Contents

List Of Figures

ABSTRACT

In today's data-driven world, traditional billboard advertising methods struggle to capture audience attention effectively, often displaying static content without considering real-time environmental factors. This project, titled "Revolutionizing Ads with Mobile Vehicles Campaigns", introduces a Smart Billboard System that leverages real-time video processing, computer vision, and machine learning to display context-aware advertisements based on traffic density. The system captures live video using a camera mounted on a moving vehicle or roadside unit. Using the YOLOv8 object detection model, it detects and counts vehicles such as cars, buses, trucks, and motorcycles. The vehicle count is then passed to a machine learning classifier, which categorizes the traffic as low, medium, or high density. Depending on this classification, the system selects and displays the most relevant advertisement from a pool of ads, each associated with a bid value. The highest-bidding ad for the current traffic level is displayed to maximize engagement and revenue. Built using Python and the Flask web framework, the system includes a user-friendly web interface that streams both the live camera feed and the selected advertisement in real time. This intelligent, responsive approach to outdoor advertising demonstrates how artificial intelligence can transform traditional marketing methods into dynamic, adaptive, and more profitable campaigns.

KEYWORDS: Smart Billboard, YOLOv8, object detection, traffic analysis, machine learning, real-time video processing, dynamic advertisement, computer vision, Flask web application, intelligent ad selection, digital marketing, context-aware system

CHAPTER 1

INTRODUCTION

In the fast-paced and highly competitive world of marketing, the effectiveness of traditional advertising methods is diminishing. Static billboards and posters, which have served as a primary medium for decades, lack the ability to adapt to changing environmental and contextual conditions. As a result, advertisements often reach audiences at suboptimal times, reducing both their impact and return on investment. With the rise of smart cities and increasing urbanization, there is a growing demand for advertising systems that are not only dynamic but also intelligent—capable of responding in real time to the surrounding environment.

Recent advancements in computer vision, deep learning, and machine learning have opened the door to new and innovative applications in the advertising industry. One such opportunity lies in the integration of real-time object detection with traffic analysis to create a context-aware advertising platform. This concept aligns with the broader trend of digital transformation, where automation and intelligence are used to optimize decision-making and enhance user engagement.

The project titled “Revolutionizing Ads with Mobile Vehicles Campaigns” aims to bridge the gap between static billboard systems and dynamic, data-driven advertising by introducing a Smart Billboard System. This system uses a camera—mounted either on a mobile vehicle or a fixed roadside structure—to capture live video feeds of nearby traffic. These video frames are analysed using YOLOv8, a state-of-the-art object detection model, to detect and count different types of vehicles such as cars, buses, motorcycles, and trucks.

Once vehicle detection is performed, the total count is fed into a machine learning classifier that categorizes the traffic density as low, medium, or high. The system then automatically selects the most appropriate advertisement from a pool of predefined ads, categorized by both traffic level and associated bid value. The advertisement with the highest bid in the matching category is chosen to maximize ad revenue and relevance.

This entire process is streamlined through a web interface built using the Flask web framework. The interface provides users with a real-time view of both the camera feed and the selected advertisement, along with basic controls to start or stop the system. The smart billboard system not only improves operational efficiency but also ensures that ads are targeted, timely, and contextually appropriate.

The proposed solution is especially effective for mobile billboard campaigns, such as those mounted on delivery trucks or public transport vehicles, where the traffic environment is constantly changing. It is also suitable for deployment in smart city infrastructure, where real-time responsiveness and automation are crucial for scalability.

This project demonstrates the practical application of artificial intelligence in the field of digital marketing, showcasing how machine learning and computer vision can work together to modernize advertising strategies. It also illustrates the potential for AI-driven systems to improve decision-making, reduce human intervention, and enhance user engagement in outdoor advertising.

1.1 Problem Statement

Despite the rise of digital technologies, many billboard advertising systems still rely on static or pre-scheduled content, lacking the ability to adapt to their environment. These conventional methods do not account for real-time factors such as traffic flow or audience density, leading to inefficient ad placements and reduced effectiveness in capturing viewer attention.

For example, a high-value advertisement may be displayed during times of low vehicle traffic, limiting its reach and reducing its return on investment. Conversely, during peak traffic hours, less relevant or lower-bid ads might be shown, missing the opportunity to engage a larger audience and generate higher revenue. Furthermore, the manual process of updating and managing billboard content is time-consuming, labour-intensive, and prone to delays, making it difficult to respond promptly to changing conditions.

Although some digital billboards have incorporated basic automation or sensor-based triggers, these systems often lack real-time traffic analysis and intelligent ad selection mechanisms. Most do not integrate advanced computer vision models or machine learning classifiers capable of making context-aware decisions.

There is a clear need for an automated, intelligent, and real-time advertising system that can monitor traffic conditions, classify traffic density, and dynamically select the most suitable advertisement based on viewer volume and ad bid value. Such a system should also be flexible enough to operate on both fixed and mobile billboard platforms, offering a scalable solution for modern, data-driven outdoor advertising.

1.2 Objectives

1. To develop a real-time vehicle detection system using the YOLOv8 object detection model to identify and count vehicles from a live camera feed.
2. To classify traffic density into three categories—low, medium, and high—based on the number of detected vehicles using a machine learning classifier.
3. To design an intelligent advertisement selection mechanism that chooses the most suitable ad from a predefined set, based on both traffic level and bid value.
4. To implement a web-based interface using the Flask framework that allows users to:
    a. View the live video stream.
    b. See the currently displayed advertisement.
    c. Start or stop the system in real time.
5. To ensure ad targeting is optimized by displaying high-bid advertisements during heavy traffic periods and more general ads during low-traffic conditions.
6. To support both mobile and stationary billboard platforms, making the system adaptable for a wide range of real-world deployment scenarios.
7. To showcase the practical application of artificial intelligence, specifically computer vision and machine learning, in enhancing the efficiency and profitability of outdoor advertising.

CHAPTER 2

LITERATURE SURVEY

The integration of computer vision and machine learning techniques has significantly advanced vehicle detection and counting, which is crucial for traffic analysis and intelligent advertising systems.

Ribeiro and Hirata (2023) introduced an efficient vehicle counting method by combining the YOLO object detection framework with Visual Rhythm, a technique that highlights key video frames and eliminates the need for complex tracking algorithms. Their approach achieved a high mean counting accuracy of approximately 99.15% and operated at speeds three times faster than traditional tracking-based methods, demonstrating the potential for real-time applications in traffic monitoring.

Morshed et al. (2023) proposed a road vehicle counting model based on the latest YOLOv8 architecture. Their system enhanced accuracy by incorporating Kalman filtering for object tracking and irrelevant region masking to reduce false positives. The model showed stable performance with high precision, recall, and F1 scores, effectively estimating road congestion levels, which is vital for adaptive advertising systems responding to traffic density.

