. Check if the detected object is an ambulance:
      - Print a message or take appropriate action.
      - Send a notification to a predefined URL (in this case, an NGROK endpoint).
  - c. If no object is detected:
    - i. Draw a placeholder box indicating "NO OBJECT."



7. Write the processed frame to the output video.

8. Release resources:

- a. Release the video capture and output writer.
- b. Close all OpenCV windows.

9. End the process.

Note: The pseudocode focuses on the high-level steps and logic, abstracting away specific implementation details and Python syntax. It provides an overview of the algorithm's structure and flow.



## 4.6 Alternative Design Methods

There was Alternative design, we chose YOLOv8 because it's considered the best version of YOLO compared to the older versions. For the mobile app flutter framework and dart was the best option instead of Swift, there are couple reasons for not choosing Swift.

This comparison table to show the differences:

Table 3. Comparison Table Between Flutter and Swift

Feature	Swift	Flutter
<b>Purpose</b>	iOS, macOS, watchOS, tvOS app development	Cross-platform app development (iOS, Android, Web, Desktop)
<b>Language</b>	Swift	Dart
<b>Development Environment</b>	Xcode (Official IDE)	Visual Studio Code, Android Studio, IntelliJ IDEA
<b>User Interface</b>	Native iOS components	Customizable widgets for consistent UI across platforms
<b>Cross Platform Development</b>	iOS only	Cross-platform support with a single codebase
<b>Community And Ecosystem</b>	Strong community and extensive resources	Growing community with active support and ongoing development
<b>Hot Reload</b>	Live preview with some limitations	Hot reload for instant code changes without restarting
<b>Learning Curve</b>	Easy to learn with a familiar syntax	Learning Dart may have a steeper curve for some developers
<b>Customization</b>	Limited customization for cross-platform UI	Highly customizable UI with Flutter widgets



Also, we could've chosen Web Socket instead of API, but it's more complicated and API is straighter forward to implement.

This comparison table to show the differences:

*Table 4. Comparison Table Between API and WebSocket*

<b>Feature</b>	<b>RESTful API</b>	<b>WebSockets</b>
<b>Communication Type</b>	Synchronous (HTTP requests)	Full-duplex communication
<b>Data Format</b>	Typically JSON or XML	Custom data formats (often binary)
<b>Connection Establishment</b>	Stateless, on-demand	Persistent connection
<b>Real-time Communication</b>	Limited real-time capabilities	Real-time, bidirectional communication
<b>Ease of Implementation</b>	Easier to implement for basic use cases	Requires more complex implementation
<b>Scalability</b>	Easier to scale horizontally	Slightly more challenging with persistent connections
<b>Security</b>	Relies on HTTPS for secure communication	Requires additional security considerations



