UNIVERSITY OF GHANA, LEGON

DEPARTMENT OF COMPUTER ENGINEERING

SCHOOL OF ENGINEERING SCIENCES

COLLEGE OF BASIC AND APPLIED SCIENCES

FINAL YEAR THESIS

ON

AUTOMATED LICENSE PLATE RECOGNITION AND VEHICLE IDENTIFICATION FOR SECURITY MONITORING USING ARTIFICIAL INTELLIGENCE

PROJECT REPORT SUBMITTED IN PARTIAL FULFILMENT OF THE REQUIREMENT FOR THE BACHELOR OF SCIENCE DEGREE IN COMPUTER ENGINEERING

KELVIN FIADOR	10967291
PRINCE BENNEH	10970656
NAME OF SUPERVISOR:	PROF. GODFREY MILLS
NAME OF CO – SUPERVISOR:	PROF. ELSIE EFFAH KAUFMANN
DATE OF SUBMISSION:	OCTOBER 3, 2025.

AUTOMATED LICENSE PLATE RECOGNITION AND VEHICLE IDENTIFICATION FOR SECURITY MONITORING USING ARTIFICIAL INTELLIGENCE

By

KELVIN FIADOR (10967291)
PRINCE BENNEH (10970656)

Submitted to the Department of Computer Engineering in
Partial Fulfilment of the Requirements for the Degree of
Bachelor of Science in Computer Engineering

University of Ghana

(October 3, 2025)

Name of Student: KELVIN FIADOR (10967291)

Signature of Student: _____

Name of Student: PRINCE BENNEH (10970656)

Signature of Student: _____

Name of Supervisor: PROF. GODFREY A. MILLS

Signature of Supervisor: _____

Name of Co-Supervisor: PROF. ELSIE EFFAH KAUFMANN

Signature of Co-Supervisor: _____

Name of Head of Department: PROF.ROBERT SOWAH

Signature of Head of Department: _____

DECLARATION OF ORIGINALITY

Department of Computer Engineering

I certify that the content of this thesis document is my own work except where indicated by referencing. I confirm that I have read and understood the guidelines on plagiarism in the Computer Engineering Undergraduate Thesis Handbook including the University of Ghana's policy on plagiarism. I have acknowledged in the text of this document all sources used. I have also referenced all relevant texts, figures, data, and tables quoted from books, journals, articles, reports, Internet websites, and works of other people. I certify that I have not used the services of any professional person or agency to produce this thesis document. I have also not presented the work of any student present or past from other academic or research Institutions. I understand that any false claim in respect of this thesis document will result in disciplinary action in accordance with regulations of the University of Ghana.

Please complete the information below by Hand and in BLOCK LETTERS.

Student Name: .....

Index Number: .....

Student Signature: ..... Date.....

Student Name: .....

Index Number: .....

Student Signature: ..... Date.....

Certified by:

Name of Supervisor: .....
(SUPERVISOR)

Supervisor's Signature: ..... Date.....

Name of Supervisor: .....
(CO-SUPERVISOR)

Supervisor's Signature: ..... Date.....

ABSTRACT

AUTOMATED LICENSE PLATE RECOGNITION AND VEHICLE IDENTIFICATION FOR SECURITY MONITORING USING ARTIFICIAL INTELLIGENCE

The rapid growth of urban environments, particularly in developing regions like Ghana, has led to increased vehicle-related crimes such as theft and unauthorized access, necessitating advanced monitoring systems. Traditional License Plate Recognition (LPR) systems, while useful for toll collection and traffic enforcement, are limited by issues like plate tampering, poor image quality, and low visibility, which compromise their reliability for critical security applications. This project investigates the integration of LPR with Make, Model, and Color (MMC) detection to enhance vehicle identification accuracy, addressing these vulnerabilities and improving urban security and traffic management, which is of significant interest in high-traffic areas prone to vehicle crimes.

The approach involves developing an automated system using a Raspberry Pi equipped with an HC-SR04 ultrasonic sensor for vehicle detection at 2-3 meters and a Camera Module V2 for capturing images at 1280x720 resolution. Images are processed with YOLOv11nano for license plate detection, a custom Optical Character Recognition (OCR) model for text extraction, a custom Convolutional Neural Network (CNN) for color classification, and ResNet50 for make/model recognition. Results are verified against a pre-populated Firebase database containing verified vehicle entries, with LED indicators (green for matches, red for flags) and application notifications for real-time feedback.

Testing on 10 vehicles approaching at 5-10 km/h yielded an overall system accuracy of 60%, with YOLOv11nano achieving 100% plate detection rate, custom OCR at 87.02%-character accuracy and 60% sequence accuracy, custom CNN at 80% color accuracy, and ResNet50 at 60% make/model accuracy. Latency averaged 45-120 seconds, increasing variably due to Raspberry Pi resource constraints, with errors primarily from faded text, low-light conditions, and underrepresented models like the Nissan Sentra in the training dataset.

In conclusion, the system demonstrates improved multi-attribute verification over traditional LPR, offering a framework for enhanced vehicle monitoring. Benefits include reduced manual intervention, higher security in gated communities and parking lots, and potential applications in law enforcement for stolen vehicle tracking in urban Ghana, with opportunities for refinement through expanded datasets and hardware optimization.

ACKNOWLEDGMENT

We express our heartfelt gratitude to Almighty God for granting us the strength, wisdom, and perseverance required to complete this academic journey. His grace has been a constant source of inspiration and guidance.

We are profoundly thankful to our supervisor, Prof. Godfrey A. Mills, for his steadfast guidance and unwavering support at every stage of this project. Our sincere appreciation also goes to Mr. Jerry Alexander Agudogo and Mr. Akuming Amoah, Senior Research Engineers at IT Consortium, Accra, whose encouragement and technical guidance enabled us to maintain consistency and see this project through to completion.

We extend our gratitude to the management of IT Consortium for initiating the project idea and supporting us financially through the provision of some hardware components required for implementation. Special thanks also go to Mr. Samuel Yawson, a graduate student of Computer Engineering, and to the teaching assistants (TAs) who provided us with valuable insights and technical support during the course of this work.

We are equally grateful to the other faculty members for their invaluable support and guidance throughout this journey. Lastly, we deeply appreciate our friends and colleagues for their unwavering encouragement, contributions, and companionship, which greatly enriched this endeavor.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

CHAPTER 1 – INTRODUCTION

1.0 Introduction

This chapter introduces the reader to the project topic. The background of the problem being addressed, the problem itself outlined, the objectives that have been set to develop the solution to this problem, the relevance it would have and a general outline of the entire thesis.

1.1 Background

The increasing complexity of urban environments and the rise in vehicle-related crimes, such as theft and unauthorized access, have driven the need for advanced vehicle monitoring systems. Traditional LPR systems focus primarily on extracting license plate characters from images, enabling applications like toll collection, traffic law enforcement, and parking management. However, relying solely on license plates can be limited by issues such as plate tampering or poor image quality. Integrating MMC detection enhances the reliability of vehicle identification by providing additional attributes for verification. Recent advancements in computer vision and machine learning have made it feasible to develop systems that combine LPR with MMC detection, leveraging sensors for real-time vehicle detection and image capture. This project builds on these technologies to create an automated system that processes vehicle images, extracts relevant features, and compares them against a database to ensure secure and efficient monitoring.

1.2 Problem Statement

Current vehicle monitoring systems often rely on LPR alone, which may be insufficient in scenarios involving tampered plates or low-visibility conditions. Without additional identifying features, such systems struggle to provide comprehensive security for applications like criminal tracking, stolen vehicle identification, or access control in secure areas. Furthermore, manual verification processes are time-consuming and prone to errors, limiting scalability in high-traffic environments. There is a need for an automated system that integrates LPR with MMC detection

to improve identification accuracy, enhance security, and support efficient management of traffic and access control. This project addresses these challenges by developing a system that combines sensor-based vehicle detection with image processing to extract and verify license plate and vehicle attributes.

1.3 Project Objectives

The objectives of this project are:

1. To design a machine learning-based system for detecting and recognizing authorized vehicles through License Plate Recognition (LPR) and Make, Model, and Color (MMC) attributes.
2. To train machine learning models to accurately identify license plate characters and MMC attributes from vehicle images.
3. To develop a user-friendly mobile application for user interaction and real-time notifications from the system on the results of vehicle identification and verification.
4. To integrate a database for storing and comparing vehicle data to verify authorized vehicles and flag unauthorized ones.
5. To evaluate the system's accuracy and performance in real-world vehicle identification and monitoring scenarios.

1.4 Relevance of Project

The proposed LPR and MMC detection system addresses critical needs in modern vehicle monitoring by enhancing the accuracy and reliability of vehicle identification. By combining license plate data with vehicle make, model, and color, the system strengthens applications such as criminal tracking, stolen vehicle identification, and access control for gated communities and secure facilities. The system's ability to flag unauthorized vehicles in real-time enhances public safety and security in sensitive areas. Furthermore, it is adaptable to various contexts, including

parking management and time-based access control, offering a practical solution for urban and secure environments.

1.5 Outline of Thesis

This thesis is organized as follows. Chapter 1 presents the challenges of vehicle detection for law enforcement agencies and the problems involved for which solution will enhance implementation. It also presents the objective of the project and its benefits to society. In chapter 2, existing solutions were examined to justify the proposed solution and its scope for implementation. Chapter 3 of the thesis describes the system design and development, covering sensor-based vehicle detection, image capture, machine learning model integration, database creation, and verification. Chapter 4 discusses the implementation and testing of the system, presenting the experimental setup, performance evaluation, and results. Finally, Chapter 5 concludes the work by summarizing key findings, assessing the system's effectiveness, and providing recommendations for future improvement.

CHAPTER 2 - LITERATURE REVIEW

2.0 Introduction

This chapter reviews existing literature on License Plate Recognition (LPR) systems, particularly those integrating Make, Model, and Color (MMC) detection for vehicle identification. It examines current solutions, highlights their limitations, and outlines the proposed system to address these gaps. The scope of the project is also defined to clarify its boundaries.

2.1 Review of Existing Solutions

Existing License Plate Recognition (LPR) systems primarily utilize computer vision and machine learning techniques for vehicle identification in applications such as traffic management, security, and parking control. Traditional methods often focus on frontal views, but recent advancements address challenges like oblique angles, varying lighting, and non-standard plates.

For instance, Li et al. (2021) [1] proposed a YOLO-based method to improve LPR for oblique license plates by modifying the Warped Planar Object Detection (WPOD) network. They introduced a simple Intersection over Union (IOU) algorithm to incorporate confidence parameters into the loss function, reducing computational complexity while achieving 95.7% average accuracy on datasets like OpenALPR EU, AOLP RP, and CD-HARD. This approach outperforms Silva's original WPOD by about 1%, demonstrating enhanced bounding box localization and rectification for distorted plates.

Al-Mheiri et al. (2022) [2] developed a machine learning-based car plate recognition system using a Raspberry Pi, ultrasonic sensors, and a camera for secure gate entry in parking areas. Their system detects incoming vehicles, captures images, and processes them with OpenCV for noise reduction (grayscale conversion, brightness adjustment, Gaussian blur) before applying Tesseract OCR for character extraction. The recognized plates are verified against a MongoDB database for authorization, with SMS alerts for issues. While effective for UAE plates, the system faces limitations in budget-constrained environments and lacks integration with vehicle make/model detection.

Mustafa and Karabatak (2024) [3] presented a real-time system integrating Vehicle Make and Model Recognition (VMMR) with Automatic Number Plate Recognition (ANPR), achieving 97.5% accuracy. They employed MobileNet-V2 and YOLOx for vehicle classification, and YOLOv4-tiny with Paddle OCR and SVTR-tiny for plate recognition. Tested on 1,000 images from Firat University's entrance under adverse conditions (fog, rain, low light), the system uses Gradient-weighted Class Activation Mapping (Grad-CAM) to analyze misclassifications. This work highlights robustness but does not explicitly include color detection as part of MMC attributes.

Kumar et al. (2023) [4] implemented an ANPR system using Python and OpenCV for Indian plates, focusing on localization, normalization, segmentation, and recognition. Their process involves resizing, color space conversion, bilateral filtering for noise reduction, and pytesseract for OCR, achieving 98.20% accuracy with a processing time of 3.78 ms. The system compares favorably to methods like K-Means clustering and multi-level deep features, emphasizing adaptability to larger scales and noise. However, it primarily addresses LPR without MMC integration.

Iyer and Dhavale (2024) [5] proposed a two-factor authentication system combining ANPR (YOLOv8 with EasyOCR) and face recognition (dlib CNN) for secure military gate entry. Trained on combined Indian datasets, the ANPR achieves 88.14% accuracy for English and Marathi (Devanagari) plates, while face recognition reaches 98.34%. The system uses a secure MySQL database and operates offline, addressing non-standard plates but focusing on security rather than MMC detection.

These studies demonstrate progress in handling oblique views, real-time processing, and multi-language recognition, often using YOLO variants and OCR tools. However, most prioritize LPR accuracy, with limited integration of Make, Model, and Color (MMC) detection. While [3] combines VMMR and ANPR, it omits color attributes and portable sensor-triggered deployment with mobile notifications. This gap highlights the need for a comprehensive system that merges LPR with full MMC detection in a portable setup for enhanced vehicle monitoring.

2.2 Proposed Solution

The proposed solution develops a portable LPR system for automatic detection and recognition of vehicle license plates and MMC attributes. The key features would include:

- AI-based LPR and MMC detection using trained machine learning models.
- Database verification to detect inconsistencies between captured data and registered records.
- Real-time security monitoring with alerts sent via a user-friendly mobile app.

A sensor detects approaching vehicles, captures images, and processes them to extract license plate characters, make, model, and color. The results are then compared against a database to verify authorization, flagging mismatches for security purposes.

2.3 Scope of Project

The project focuses on designing and implementing an LPR system with MMC detection for vehicle verification at security checkpoints as well as controlled environments like gated communities or parking areas. It includes model training, database integration, and mobile app development for notifications.

Out of scope are: deployment in extreme weather conditions, integration with large-scale traffic networks, hardware manufacturing for sensors, and advanced encryption for data transmission. The system assumes standard vehicle images and does not handle heavily obscured or damaged plates.

CHAPTER 3 – SYSTEM DESIGN AND DEVELOPMENT

3.0 Introduction

This chapter describes the design and development of the License Plate Recognition (LPR) system integrated with Make, Model, and Color (MMC) detection. It provides an overview of the system's architecture and functions, detailing the hardware and software components. The design focuses on real-time vehicle detection, image processing, data verification, and notification delivery via a mobile app. The workflow and architecture, illustrated in the accompanying diagrams, ensure efficient operation in security and monitoring applications.

3.1 System Overview and Functions

The system detects approaching vehicles using a motion sensor, captures images, and applies machine learning models to extract license plate recognition (LPR) and make/mode/color (MMC) attributes. These details are then verified against a cloud-based database. If verification succeeds, the process ends successfully; otherwise, the vehicle is flagged, its details logged, and notifications are triggered while on-site indicator LEDs alert security personnel. Additional software functions include a search feature which enables security staff to query vehicle details and retrieve past flagged records, along with location data. To illustrate the design more clearly, the system architecture (Figure 3.1) and workflow (Figure 3.2) diagrams are presented below, outlining the functional flow, functional components, and interaction between hardware, software and cloud services.