Liu et al. (2023) developed an adaptive vehicle counting method tailored for drone imagery. By utilizing spatial attention mechanisms and multiscale receptive fields, their model improved detection accuracy in complex aerial views, making it suitable for large-scale traffic surveillance, a feature that could be adapted for mobile billboard monitoring from different vantage points.

In the domain of outdoor advertising, Cerenkov et al. (2023) explored the significance of roadside billboards from the driver's perspective using a combination of eye-tracking data and YOLOv8-based object detection. Their pipeline achieved 97.5% accuracy in classifying billboard relevance, underscoring the importance of context-aware advertisement delivery, which aligns with the objectives of smart billboard systems.

These studies highlight the effectiveness of combining advanced detection techniques with contextual analysis to enhance both traffic monitoring and targeted advertising. Building upon these foundations, the current project integrates YOLOv8-based vehicle detection with machine learning-driven traffic classification to dynamically optimize advertisement display based on real-time traffic density. This approach not only improves ad relevance and engagement but also demonstrates the practical fusion of AI and computer vision in the evolving field of digital marketing and smart city applications.

2.1 Existing System

Traditional billboard advertising systems generally display static content that does not change based on real-time conditions. Advertisements are manually scheduled and remain fixed regardless of the current traffic density or audience presence. This lack of adaptability often leads to inefficient use of advertising space, as high-value ads may be displayed during low-traffic periods, reducing their impact and return on investment.

Recent advancements have introduced digital billboards that can display dynamic content. However, many of these systems rely on pre-programmed schedules or simple sensors that detect basic motion or light changes without detailed analysis of traffic or viewer behaviour. This limits their ability to tailor advertisements based on the actual audience or environmental context.

Some modern systems have started incorporating computer vision techniques for vehicle detection and counting, primarily using deep learning models like YOLO. These systems analyse real-time video feeds to estimate traffic density and adjust advertisements accordingly. However, many existing solutions still face challenges such as high computational costs, latency issues, and limited integration with machine learning models for intelligent decision-making.

Overall, while progress has been made towards more responsive advertising, there remains a need for fully automated, intelligent billboard systems that dynamically select advertisements based on accurate, real-time traffic analysis, maximizing both engagement and revenue.

Limitations of Existing Systems

1. Static Content Display: Traditional billboards and many digital advertising systems show fixed content that does not adapt to real-time traffic or environmental changes, leading to ineffective audience targeting.
2. Manual Operation: Most existing systems require manual scheduling and management of advertisements, which is time-consuming, inefficient, and not scalable for dynamic contexts.
3. Limited Context Awareness: Many current digital billboards lack sophisticated analysis of traffic density or viewer behaviour, reducing the relevance and impact of displayed ads.
4. High Computational Demand: Some computer vision-based systems for vehicle detection and counting rely on complex tracking algorithms, resulting in slower processing speeds and higher hardware requirements.
5. Latency Issues: Real-time responsiveness can be compromised due to processing delays, impacting the timely selection and display of relevant advertisements.

6. Inadequate Integration: Existing systems often do not fully integrate machine learning models with detection modules, limiting their ability to make intelligent decisions for ad selection based on traffic conditions.
7. Limited Adaptability: Systems tailored to fixed installations struggle to accommodate mobile billboard platforms or varying viewpoints, restricting their deployment flexibility.

2.2 Proposed System

The proposed system aims to revolutionize outdoor and mobile advertising by integrating real-time traffic analysis with intelligent advertisement selection. This smart billboard system uses computer vision and machine learning to adapt ad content dynamically based on live traffic conditions, maximizing relevance and potential revenue.

A live video feed is captured using a camera, typically mounted on a vehicle or roadside. The frames are processed using the YOLOv8 object detection model to detect and count vehicles such as cars, buses, and trucks. Based on the number of vehicles detected, a pre-trained machine learning classifier determines the current traffic density level—low, medium, or high.

Each traffic level is associated with a pool of advertisements categorized by bid value. The system selects and displays the highest-bidding ad corresponding to the current traffic level. This ensures that more valuable ads are shown during high-traffic periods, optimizing exposure and profitability.

The entire system is implemented using Python and Flask, with live video and advertisement streams accessible through a web interface. Users can monitor the feed and control system operation in real time.

This solution provides a cost-effective, responsive, and intelligent alternative to traditional billboard systems, making it suitable for both fixed and mobile advertising platforms.

2.2.1 YOLOv8 – Object Detection Model

`YOLOv8 (You Only Look Once, version 8) is a state-of-the-art real-time object detection model developed by Ultralytics. It is designed to efficiently detect and classify multiple objects within images and video streams with high precision and low latency. The YOLO architecture has undergone several iterations, and version 8 brings significant improvements in terms of detection accuracy, model efficiency, and support for various deployment scenarios, including edge devices.

In this project, YOLOv8 is used as the core component for real-time vehicle detection. A live video stream is captured through a camera mounted on a mobile

vehicle or stationary roadside unit. Each frame from this stream is processed using the YOLOv8 model to identify and count different types of vehicles. The model supports detection of multiple vehicle classes such as cars, buses, trucks, and motorcycles, which are the primary indicators of road traffic volume.

One of the key reasons YOLOv8 was chosen for this application is its compact model architecture and real-time performance capability. The lightweight variant, yolov8n.pt (nano version), is used in the implementation to ensure high-speed inference even on systems without a dedicated GPU. Despite its small size, YOLOv8n maintains strong detection accuracy, making it ideal for embedded and web-based deployment.

Each frame passed to YOLOv8 is analysed to return:

- **Bounding boxes** around detected objects
- **Class labels** indicating the type of object (e.g., car, truck)
- **Confidence scores** showing the probability of correct classification

The object classes returned by the model are then filtered to include only relevant vehicle types (class IDs for car = 2, motorcycle = 3, bus = 5, and truck = 7). The total count of these detected vehicles is used as the input for traffic classification.

Key Advantages of Using YOLOv8:

- **Real-Time Processing:** Enables frame-by-frame vehicle analysis at over 20–30 FPS on standard machines.
- **High Accuracy:** Delivers robust detection with minimal false positives, even in challenging environments.
- **Lightweight Deployment:** The nano version can run on CPUs, making it suitable for edge devices or mobile units.
- **Versatility:** Can detect over 80 object classes, with configurable options to limit detection to only vehicles.
- **Integrated API:** Easily integrates into Python-based workflows, including Flask and OpenCV applications.

By using YOLOv8, the system ensures timely and accurate detection of vehicle density, which is a critical factor for determining current traffic levels. This real-time data serves as the foundation for the machine learning module to classify traffic and select suitable advertisements. The combination of YOLOv8's speed and accuracy allow the entire smart billboard system to function responsively and adaptively in changing road conditions.

2.2.2 Machine Learning Classifier – Traffic Level Prediction

After extracting the number of vehicles from each video frame using the YOLOv8 object detection model, the next critical step in the system is to classify the current traffic density level. This classification enables the system to dynamically select and display an advertisement that is most relevant for the current traffic conditions. To achieve this, a machine learning classifier is employed.