## 4.7 Graphical User Interface Design

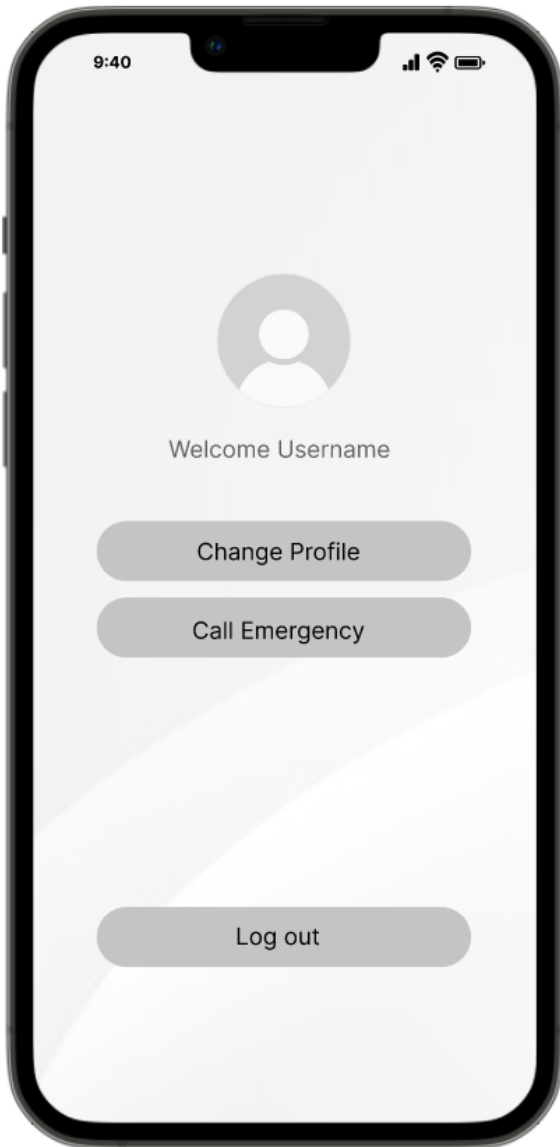


Figure 21. Profile Page Prototype

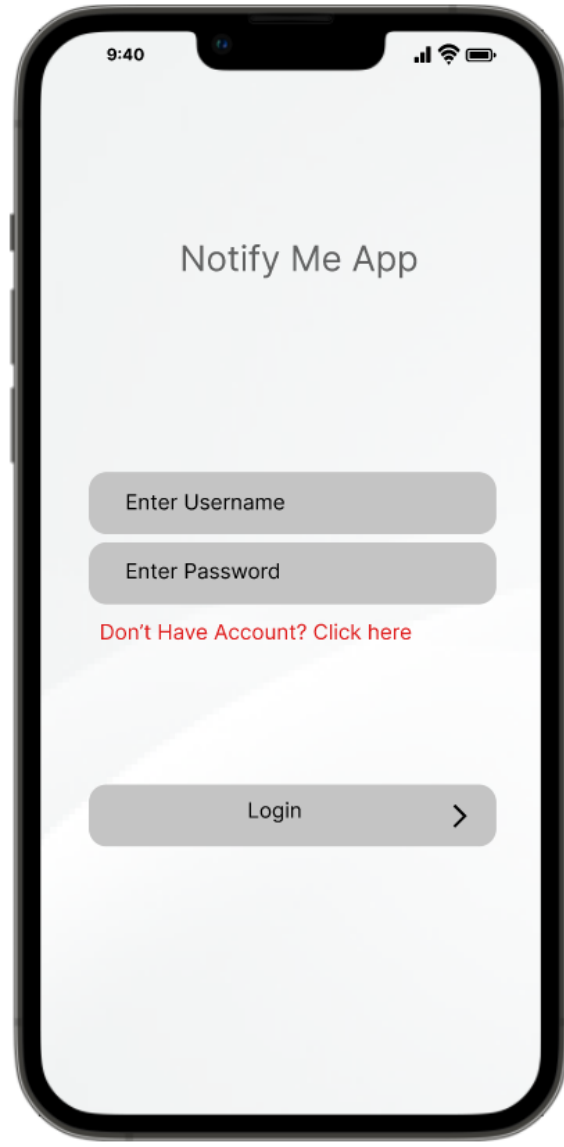
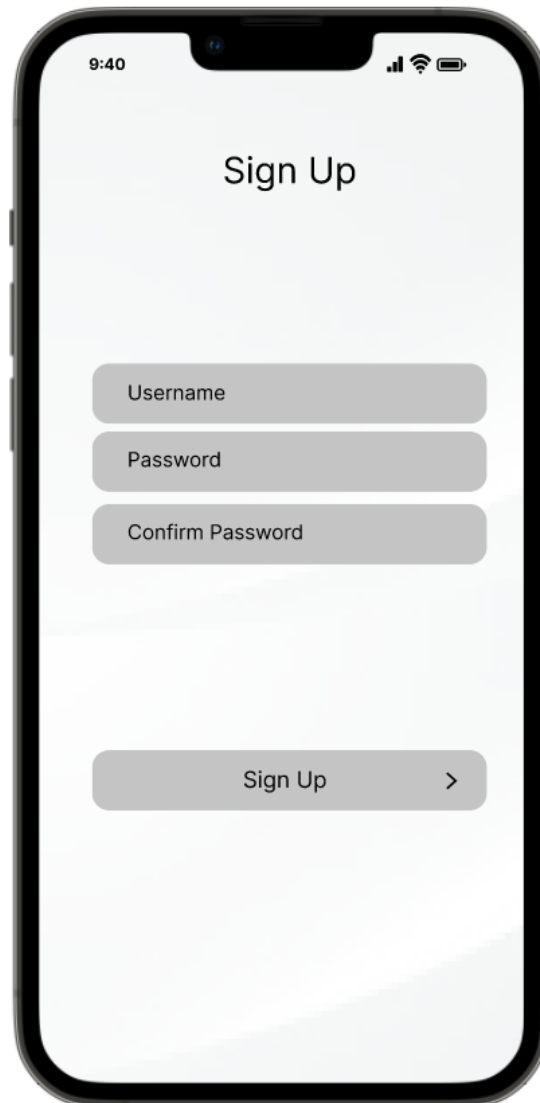


Figure 22. Login Page Prototype





*Figure 23. Signup Page Prototype*

This prototype was designed using Figma [13] and it shows the login and signup and the home page, the colors we chose are bright and not sharp for the ease of use.



## 5. Implemantion Chapter

In this chapter we will show the content of our system, the critical parts of the code, and how the model

### 5.1 Implementation Requiremnts

#### 5.1.1 Hardware Requirements

The system contains two sub-systems:

- 1- The YOLOv8 model, which was trained using Google Colab Tesla T4 GPU it can delivers enhanced performance with faster training session, making it well-suited for the model training. The training was done on 100 epochs and took nearly 2 hours.
- 2- The mobile app, which is an application was built using Flutter and Dart, and coded using Visual Studio Code, so it doesn't require a strong hardware to run.

#### 5.1.2 Software Requirements

The software requirements for the Our system encompass various components, each chosen for specific functionalities within the architecture.

For the Flask backend written in Python, the “Flask framework” was selected due to its lightweight nature and flexibility, making it well-suited for the development. Additionally, SQLite was chosen as the database engine, providing a self-contained, serverless, and easily deployable solution, fitting the scale and requirements of the project.

Integrating “Ngrok” into Our project for API is a practical choice during development. Ngrok securely exposes the local Flask API to the internet, allowing real-device to connect.

To facilitate Firebase integration within the Flask backend, the pyfcm library was employed. This Python library enables seamless interaction with Firebase Cloud Messaging (FCM), allowing the backend to send push notifications efficiently to mobile devices.