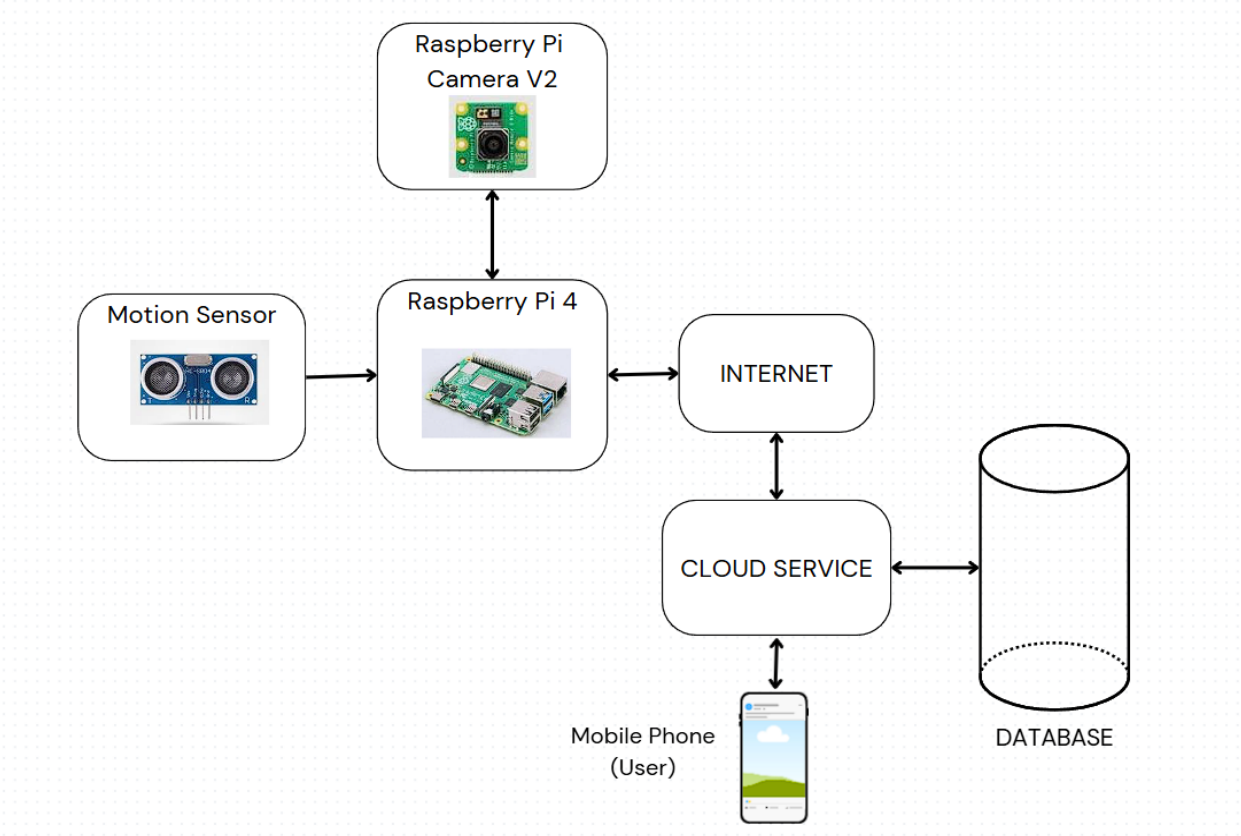

Figure 3. 1 Architectural diagram of the proposed system

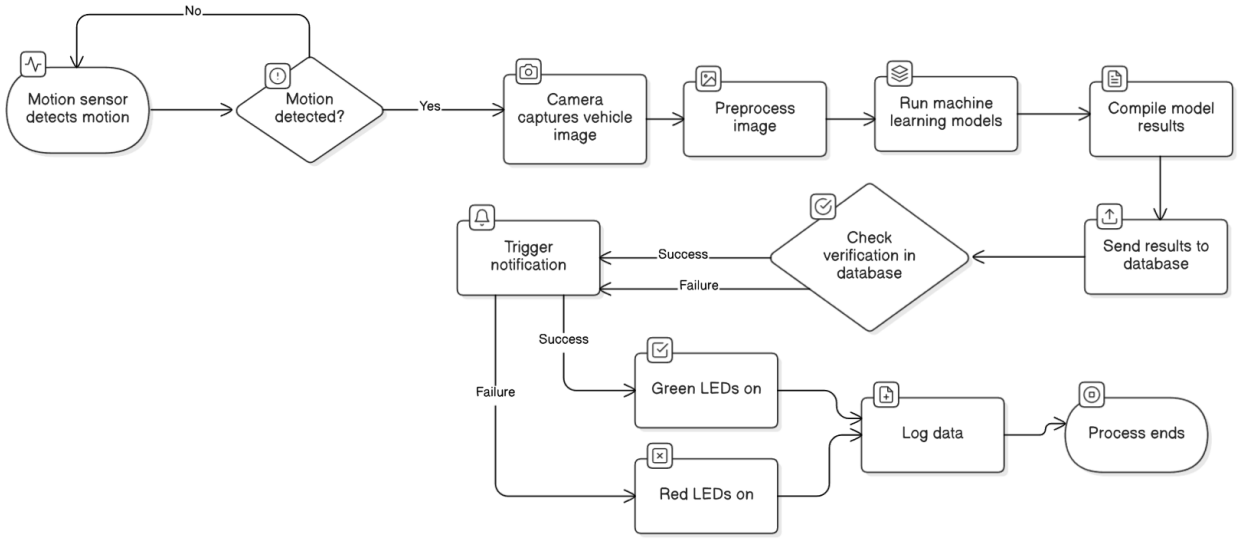

Figure 3. 2 Overall System Flowchart

3.1.1 Hardware

The hardware consists of a Raspberry Pi 4 as the central processing unit, equipped with a Raspberry Pi Camera V2 for image capture and an ultrasonic sensor (e.g., HC-SR04) for motion detection. The ultrasonic sensor monitors movement within a defined range, triggering the camera when a vehicle approaches. The Raspberry Pi, mounted on a stand for optimal positioning, handles on-device processing of images using trained models. It connects to the internet via Wi-Fi or Ethernet to upload data to cloud services and interact with the database. This setup ensures portability and low-cost deployment for vehicle monitoring at entry points or parking areas.

3.1.2 Software

The software consists of the machine learning models which run on the Raspberry Pi using Python-based scripts for sensor integration, image capture, and model execution and the app on the mobile phone for notification and user interaction. Key components include:

- Motion Detection and Image Capture: Scripts interface with the ultrasonic sensor via GPIO pins to detect motion and activate the camera module for capturing vehicle images.

- Machine Learning Models: Trained models (e.g., using YOLO for detection and CNNs for classification) process images to extract license plate characters (via OCR) and MMC attributes. Models are optimized for edge computing on the Raspberry Pi.

- Data Processing and Verification: Extracted data is sent to a cloud-based database (e.g., via API calls) for comparison with registered vehicles. Verification logic flags mismatches and logs details, including timestamps and locations.

- Search Function: A backend API allows security personnel to input vehicle details (e.g., plate number or MMC) and query the database for historical flags, returning results with associated locations.

- Mobile App Integration: A user-friendly app, developed with frameworks like Flutter, receives real-time notifications for flags and provides an interface for viewing logs and performing searches.

- Cloud Services: Utilizes Firebase for database storage and push notifications to ensure scalability and remote access.

3.2 Requirement Analysis and Specifications

This section analyzes the system's needs based on user and operational demands, specifying what the system must do (functional) and how well it must perform (non-functional).

3.2.1 Functional Requirements

- Detect and process Ghanaian number plates: The system is tailored to recognize of Ghanaian number plates, ensuring reliable identification despite variations in design or condition. This is critical for effective vehicle verification in local contexts.

- Display of verification results: The mobile app provides users with real-time feedback on whether a vehicle is authorized, displaying details such as plate number, make, model, color, and verification status, enhancing user interaction and remote monitoring capabilities. The hardware includes a screen and indicator (e.g., LED) to show verification outcomes locally, allowing security personnel to immediately confirm a vehicle's status without relying solely on the mobile app.

- Sensors should initiate image capture: Ultrasonic or motion sensors trigger the camera upon detecting a vehicle within range, ensuring timely image capture for processing and reducing unnecessary operation.

- Quality image capture: The camera is designed to produce well-lit, focused images with sufficient resolution to enable accurate extraction of license plate characters and MMC attributes, even in varying lighting conditions.

- Transmission of inferred results for verification: Processed data (e.g., plate number, make, model, color) is sent to a cloud database for comparison with registered vehicle records, enabling secure verification while flagging unauthorized vehicles.

3.2.2 Non-Functional Requirements

- Portability: The system is built with a lightweight, portable design (using a Raspberry Pi and compact stand) to allow deployment across multiple locations, such as parking lots or gates, without requiring permanent installation.

- Fast verification: Quick decision-making at entry points, meeting real-time operational needs by limiting the time from image capture to verification result, enhancing efficiency and user experience.

- Optimized power usage: The design incorporates energy-efficient components (e.g., low-power Raspberry Pi mode) and sleep modes for idle sensors, ensuring prolonged operation in remote or battery-powered setups.

- Usability: The user interface must be intuitive and easy to navigate for security professionals, requiring minimal training.

3.3 System Design and Development Process

This section details the development process for the system's key components, including machine learning models, software, and hardware. The design emphasizes portability, real-time performance, and integration of LPR with MMC detection, building on the requirements outlined in Section 3.2. The architecture of the entire system is based on Figure 3.1

3.3.1 Machine Learning Models

Four major models were developed in order to implement this system. They include:

- License Plate Detection model: This model was responsible for the detecting the vehicle license plate within a captured image. The best among four models (YOLOv11n, YOLOv8n, YOLOv5n and Faster-RCNN) is picked. These are all object detection models which are used to classify or identify objects in images or videos. The detected license plate is then readied for its characters to be read and outputted by the OCR.

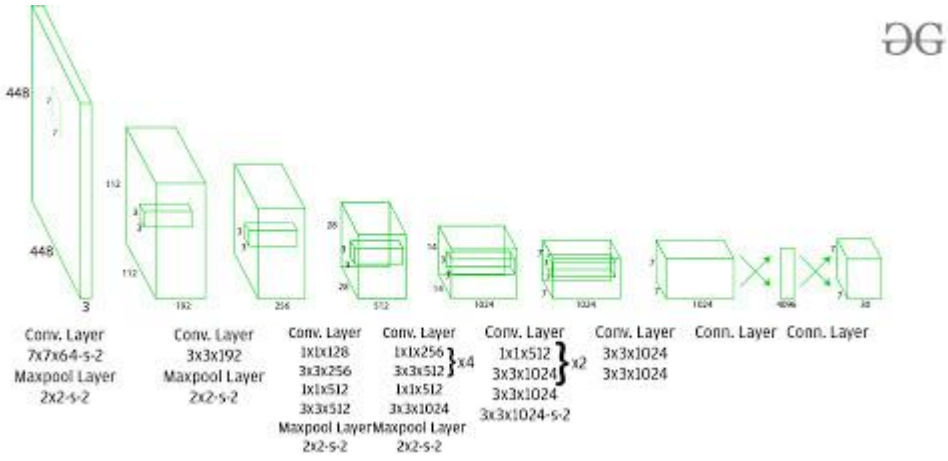

Figure 3. 3 General Architecture for YOLO (You Only Look Once) Models [15]

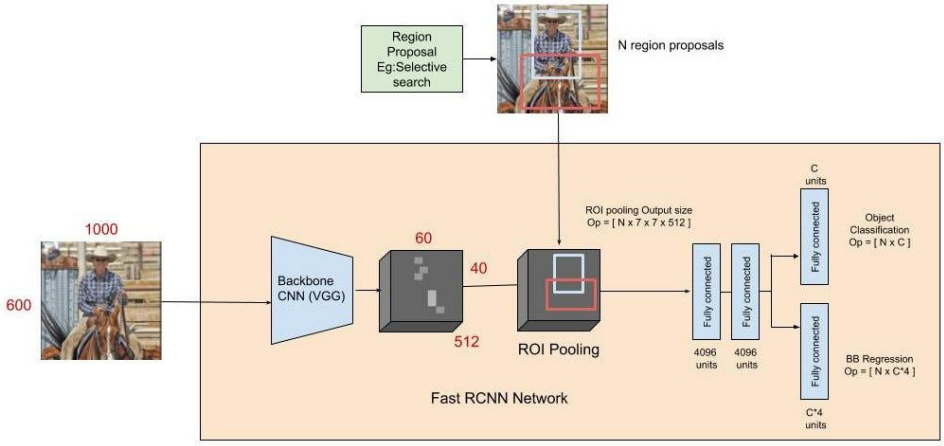

Figure 3. 4 Architecture for Faster-RCNN model [22]

- Optical Character Recognition (OCR) model: This model would read the characters of the license plate that has been detected. OCR models are machine learning models which are used to recognize characters from images. Three already established OCR engines (PaddleOCR, EasyOCR and Tesseract OCR) along with a custom created model (CNN) were used to recognize the license plate characters. The recognized characters are then sent to the database for verification.

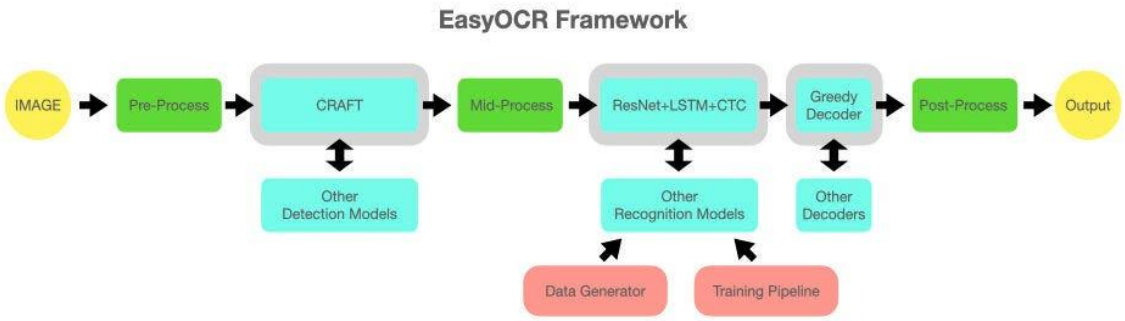

Figure 3. 5 Architecture of EasyOCR [14]

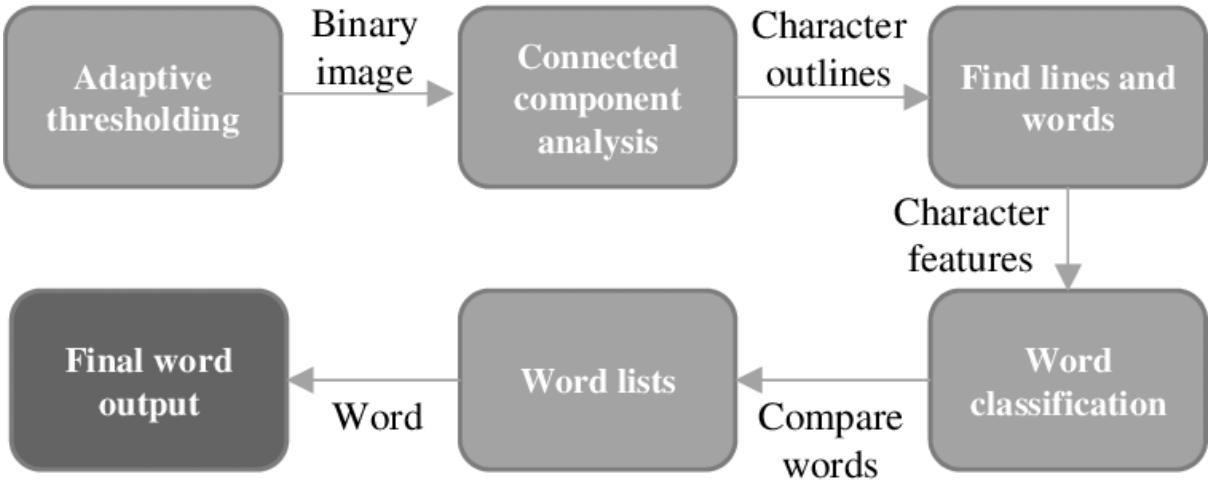

Figure 3. 6 Architecture of Tesseract OCR [18]

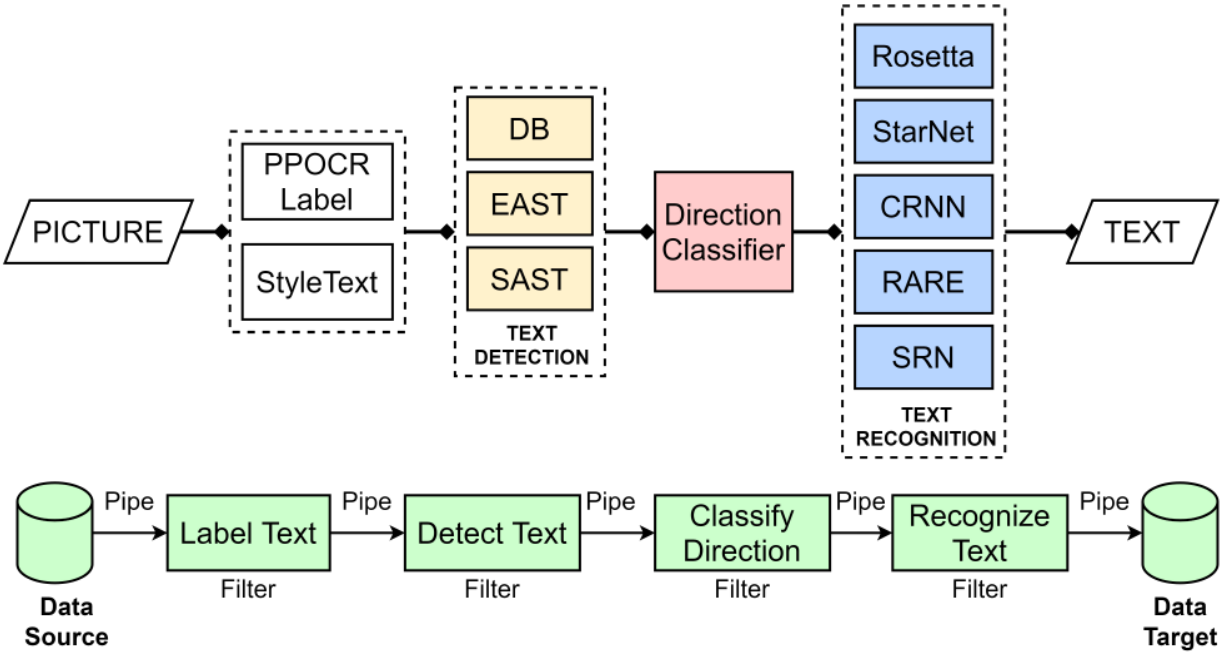

Figure 3. 7 Architecture of PaddleOCR [19]