The classifier is trained using a supervised learning approach, where historical vehicle count data is labelled with corresponding traffic density levels: Low, Medium, and High. The input feature for the model is a single numeric value representing the count of vehicles detected in a frame, while the output is a categorical label representing the traffic condition.

Traffic Density Classification Logic:

- **Low Traffic**: 0 to 5 vehicles
- **Medium Traffic**: 6 to 15 vehicles
- **High Traffic**: More than 15 vehicles

These thresholds are determined based on domain knowledge and traffic patterns observed during system testing. The classifier learns these patterns and applies them in real-time to new data coming from the detection pipeline.

Model Used and Implementation:

- A **Decision Tree Classifier** is used for this system. It was selected for its interpretability, low inference time, and suitability for small, structured datasets.
- The model is developed and trained using **scikit-learn**, a popular machine learning library in Python.
- After training, the model is saved using joblib as a .pkl (Pickle) file to enable fast loading during runtime.
- At runtime, the model is loaded once at system startup and used repeatedly to classify traffic levels from incoming vehicle counts.

Runtime Workflow:

1. YOLOv8 detects and counts vehicles in the current frame.
2. The total count is passed as input to the trained classifier.
3. The classifier predicts the traffic level as low, medium, or high.
4. This traffic level is used to query a pool of pre-categorized advertisements.
5. The system then selects the **highest-bid advertisement** for that traffic level and displays it in real-time.

Why Use a Machine Learning Classifier?

- **Flexibility**: The classifier can be retrained with more data to adapt to changing traffic patterns or new locations.
- **Scalability**: Allows easy expansion to include more traffic levels or features in the future (e.g., time of day, weather).
- **Speed**: Lightweight models like Decision Trees offer millisecond-level inference times, ensuring the system remains responsive.
- **Autonomy**: Replaces manual thresholds with a data-driven method that can learn from patterns and make intelligent predictions.

This intelligent traffic classification system adds a layer of decision-making to the smart billboard, enabling **context-aware advertising**. By linking real-world conditions to ad content in real time, the system maximizes the impact and relevance of displayed ads while also increasing monetization potential based on ad bidding strategies.

CHAPTER 3

SYSTEM DESIGN

3.1 Importance of the Design

The system starts by collecting diverse human action video datasets from various online sources. These videos cover activities like walking, running, and jumping, ensuring the model can detect actions in different environments. The data undergoes preprocessing and augmentation to improve model performance.

The user interface enables easy interaction, allowing users to upload videos for human action detection. Once uploaded, videos are pre-processed by extracting frames, resizing, and normalizing pixel values. This standardization ensures the deep learning models can process the videos effectively, enabling efficient and accurate human action recognition.

The system employs a combination of 3D CNNs and LSTM networks for human action detection. The 3D CNN captures spatial and temporal patterns in videos, while the LSTM learns long-term dependencies across frames. This combined approach enhances the system's ability to accurately recognize actions, as each model addresses different aspects of video analysis, providing complementary strengths.

After training, the models' performance is evaluated using metrics like accuracy, precision, recall, and F1 score. These metrics help assess how well the models detect actions. If accuracy is low, adjustments like hyperparameter tuning or data expansion are made to improve performance iteratively.

Real-time processing is crucial for the system’s functionality, allowing users to receive quick feedback on detected actions. The system processes videos swiftly, making it suitable for applications like surveillance and sports analytics. Scalability ensures the system can handle multiple user requests, accommodating high demand in practical deployment scenarios.

The user interface is designed for ease of use, displaying detected actions alongside bounding boxes and confidence scores. This visual feedback enhances user understanding by making predictions transparent. Additionally, the system explains its detection process, building trust and enabling users to interpret results clearly.

The system is adaptable for future scalability, able to integrate new datasets and keep pace with emerging trends. As new action categories and detection techniques arise, the system can be updated and retrained, ensuring it remains effective in diverse real-world settings.

3.2 UML Diagrams

3.2.1 Class Diagram

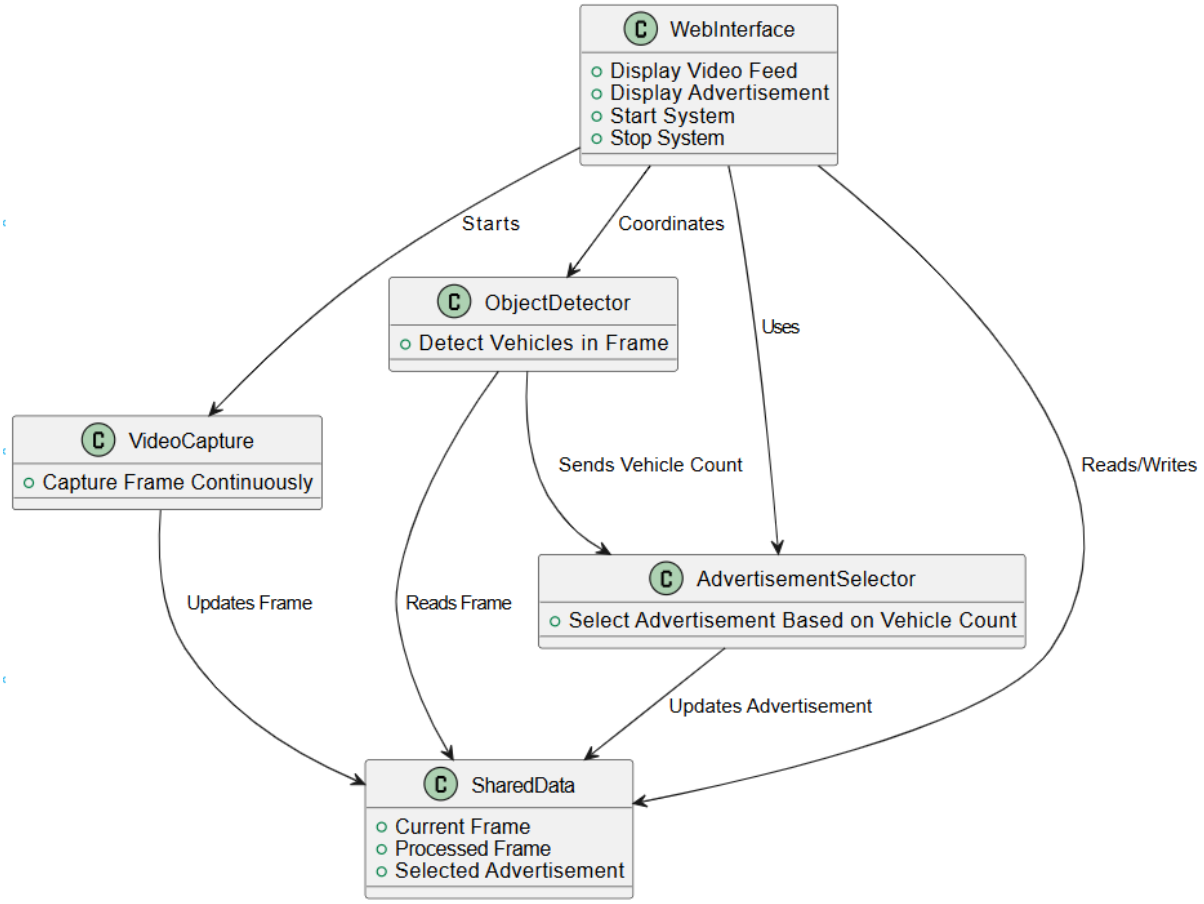

Figure 1: Class Diagram

The class diagram represents the object-oriented structure of the Smart Billboard system, outlining the major functional components and their interactions. It provides a visual and conceptual overview of how the system is organized and how each module contributes to the overall workflow, from video input to advertisement display.