For the video processing script, Python's OpenCV library was chosen for its robust capabilities in computer vision, while Ultralytics YOLOv8 (You Only Look Once) implementation was selected for real-time object detection in videos, with a particular focus on identifying ambulances.

The Flutter mobile app, developed in Dart, leverages the Flutter framework, which is a UI toolkit famous for creating natively compiled applications across various platforms from a single



codebase. Firebase for Flutter integrates Firebase services, including Firebase Cloud Messaging, facilitating the receipt and handling of push notifications within the mobile app.

To ensure a consistent development environment, Visual Studio Code (VSCode) was selected as the integrated development environment (IDE) for both Python and Dart/Flutter development. Git, a distributed version control system, is employed for tracking changes, with GitHub or GitLab chosen as the hosting platform for collaborative development, code review, and issue tracking.

### 5.1.3 Programming Language

The selection of programming languages for this project is purposeful, to provide distinct aspects of the system. Python is adopted for the Flask backend and video processing, leveraging its flexible and rich library ecosystem.

Flask, a lightweight web framework, accelerate API development. The video processing script employs OpenCV and Ultralytics YOLO, established libraries known for efficient computer vision tasks.

Dart is chosen for the NotifyMe app, aligning with the Flutter framework to ensure a responsive and visually appealing user interface.

Moreover, the decision to use Python is reinforced by its seamless integration with Firebase Cloud Messaging (FCM), a great way to connect the component for the push notifications in the Flutter app. This association between the backend and mobile app languages enhances compatibility.



### 5.1.4 Tools and Technologies

In the realm of software development, our project adopts a set of tools and technologies that emphasizing efficiency and effectiveness. The Flask backend leverages Python, a adaptable language known for its readability and rich libraries. Flask, chosen for its simplicity and scalability. SQLite is employed for database operations due to its lightweight nature, making it suitable for small to medium-scale applications. The integration of Firebase Cloud Messaging (FCM) into the Python backend enhances real-time push notifications, ensuring timely communication with the Flutter mobile app.

On the client side, the Flutter framework is selected for the mobile app, offering a single codebase for both iOS and Android platforms, reducing development effort. Dart, as the language for Flutter. OpenCV and Ultralytics YOLO are employed for video processing in Python, providing robust computer vision capabilities for object detection.

During the analysis and design phase, tools like Lucidchart[14] and Creately[15] are used, which offers a collaborative platform for creating diagrams. The choice of these tools is grounded in their industry-wide acceptance, clear communication and efficient design collaboration.

This comprehensive selection of tools and technologies reflects a strategic approach to address the unique requirements of each project component. The use of proven technologies, well-established libraries, and recognized tools contributes to the project's robustness, scalability, and maintainability.



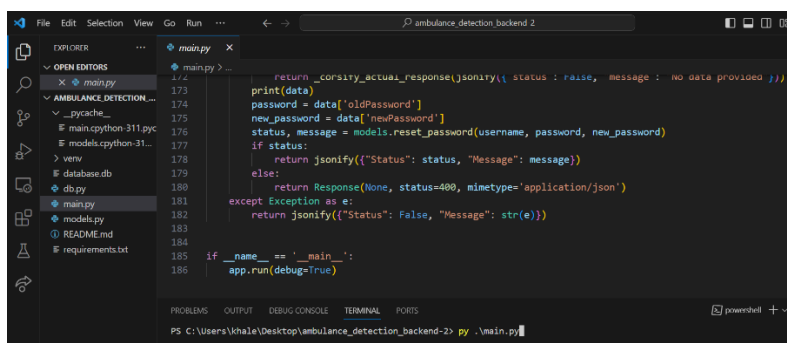
## 5.2 Implementation Details

### 5.2.1 Deployment and Installation:

The deployment process involves setting up the Flask backend on a server, utilizing Ngrok for global access instead of the local during development. The steps include installing Python, Flask, and required dependencies, initializing the Flask app, and configuring the server. For the Flutter mobile app, developers need to install Flutter SDK, set up the project, and run it on an emulator or physical device.

The Flask backend features procedures for user authentication, and push notification handling. Functions like login, signup, interact with the SQLite database to manage user-related operations. The video processing script employs procedures to detect objects, specifically focusing on identifying ambulances within video frames.

For deploying the system, we need to have VS Code as an environment and Ngrok installed for the API step and connecting the model to the back end of the app.



```
173 return corsify_actual_response(jsonify({'status': False, 'message': 'No data provided'}))
174
175 print(data)
176 password = data['oldPassword']
177 new_password = data['newPassword']
178 status, message = models.reset_password(username, password, new_password)
179 if status:
180     return jsonify({'Status': status, 'Message': message})
181 else:
182     return Response(None, status=400, mimetype='application/json')
183 except Exception as e:
184     return jsonify({'Status': False, 'Message': str(e)})
185
186 if __name__ == '__main__':
187     app.run(debug=True)
```

Figure 24. Running the Backend Code

At the beginning, [Figure 24] shows how we will run the back end code using `py ./main.py`

```
C:\Users\khale\Downloads\ngrok-v3-stable-windows-amd64>ngrok.exe http --host-header=rewrite localhost:5000
```