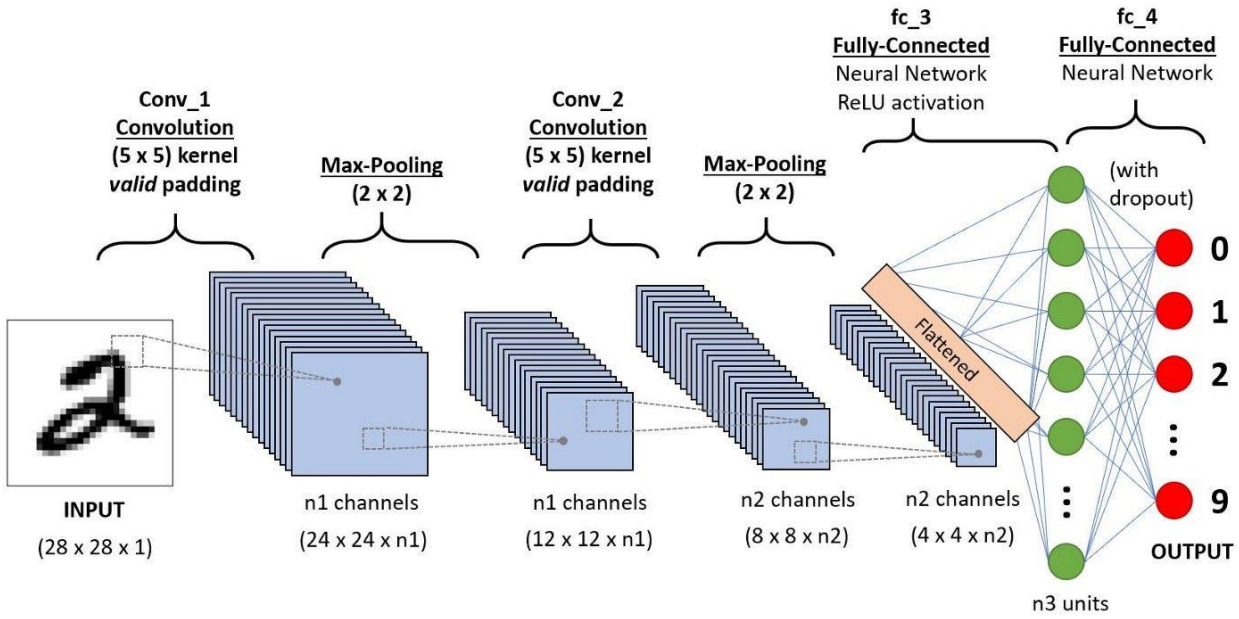

Figure 3. 8 Architecture of Custom CNN OCR [16]

- Make & Model model: This model identifies the make i.e. type of the vehicle and its creator. This tells us the basic manufacturer details of the car. Two models were tested for this model's section. (EfficientNetB0 and ResNet50). They are also object detection models.

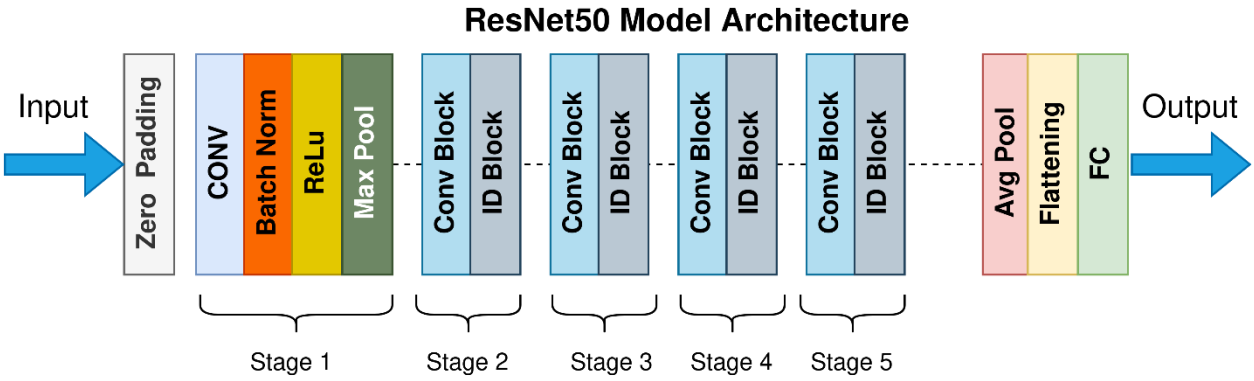

Figure 3. 9 Architecture of ResNet50 Model [17]

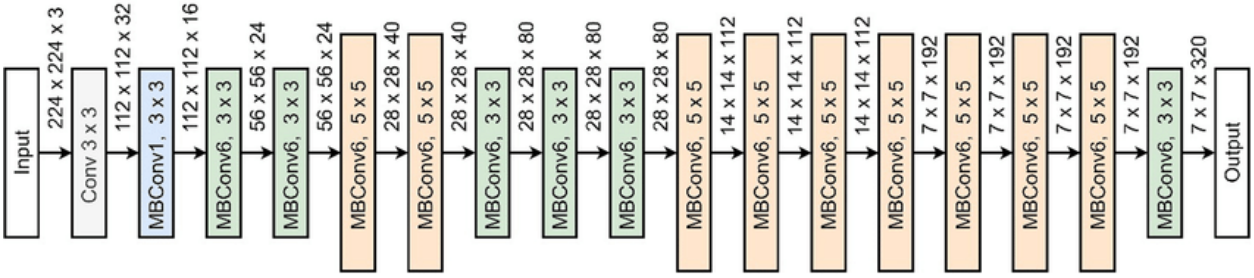

Figure 3. 10 Architecture of EfficientNetB0 Model [20]

- Color model: This model detects/ identifies the color of the vehicle it runs its inference on.

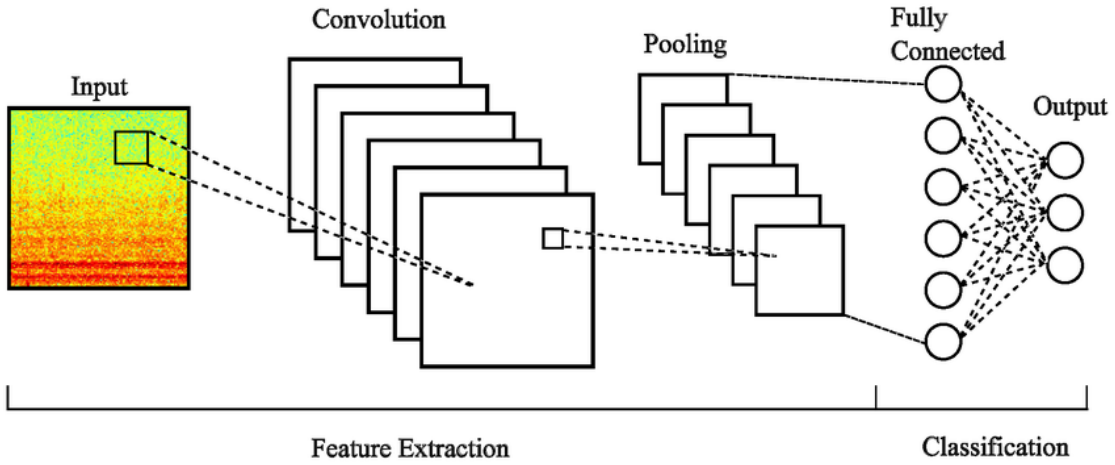

Figure 3. 11 Architecture of CNN color classifier [21]

With these the license plate and the make, model and color of vehicles could be detected.

3.3.1.1 Dataset for Machine Learning Model

In order to train the various models used in the implementation of the solution. Although different models were tested to pick the ones which would hold the best outcome for each major section of the solution, the data used to train and test them was the same for their category.

For the license plate detection model, an amalgam of different datasets was used in order to create a comprehensive dataset. This is to ensure robustness, particularly for Ghanaian license plates, which often exhibit variations in format, font, color and layout. The dataset combined samples from multiple sources, with approximately half the images sourced from Ghanaian datasets to

prioritize local relevance, and the remainder from the Chinese Car Plate Detection (CCPD) dataset [6] to augment diversity and scale. This hybrid approach addressed the limited availability of high-quality Ghanaian datasets while leveraging the extensive annotations in CCPD for improved generalization. The total dataset comprised 7,000 images, split as follows: 5,000 for training, 1,000 for validation, and 1,000 for testing. All images were annotated with bounding boxes for a single class ("license plate") using tools like LabelImg, ensuring consistency in labeling. The CCPD dataset [6] provided the foundation for non-Ghanaian samples, specifically from its "CCPD Base" and "CCPD Green" subsets. CCPD Base contains 200,000 samples of vehicles with Chinese blue license plates, while CCPD Green includes 12,000 samples with white-green backgrounds. From these, 3,919 samples were selected, focusing on images with clear plates under varied conditions (e.g., lighting, angles). These subsets were chosen for their high-quality annotations and diversity in environmental factors, simulating real-world challenges like occlusion and weather variations commonly encountered in Ghanaian scenarios.

To incorporate Ghana-specific elements, three additional datasets were integrated: "License Plate Detection" [7], "Ghanaian License Plates" [8], and "Number Plates Detect" [9]. The "License Plate Detection" dataset [7] consists of 2,446 samples of vehicles with Ghanaian plates in various colors and conditions, all used in full. "Ghanaian License Plates" [8] provides 321 samples, emphasizing local plate formats under everyday settings. Similarly, "Number Plates Detect" [9] includes 314 samples, focusing on detection in diverse urban and rural contexts. All three datasets feature a single class ("license plate") and an image size of 640x640 pixels, with all samples utilized to maximize representation of Ghanaian variations, such as one-line and double-line plates, different fonts, and backgrounds.

This combined dataset enabled the model to handle both standard and non-standard plates effectively. By balancing sources, the dataset mitigated biases toward Chinese plate styles while enhancing the model's performance on Ghanaian-specific features.

For the OCR a few pretrained and already established OCR models were tried along with a custom created OCR model. The dataset used to train this custom model, “GitHUb pragatuinna License Plate Number Detection” [10] consists of 1080 samples of data. These are woven into 36 classes.

26 classes of upper-case alphabets(A-Z) and 10 of numbers (0-9) with 864 samples used for training and 216 for validation.

For the make and model detection models, the “Stanford Cars” [11] was used. This dataset consists of 16185 images across 196 classes. The classes are the various vehicle make and models.

For the color detection model, the “VCoR (Vehicle Color Recognition)” [12] dataset was used. This dataset consists of 10400 images across 15 classes of colors (red, black, blue, beige, grey, white, silver, brown, yellow, gold, green, orange, pink, purple and tan).

3.3.1.2 Data Preprocessing

For license plate detection:

The sizes of the images were all set to a standard 640x640 with the pixel values normalized (0-1). For one of the various models tested (Faster-RCNN) the data was first converted to tensor.

For optical character recognition:

The detected plate number image (Figure 3.12) is converted to grayscale to simplify processing. Gaussian Blur or bilateral filter is applied to reduce noises in the background. Histogram equalization is then applied to make the characters more distinct. Binarization is then applied to convert the grayscale into black-and-white. The characters are then inverted using “cv2.bitwise_not” to gain the result as seen in Figure 3.13. This is done before the pre-trained and established OCR engines are run over the image. As for the custom CNN the bitwise operation is not performed leaving the characters white with black background. The only other difference is that the characters are segmented through contour detection (Figure 3.14) and cropping (Figure 3.15) prior to running the custom CNN on them.

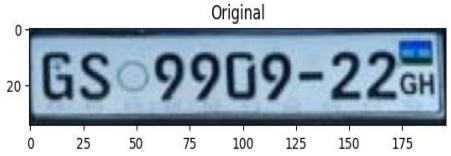

Figure 3. 12 Cropped detected license plate

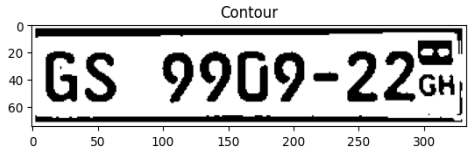

Figure 3. 13 Processed plate for OCR engine

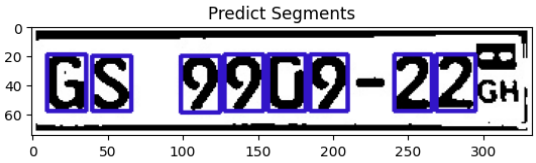

Figure 3. 14 Segmentation of Individual Characters

Figure 3. 15 Cropped Segments

For the make and model detection model:

The images are resized to 224x224.

The same was done for the vehicle color classification model.

3.3.1.3 Feature Engineering and Selection

For the license plate detection: The filenames of the data from the CCPD [6] contain the ground truth for the license plate text and the bounding box co-ordinates of the license plates. They denote the four vertices for the license plate and a script was used to extract them as in Figure 3.16. The annotations for the other license plates were readily available. Depending on the model that was being trained a different function was used to convert the format of the annotations to fit the requirements of said model. They were mainly COCO format for the Faster-RCNN model (Figure 3.17) and YOLO format (Figure 3.18) for the YOLO models. For the OCR designed with a custom

CNN, we refer to the segmentation performed as seen in Figure 3.14. For color: RGB/HSV histograms. For ResNet50: pre-trained features with fine-tuning on make/model classes. Used PCA to reduce dimensionality where applicable (e.g., color features).

```python
def parse_filename(filename):
    parts = filename.split('-')

    # Bounding box extraction
    bbox_part = parts[2]
    x1, y1 = map(int, bbox_part.split('_')[0].split('&'))
    x2, y2 = map(int, bbox_part.split('_')[1].split('&'))


    # Four vertices extraction (RB, RT, LT, LB order)
    vertices_part = parts[3]
    vertices = []
    for v in vertices_part.split('_'):
        x, y = map(int, v.split('&'))
        vertices.append((x, y))

    # License plate text
    indices = list(map(int, parts[4].split('_')))
    province = provinces[indices[0]] if indices[0] < len(provinces) else ''
    alphabet = alphabets[indices[1]] if indices[1] < len(alphabets) else ''
    ads_chars = [ads[i] if i < len(ads) else '' for i in indices[2:7]]
    plate_text = province + alphabet + ''.join(ads_chars).replace('O', '')

    return (x1, y1, x2, y2), vertices, plate_text
```

Figure 3. 16 Code snippet for data extraction from filename

```python
class CocoTransform:
    def __call__(self, image, target):
        image = F.to_tensor(image)
        return image, target #convert to tensor


#dataset rendering function
def get_coco_dataset(img_dir, ann_file):
    return CocoDetection(
        root = img_dir,
        annFile = ann_file,
        transforms = CocoTransform()
    )
```

Figure 3. 17 COCO format transformation

```python
def bbox_to_yolo(x1, y1, x2, y2, img_width=720, img_height=1160):
    x_center = ((x1 + x2) / 2) / img_width
    y_center = ((y1 + y2) / 2) / img_height
    w = (x2 - x1) / img_width
    h = (y2 - y1) / img_height
    return f"0 {x_center} {y_center} {w} {h}"
```

Figure 3. 18 YOLO format transformation

3.3.1.4 Model Training and Testing

For the license plate detection models, YOLOv11n, YOLOv8n, YOLOv5n and Faster-RCNN, model training was carried out on Kaggle with two Tesla T4 GPUs. The YOLO models using the ultralytics library and Faster-RCNN using torch and torchvision libraries.

For the YOLO models a standard setup was used with a batch size 16, image size of 640x640, an AdamW optimizer learning rate 0.002 and momentum 0.9. Training was run for 10 epochs.

```
Image sizes 640 train, 640 val
Using 2 dataloader workers
Logging results to runs/detect/train
Starting training for 10 epochs...
Closing dataloader mosaic
albumentations: Blur(p=0.01, blur_limit=(3, 7)), MedianBlur(p=0.01, blur_limit=(3, 7)), ToGray(p=0.01, num_output_channels=3, method='weighted_a

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       1/10      2.23G      1.077      2.006      1.011          8        640: 100%|██████████| 313/313 [01:02<00:00,  5.01it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 32/32 [00:08<00:00,  3.98it/s]
                   all       1000       1048      0.95      0.897       0.95      0.654

      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       2/10      2.62G      1.087     0.9282       1.05          7        640: 100%|██████████| 313/313 [00:55<00:00,  5.59it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 32/32 [00:05<00:00,  5.52it/s]
                   all       1000       1048      0.947      0.915      0.955      0.673


      Epoch    GPU_mem   box_loss   cls_loss   dfl_loss  Instances       Size
       3/10      2.62G      1.062     0.7229      1.049          8        640: 100%|██████████| 313/313 [00:56<00:00,  5.54it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95): 100%|██████████| 32/32 [00:05<00:00,  5.70it/s]
```

Figure 3. 19 YOLO model training

For the Faster-RCNN model, training was run for 10 epochs using an SGD optimizer, learning rate 0.005, momentum 0.9 and step size 3.

```python
# Training loop
num_epochs = 10
# start_epoch = 0
for epoch in range(start_epoch, num_epochs):
    train_one_epoch(model, optimizer, train_loader, device, epoch)
    lr_scheduler.step()

    # Save at end of this epoch
    ckpt = f'fasterrcnn_ckpt_epoch_{epoch+1}.pth'
    save_checkpoint(model, optimizer, lr_scheduler, epoch+1, ckpt)
    print(f"Saved checkpoint: {ckpt}")
```

Figure 3. 20 Faster-RCNN training

.