The class diagram flow of the Smart Billboard System illustrates the interaction and data exchange among its core modules to achieve intelligent, real-time advertisement display. The process begins when the WebInterface initiates the system based on user input, such as starting or stopping the application. Once started, the VideoCapture class continuously captures frames from a live video source (e.g., a webcam mounted on a mobile vehicle or roadside). These captured frames are forwarded to a shared storage unit called SharedData, which acts as a centralized thread-safe buffer accessible by all system components.

The ObjectDetector then retrieves the latest video frame from SharedData and applies the YOLOv8 object detection algorithm to identify and count vehicles in the scene. This detection includes various vehicle types such as cars, trucks, buses, and motorcycles. Once the number of detected vehicles is determined, this information is sent to the AdvertisementSelector.

The AdvertisementSelector uses a pre-trained machine learning classifier to classify the traffic level into three categories: low, medium, or high. Based on the current traffic density, it selects the highest-bidding advertisement from a predefined list corresponding to the current traffic level. The chosen advertisement image is then updated in SharedData.

Finally, the WebInterface fetches the processed frame and the selected advertisement from SharedData and renders both in real time on the web dashboard for monitoring. This continuous loop ensures that the advertisements shown are always aligned with live traffic conditions, enabling the system to adapt dynamically and maximize viewer engagement and advertiser revenue.

The flow of the class diagram demonstrates a clear modular structure, with each component performing a distinct role while sharing necessary data via a synchronized interface. This architecture supports real-time performance, extensibility, and maintainability.

3.2.2 Use case diagram

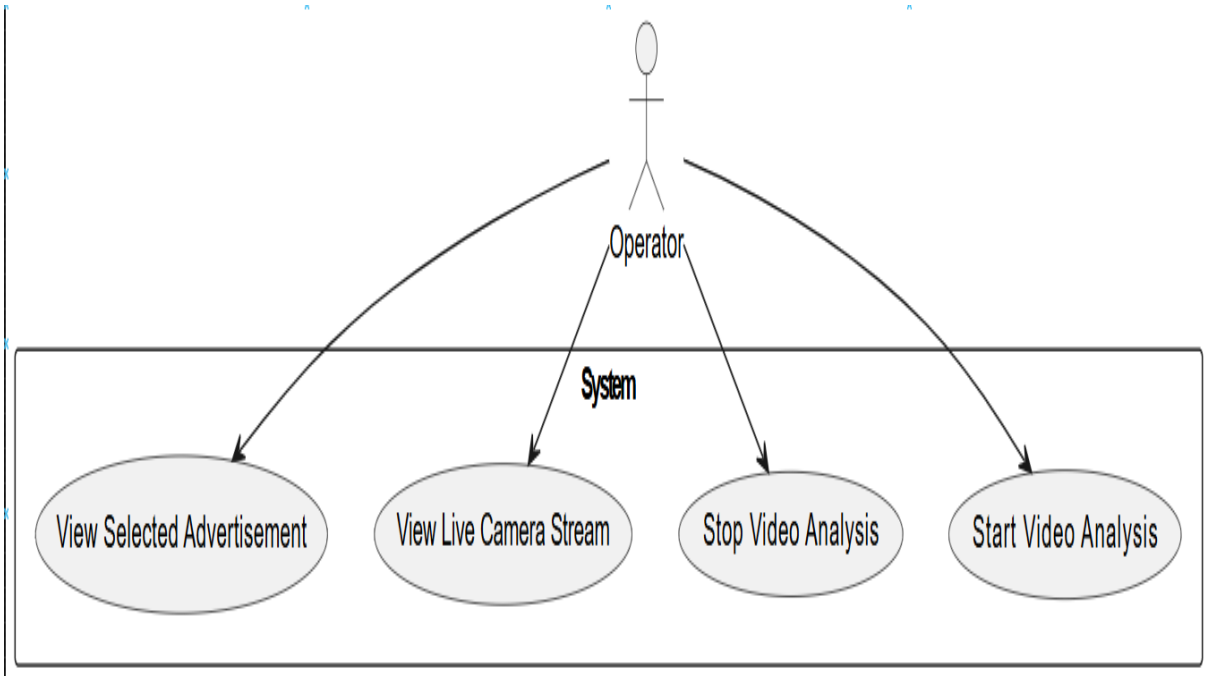

Figure 2: Use Case Diagram

A Use Case Diagram is a fundamental component of the Unified Modelling Language (UML) that illustrates how users (actors) interact with the system to perform specific operations. It helps define the functional scope of the application by showing the relationships between the user and the system's primary functionalities. In this project, the use case diagram provides a clear overview of the interaction between the Operator and the Smart Billboard System.

The system is primarily operated through a web interface and is controlled by a single actor, referred to as the Operator. The operator is responsible for initiating and managing the video processing workflow and monitoring its output. The operator interacts with the system through the following core use cases:

- Start Video Analysis: This use case represents the action where the operator starts the system. Internally, this triggers the loading of the YOLOv8 model, begins video capture from the connected webcam, and starts the processing pipeline for vehicle detection and traffic classification.
- Stop Video Analysis: Through this use case, the operator can halt the video stream processing. This ensures resource management and allows the system to safely shut down ongoing threads and processes.
- View Live Camera Stream: Once the system is running, the operator can view the live video feed from the connected camera. This allows real-time monitoring of the traffic being analysed.
- View Selected Advertisement: Simultaneously with the live video feed, the operator is able to see the advertisement currently being displayed, which is dynamically selected based on detected traffic density and bid values.

The use case diagram highlights a user-cantered design, where the operator has direct control over the main operational features. By abstracting complex backend processes like object detection, machine learning classification, and advertisement bidding into simple UI actions, the system ensures ease of use and accessibility without compromising on intelligence or automation.

This modular approach improves system usability and scalability, enabling future enhancement such as user-based authentication, multiple advertisement categories, or remote control interfaces. Overall, the use case diagram effectively models how the user interacts with the Smart Billboard System, ensuring all critical functions are accessible through a simplified interface.