Figure 25. Terminal Command to Run API

[Figure 25] shows how we will run the API using Ngrok with the port 5000 using this command:



```
C:\Users\khale\Downloads\ngrok-v3-stable-windows-amd64\ngrok.exe - ngrok.exe http --host-header=rewrite localhost:5000
ngrok
Build better APIs with ngrok. Early access: ngrok.com/early-access

Session Status      online
Account             abdulrahman (Plan: Free)
Version             3.5.0
Region              India (in)
Latency             175ms
Web Interface       http://127.0.0.1:4040
Forwarding           https://b31c-2a02-cb80-40bb-3066-8506-fdff-dcd9-46a2.ngrok-free.app -> http://localhost:5000

Connections
  ttl    opn    rt1    rt5    p50    p90
  220    0      0.00   0.00   0.85   0.96

HTTP Requests
-----
POST /api/v1/usermg/signin      200 OK
POST /api/v1/usermg/signin      200 OK
POST /api/v1/usermg/send-notification 200 OK
POST /api/v1/usermg/send-notification 200 OK
POST /api/v1/usermg/send-notification 200 OK
POST /api/v1/usermg/send-notification 200 OK
POST /api/v1/usermg/send-notification 200 OK
POST /api/v1/usermg/send-notification 200 OK
POST /api/v1/usermg/send-notification 200 OK
POST /api/v1/usermg/send-notification 200 OK
```

Figure 26. Ngrok Interface

This terminal page in [Figure 26] shows “200 OK” referring to the http status, you can see all the status that may occur in “Ngrok” terminal from Ngrok website[16]. This means the backend is running successfully.

### 5.2.2 Data Structures Description:

The project relies on SQLite as the database, employing tables for user information. The user table includes fields such as username, password hash. This relational database structure ensures efficient data organization and retrieval.

### 5.2.3 Procedures Description:

Let's start with the first part which is the model training we trained nearly 900 images that we annotated using CVAT[17] which is a website used for images annotation, the model was trained using Google Colaboratory for a higher performance with a faster run time.

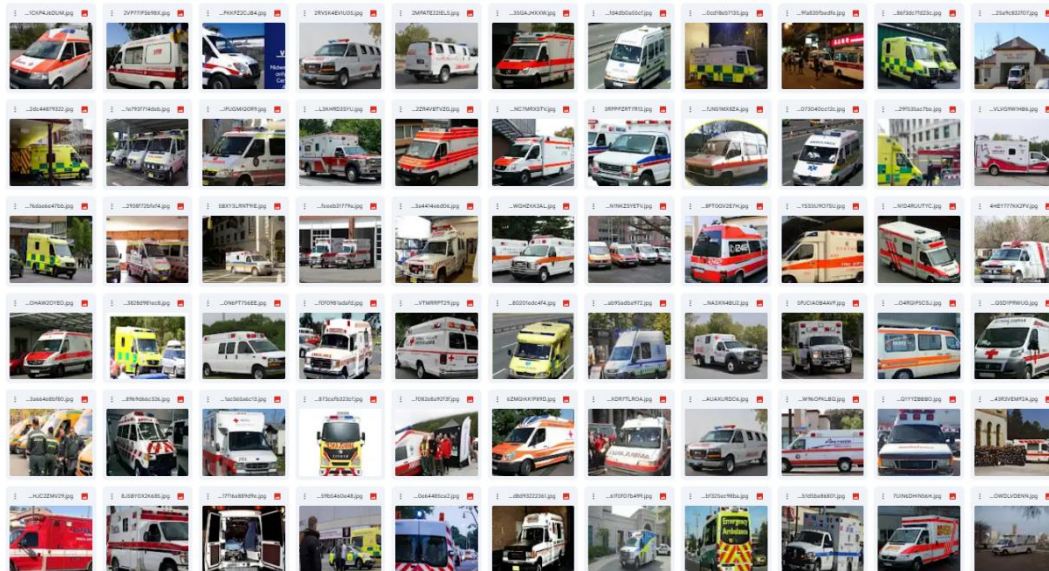


Figure 27. Sample from the Ambulance Dataset

In [Figure 27] an example from the ambulance dataset which contains 939 images.

```
!pip install ultralytics
```

Figure 28. Ultralytics Installing

In [Figure 28] installing the ultralytics to import the YOLO model

```
import os

from ultralytics import YOLO

!yolo task=detect mode=train data={'/content/gdrive/MyDrive/TrainModel/'} /google_colab_config.yaml epochs=100 imgsz=800 plots=True
```

Figure 29. Training the YOLOv8 Model

[Figure 29]:

Import os, which provides functions for interacting with the operating system.

From ultralytics import YOLO: import the yolo model.