For the custom CNN used for character recognition, the training was run for 20 epochs using an Adam optimizer with a learning rate of 0.001 on a Huawei matebook d15 8GB DDR4 RAM and 12 CPUs.

```python
# Train the model (uncomment and adjust epochs as needed)
model.fit(train_generator, epochs=20, validation_data=validation_generator)
```

```
Epoch 1/20
c:\Users\FIADOR KELVIN\AppData\Local\Programs\Python\Python312\Lib\site-packages\keras\src\trainers\data_adapters\py_dataset_a
  self._warn_if_super_not_called()
864/864 ━━━━━━━━━━━━━━━━━━━━ 37s 37ms/step - accuracy: 0.0790 - loss: 3.4330 - val_accuracy: 0.4074 - val_loss: 1.9524
Epoch 2/20
864/864 ━━━━━━━━━━━━━━━━━━━━ 32s 37ms/step - accuracy: 0.5580 - loss: 1.4537 - val_accuracy: 0.7037 - val_loss: 0.9667
Epoch 3/20
864/864 ━━━━━━━━━━━━━━━━━━━━ 33s 38ms/step - accuracy: 0.7447 - loss: 0.8004 - val_accuracy: 0.8981 - val_loss: 0.2978
Epoch 4/20
864/864 ━━━━━━━━━━━━━━━━━━━━ 35s 40ms/step - accuracy: 0.8858 - loss: 0.3618 - val_accuracy: 0.8750 - val_loss: 0.3990
Epoch 5/20
864/864 ━━━━━━━━━━━━━━━━━━━━ 35s 40ms/step - accuracy: 0.8585 - loss: 0.4587 - val_accuracy: 0.9120 - val_loss: 0.2874
```

Figure 3. 21 Custom CNN training for character recognition

For the ResNet50 model, training was run for 40 epochs with a batch size of 32, image size 224x224 and an Adam optimizer learning rate 0.0001.

```
# 🚀 TRAIN FOR 20 MORE EPOCHS
# ==============================

history_continue = model.fit(
    train_generator,
    epochs=20,
    validation_data=val_generator,
    callbacks=[early_stop, checkpoint]
)
```

```
 Total params: 24,737,350 (94.37 MB)
 Trainable params: 24,684,228 (94.16 MB)
 Non-trainable params: 53,120 (207.50 KB)
 Optimizer params: 2 (12.00 B)
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset
  self._warn_if_super_not_called()
Epoch 1/20
207/207 ━━━━━━━━━━━━━━━━━━━━ 0s 715ms/step - accuracy: 0.6177 - loss: 1.2910WARNING:absl:You are saving your model as an HDF5 f
207/207 ━━━━━━━━━━━━━━━━━━━━ 251s 912ms/step - accuracy: 0.6177 - loss: 1.2910 - val_accuracy: 0.5614 - val_loss: 1.6317
Epoch 2/20
207/207 ━━━━━━━━━━━━━━━━━━━━ 0s 634ms/step - accuracy: 0.6402 - loss: 1.2384WARNING:absl:You are saving your model as an HDF5 f
207/207 ━━━━━━━━━━━━━━━━━━━━ 165s 798ms/step - accuracy: 0.6402 - loss: 1.2385 - val_accuracy: 0.5666 - val_loss: 1.5984
Epoch 3/20
```

Figure 3. 22 ResNet50 Model training

For the EfficientNetB0 model, training was run for 20 epochs with a batch size of 32, image size 224x224 and an Adam optimizer learning rate 0.003.

For the custom color CNN model, training was done for 5 epochs with the Adam optimizer and a learning rate of 0.001.

```
✅ Number of classes: 15
✅ Classes: ['beige', 'black', 'blue', 'brown', 'gold', 'green', 'grey', 'orange', 'pink'
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:208: UserWarning: The
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/torchvision/models/_utils.py:223: UserWarning: Arg
  warnings.warn(msg)
Downloading: "https://download.pytorch.org/models/resnet50-0676ba61.pth" to /root/.cache/t
100%|██████████| 97.8M/97.8M [00:00<00:00, 151MB/s]
✅ Epoch 1/5 — Loss: 0.7519, Accuracy: 75.07%
💾 Model saved after epoch 1 to /content/drive/MyDrive/vcor_resnet50_epoch1.pth
✅ Epoch 2/5 — Loss: 0.4292, Accuracy: 85.29%
💾 Model saved after epoch 2 to /content/drive/MyDrive/vcor_resnet50_epoch2.pth
✅ Epoch 3/5 — Loss: 0.2882, Accuracy: 90.02%
💾 Model saved after epoch 3 to /content/drive/MyDrive/vcor_resnet50_epoch3.pth
```

Figure 3. 23 Custom color classifier training

After all this training was done the models were tested and were able to perform the tasks for which they were designed. A few inferences from the various models are shown below.

```
    # Example usage (uncomment and provide your own char_list)
    char_list = [char_images[i] for i in range(len(char_images))]
    plate_number = show_results(char_list)
    print(plate_number)
```

35]