3.2.3 Sequence Diagram

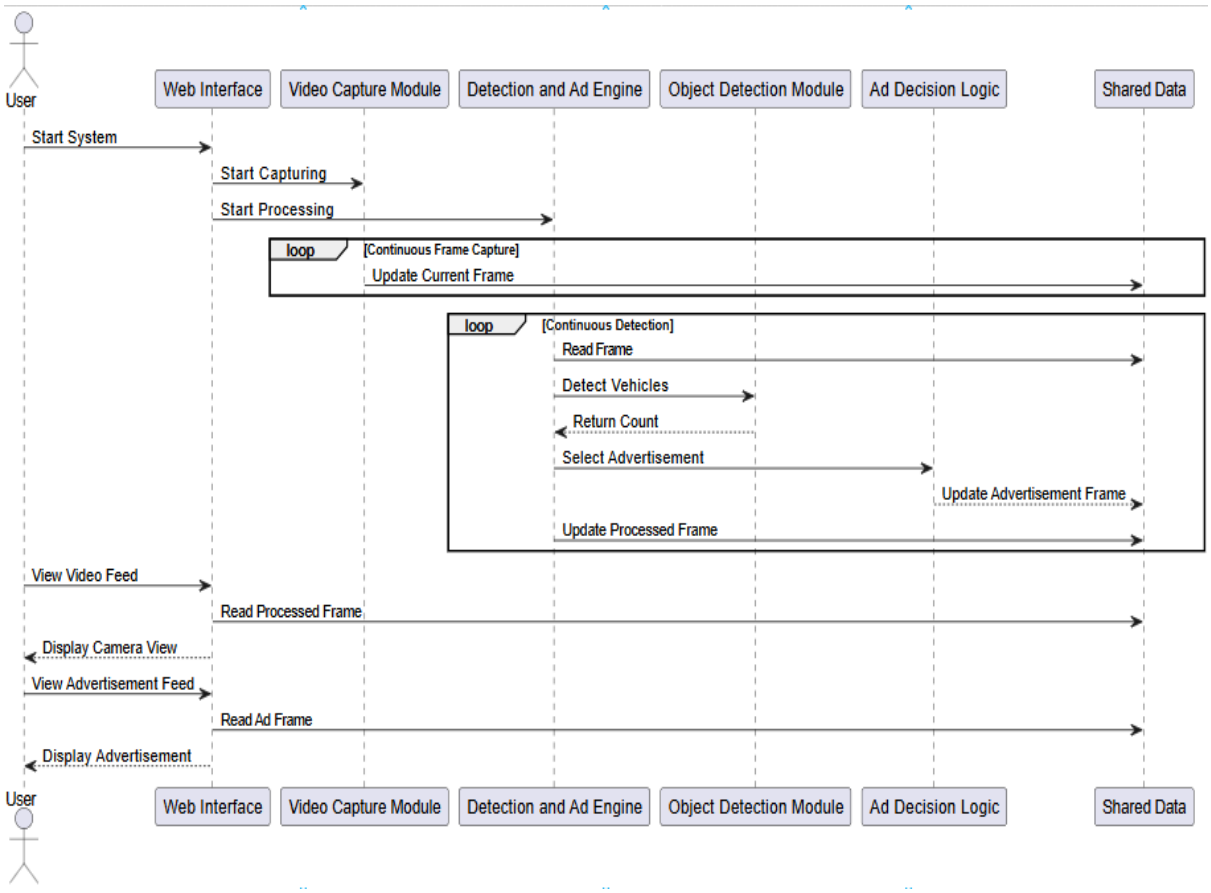

Figure 3: Sequence Diagram

The Sequence Diagram illustrates the dynamic behaviour of the Smart Billboard System by modelling the interaction between the system components over time. It provides a clear step-by-step flow of control and data exchange from the moment the user initiates the system to the final real-time display of advertisements based on live traffic analysis. The sequence diagram includes the following key actors and components: User, Web Interface, Video Capture Module, Detection and Ad Engine, Object Detection Module, and Ad Decision Logic.

The process begins when the User interacts with the Web Interface to initiate the system by clicking the “Start” button. This action triggers the Video Capture Module, which begins continuously capturing frames from the webcam or video source. These frames are periodically updated and stored in a shared memory space accessible by other system components.

Concurrently, the Web Interface initiates the Detection and Ad Engine, which begins reading the latest frame from shared memory. The engine then forwards the frame to the Object Detection Module, where the YOLOv8 model processes the image to detect and count vehicles

such as cars, buses, trucks, and motorcycles. The vehicle count is then returned to the Detection Engine.

The Detection Engine passes this vehicle count to the Ad Decision Logic, which employs a pre-trained machine learning classifier to categorize the traffic density into one of three classes: low, medium, or high. Based on this classification, the module selects the most appropriate advertisement from a pool of predefined ads, using the highest-bid strategy to optimize revenue. The selected advertisement frame is written back into the shared data store for rendering.

The User can then view both the real-time video feed and the advertisement feed through the Web Interface. The interface reads the most recent processed frame and the selected advertisement frame from shared memory and renders them side by side on the dashboard. This visualization loop continues seamlessly as long as the system remains active.

This sequence ensures a modular, efficient, and concurrent workflow, allowing the system to dynamically adapt to traffic conditions and intelligently display context-aware advertisements. By following this well-structured sequence, the system maintains high responsiveness and user engagement, while also ensuring technical scalability and real-time performance.