!yolo task = detect: the training will start from this line



```

Epoch  GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
98/100   3.53G    0.5808    0.3606    1.226     15         800: 100% 59/59 [00:24<00:00, 2.37it/s]
      Class  Images  Instances  Box(P      R      mAP50  mAP50-95): 100% 30/30 [00:14<00:00, 2.11it/s]
      all    939    1133      0.98      0.977    0.993    0.866

Epoch  GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
99/100   3.52G    0.5718    0.3474    1.212     11         800: 100% 59/59 [00:27<00:00, 2.11it/s]
      Class  Images  Instances  Box(P      R      mAP50  mAP50-95): 100% 30/30 [00:13<00:00, 2.18it/s]
      all    939    1133      0.983     0.974    0.993    0.867

Epoch  GPU_mem  box_loss  cls_loss  dfl_loss  Instances  Size
100/100  3.55G    0.5729    0.3464    1.213     13         800: 100% 59/59 [00:25<00:00, 2.29it/s]
      Class  Images  Instances  Box(P      R      mAP50  mAP50-95): 100% 30/30 [00:16<00:00, 1.82it/s]
      all    939    1133      0.983     0.974    0.993    0.872

100 epochs completed in 1.233 hours.
Optimizer stripped from runs/detect/train/weights/last.pt, 6.3MB
Optimizer stripped from runs/detect/train/weights/best.pt, 6.3MB

Validating runs/detect/train/weights/best.pt...
Ultralytics YOLOv8.1.0 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
      Class  Images  Instances  Box(P      R      mAP50  mAP50-95): 100% 30/30 [00:14<00:00, 2.02it/s]
      all    939    1133      0.983     0.974    0.993    0.873

Speed: 0.4ms preprocess, 3.2ms inference, 0.0ms loss, 2.9ms postprocess per image
Results saved to runs/detect/train
🔗 Learn more at https://docs.ultralytics.com/modes/train

```

Figure 30. Training Result

[Figure 30] shows the end of the training and it took a little more than 1 hour, 939 images were trained on 100 epochs.

Epochs: number of iterations that the code takes around, which means the training was done in 100 cycle.

```

[10] import os

from ultralytics import YOLO

!yolo task=detect mode=val model={ROOT_DIR}/runs/detect/train/weights/best.pt data={ROOT_DIR}/google_colab_config.yaml

Ultralytics YOLOv8.0.196 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3005843 parameters, 0 gradients, 8.1 GFLOPs
val: Scanning /content/gdrive/MyDrive/TrainModel/data/labels/train.cache... 932 images, 7 backgrounds, 0 corrupt: 100% 939/939 [00:00<?, ?it/s]
      Class  Images  Instances  Box(P      R      mAP50  mAP50-95): 100% 59/59 [00:24<00:00, 2.43it/s]
      all    939    1133      0.982     0.974    0.993    0.873

Speed: 1.3ms preprocess, 7.2ms inference, 0.0ms loss, 2.8ms postprocess per image
Results saved to runs/detect/val
🔗 Learn more at https://docs.ultralytics.com/modes/val

```

Figure 31. Validation Code

The [Figure 31] shows the Validation code and where the result where saved.





Figure 32. Validation Result

[Figure 32] shows the result of the Validation.

```

7
8 VIDEO5_DIR = os.path.join('.', 'videos')
9
10 video_path = os.path.join(VIDEO5_DIR, 'VIDEO PATH ')
11 video_path_out = '{}_out.mov'.format(video_path)
12
13 cap = cv2.VideoCapture(video_path)
14 ret, frame = cap.read()
15 H, W, _ = frame.shape
16 out = cv2.VideoWriter(video_path_out, cv2.VideoWriter_fourcc(*'MP4V'), int(cap.get(cv2.CAP_PROP_FPS)), (W, H))
17
18 model_path = os.path.join('.', 'runs', 'detect', 'train15', 'weights', 'last.pt')
19 model = YOLO([model_path])
20
21 threshold = 0.5
22

```

Figure 33. Detection Code

[Figure 33] shows the main part of the model code, in Line 10 we can put any video path either it's a live video or a downloaded video from your device.

```

37
38 # Check if the detected object is an ambulance
39 if results.names[int(class_id)] == 'ambulance':
40     print("Ambulance detected! Take appropriate action here.")
41     requests.post('https://b31c-2a02-cb80-40bb-3066-8506-fdff-dcd9-46a2.ngrok-free.app/api/v1/usermng/send-notification',
42                 data = {'ambulance': 'true'})
43

```

Figure 34. Model API Code

[Figure 34] is another important part of the model code, which will send to the back end a signal throw API if it detects an ambulance.

```

5 class API {
6     static const String baseUrl =
7         'https://logically-logical-bear.ngrok-free.app/api/v1/usermng';
8

```

Figure 35. Application API Code

[Figure 35] is from the flutter front-end application, which will receive the signals via API from the model.



### 5.2.4 Graphical User Interface Description

NotifyME features a minimalist design theme, employing the gray colors with its shades as the primary color to ensure simplicity and user-friendliness across various age groups.

The secondary color, highlighted in red, is strategically used to draw attention to crucial elements such as the "Signup" prompt.

The inspiration for our logo came from the title of our graduation project “Make Way for Lifesaver”, which is; an ambulance vehicle driving fast with an injured patient to try to reach the hospital in the fastest way possible.

The app's layout begins with a Login page displaying fields for user name and password, accompanied by a red notice encouraging new users to sign up. After the signup process, users are redirected to the login page for account verification. The Home Page welcomes users with their username, featuring a top bar with a hospital sign for immediate access to emergency services (997) and a button leading to the user's profile, where password changes can be made.

Default typography is chosen to seamlessly integrate with the minimalist design, and the logo, depicting an ambulance, further reinforces the app's focus on emergency services.

The app's formal notification system serves the primary purpose of alerting users when an ambulance is in close proximity, enhancing the overall functionality of NotifyME.

The logo describes the system idea when an ambulance has an emergency call and the logo presented in login page to show the user, so the user can have a slight idea of the system.

## 6. Testing Chapter

In this chapter, we will test our entire system and all components including the model and all the application function.