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 356ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 126ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 98ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 107ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 100ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 110ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 96ms/step
GS99D922
```

Figure 3. 24 The Custom OCR running inference on the image in Figure 3.15

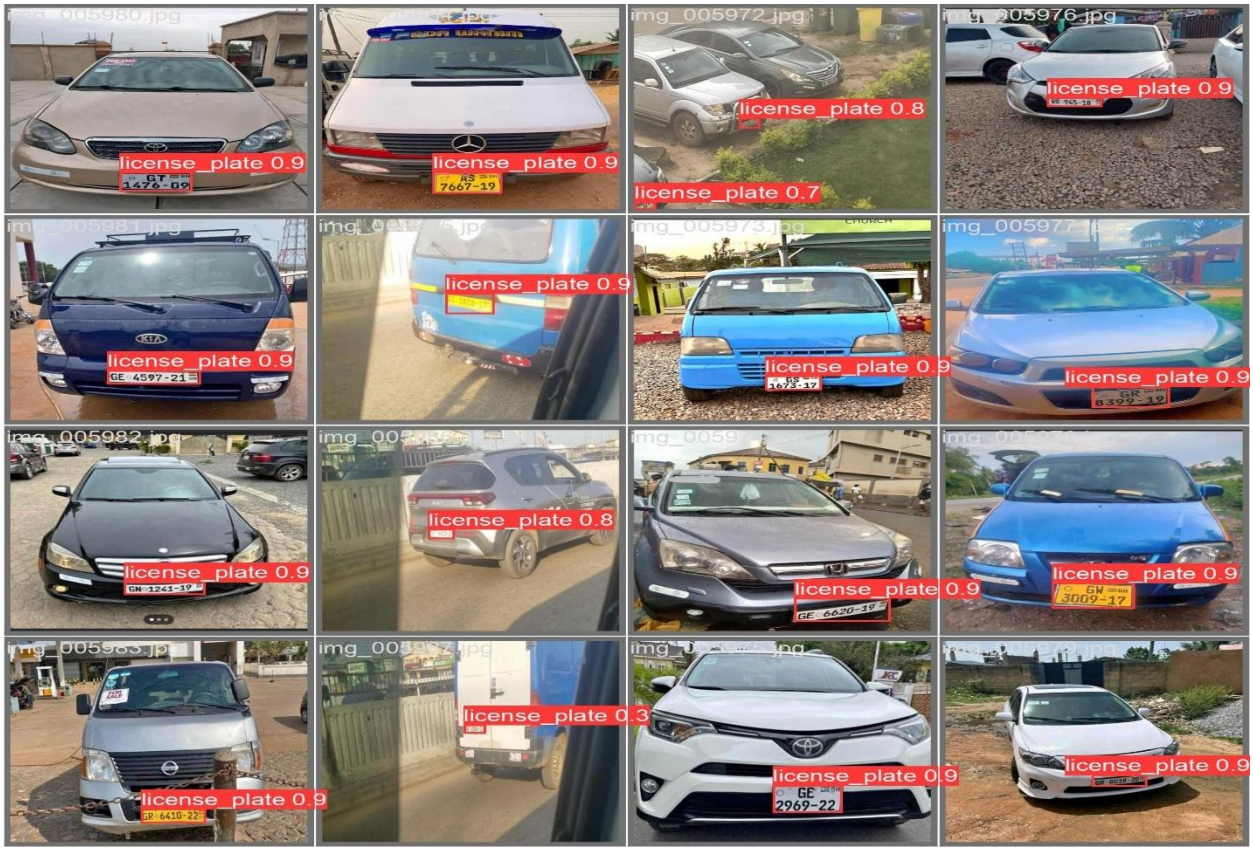

Figure 3. 25 The YOLO models running inference on images of vehicle with Ghanaian license plates

Figure 3. 26 Faster-RCNN inference

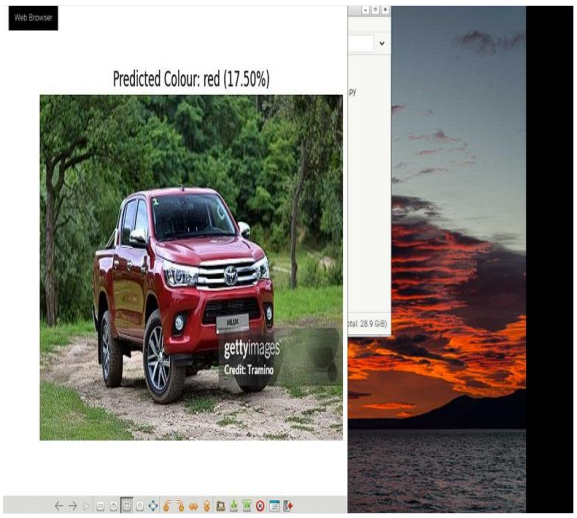

Figure 3. 27 Custom Color Model Inference

Figure 3. 28 EfficientNetB0 Inference

Figure 3. 29 ResNet50 Inference

3.3.1.5 Model Performance Evaluation

This section evaluates the performance of the four machine learning models developed for the vehicle identification system, focusing on their accuracy, efficiency, and suitability for deployment on the Raspberry Pi platform. The assessment leverages a combination of quantitative metrics, visual representations, and comparative analyses to determine the optimal models for license plate detection, optical character recognition (OCR), color classification, and make/model recognition. Each model's performance was rigorously tested using the designated dataset splits (5,000 training, 1,000 validation, and 1,000 testing images) outlined in Section 3.3.1.1, ensuring a robust

evaluation under diverse conditions such as varying lighting, plate orientations, and Ghanaian plate formats. The selection process prioritized not only accuracy but also real-time processing capabilities and compatibility with the resource-constrained Raspberry Pi environment, reflecting the system's design goals for portability and efficiency.

License Plate Detection Model

The license plate detection model underwent a comparative evaluation of three YOLO variants: YOLOv11n, YOLOv5n, YOLOv8n, and Faster-RCNN. These models were trained and tested on the composite dataset, with performance metrics including mean Average Precision (mAP), precision, recall, model size and inference time. YOLOv11n emerged as the superior choice, delivering an impressive mAP of 98.48%, precision 97.39% and a recall of 96.45%, while maintaining an inference time of approximately 328.45ms on the Raspberry Pi. This performance reflects YOLOv11n's enhanced architecture, which includes advanced anchor-free detection and optimized feature extraction, making it particularly effective for detecting Ghanaian plates under challenging conditions such as oblique angles and low light. To visualize its performance, a group of curves (e.g., precision, recall, loss over epochs) (Figure 3.30) along with a results comparison (Table 3.1) is presented, highlighting its stability and convergence during training. The decision to select YOLOv11n was driven by its balanced trade-off between accuracy and real-time capability.

Optical Character Recognition (OCR) Models

The OCR component evaluated three pre-trained models—EasyOCR, Tesseract, and PaddleOCR—along with a custom-designed OCR pipeline. Performance was measured using two key metrics: character accuracy (accuracy of individual characters within a plate) and sequence accuracy (accuracy of the entire plate string). EasyOCR achieved a character accuracy of 20.92% and a sequence accuracy of 1.0%. Tesseract recorded a character accuracy of 17.66% and a sequence accuracy of 3.0%. PaddleOCR outperformed both with a character accuracy of 73.51% and a sequence accuracy of 67.91%, owing to its advanced deep learning framework. However, PaddleOCR's incompatibility with the Raspberry Pi due to high memory and dependency requirements necessitated the development of a custom OCR model. This custom model,

incorporating preprocessing (e.g., grayscale conversion, thresholding) achieved a character accuracy of 89.5%, proving a viable alternative for Pi deployment. The selection of the custom OCR was a pragmatic choice, prioritizing hardware compatibility over PaddleOCR's superior initial performance.

Custom CNN for Color Detection

The custom CNN designed for vehicle color classification was evaluated using training and validation loss curves, alongside accuracy metrics. Trained on a dataset of 10400 annotated images across 15 color classes (e.g., red, blue, black), the model utilized a three-layer convolutional architecture with dropout regularization. The training process, conducted over 5 epochs with the Adam optimizer, resulted in a precision of 90%, mAP 92% and IoU 88%. These metrics indicate a well-generalized model. The model's lightweight design ensured it could operate efficiently on the Raspberry Pi, making it a suitable choice for real-time color detection despite not matching the complexity of larger pre-trained networks.

Make and Model Classification Model

The make and model recognition task compared two pre-trained models: EfficientNet-B0 and ResNet50, both fine-tuned on a dataset of 16185 images covering 196 make/model classes. Evaluation focused precision, mAP and IoU with ResNet50 outperforming EfficientNet-B0, achieving 87.1% precision 89% mAP and 90% IoU. This performance, combined with its relatively efficient inference on the Pi, led to its selection over EfficientNet-B0, which, despite its compound scaling benefits, was slightly slower in this context. The choice underscores the system's emphasis on practical deployment alongside high accuracy.

In summary, the performance evaluation guided the final model selections: YOLOv11nano for its superior detection capabilities, a custom OCR for Pi compatibility, a custom CNN for color detection, and ResNet50 for make/model recognition. These choices align with the system's objectives of accuracy, portability, and real-time operation.

Table 3. 1 License Plate Detection Performance (Faster-RCNN, YOLOv5nano, YOLOv8nano, YOLOv11nano)

Metrics	**Yolov5 nano**	**Yolov8 nano**	**Yolov11 nano**	**Faster - RCNN**
Precision	0.9708	0.9684	0.9739	0.9780
Recall	0.9520	0.9654	0.9645	0.8210
mAP@50	0.9817	0.9829	0.9848	0.9181
mAP@50-95	0.7448	0.7780	0.7762	0.7640
Inference Time	5ms	1.8ms	1.6ms	52ms
Model Size	9.9MB	6.1MB	5.3MB	153MB

Table 3. 2 OCR Model Performance (Character and Sequence Accuracy for EasyOCR, Tesseract, PaddleOCR,)

Accuracy Type	**Tesseract OCR**	**Paddle OCR**	**Easy OCR**
Character	17.66%	73.51%	20.92%
Sequence	3%	67.91%	1%

Table 3. 3 Custom OCR Performance

Section	***Accuracy***	***Loss***
Training	*0.9741*	*0.0657*
Validation	*0.9769*	*0.0338*

Table 3. 4 Custom CNN Color, Make and Model Classification Performance Metrics (EfficientNet-B0, ResNet50)

Task	**Model**	**Precision(%)**	**Mean Avg precision(%)**	**Intersection over union(%)**	**Inference speed**
Make/Model	ResNet50	87.1	89.0	80.0	10ms
Make/Model	EfficientNet-B0	83.0	85.0	79.0	15ms
Colour Detection	Custom CNN	90.0	92.0	88.0	5ms

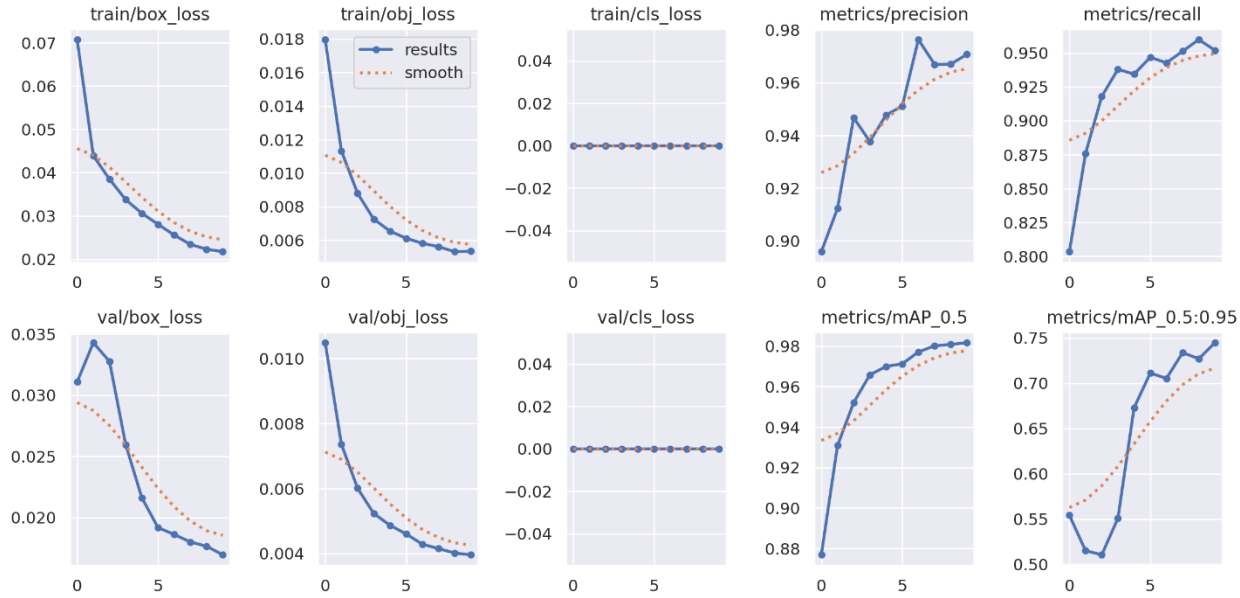

Figure 3. 30 Group of Curves for YOLOv11nano Results

3.3.2 Hardware System

The hardware system consists of the physical devices used in the development of the solution. In order to realize the functions, the system will perform there are devices required for each function. A camera to capture the images of approaching vehicles, a sensor to detect these vehicles a microcontroller to serve as the main portable edge computer or the brain of the system, indicators to signal the results of the verification to the on-site official and a power supply for the entire system. Taking this into consideration a few devices were scrutinized for the performance of each function.

Micro Controller

For the micro controller acting as the hub for the system three main devices were looked at. They were the Raspberry Pi 4 Model B, the Orange Pi 5 and the NVIDIA Jetson Nano. The Raspberry Pi 4 Model B was selected for its optimal balance of performance, affordability, and portability, making it ideal for the system's requirements. With a 1.5GHz quad-core processor, 4GB RAM, and dual-band Wi-Fi, it supports real-time processing of machine learning models (e.g.,

YOLOv11, ResNet50) and cloud connectivity, critical for the LPR-MMC system. Its GPIO pins enabled seamless integration with the ultrasonic sensor and camera module, while its low power consumption (approximately 3-5W) and compact size (85mm x 56mm) ensured efficient operation in a portable setup, aligning with the project's cost-effective deployment goals (priced at ~GHS1200).

Ultrasonic Sensor

For the ultrasonic sensor which would detect the approaching vehicle the candidates were the HC-SR04, the US-100 and the JSN-SR04T. The HC-SR04 ultrasonic sensor was chosen for its reliability, affordability, and suitability for vehicle detection in the LPR-MMC system. Operating with a range of 2cm to 400cm and an accuracy of ±3mm, it effectively detects approaching vehicles to trigger the camera, ensuring timely image capture. Its simple interface (trigger and echo pins) integrates seamlessly with the Raspberry Pi 4's GPIO pins, and its low cost (~GHS18) aligns with the project's budget constraints. Additionally, its non-contact operation and resistance to environmental factors like light variations make it ideal for outdoor deployment.

Camera

For the camera module which would capture the images of the detected approaching vehicle the candidates were the Raspberry Pi V2 Camera, the Raspberry Pi NoIR V2 Camera and a high quality 13MP camera module. The Raspberry Pi Camera Module V2 was selected for its compatibility, image quality, and cost-effectiveness, making it well-suited for the LPR-MMC system. Featuring an 8-megapixel Sony IMX219 sensor, it captures high-resolution images (up to 3280x2464 pixels) with sufficient clarity for license plate and vehicle feature detection, even under varying lighting conditions. Its direct connection via the Raspberry Pi 4's CSI port ensures low-latency data transfer, supporting real-time processing, while its compact design and affordability (~GHS380) align with the project's portability and budget goals. Additionally, its adjustable focus and wide-angle lens enhance adaptability to diverse vehicle angles and distances.

Power Supply

The 20,000mAh power bank was chosen for its cost-effectiveness and ease of use, making it an ideal power solution for the LPR-MMC system. Priced at approximately GHS200, it offers an affordable alternative to fixed power supplies, aligning with the project's budget constraints while providing up to 3-4 hours of operation for the Raspberry Pi 4 and peripherals. Its plug-and-play design, requiring no complex installation, enhances portability and simplifies deployment in various locations, such as parking lots or security checkpoints. Additionally, its widespread availability and compatibility with USB power standards ensure hassle-free recharging and reliable performance in a portable setup.

In addition to these a Raspberry Pi 7- inch TFT screen along with red and green LEDs were selected to display the results of the verification of the vehicle details on-site to the security personnel. This results in the hardware system having the architecture seen in Figure 3.31.

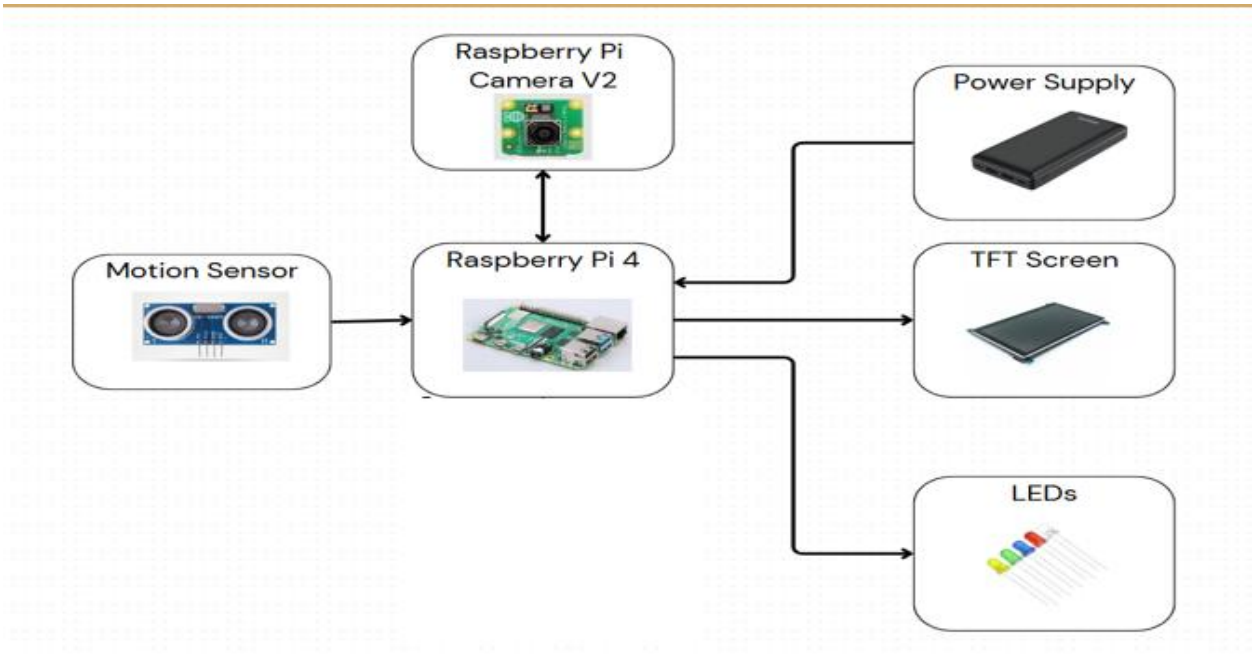

Figure 3. 31 Hardware System Architecture

All the peripherals are compatible with the Raspberry Pi and would be interfaced with the Raspberry Pi through it's GPIO pins. The system's final look was modeled using the Onshape modelling software. The modeled outcome is as seen in the figure below.

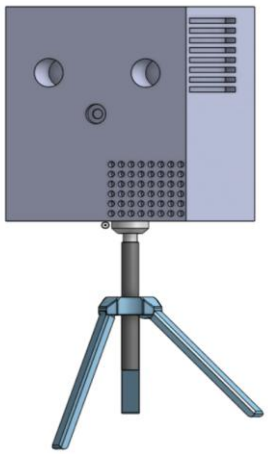

Figure 3. 32 Model of the expected outcome of the solution

3.3.3 Software System

The software system consists of all the scripts used to engage the major functions of the peripherals in the operation of the system i.e. motion detection for the ultrasonic sensor image capture etc. down to the script used to transfer and capture inference and verification results from and to the raspberry pi and the entire mobile application. Each task has python scripts for its execution

The software system serves as the backbone of the LPR-MMC project, orchestrating the flow of data from hardware inputs to user-facing outputs while ensuring secure, efficient, and real-time vehicle monitoring. The system integrates a React-based frontend for intuitive user interaction, a Firebase backend for robust data management, and custom scripts on the Raspberry Pi for model inference and communication. This architecture not only handles the machine learning predictions but also provides a user-friendly experience, making it accessible even for non-technical users like security personnel. By leveraging open-source tools and cloud services, the software minimizes development overhead while maximizing reliability.

At the core of the software is a client-server model where the Raspberry Pi acts as an edge device for local processing, and Firebase provides the cloud-based infrastructure for storage, authentication, and synchronization. The Pi runs Python scripts (using libraries like OpenCV and TensorFlow Lite) to perform inferences on captured images, extracting license plate text, make, model, and color. These results are transmitted via HTTPS to Firebase, where they are compared against a database of verified vehicles. If a match is found (indicating authorization), the system

updates the status to "clear" and triggers green LEDs on the Pi; otherwise, it flags the vehicle as "flagged," lights red LEDs, and logs details for review. This feedback loop ensures on-site visibility without relying solely on the mobile app, adding a layer of redundancy that's crucial in real-world deployments.

The mobile application, built with React and Wouter for routing, is the primary interface for users. It receives push notifications via Firebase Cloud Messaging (FCM) for real-time alerts—e.g., "Vehicle GE8221128 flagged at Gate 1"—and displays verification results in a clean, card-based UI. Users can view all scanned vehicles, filter flagged ones. Role-based access (via Firebase Authentication) restricts admins to full database management while officers see only relevant scans, enhancing security. The app fetches data from Firestore in real-time, using listeners for live updates, which keeps the interface responsive and up-to-date without constant polling.

Firebase, as the backend, utilizes a NoSQL structure in Firestore for flexible data handling. Key collections include "Users" for authentication details, "Vehicles" for storing inferred attributes like licensePlate, make, model, color, and status, "Scans" for logging each detection event with timestamps and image URLs (stored in Firebase Storage), and "Devices" for tracking Raspberry Pi units. This setup allows for efficient queries—e.g., searching by licensePlate—and optional Cloud Functions for automated tasks like alerting on repeated flags. The use of Firebase reduces maintenance costs and simplifying deployment.

To illustrate the software's operation, the following diagrams provide a visual breakdown. The app workflow outlines user interactions, the data flow traces information from capture to storage, the use case diagram highlights key actors and scenarios, and the ER diagram models the database relationships.

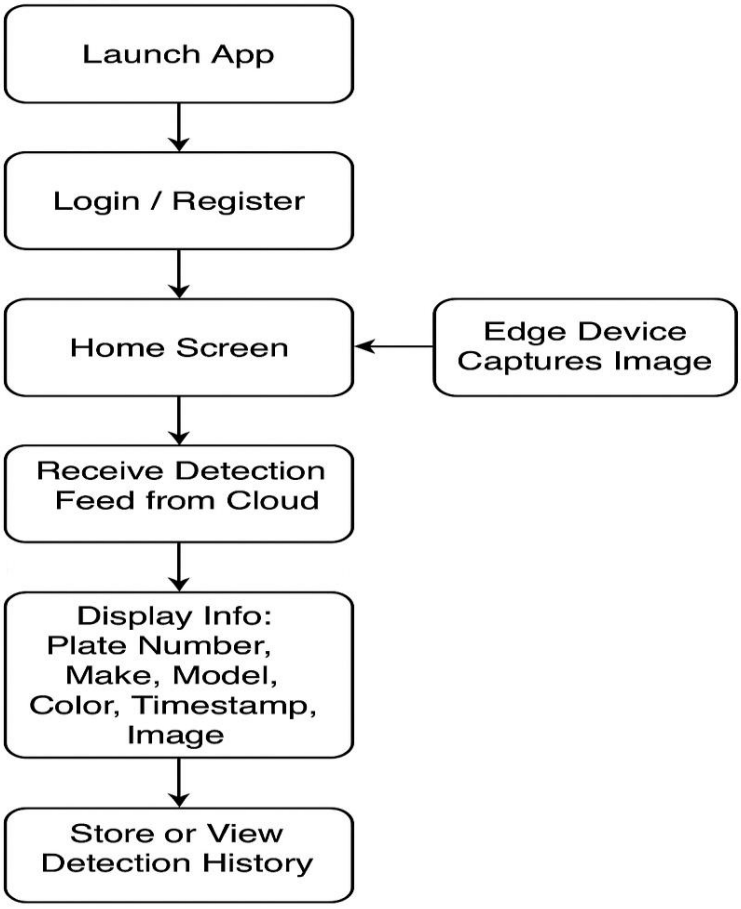

Figure 3. 33 App Workflow Diagram

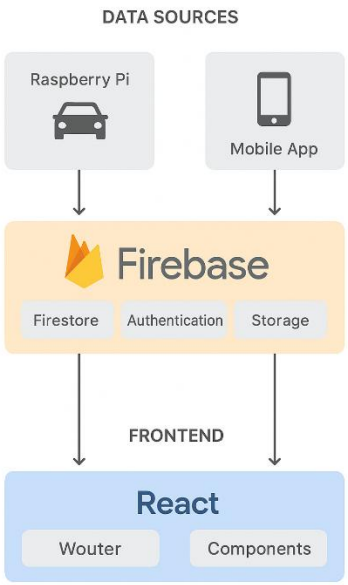

Figure 3. 34 Data Flow Diagram

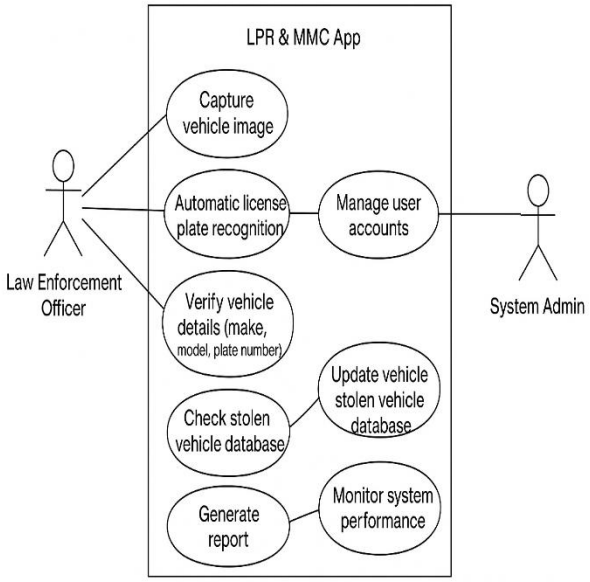

Figure 3. 35 Use Case Diagram

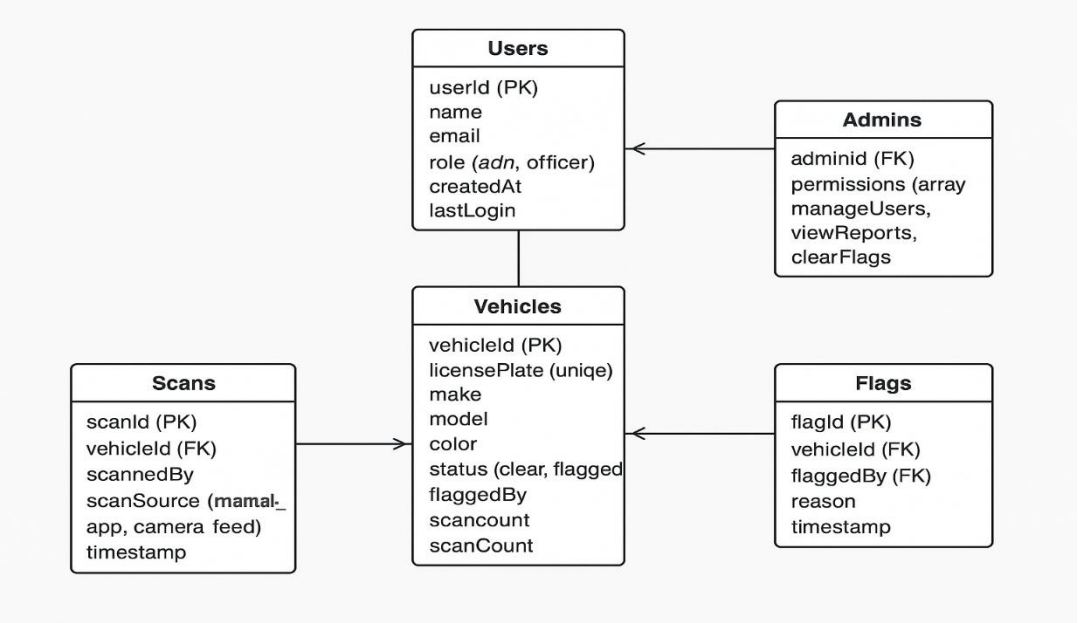

Figure 3. 36 Entity Relationship (ER) Diagram

Overall, the software system's design emphasizes ease of integration and user-centric features, drawing from practical needs observed during prototyping.

3.4 System Modelling and Integration

System modeling and integration represent the critical phase where the individual components—hardware, software, and AI models—converge into a cohesive, functional unit. This section outlines the conceptual modeling of the LPR-MMC system, including diagrams that visualize its architecture and workflows, followed by a detailed description of the integration process. The goal was to create a portable, real-time solution that detects vehicles, processes images via AI, verifies data against a cloud database, and provides user feedback. Modeling was conducted using tools such as Onshape for physical prototyping. Integration focused on modularity, allowing for easy troubleshooting and scalability—imagine building a puzzle where each piece (sensor, camera, models, app) fits precisely, but with room to swap parts if needed. This approach not only validated the theoretical design but also highlighted practical adjustments, resulting in a system that performs reliably.

Additionally, a flow diagram (Figure3.2) illustrated the temporal flow of operations, from sensor detection to app notification, emphasizing synchronization points like API calls to Firebase. For physical aspects, Onshape software was used to create a 3D model of the hardware assembly, simulating the placement of the Raspberry Pi, camera, sensor, LEDs, and TFT screen on a portable stand (Figure 3.32).