3.2.4 Activity Diagram

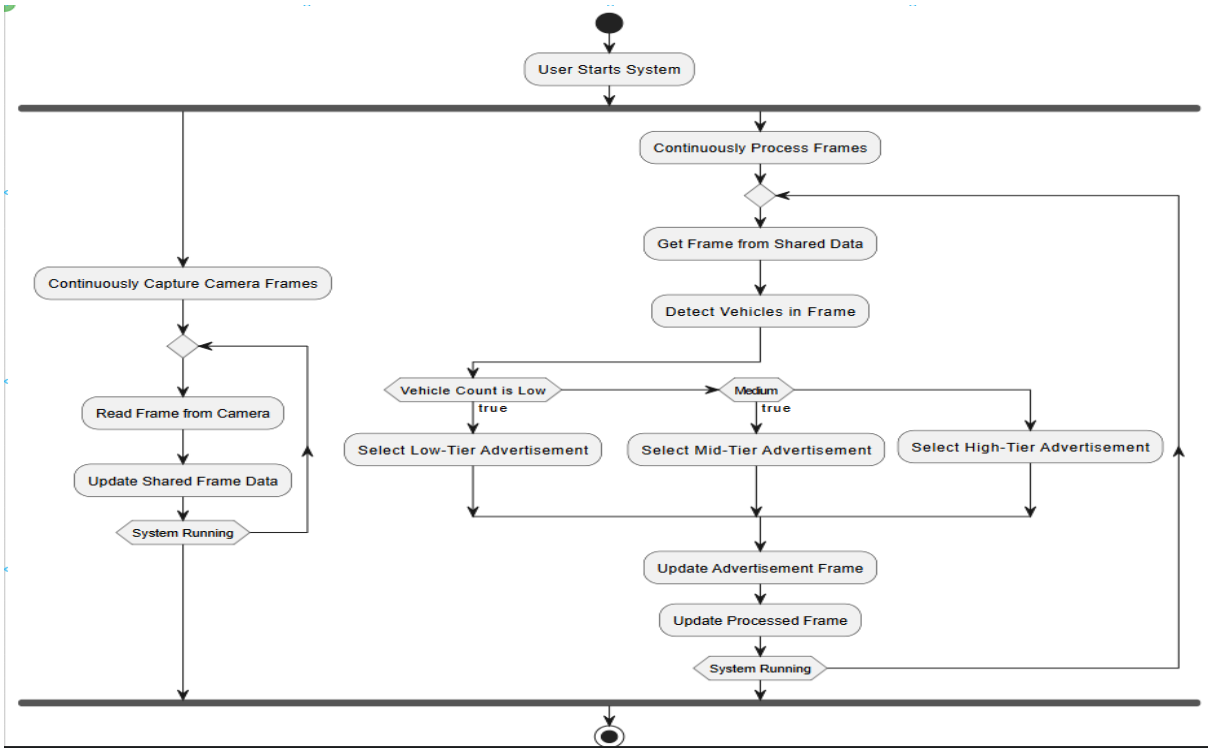

Figure 4: Activity Diagram

The Activity Diagram provides a high-level visualization of the workflow and internal logic of the Smart Billboard System. It represents the step-by-step sequence of operations, from system initialization to dynamic advertisement selection, and showcases how concurrent activities are managed during execution. The diagram is divided into two major concurrent operations—frame capture and intelligent frame processing—which run in parallel for efficient performance.

The activity begins when the User starts the system via the web interface. This action initiates two concurrent threads. The first thread, Camera Frame Capture, continuously reads frames from the live video feed using a webcam. Each captured frame is stored in a shared memory buffer, making it accessible to the processing unit.

Simultaneously, the second thread, Frame Processing, retrieves the latest video frame from shared memory and runs it through the YOLOv8 object detection model to identify and count the number of vehicles in the frame. Once vehicle detection is complete, the system evaluates the traffic density level based on the number of vehicles: if the count is low, a low-tier advertisement is selected; if the count is moderate, a mid-tier advertisement is displayed; and for a high vehicle count, a high-tier, premium advertisement is chosen.

The selected advertisement frame is then updated and stored, along with the processed video frame, in the shared memory. This ensures that the web interface can continuously display both the live traffic view and the corresponding context-aware advertisement side-by-side to the user in real time.

This activity flow continues in a loop until the system is manually stopped. The use of parallel operations allows the system to perform real-time detection and decision-making without introducing latency or lag, ensuring smooth performance. The diagram highlights the responsiveness, modularity, and intelligent decision logic embedded in the system's architecture.

3.2.5 System architecture

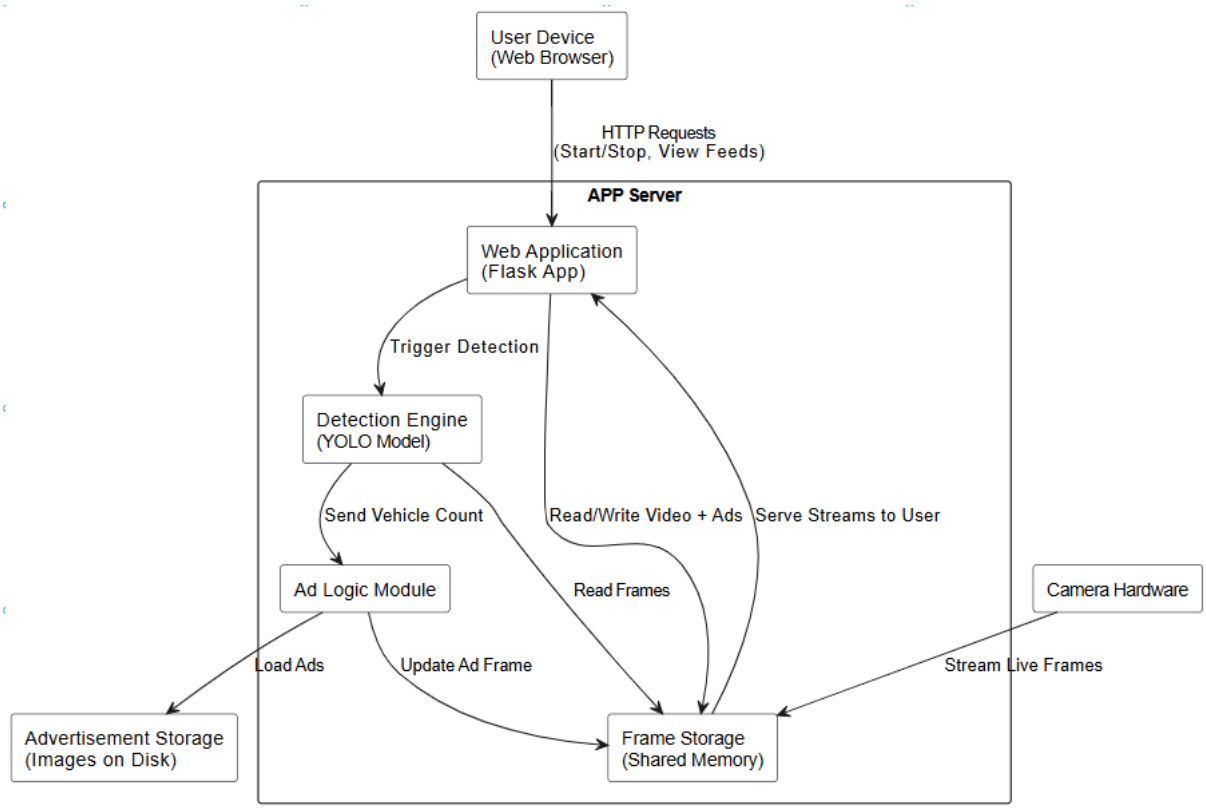

Figure 5: System Architecture

The System Architecture Diagram illustrates the high-level design and interaction between different hardware and software components in the Smart Billboard System. It highlights how various modules work together to achieve the goal of real-time vehicle detection and advertisement selection based on traffic density.

At the top of the architecture, the User Device, typically a web browser on a computer or mobile device, serves as the primary interface for interacting with the system. The user can initiate or stop the system and monitor live video and advertisement feeds through the web interface. These interactions are managed by the Web Application Layer, built using the Flask framework, which acts as the central controller for all system operations.

The Camera Hardware is responsible for continuously streaming live video frames. These frames are stored in a Shared Memory Module (Frame Storage) that acts as a communication buffer for real-time data exchange between the detection engine, advertisement logic, and the web interface. This shared memory ensures that all components have quick and synchronized access to the latest frame data.