### 6.1 Model Results

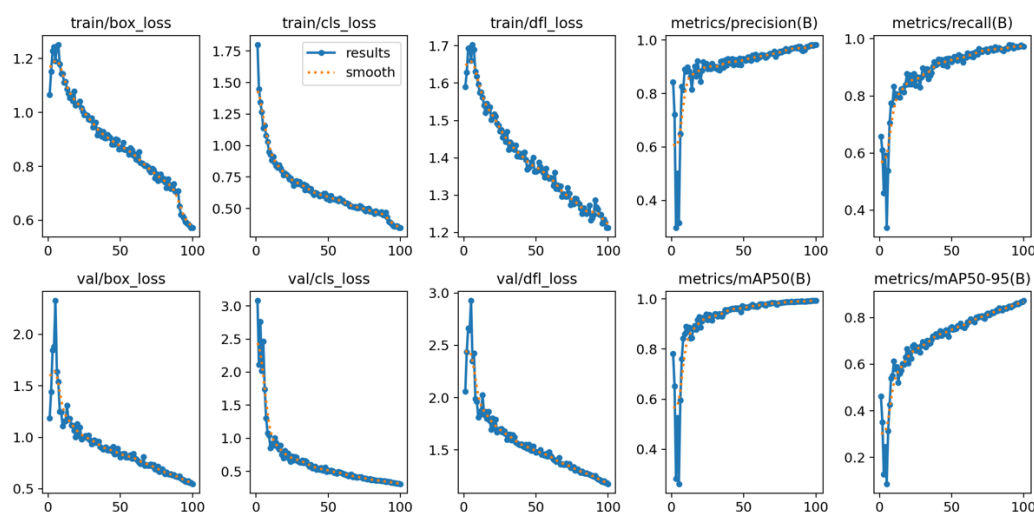


Figure 36. Model Training Results

- box loss: This measures how well the model predicts the bounding boxes of objects in the images.
- cls loss: This measures how well the model classifies the objects in the images.
- dfl loss: This measures how well the model detects objects in the images.
- metrics/precision(B): This metric measures the precision of the model's object detections.
- metrics/recall(B): This metric measures the ability of the model to find all the positive instances.
- metrics/mAP50(B): This metric summarizes the model's performance across all classes of objects, calculated at an Intersection over Union (IoU) threshold of 0.5.
- metrics/mAP50-95(B): This metric summarizes the model's performance across all classes of objects, calculated at an Intersection over Union (IoU) threshold ranging from 0.5 to 0.95.

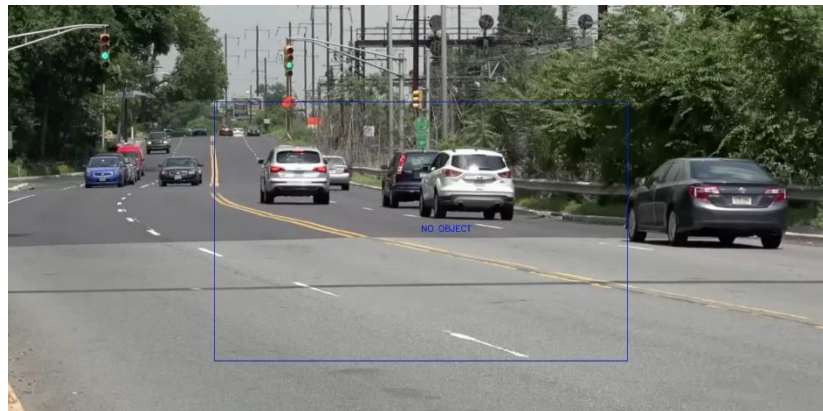
As the model is trained, the graph at [Figure 36] shows that the loss function is decreasing, indicating that the model is learning to make better predictions. The graphs show that all three loss functions decrease over time. The mAP is a measure of how well the model predicts the bounding boxes and classes of objects in the images. The higher the mAP, the better the model is performing. The graphs show that the mAP of the model increases over time, indicating that the model is learning to make better predictions. Overall, the graphs appear to show that the model is training well and making good progress.

## 6.2 Model Testing:



Figure 37. Ambulance Images Before and After the Model Prediction





*Figure 38. Output of the Model without Ambulance Vehicle*

The [Figure 38] shows the output of the frame if there are no detections. In the [Figure 37] there are some examples of the model detection before and after the detection process.

## 6.3 Register Page Testing:

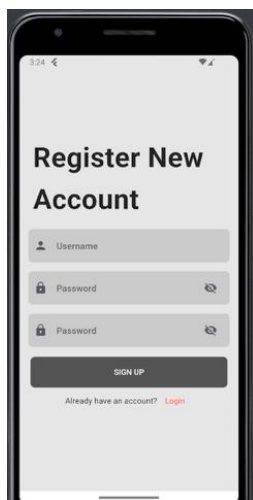


Figure 39. Register Page

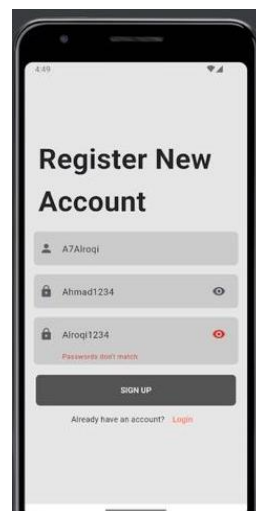


Figure 40. Register Page with Error

In the [Figures 39] it shows that there are input fields:

Username: This field requires the user to enter a unique username.