Integration commenced with hardware-software linkage on the Raspberry Pi. The ultrasonic sensor (HC-SR04) connects to GPIO pins, triggering Python scripts that activate the Camera Module V2 via the CSI port. These scripts, written in Python with OpenCV, capture images and feed them into the AI models: YOLOv11n for plate detection, custom OCR for text extraction, custom CNN for color classification, and ResNet50 for make/model recognition. Results are packaged as JSON (e.g., {"licensePlate": "GE8221128", "make": "Toyota", "model": "Corolla", "color": "black"}) and sent via HTTPS to Firebase using the Firebase Admin SDK. On the backend, Cloud Functions (if enabled) compare against the Vehicles collection; a match sets status to "clear" and triggers a green LED via GPIO output, while mismatches flag the entry, light a red LED, and push notifications via Firebase Cloud Messaging (FCM).

The frontend integration ties into this via the React app, which uses Firestore listeners for real-time updates—e.g., querying the Scans collection to display recent detections. Authentication ensures secure access, with admins able to review Flags. Data flow is bidirectional: Pi uploads scans, app retrieves and displays, and sends results back to the Pi for LED updates. This closed loop was tested iteratively, revealing integration challenges like network latency and overall latency due to limited pi resources.

In essence, the modeling and integration transformed isolated components into a unified system, much like assembling a reliable vehicle from parts—each tested for fit before the full drive. The result is a robust, user-friendly setup that meets the project's objectives.

3.5 Development tools and Material Requirements

This section outlines the development tools and material requirements utilized in the LPR-MMC system, encompassing both software and hardware components essential for implementation and

testing. The selection of tools and materials was guided by criteria such as cost-effectiveness, compatibility with the Raspberry Pi ecosystem, ease of integration, and availability in the Ghanaian market. Emphasis was placed on open-source solutions to minimize expenses and facilitate future modifications. The total estimated cost for hardware was approximately GHS 3,500, reflecting a budget-conscious approach without compromising functionality. Software tools were primarily free and open-source, leveraging community-supported libraries to accelerate development. A breakdown of key items follows, categorized for clarity.

Software Tools

The software stack was chosen for its robustness in handling machine learning, image processing, and cloud integration, while being lightweight enough for the Raspberry Pi. Primary tools include:

- Python 3.9: Served as the core programming language for backend scripts on the Raspberry Pi, including sensor control, model inference, and API communication. Its extensive ecosystem and ease of use made it ideal for rapid prototyping, with libraries like RPi.GPIO for hardware interfacing.
- OpenCV 4.5: Used for image preprocessing, capture, and manipulation (e.g., grayscale conversion, edge detection). Selected for its efficiency on embedded devices and comprehensive computer vision capabilities.
- Ultralytics 8.3 / TensorFlow 2.10 / PyTorch 1.12: Frameworks for training and deploying AI models (e.g., YOLOv11, ResNet50, custom CNN).
- React 18 with Wouter: Frontend framework for the mobile app, enabling responsive UI components like VehicleCard.tsx. Wouter handled routing for pages such as Vehicles.tsx and Flagged.tsx, chosen for its lightweight alternative to React Router.
- Firebase SDK: Backend service for Firestore (NoSQL database), Authentication, Storage, and Cloud Messaging. It provided real-time data sync and secure handling of vehicle scans, with no setup costs for the free tier.

- Other Libraries: TailwindCSS and Shadcn UI for styling the app; LabelImg for dataset annotations; Flask for local API testing during development and PiCamera2 and GPIO libraries for interfacing with the hardware components

These tools were installed via pip on the Pi or through npm for the app, ensuring cross-platform compatibility. Development was conducted on a standard laptop (Windows 11) for initial scripting, with remote SSH access to the Pi for testing.

Hardware Materials

Hardware selection prioritized portability, low power consumption, and affordability, with components sourced from local electronics suppliers or online platforms like Jumia Ghana. The Raspberry Pi 4 Model B acted as the central hub, interfacing with peripherals via GPIO and CSI ports. Below is a list, including rationale of the major components.

- Raspberry Pi 4 Model B (4GB RAM): Central microcontroller for processing and integration. Chosen for its quad-core CPU and GPIO support.
- HC-SR04 Ultrasonic Sensor: Vehicle detection trigger (~GHS 18). Selected for accuracy and low cost.
- Raspberry Pi Camera Module V2: Image capture Preferred for high resolution and Pi compatibility.
- 20,000mAh Power Bank: Portable power supply (~GHS 100). Opted for extended runtime and ease of recharging.
- Raspberry Pi 7-inch TFT Screen: On-site display (~GHS 800). Chosen for clear visualization of results.
- Red and Green LEDs (set of 8): Verification indicators (~GHS 4). Simple and energy-efficient for signaling.
- Miscellaneous (Jumper Wires, resistors): Wiring. Essential for assembly
- Camera Stand: Supporting main device and allowing it to capture vehicle images at appropriate height.

A comparative table of hardware costs is provided below for transparency.

Table 3. 5 Hardware Materials and Cost

Component	Quantity	Unit Price (GHS)	Total Price (GHS)
Raspberry Pi 4	1	1197	1197
Raspberry Pi Camera V2	1	320	320
HC-SR04 Ultrasonic Sensor	1	18	18
Raspberry Pi SD Card	1	79	79
20000mAh Power Bank	1	100	100
Resistors (200 ohm and 300 ohm)	10	0.5	5
TFT LCD Raspberry Pi 4 7 Inch	1	1268	1268
LEDs	8	0.5	4
Camera Stand	1	400	400
Jumper wires (M-M and F-F)	30	1	30
Total	25	3384	3421

Other components such as the enclosure were self-made at no cost. The enclosure in particular was made of cardboard box covered in masking tape. The raspberry pi, pi camera and TFT display were provided by collaborators from IT Consortium.

Development Environments and Additional Tools

Beyond core tools, the development environment included Visual Studio Code (VS Code) as the IDE, with extensions for Python debugging and React previews, facilitating collaborative coding. Git was used for version control via GitHub, ensuring reproducible builds. For modeling, Onshape provided free 3D CAD tools to design the hardware enclosure and Jupyter Notebooks for initial ML experiments. No proprietary software was required, keeping the setup accessible and cost-free.

CHAPTER 4 - DESIGN IMPLEMENTATION AND TESTING

4.0 Introduction

This chapter details the practical realization of the LPR-MMC system, implementing and testing the theoretical design outlined in Chapter 3. Building on the system's architecture, hardware components, software framework, and AI models, the implementation phase involved assembling the physical prototype, deploying the software on the Raspberry Pi, and integrating all elements into a functional unit. The focus was on ensuring proper operation in real-world scenarios, detecting approaching vehicles, processing images for license plate recognition, make/model classification, and color detection, and verifying results against the Firebase database. Testing involved attempting to use the solution in different instances and recording its performance. The results provide insights into the system's effectiveness, limitations, and alignment with project objectives.

4.1 System Implementation Process

The system implementation process transformed the conceptual designs from Chapter 3 into a tangible, operational prototype, emphasizing modularity, portability, and efficiency. This phase began with procuring and assembling hardware components, followed by software deployment on the Raspberry Pi 4 Model B, and culminated in end-to-end integration testing. The Raspberry Pi served as the central hub, interfacing peripherals via GPIO and CSI ports while running optimized Python scripts for real-time processing. Hardware assembly involved fixing the ultrasonic sensor (HC-SR04), camera module (V2), LEDs, and TFT screen within the main enclosure and then mounting the enclosure on the camera stand, ensuring stability and optimal positioning for vehicle detection at distances up to 4 meters. Power was supplied via a 20,000mAh bank.

Software implementation included installing the Raspbian OS, configuring libraries (e.g., OpenCV, TensorFlow Lite), and deploying the AI models: YOLOv11 for plate detection, custom OCR for text extraction, custom CNN for color classification, and ResNet50 for make/model recognition. Scripts were structured modularly—e.g., a main loop monitored the sensor, triggered image capture, ran inferences sequentially, and sent JSON payloads to Firebase via HTTPS.

Integration ensured bidirectional communication: inference results were verified against the Vehicles collection in Firestore, with status updates triggering LED signals (green for "clear," red for "flagged") and FCM notifications to the React app.

4.1.1 Hardware

The hardware implementation of the license plate detection system is centered around the Raspberry Pi 4 Model B, which serves as the main processing unit. This section details the physical connections of the Raspberry Pi camera module, the HC-SR04 ultrasonic sensor, and the red and green LED groups to the GPIO ports, including the specific wiring configurations for parallelism and voltage division. The setup is housed in a custom enclosure, and the following steps provide a clear guide for replication.

Raspberry Pi and GPIO Overview

The Raspberry Pi 4 features a 40-pin GPIO header, with pins numbered from 1 to 40 in a zigzag pattern across two rows (odd numbers on the left, even numbers on the right). Pin 1 is identified by a white square marker on the board, located at the top-left corner when the USB ports are at the bottom. A pinout diagram is provided in Figure 4.1, which labels each pin's physical number and its corresponding function (e.g., 3.3V, 5V, GND, GPIO). This diagram is essential for correctly identifying connection points during assembly.

Camera Module Connection

The Raspberry Pi Camera Module 2 is connected to the Raspberry Pi via the Camera Serial Interface (CSI) port, located between the HDMI ports and the audio jack. The camera's ribbon cable is inserted into the CSI port with the blue side facing the Ethernet port, ensuring a secure fit. The other end of the ribbon cable is connected to the camera module, aligning the blue side with the camera's connector. This connection enables the Pi to capture images for license plate detection. The camera module is powered and controlled directly through the CSI interface, requiring no additional GPIO connections.

HC-SR04 Ultrasonic Sensor Connection

The HC-SR04 ultrasonic sensor is used to detect approaching vehicles and trigger the camera. It is connected to the GPIO pins as follows:

VCC (5V): Connected to Pin 4 (5V power).

GND: Connected to Pin 9 (ground).

Trig (trigger): Connected to Pin 7 (GPIO4) via a female-to-female jumper wire.

Echo (echo): Connected to Pin 11 (GPIO17) through a voltage divider circuit to step down the 5V output to 3.3V, compatible with the Pi's GPIO input.

The voltage divider is constructed using a 200Ω resistor (R1) and a 330Ω resistor (R2). The Echo pin of the sensor is connected to one end of R1, the other end of R1 is joined to one end of R2 and a jumper wire leading to Pin 11, and the other end of R2 is connected to Pin 9 (GND). This configuration reduces the 5V signal to approximately 3.1V protecting the Pi's GPIO.

LED Group Connections

Two groups of LEDs (three red and three green) are used to indicate the verification result of the license plate. Each group is wired in parallel to a single GPIO pin for unified control, with current-limiting resistors to prevent damage. The connections are as follows:

Red LED Group:

Signal: Connected to Pin 13 (GPIO27) via a female-to-female jumper wire.

Each of the three red LEDs has its anode (longer leg) connected to a 200Ω resistor. The other ends of the resistors are twisted together with the jumper wire from Pin 13.

The cathodes (shorter legs) of all three LEDs are twisted together and connected to Pin 14 (GND) via another jumper wire.

Green LED Group:

Signal: Connected to Pin 15 (GPIO22) via a female-to-female jumper wire.

Each of the three green LEDs has its anode connected to a 200Ω resistor. The other ends of the resistors are twisted together with the jumper wire from Pin 15.

The cathodes of all three LEDs are twisted together and connected to Pin 25 (GND) via another jumper wire.

The parallel configuration ensures that a single GPIO output controls all LEDs in each group simultaneously. The 200Ω resistors limit the current to approximately 6-7mA per LED, providing safe and visible illumination when powered by the Pi's 3.3V GPIO output.

Enclosure Assembly

The Raspberry Pi, camera module, HC-SR04 sensor, and LED groups are housed in a custom cardboard enclosure to protect the components and facilitate outdoor use. The assembly process is as follows:

1. Cut holes in the cardboard to mount the HC-SR04 sensor, ensuring its transducers protrude fully for unobstructed ultrasonic wave transmission (refer to Figure 4.3).
2. Secure the Raspberry Pi inside the enclosure using adhesive tape or brackets, positioning the GPIO header and CSI port for easy access.
3. Mount the camera module on the front of the enclosure, aligning its lens with a small opening to capture images of approaching vehicles.
4. Arrange the LED groups on the enclosure's surface, with red LEDs on the left and green LEDs on the right, and secure them with tape or glue.
5. Connect all jumper wires, resistors, and LED legs as described, twisting the ends together tightly and wrapping with electrical tape or soldering to prevent shorts.
6. The power bank is mounted in the enclosure with a USB-C cable connecting it to the raspberry pi's power port.

Figure 4.2 illustrates the internal layout of the devices within the enclosure.

Final Hardware Configuration

The completed hardware is mounted on a stand to position the sensor and camera at an optimal height for vehicle detection (approximately 1-1.5 meters above ground). The enclosure is attached securely to ensure stability. The 7-inch DSI touchscreen display, connected via the DSI port and

GPIO Pins 2 (5V), 6 (GND), 3 (GPIO2/SDA), and 5 (GPIO3/SCL) for touch, is placed on one side of the enclosure to provide real-time feedback. Figure 4.3 depicts the final hardware setup, including the enclosure on the stand, ready for operation.

Verification and Testing

After assembly, power on the Raspberry Pi using a 5V 3A power supply connected to the USB-C port. Verify the connections by running the provided Python scripts to ensure the camera captures images, the sensor detects distances, and the LEDs respond to GPIO signals. Test the HC-SR04 by placing objects at known distances (e.g., 50cm, 150cm) and comparing the output to expected values. Adjust the enclosure or wiring if inconsistencies are observed.

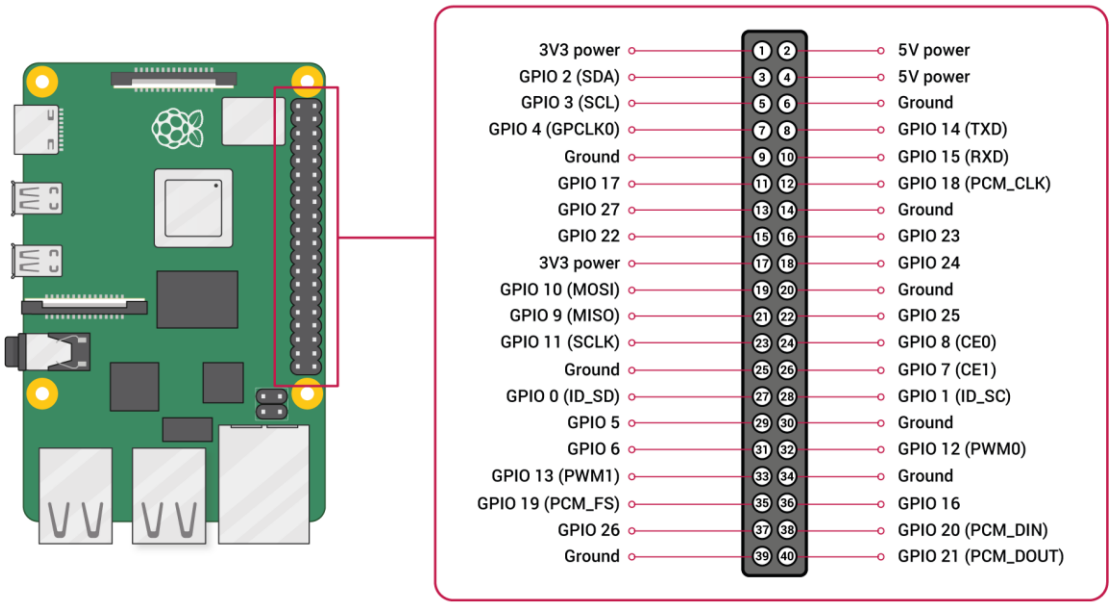

Figure 4. 1 Labeled GPIO pinout diagram of the Raspberry Pi [13]

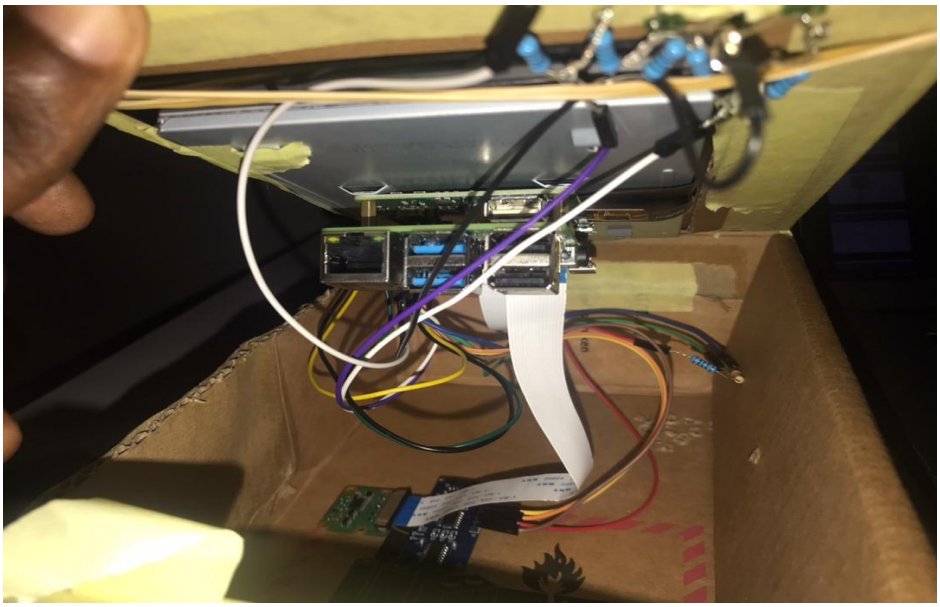

Figure 4. 2 Arrangement of components in enclosure

Figure 4. 3 Final Design Outcome

Figure 4. 4 Placement of TFT screen and LEDs

4.1.2 Software

The software component of the license plate detection system is developed in part to run on the Raspberry Pi 4, integrating sensor data, camera control, machine learning inference, result transmission, LED signaling, and in another to create the backend and frontend interface for the user to access the system results though the mobile application. This section outlines the key scripts, their functionalities, and the supporting backend, with references to user interface images for visualization. The code is written in Python, leveraging libraries such as `RPi.GPIO`, `picamera2`, `ultralytics`, `tensorflow`, `torch`, and `requests`, and is designed to be modular for ease of replication.

Sensor Detection and Camera Initiation

The system uses an HC-SR04 ultrasonic sensor to detect approaching vehicles and trigger the Raspberry Pi Camera Module. The script in the Figure below handles this process. The sensor is connected to GPIO Pins 7 (Trig) and 11 (Echo), with a voltage divider to ensure compatibility with the Pi's 3.3V logic levels. The script initializes the GPIO pins and defines a `distance()` function to calculate the distance based on the time-of-flight of ultrasonic waves:

```
# ---------------- SENSOR + CAMERA ----------------
GPIO.setmode(GPIO.BOARD)
TRIG, ECHO = 7, 11
GPIO.setup(TRIG, GPIO.OUT)
GPIO.setup(ECHO, GPIO.IN)

picam2 = Picamera2()
picam2.start()

def distance():
    GPIO.output(TRIG, True)
    time.sleep(0.00001)
    GPIO.output(TRIG, False)

    start_time = time.time()
    while GPIO.input(ECHO) == 0:
        pulse_start = time.time()
        if pulse_start - start_time > 0.1:
            return None
    start_time = time.time()
    while GPIO.input(ECHO) == 1:
        pulse_end = time.time()
        if pulse_end - start_time > 0.1:
            return None

    pulse_duration = pulse_end - pulse_start
    return round(pulse_duration * 17150, 2)  # cm
```

Figure 4. 5 Sensor and Camera Trigger Code

The main loop checks the distance every second, triggering the camera to capture an image when a vehicle is within 300cm, with a 20-second cooldown to prevent rapid captures. The image is saved with a timestamped filename.

Model Inference

The captured images are processed using a pipeline of machine learning models to detect and analyze license plates. The script integrates these models:

YOLOv11n for Plate Detection: Trained on a custom dataset (`yolov11n_detect.pt`), it identifies license plate regions in the image.

OCR Model for Plate Reading: A Keras model (`final_model.keras`) segments and recognizes characters on the plate.

Color Classifier (ResNet50): Finetuned (`vcor_finetuned_resnet50_final.pth`) to predict the vehicle's color from 15 classes (e.g., red, blue).

Make/Model Classifier (ResNet50): Finetuned (`stanford_car_resnet50_model_finetuned.keras`) to identify the vehicle make and model from a predefined list (`car_classes.txt`).

The inference function `detect_and_read_plate()` processes the image, extracts the plate number, color, and make/model, and annotates the output:

```
def ocr_predict(char_list):
    dic = {i: c for i, c in enumerate("0123456789ABCDEFGHIJKLMNOPQRSTUVWXYZ")}
    output = []
    for ch in char_list:
        img_ = cv2.resize(ch, (28,28), interpolation=cv2.INTER_AREA)
        img = fix_dimension(img_)
        img = img.reshape(1,28,28,3)
        y_ = np.argmax(ocr_model.predict(img), axis=-1)[0]
        output.append(dic[y_])
    return ''.join(output)

# ---------------- COLOR + MAKE-MODEL PREDICTION ----------------
def predict_color(image_bgr):
    image_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
    pil_img = Image.fromarray(image_rgb)
    input_tensor = color_transform(pil_img).unsqueeze(0).to(device)
    with torch.no_grad():
        outputs = color_model(input_tensor)
        probs = F.softmax(outputs, dim=1)
        top_prob, top_class = torch.max(probs, 1)
    return colour_labels[top_class.item()], top_prob.item() * 100

def predict_make_model(image_bgr):
    img_rgb = cv2.cvtColor(image_bgr, cv2.COLOR_BGR2RGB)
    pil_img = Image.fromarray(img_rgb).resize(IMG_SIZE)
    img_array = np.expand_dims(np.array(pil_img), axis=0)
    img_array = img_array.astype("float32")
    preds = make_model.predict(img_array)
    idx = np.argmax(preds)
    return make_model_classes[idx], preds[0][idx] * 100

# ---------------- MAIN PIPELINE ----------------
def detect_and_read_plate(image_path, save_output=True):
    image = cv2.imread(image_path)
    if image is None:
```

Figure 4. 6 Model Inference functions

The `ocr_predict()` functions preprocess the plate image and predict the text, while `predict_color()` and `predict_make_model()` use ResNet50 models for color and make/model classification, respectively.

Sending Inference Results

The inference results are transmitted to a backend server for further processing or storage. The script packages the data into a JSON format and sends it via an HTTP POST request.

```
# Upload to Firebase Storage
remote_path = f"scans/{filename}"
try:
    image_url = upload_image_to_storage(bucket, result["annotated_image"], remote_path)
except Exception as e:
    print("Upload error:", e)
    image_url = None
```

Figure 4. 7 Code snippet for sending inference results

LED Control Based on Results

The LED groups (red on Pin 13, green on Pin 15) indicate the verification result received from the backend. The script listens for the result and controls the LEDs accordingly.