The Detection Engine leverages the YOLOv8 model to process frames retrieved from shared memory. It detects and counts vehicles in each frame, identifying types such as cars, buses, and trucks. Once the vehicle count is determined, this data is sent to the Ad Logic Module, which contains a pre-trained machine learning model. The model classifies the traffic level as low, medium, or high based on vehicle density.

The Ad Logic Module then accesses the Advertisement Storage, which holds categorized advertisement images based on bid value and relevance. It selects the highest-bidding ad for the identified traffic level and updates the shared memory with the selected ad frame. Both the live video stream and the selected advertisement are then rendered on the web interface in real time for the user to view.

This modular and decoupled architecture ensures smooth performance, scalability, and easy maintenance. By separating concerns across detection, decision-making, and user interaction, the system achieves high efficiency and adaptability, making it suitable for deployment in both roadside and mobile vehicle environments.

CHAPTER 4

IMPLEMENTATION

4.1 Steps Description

The implementation of the Smart Billboard Advertisement System involves integrating multiple technologies, including real-time object detection, machine learning, and web development. The system is built in Python using the Flask framework for the backend and web interface. For real-time object detection, the YOLOv8 (You Only Look Once) model developed by Ultralytics is used, which is well-suited for detecting vehicles such as cars, buses, trucks, and motorcycles in live video feeds. Additionally, OpenCV is used to capture and process video frames from a live camera source. The decision-making logic for advertisement display is handled by a custom machine learning model trained to classify traffic levels based on vehicle count.

The environment setup includes Python 3.10+, Flask for handling HTTP requests and routing, OpenCV for image and video processing, and joblib for loading the pre-trained traffic classification model. The model (ad_selector_model.pkl) was trained using scikit-learn on a small dataset consisting of different vehicle count scenarios mapped to traffic levels: low, medium, and high. Each count corresponds to a specific traffic density, and the model learns to categorize the scene based on the number of detected vehicles. These predictions are then used to decide which category of advertisement should be shown on the digital billboard.

The advertisement images are stored locally under the static/ads/ directory. Each ad image is tagged with metadata that includes its traffic level relevance and a bid value. These ads are categorized into low, medium, and high-traffic types. When a frame is captured and processed by the YOLO model, the number of vehicles is counted, passed to the machine learning model, and based on the predicted traffic level, the ad with the highest bid from the corresponding category is selected for display. This approach ensures that high-value ads are shown during peak traffic, thereby increasing the advertisement's effectiveness and monetization potential.

The backend consists of multithreaded components. One thread is responsible for continuously grabbing frames from the camera, while another handles the detection and advertisement selection logic. Shared memory is protected using locks to ensure thread safety when reading and writing frame data. The Flask server provides live video and advertisement feeds to the client browser using MJPEG streaming. The user can start and stop the system through the web interface, which interacts with the server through POST requests. This modular and efficient architecture enables seamless real-time performance and a responsive web-based control panel.

In summary, the implementation combines the strengths of YOLOv8's fast detection, a machine learning-based traffic classifier, and a bid-driven ad selection strategy. This results in

a fully automated, real-time system capable of making intelligent advertisement decisions based on live traffic conditions, with a user-friendly interface and robust performance.

4.2 Sample Code

4.2.1 Backend

```
from flask import Flask, render_template, Response, request
import cv2
import numpy as np
from threading import Thread, Lock
from ultralytics import YOLO
import time
import joblib

app = Flask(__name__)

# Load YOLOv8 model
model = YOLO("yolov8n.pt")
# Load ML model for traffic classification
ad_selector = joblib.load('ad_selector_model.pkl')

# Resize size for frames and ads
billboard_size = (640, 480)

# Ad metadata with pricing
ad_data = [
    {"name": "bmw", "path": "static/ads/bmw.jpg", "level": "low", "bid": 0.5},
    {"name": "star", "path": "static/ads/star.jpg", "level": "medium", "bid": 1.2},
    {"name": "diary", "path": "static/ads/diary.jpg", "level": "high", "bid": 1.8},
]
```

```
# Organize ads by traffic level
ads_by_level = {"low": [], "medium": [], "high": []}
for ad in ad_data:
    img = cv2.imread(ad["path"])
    if img is None:
        print(f"Warning: Could not read {ad['path']}")
        continue
    image = cv2.resize(img, billboard_size)
    ads_by_level[ad["level"]].append({
        "name": ad["name"],
        "image": image,
        "bid": ad["bid"]
    })

def select_ad_by_bid(traffic_level):
    candidates = ads_by_level.get(traffic_level, [])
    if not candidates:
        return np.zeros((480, 640, 3), dtype=np.uint8)
    return max(candidates, key=lambda ad: ad["bid"])["image"]

# Shared variables
frame_lock = Lock()
latest_frame = None
output_frame = np.zeros((480, 640, 3), dtype=np.uint8)
current_ad_frame = np.zeros((480, 640, 3), dtype=np.uint8)
running = False

# Thread for grabbing frames
```

```
class FrameGrabber(Thread):
    def __init__(self):
        super().__init__()
        self.cap = cv2.VideoCapture(0)
        self.daemon = True

    def run(self):
        global latest_frame
        while running and self.cap.isOpened():
            self.cap.grab()
            ret, frame = self.cap.retrieve()
            if not ret:
                time.sleep(0.05)
                continue
            with frame_lock:
                latest_frame = frame
        self.cap.release()

# Detection + ad selection loop
def detect_and_stream():
    global latest_frame, output_frame, current_ad_frame
    last_detection_time = 0
    detection_interval = 0.3  # seconds (about 3 FPS)

    while running:
        if time.time() - last_detection_time < detection_interval:
            time.sleep(0.01)
            continue
        last_detection_time = time.time()
```

```
        with frame_lock:
            if latest_frame is None:
                continue
            frame = cv2.resize(latest_frame, (640, 480))

        results = model(frame, verbose=False)
        boxes = results[0].boxes

        vehicle_classes = [2, 3, 5, 7]  # car, motorcycle, bus, truck
        vehicle_count = 0

        if boxes and boxes.cls is not None:
            classes = boxes.cls.cpu().numpy().astype(int)
            vehicle_count = sum(cls in vehicle_classes for cls in classes)

        ad_index = ad_selector.predict([[vehicle_count]])[0]
        traffic_levels = ["low", "medium", "high"]
        traffic_level = traffic_levels[ad_index]

        ad = select_ad_by_bid(traffic_level)

        with frame_lock:
            output_frame = frame
            current_ad_frame = ad

def generate_video():
    while True:
        with frame_lock:
```

```
            frame = output_frame
        _, buffer = cv2.imencode('.jpg', frame)
        yield (b'--frame\r\nContent-Type: image/jpeg\r\n\r\n' + buffer.tobytes() + b'\r\n')

def generate_ad():
    while True:
        with frame_lock:
            ad = current_ad_frame
        _, buffer = cv2.imencode('.jpg', ad)
        yield (b'--frame\r\nContent-Type: image/jpeg\r\n\r\n' + buffer.tobytes() + b'\r\n')

@app.route('/')
def index():
    return render_template('index.html')

@app.route('/video_feed')
def video_feed():
    return Response(generate_video(), mimetype='multipart/x-mixed-replace;
boundary=frame')

@app.route('/ad_feed')
def ad_feed():
    return Response(generate_ad(), mimetype='multipart/x-mixed-replace; boundary=frame')

@app.route('/control', methods=['POST'])
def control():
    global running
    action = request.form.get('action')
    if action == 'start' and not running:
```

```
        running = True
        FrameGrabber().start()
        Thread(target=detect_and_stream, daemon=True).start()
    elif action == 'stop':
        running = False
    return ('', 204)

if __name__ == '__main__':
    app.run(debug=False, use_reloader=False)
```