Password: This field requires the user to enter a password.

There are two password fields to confirm the user has entered their password correctly.

If the user enters a different password, it will alert that the passwords doesn't match [Figures 40].

Sign Up button: This button, when tapped, will create a new user account with the entered information and navigate to the login page.



## 6.4 Login Page Testing:

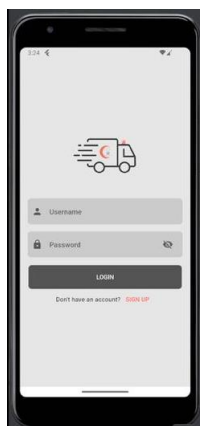


Figure 41. Login Page

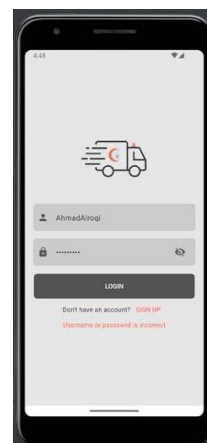


Figure 42. Login Page with Error

The [Figures 41] shows the login input fields, which includes fields for username and password. Below the login form, there is a button labelled "LOGIN".

If the user enters the wrong information, it will alert the user with the message "Username or password is incorrect" [Figures 42].

Under the form, there is a message for new users to sign up if they don't have an account "Don't have an account? SIGN UP".

## 6.5 Password Changing Testing:



Figure 43. Changing the Password Successfully

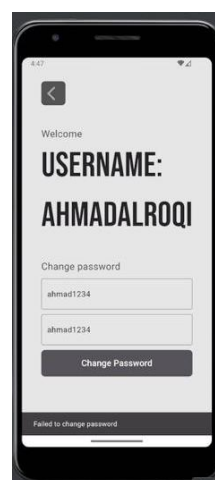


Figure 44. Error while Changing the Password

In [Figure 43], it would alert the user if they changed the password is successfully, and the [Figure 44] shows if the old password matched the new password.

## 6.6 Calling Emergency Testing:

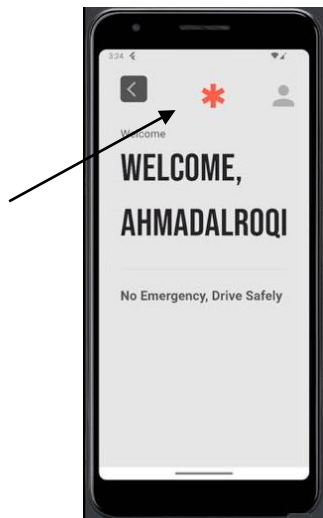


Figure 45. Showing the Emergency Call Button



Figure 46. Navigating to Calling 997

In [Figure 45 and 46], pressing the Red Icon which will immediately call the Emergency.

## 6.7 Messaging Alert Testing

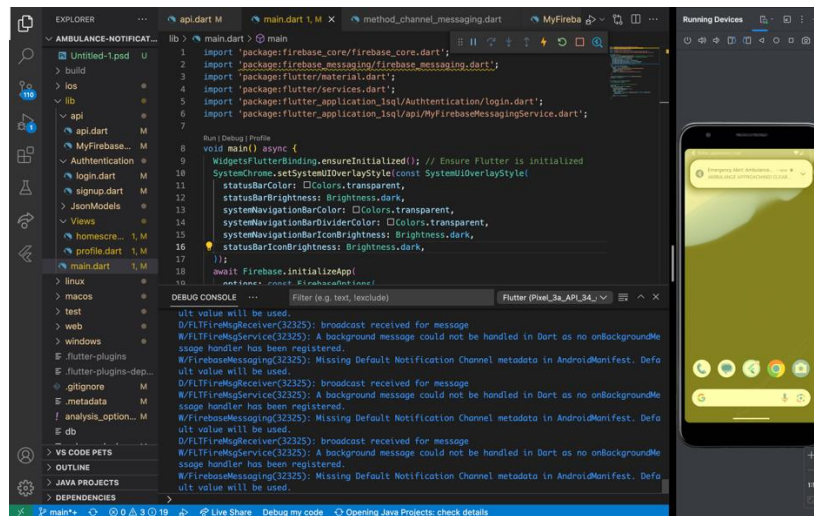


Figure 47. Notification Alert

This [Figure 47] shows the communication between the back end and the front end when the model sends a signal via API to the back end, which will notify the user “AMBULANCE APPROACHING! CLEAR THE WAY. THANK YOU.” to let the user know that there is an ambulance nearby.





## 7. Future Work

- **Model Accuracy:** Continuously enhancing the accuracy of the YOLOv8 model for better vehicle detection, especially in challenging conditions.
- **Expansion of Supported Vehicles:** Extending the model to recognize and differentiate among various types of emergency vehicles, not just ambulances, also Police and Fire Fighters vehicles to be identified.
- **Integration with Additional Sensors:** Exploring the integration of other sensors or technologies, such as LiDAR or radar, to improve the system's overall performance and reliability.
- **User Interface Improvements:** Enhancing the user interface of the mobile application for better user experience and adding more features that could help in emergency response.
- **Collaboration with Emergency Services:** Establishing collaborations with local emergency service centers to integrate the system seamlessly into their operations and improve emergency response times.
- **Continuous Monitoring and Maintenance:** Implementing a monitoring system to continuously track the system's performance and promptly address any issues that may arise.