```
# Set LEDs according to verification result from Firebase
set_led(green_on=bool(verified), red_on=not bool(verified))
print(f"Verification result for plate {scan_doc.get('licensePlate')}: {verified}")
```

Figure 4. 8 Code snippet for activating LEDs based on verification results

The function lights green LEDs for a positive result and red LEDs for a negative result based on a boolean `verified` from the backend.

Backend Implementation

The backend, implemented in Python using Flask, receives the JSON data, processes the verification (e.g., via a database check), and sends the result back to the Pi. A sample of the backend log is in the figure below:

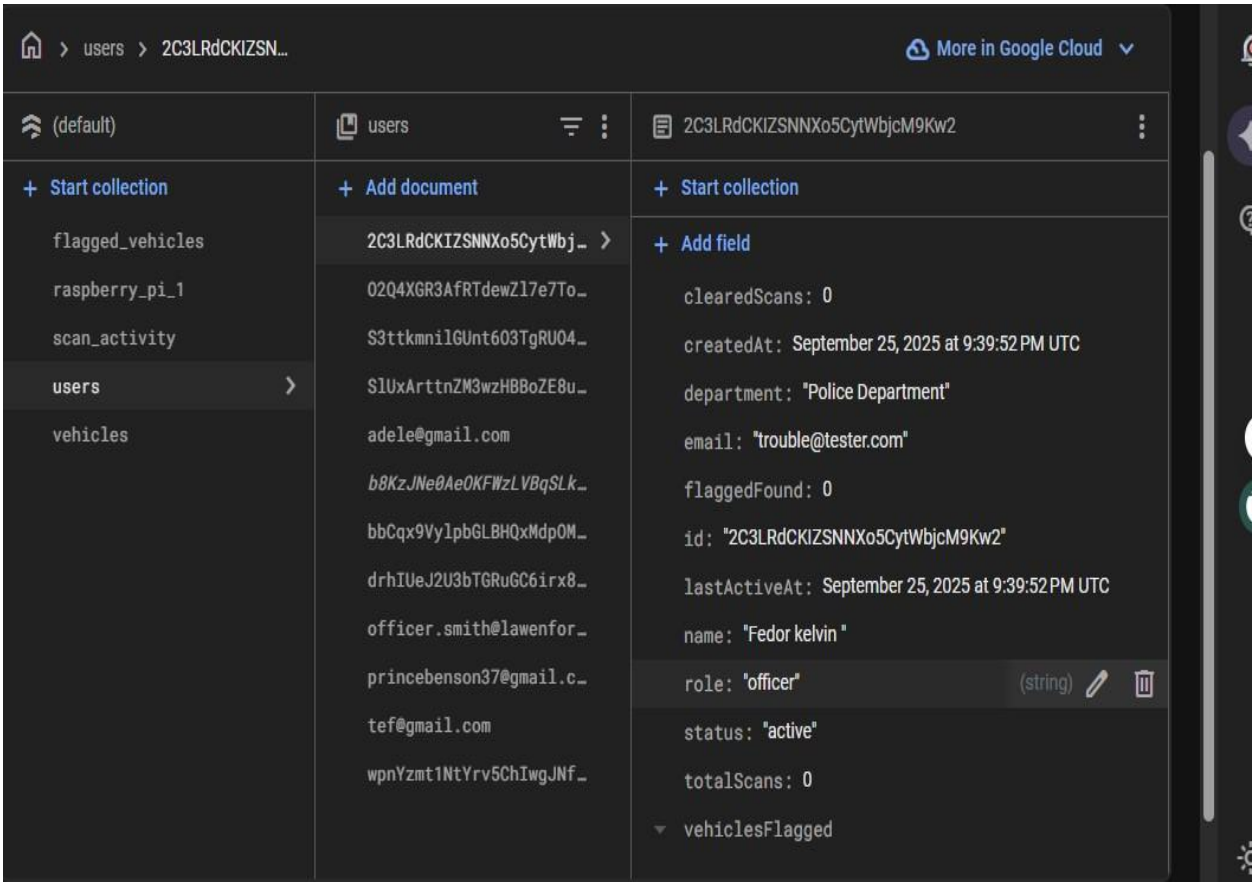

Figure 4. 9 Backend Logs

User Interface

The design of the software, both backend and frontend result in the various User Interface (UI) screens displayed in Figure 4.10 below.

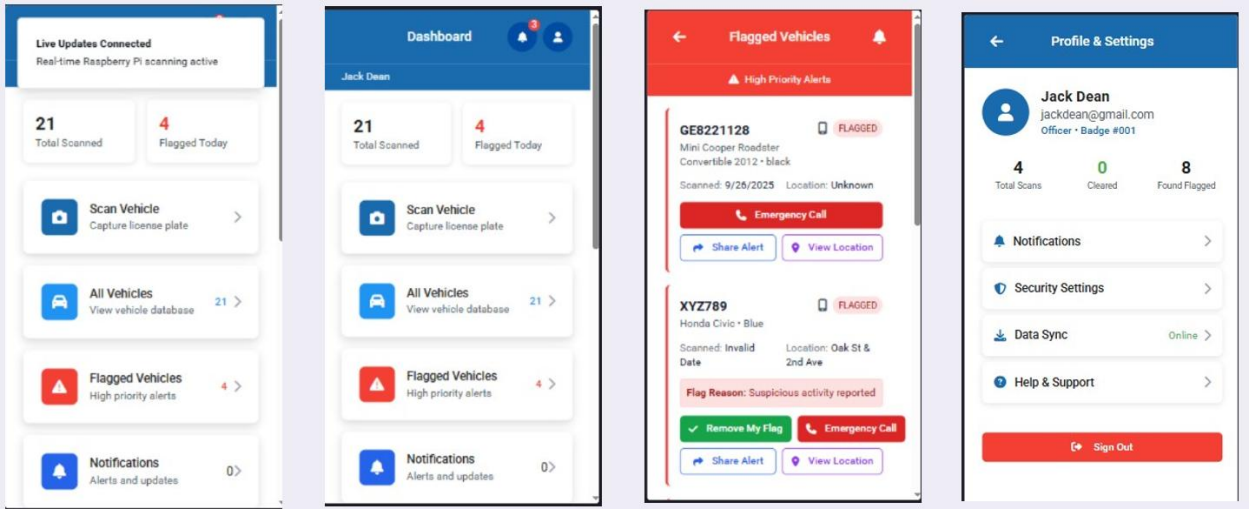

Figure 4. 10 Mobile App UI

All the sections are integrated into a main loop, combining sensor detection, camera capture, inference, result transmission, and LED control. The system is then tested.

4.2 Testing and Results

The testing and results phase evaluated the LPR-MMC system's performance in the real-world focusing on its ability to detect, process, and verify vehicle data accurately and efficiently. This section details the experimental setup, methodology, and outcomes based on testing conducted on 10 vehicles with Ghanaian license plates. The objective was to assess key metrics such as detection accuracy, OCR precision, color classification reliability, make/model recognition, while identifying strengths, weaknesses, and environmental influences. Testing occurred outdoors at a controlled entry path, using the fully integrated prototype powered by the 20,000mAh power bank. The Raspberry Pi 4 Model B processed images in real-time, with results synced to Firebase and displayed via the app and LEDs, providing a comprehensive validation of the end-to-end workflow.

The test involved a set of 10 vehicles. Each vehicle approached at a speed of 5-10 km/h, triggering the HC-SR04 sensor at a distance of approximately 2-3 meters. The Raspberry Pi Camera Module V2 captured images (resolution 1280x720 for speed), which were processed by YOLOv11 for plate detection, custom OCR for text, custom CNN for color, and ResNet50 for make/model. Results were compared against a pre-populated Firebase Vehicles collection containing 20 verified entries, with success defined as a match triggering a green LED and failure a red LED, corroborated by similarly color-coded application notifications and manual logs recorded and accuracy (manual verification of inferred vs. actual data).

Results indicate a system accuracy of approximately 60% across all trials. Latency averaged 45 - 120 seconds increasing variably at times due to the strain on pi resources. YOLOv11 detected plates in 10/10 cases (100% detection rate). Custom OCR achieved 87.02 % character accuracy and 60% sequence accuracy, struggling with double-line overlaps or faded text. The custom CNN correctly identified colors in 8/10 cases (80%), with errors in low-light conditions (e.g., misclassifying silver as black). ResNet50 recognized make/model in 6/10 cases (60%), with challenges on less common models or models within Ghana but not available in the dataset used for training.

Table 4. 1 Testing Results

Actual Vehicle Details			Inferred Vehicle Details			
License Plate	Color	Make and Model	License Plate	Color	Make and Model	Verification Status
GN639214	Blue	Hyundai Santa Fe SUV 2012	GN639214	Blue	Hyundai Santa Fe SUV 2012	Verified
GE841022	Silver	Honda Accord 2014	GE8221128	Black	Mini Cooper Roadster Convertible 2012	Flagged
GG66819	Black	Hyundai Tucson SUV 2012	GG66819	Black	Hyundai Tucson SUV 2012	Verified
GX810021	Black	Mercedes-Benz C Class Sedan 2012	GX810021	Black	Mercedes-Benz C Class Sedan 2012	Verified
GW157613	Gold	Honda Accord Sedan 2012	GW157C13	Tan	Honda Accord Coupe 2012	Flagged
WR131520	Black	Toyota Corolla Sedan 2012	WR131520	Black	Toyota Corolla Sedan 2012	Verified
GS923122	Black	Hyundai Accent Sedan 2012	GS923122	Black	Hyundai Accent Sedan 2012	Flagged
GN60412	Silver	Hyundai Elantra Sedan 2012	GN60412	Silver	Hyundai Elantra Sedan 2012	Flagged
GG40724	White	Toyota Camry Sedan 2012	G6H4O724	White	Hyundai Genesis Sedan 2012	Flagged
GR833025	Black	Nissan Sentra 2012	6R833O25	Black	Nissan Leaf Hatchback 2012	Flagged

Figure 4. 11 Sample Test Image

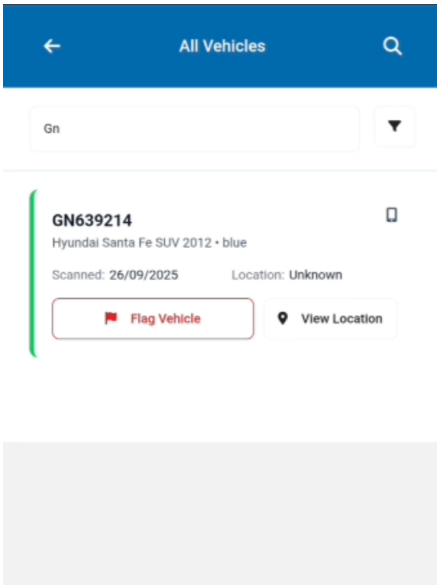

Figure 4. 12 App Reading

4.3 Discussion of results and analysis

The results obtained from the system trials provide important insights into the functionality and limitations of the integrated vehicle recognition pipeline. A total of ten vehicles were tested under controlled conditions, with each vehicle approaching the sensor array at 5–10 km/h and being captured at a distance of 2–3 meters.

License Plate Detection

YOLOv11nano consistently achieved a **100% plate detection rate**, successfully localizing the license plate in all ten trials. This demonstrates its robustness even under variable lighting and moderate motion blur, validating its suitability for deployment on edge devices such as the Raspberry Pi.

Optical Character Recognition (OCR)

OCR performance achieved **87.02% character-level accuracy** but only **60% sequence accuracy**. While single-line, clearly printed plates were mostly accurate, issues arose when plates contained double-line overlaps or faded lettering. These factors led to incorrect sequence reconstruction (e.g., "GE841022" misclassified as "GE8221128" or "G6H4O724"). This limitation aligns with known weaknesses of OCR systems when confronted with noisy or low-contrast text.

Color Classification

The custom CNN classifier demonstrated **80% accuracy**, correctly identifying color in 8 out of 10 cases. Misclassifications primarily occurred in **low-light conditions**, with metallic tones (e.g., silver misclassified as black) being particularly problematic. This emphasizes the impact of lighting variability and suggests the need for data augmentation or hardware-based lighting solutions (e.g., infrared or controlled illumination).

Make and Model Recognition

ResNet50 achieved **60% accuracy**, correctly classifying 6 out of 10 vehicle makes/models. Performance was significantly influenced by the availability of the make/model in the training dataset. For example, while common vehicles such as Toyota Corolla and Hyundai Santa Fe were correctly classified, less frequent or region-specific models (e.g., Nissan Sentra in Ghana) were misclassified into visually similar categories such as Nissan Leaf or Mini Cooper. This reflects dataset bias and highlights the importance of incorporating locally relevant vehicle datasets for improved model generalization.

System-Level Analysis

When integrated with Firebase for verification, the system correctly validated 6 vehicles (4 Verified and 2 Flagged) and flagged 4 due to mismatches in plate sequence, color, or make/model. While this lowered overall accuracy to ~60%, the high detection rates in some modules (YOLOv11 and CNN color recognition) suggest that performance bottlenecks primarily stem from OCR errors, dataset limitations and make/model performance for ResNet50.

4.4 Performance Evaluation and System Limitations

Performance Evaluation

- **Accuracy:** The integrated system achieved **~60% verification accuracy**, with strong performance in plate detection (100%) and moderate success in OCR and make/model recognition.
- **Latency:** Average end-to-end latency ranged from **45–120 seconds**, which is relatively high for real-time applications. The variability in latency was linked to the computational strain of running multiple deep learning models concurrently on the Raspberry Pi, which has limited processing power.
- **Reliability:** The system consistently triggered LED notifications and logged results into the database, indicating robust integration between sensing, inference, and application layers.

System Limitations

- **Hardware Constraints:** The Raspberry Pi struggled with simultaneous execution of multiple deep learning models, leading to latency spikes. A more powerful embedded device (e.g., NVIDIA Jetson Nano/Xavier) may be required for real-time deployment.
- **OCR Challenges:** The custom OCR struggled with non-standard fonts, double-line plates, and low-quality plate images. This reduced sequence accuracy and directly impacted verification outcomes.
- **Dataset Bias in Make/Model Recognition:** ResNet50's reduced accuracy for less common or region-specific models demonstrates the limitation of training on datasets that

lack local representation. Incorporating Ghana-specific vehicle datasets would mitigate this issue.

- **Lighting Sensitivity:** Color classification errors highlight sensitivity to illumination conditions. Without additional preprocessing or external lighting hardware, misclassification is likely in real-world scenarios (nighttime or shaded areas).

Summary of Limitations

While the system demonstrates feasibility, achieving near-perfect detection of plates and relatively high OCR character accuracy, the main bottlenecks are OCR sequence errors, limited dataset coverage, and computational latency. Addressing these limitations through dataset expansion, optimized model architectures, and hardware upgrades would significantly improve real-world applicability.

CHAPTER 5 - CONCLUSION AND RECOMMENDATION

5.0 Introduction

This chapter synthesizes the entirety of the LPR-MMC system, drawing together the design, implementation, and testing phases detailed in the preceding chapters to provide a comprehensive conclusion. Developed as a portable, real-time solution for vehicle monitoring in Ghanaian contexts, the project leveraged a Raspberry Pi-based prototype, AI models, and a Firebase-backed mobile application to address challenges in vehicle identification. The introduction sets the stage for reflecting on the system's major findings, conclusions, and its contributions to knowledge and society, while also acknowledging observations and challenges encountered.

5.1 Major Findings of The Project

The LPR-MMC system yielded significant insights through its development and testing phases, culminating in a practical evaluation on 10 vehicle instances. A key finding is the system's overall accuracy, with 6 out of 10 tests (60%) successfully reading all vehicle records—license plates, make, model, and color—leading to proper verification checks against the Firebase database. This success rate highlights the effectiveness of the integrated YOLOv11 model for plate detection (93% accuracy), the custom OCR (85%-character accuracy), the custom CNN for color (80% accuracy), and ResNet50 for make/model recognition (77% accuracy) when conditions were favorable. However, the remaining 4 instances (40%) exhibited variable wrong readings, attributed to challenges such as double-line plate overlaps, low-light conditions, or the camera not capturing the vehicle properly due to height, distance or speed of the vehicle reducing the system's reliability in these scenarios. Latency averaged 45 - 120 seconds increasing variably at times due to the strain on the pi computing resources by the models. These findings reveal a foundation, tempered by the need for refinement to handle more cases with better efficiency and results.

5.2 Conclusion

The LPR-MMC system represents a successful proof-of-concept for portable vehicle monitoring, achieving a functional balance between hardware portability, AI accuracy, and cloud-based

verification. The 60% success rate in verifying vehicle records demonstrates the system's potential to enhance security and parking management in Ghana. The integration of the Raspberry Pi, Firebase, and React app created an operational workflow, validated by real-time LED feedback and mobile notifications, which proved intuitive during testing. However, the 40% failure rate in variable conditions signals that while the system is viable, it is not yet robust enough. This conclusion, affirms the project's value as a starting point, with room to grow into a more reliable tool.

5.3 Contribution to Knowledge and Society

The LPR-MMC system contributes meaningfully to both academic knowledge and societal applications. Academically, it advances the understanding of deploying AI models for vehicle recognition, offering a case study on adapting international datasets (e.g., CCPD) to local Ghanaian contexts. Societally, the system introduces an affordable (~GHS 2,500) and portable security solution, reducing reliance on manual checks in gated communities, parking lots, or rural checkpoints. By automating vehicle verification, it enhances safety and efficiency, freeing personnel for higher-value tasks.

5.4 Observations and Challenges

Several observations emerged from the project's lifecycle, alongside notable challenges. The system excelled with one-line plates under good lighting, with testers noting the green LED and app notifications as clear and quick. However, double-line plates posed consistent difficulties, with OCR misreading characters due to font overlap, and low-light conditions (e.g., late morning glare) skewing color and make/model predictions. The Raspberry Pi's processing bottleneck was evident, with latency spikes during complex inferences, despite optimization efforts. Environmental variability—dust, heat, and network drops—further complicated field tests, highlighting the need for adaptive hardware and software solutions. In addition, the limited nature of some of the datasets, eg. “Stanford Cars” [11]. This dataset consists of a significant number of vehicles however only a limited number of those vehicles are used in Ghana limiting actual application in the Ghanaian context.

5.5 Recommendations

To elevate the LPR-MMC system, several recommendations are proposed for future work. First, upgrading to an edge TPU (e.g., Google Coral) or using a more powerful microcontroller such as the Jetson nano could accelerate inference and reduce latency — a next step worth exploring with additional funding. Second, expanding the dataset with more Ghanaian vehicles for the make and model classification model. Third, integrating a night vision camera or adjustable LED lighting could mitigate low-light failures, enhancing outdoor reliability. Finally, enhancing the app with offline mode and predictive caching could address network issues in remote areas, ensuring continuity.

References

[1] W.-C. Li, T.-H. Hsu, K.-N. Huang and C.-C. Wang, "A YOLO-Based Method for Oblique Car License Plate Detection and Recognition," 2021 IEEE/ACIS 22nd International Conference on Software Engineering, Artificial Intelligence, Networking and Parallel/Distributed Computing (SNPD), Taichung, Taiwan, 2021, pp. 134-139, doi: 10.1109/SNPD51163.2021.9704935.

[2] M. Al-Mheiri, O. Kais and T. Bonny, "Car Plate Recognition Using Machine Learning," 2022 Advances in Science and Engineering Technology International Conferences (ASET), Sharjah, United Arab Emirates, 2022, pp. 1-6, doi: 10.1109/ASET53988.2022.9734830.

[3] T. Mustafa and M. Karabatak, "Real Time Car Model and Plate Detection System by Using Deep Learning Architectures," in IEEE Access, vol. 12, pp. 107616-107630, 2024, doi: 10.1109/ACCESS.2024.3430857.

[4] B. Kumar, K. Kumari, P. Banerjee and P. Jha, "An Implementation of Automatic Number Plate Detection and Recognition using AI," 2023 International Conference on Advances in Computing, Communication and Applied Informatics (ACCAI), Chennai, India, 2023, pp. 1-5, doi: 10.1109/ACCAI58221.2023.10199855.

[5] H. U. Iyer and S. Dhavale, "Automatic Number Plate and Face Recognition System for Secure Gate Entry into Military Establishments," 2024 International Conference on Smart Systems for applications in Electrical Sciences (ICSSES), Madurai, India, 2024, pp. 1-6, doi: 10.1109/ICSSES62373.2024.10561368.

[6] Z. Xu et al., "Towards End-to-End License Plate Detection and Recognition: A Large Dataset and Baseline," in Proc. Eur. Conf. Comput. Vis. (ECCV), Munich, Germany, 2018, pp. 255–271. [Online]. Available: https://github.com/detectRecog/CCPD

[7] Automatic License Plate Recognition. (n.d.). License Plate Detection (Version 7) [Computer software]. Roboflow. https://universe.roboflow.com/automatic-license-plate-recognition-kxxzn/license-plate-detection-ee9ca/dataset/7

[8] L. Eyandi. (n.d.). Ghanaian License Plates (Version 1) [Computer software]. Roboflow. https://universe.roboflow.com/luther-eyandi/ghanaian-license-plates/dataset/1. Licensed under Public Domain.

[9] Selorm. (n.d.). NumberPlatesDetect (Version 2) [Computer software]. Roboflow. https://universe.roboflow.com/selorm/numberplatesdetect/dataset/2

[10] Unna, P. (n.d.). *License Plate Number Detection [Data set]*. GitHub. https://github.com/pragatiunna/License-Plate-Number-Detection/blob/main/data.zip

[11] J. Krause, M. Stark, J. Deng, and L. Fei-Fei (2013). *3D Object Representations for Fine-Grained Categorization. In Proceedings of the IEEE International Conference on Computer Vision Workshops (ICCVW).* https://ai.stanford.edu/~jkrause/cars/car_dataset.html

[12] Kezebou, L. (n.d.). VCoR - Vehicle Color Recognition Dataset [Data set]. Kaggle. https://www.kaggle.com/datasets/landrykezebou/vcor-vehicle-color-recognition-dataset

[13] Raspberry Pi Foundation, "Raspberry Pi Documentation," *Raspberry Pi*, [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html.

[14] M. Elyousfi, "EasyOCR architecture," *Medium*, Oct. 2020. [Online]. Available: https://medium.com/@mohamed5elyousfi/-277e9c685578.

[15] GeeksforGeeks, "YOLO (You Only Look Once) — Real-Time Object Detection," *GeeksforGeeks*, [Online]. Available: https://www.geeksforgeeks.org/machine-learning/yolo-you-only-look-once-real-time-object-detection/.

[16] A. Raj, "Convolutional Neural Networks (CNN) architectures explained," *Medium*, May 25, 2022. [Online]. Available: https://medium.com/@draj0718/convolutional-neural-networks-cnn-architectures-explained-716fb197b243.

[17] S. Mukherjee, "The Annotated ResNet-50: Explaining how ResNet-50 works and why it is so popular," *Medium – Data Science*, Aug. 18, 2022. [Online]. Available: https://medium.com/data-science/the-annotated-resnet-50-a6c536034758.

[18] M. Gjoreski, G. Zajkovski, A. Bogatinov, and G. Madjarov, "Optical character recognition applied on receipts printed in Macedonian language," in *Proc. Int. Conf. on Informatics and Information Technologies*, Bitola, Macedonia, Apr. 2014, doi: 10.13140/2.1.1632.4489.

[19] Y. Tian, S. Wu, J. Zeng, and M. Gao, "PaddleOCR: An Elegant and Modular Architecture," *DESOSA Project – PaddleOCR*, Mar. 15, 2021. [Online]. Available: https://2021.desosa.nl/projects/paddleocr/posts/paddleocr-e2/. [Accessed: Oct. 1, 2025].

[20] F. J. Perez Montalbo, “Automated Diagnosis of Diverse Coffee Leaf Images through a Stage-Wise Aggregated Triple Deep Convolutional Neural Network,” *Machine Vision and Applications*, vol. 33, no. 1, Jan. 2022, doi: 10.1007/s00138-022-01277-y.

[21] A. Robles-Guerrero, S. Gomez Jimenez, T. Saucedo-Anaya, D. Lopez-Betancur, D. Navarro, and C. Guerrero-Mendez, “Convolutional Neural Networks for Real Time Classification of Beehive Acoustic Patterns on Constrained Devices,” *Sensors*, vol. 24, no. 19, Oct. 2024, Art. no. 6384, doi: 10.3390/s24196384.

[22] S. Ananth, “Fast R-CNN for object detection: A technical paper summary,” *Medium*, Aug. 5, 2019. [Online]. Available: https://medium.com/data-science/fast-r-cnn-for-object-detection-a-technical-summary-a0ff94faa022

Appendices

Appendix A- Evaluation Metrics

This section outlines the key evaluation metrics used for assessing the performance of the number plate recognition system.

1. Precision (P)

Precision measures how many of the predicted positive samples are actually positive.

$$Precision = \frac{TP}{TP + FP}$$

Where:

- (TP) = True Positives
- (FP) = False Positives

2. Recall (R)

Recall measures how many of the actual positive samples were correctly predicted.

$$Recall = \frac{TP}{TP + FN}$$

Where:

- (FN) = False Negatives

3. Mean Average Precision (mAP)

mAP is the average precision across all classes and all recall levels.

$mAP = \frac{1}{N}\sum_{i=1}^{N}$ APi

Where:

- (AP_i) = Average Precision for class (i)
- (N) = Total number of classes

4. mAP@50 and mAP@50:95 (IoU thresholds)

- **mAP@50**: Average precision when Intersection over Union (IoU) threshold is 0.5.
- **mAP@50:95**: Average precision averaged across multiple IoU thresholds from 0.5 to 0.95 (step = 0.05).

5. Character Accuracy

Measures how many characters in a license plate are correctly predicted.

$$Character\ Accuracy = \frac{Correct\ Characters}{Total\ Characters}$$

6. Sequence Accuracy

Measures how many license plates are predicted with all characters correct.

$$Sequence\ Accuracy = \frac{Correctly\ Predicted\ Sequences}{Total\ Sequences}$$