## 8. Conclusion

This project introduces a comprehensive system aiming to address the critical issue of traffic congestion affecting emergency response times, particularly for ambulance vehicles. Through the integration of image processing algorithms and artificial intelligence (AI) models.

The methodology involves an organized approach, including problem definition, data collection, model selection, training, integration with traffic lights, real-time processing, testing, and evaluation.

The design chapter outlines the system architecture, database design, modular decomposition, system organization, algorithm, alternative design methods, and graphical user interface design.

The implementation chapter shows hardware and software requirements, implementation details, and testing procedures.

The software requirements section highlights the chosen technologies for various components, emphasizing the use of Flask, SQLite, Ngrok, pyfcm, OpenCV, and Ultralytics YOLOv8.

The programming languages chosen, such as Python and Dart, and the tools and technologies selected for development contribute to the project's efficiency and effectiveness.

The testing chapter includes model results, demonstrating the model's training progress through loss metrics, and testing of various application functionalities, including registration, login, password changing, and emergency calling.

The future work section outlines potential improvements, including enhancing model accuracy, expanding supported vehicles, integrating additional sensors, conducting real-world testing, improving the user interface, collaborating with emergency services, and implementing continuous monitoring and maintenance.

In conclusion, this project represents a significant step toward leveraging technology to address challenges in emergency vehicle response times. By combining image processing and AI models with a well-designed mobile application, the aim is to contribute to make urban environments safer and more responsive during medical emergencies. The comprehensive documentation provided serves as a valuable guide for understanding the project's potential for future enhancements.



## 9. References:

- [1] H. Song, H. Liang, H. Li, Z. Dai, and X. Yun, "Vision-based vehicle detection and counting system using deep learning in highway scenes," *Eur. Transp. Res. Rev.*, vol. 11, no. 1, p. 51, Dec. 2019, doi: 10.1186/s12544-019-0390-4.
- [2] Shubham Mishra, Mrs. Versha Verma, Dr. Nikhat Akhtar, Shivam Chaturvedi, and Dr. Yusuf Perwej, "An Intelligent Motion Detection Using OpenCV," *Int. J. Sci. Res. Sci. Eng. Technol.*, pp. 51–63, Mar. 2022, doi: 10.32628/IJSRSET22925.
- [3] Al-aneze, "70% of accident deaths due to ambulance delays." [Online]. Available: <https://www.alwatan.com.sa/article/341055>
- [4] M. Nigam, S. Bhattacharya, and G. Sumathi, "Ambulance detection using image processing and neural networks," *J. Phys. Conf. Ser.*, vol. 2115, p. 012036, Nov. 2021, doi: 10.1088/1742-6596/2115/1/012036.
- [5] O. Masurekar, O. Jadhav, P. Kulkarni, and S. Patil, "Real Time Object Detection Using YOLOv3," vol. 07, no. 03, 2020.
- [6] A. W, "Detection of Ambulance Using Computer Vision," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 10, no. 6, pp. 1299–1304, Jun. 2022, doi: 10.22214/ijraset.2022.44104.
- [7] A. Shobana, C. Sheshank, K. Vishal, S. Vishnu, and M. Srinath, "Ambulance Blocking Vehicles Identification By Artificial Intelligence," 2023.
- [8] S. Sultana, B. Ahmed, M. Paul, M. R. Islam, and S. Ahmad, "Vision-Based Robust Lane Detection and Tracking in Challenging Conditions," *IEEE Access*, vol. 11, pp. 67938–67955, 2023, doi: 10.1109/ACCESS.2023.3292128.
- [9] X. Yuan, A. Kuerban, Y. Chen, and W. Lin, "Faster Light Detection Algorithm of Traffic Signs Based on YOLOv5s-A2," *IEEE Access*, vol. 11, pp. 19395–19404, 2023, doi: 10.1109/ACCESS.2022.3204818.
- [10] "Version," Roboflow. Accessed: Oct. 25, 2023. [Online]. Available: <https://universe.roboflow.com/pouria-maleki-qsj9z/ambulance-tjeov/dataset/1>
- [11] "Download Ambulance labeled image classification dataset labeled image dataset." Accessed: Oct. 25, 2023. [Online]. Available: <https://images.cv/dataset/ambulance-image-classification-dataset>
- [12] "Emergency Vehicles Identification." Accessed: Oct. 25, 2023. [Online]. Available: <https://www.kaggle.com/datasets/abhisheksinghblr/emergency-vehicles-identification>
- [13] "Figma: The Collaborative Interface Design Tool," Figma. Accessed: Feb. 03, 2024. [Online]. Available: <https://www.figma.com/>
- [14] "Intelligent Diagramming," Lucidchart. Accessed: Feb. 03, 2024. [Online]. Available: <https://www.lucidchart.com>
- [15] "Creately | Visual Collaboration & Diagramming Platform." Accessed: Feb. 03, 2024. [Online]. Available: <https://creately.com/>
- [16] "Errors Reference | ngrok documentation." Accessed: Feb. 03, 2024. [Online]. Available: <https://ngrok.com/docs/errors/reference/>
- [17] "CVAT." Accessed: Feb. 03, 2024. [Online]. Available: <https://www.cvat.ai/>