4.2.2 Frontend

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Smart Billboard</title>
  <style>
    body {
      display: flex;
      flex-direction: column;
      align-items: center;
      font-family: Arial, sans-serif;
    }
    #feeds {
      display: flex;
      justify-content: space-around;
      width: 100%;
```

```
    }
    .feed {
      border: 2px solid #000;
      width: 640px;
      height: 480px;
    }
    .controls {
      margin: 20px;
    }
    button {
      padding: 10px 20px;
      font-size: 18px;
    }
  </style>
</head>
<body>
  <h1>App</h1>
  <div class="controls">
    <form method="post" action="/control">
      <button type="submit" name="action" value="start">Start</button>
      <button type="submit" name="action" value="stop">Stop</button>
    </form>
  </div>
  <div id="feeds">
    <div class="feed">
      <img src="{{ url_for('video_feed') }}" width="640" height="480">
    </div>
    <div class="feed">
```

```
        <img src="{{ url_for('ad_feed') }}" width="640" height="480">
      </div>
    </div>
</body>
</html>
```

CHAPTER 5

TESTING

5.1 Importance of Testing

Testing is a crucial phase in the development lifecycle that holds paramount importance for ensuring the reliability, functionality, and overall quality of a system or application. It involves systematically evaluating various aspects, such as accuracy, robustness, and user experience, to detect and rectify defects, bugs, or vulnerabilities. Testing not only verifies that the software or model meets specified requirements but also contributes to the prevention of costly errors, the optimization of performance, and the establishment of user trust. Thorough testing provides a systematic and evidence-based approach to validating the system's capabilities, identifying potential risks, and enhancing the overall quality of the end product, whether it be software, a machine learning model, or any computational system.

i. **Accuracy Verification:**

- Testing is crucial for verifying the accuracy of the counterfeit currency detection system. Rigorous testing ensures that the system can effectively differentiate between genuine and fake currency, reducing the risk of false positives or negatives that could have significant financial implications.

ii. **Robustness against Variability:**

- Counterfeit currency may exhibit a wide range of variations in terms of printing quality, paper texture, and other features. Testing allows the system to be robust against such variability, ensuring consistent and reliable performance across different instances of counterfeit notes.

iii. **Adaptability to New Threats:**

- The landscape of counterfeit currency is dynamic, with new techniques and technologies emerging over time. Testing enables the system to adapt to evolving threats, ensuring that it remains effective in detecting the latest counterfeit methods and security features.

iv. **User Trust and Confidence:**

- Thorough testing instils trust and confidence in users, whether they are financial institutions, businesses, or individuals. Knowing that the counterfeit currency detection system has undergone comprehensive testing builds credibility and encourages widespread adoption.

v. **Mitigation of False Alarms:**

- Testing helps identify and address scenarios that may lead to false alarms. By refining the detection algorithm and minimizing false positives, the system becomes more reliable and user-friendly, reducing unnecessary disruptions and alerts.

vi. **Compliance with Regulations:**

- In the financial sector, compliance with regulatory standards is paramount. Testing ensures that the counterfeit currency detection system adheres to relevant regulations, safeguarding against legal and compliance-related risks.

5.2 Types of Testing

Testing is a comprehensive process, and various types of testing are employed to ensure the quality, reliability, and functionality of software, systems, or applications. Here are some key types of testing:

i. **Unit Testing:**

- Involves testing individual units or components of a software independently to ensure they function correctly. Developers often perform unit testing during the development phase.

ii. **Integration Testing:**

- Focuses on verifying the interactions and interfaces between integrated components or systems. It ensures that different modules work together seamlessly.

iii. **Functional Testing:**

- Verifies that the software functions according to specified requirements. It involves testing the system's features, capabilities, and user interactions.

iv. **Non-Functional Testing:**

- Focuses on non-functional aspects such as performance, usability, reliability, and security. Examples include performance testing, usability testing, and security testing.

v. **Regression Testing:**

- Ensures that new changes or updates to the software do not negatively impact existing functionalities. It involves retesting previously tested features.

vi. **User Acceptance Testing (UAT):**

- Involves testing the software from the end user's perspective to ensure that it meets their requirements and expectations. Users typically perform this testing.

vii. **System Testing:**

- Evaluates the complete and integrated system to ensure that it behaves as intended. It assesses the system's compliance with specified requirements.

viii. **Performance Testing:**

- Evaluates the system's performance, responsiveness, and stability under various conditions, such as load testing, stress testing, and scalability testing.

ix. **Usability Testing:**

- Assesses the software's user interface and overall user experience. It ensures that the system is user-friendly and meets user expectations.

x. **Security Testing:**

- Identifies vulnerabilities and weaknesses in the software's security features. It involves testing for potential threats and ensuring data protection.

xi. **Compatibility Testing:**

- Verifies that the software functions correctly across different platforms, browsers, devices, and operating systems.

xv. **Exploratory Testing:**

- Involves ad-hoc testing where testers explore the software to discover defects without predefined test cases. It is often used to identify unexpected issues.

xvi. **Beta Testing:**

- Involves releasing a version of the software to a limited group of users for testing in a real-world environment. It helps gather user feedback before the official release.

xvii. **Alpha Testing:**

- Conducted by internal teams before beta testing. It aims to identify issues within the software before it is made available to a wider audience.

xviii. **Ad-hoc Testing:**

- Informal testing without predefined test cases. Testers use their experience, creativity, and domain knowledge to identify defects.

xix. **Load Testing:**

- Assesses how the system performs under expected and peak loads. It helps ensure that the software can handle a specific number of concurrent users or transactions.

xx. **White Box Testing:**

- Examines the internal logic, structure, and code of the software. Testers have knowledge of the internal workings of the system.

xxi. **Black Box Testing:**

- Focuses on testing the software's functionalities without knowledge of its internal code or logic. Testers assess the system based on input and output.

These testing types can be combined or customized based on the specific needs of a project to achieve a comprehensive testing strategy.

5.2.1 Functional Testing

Testers follow the following steps in the functional testing:

- Tester does verification of the requirement specification in the software application.
- After analysis, the requirement specification tester will make a plan.
- After planning the tests, the tester will design the test case.
- After designing the test, case tester will make a document of the traceability matrix.
- The tester will execute the test case design.
- Analysis of the coverage to examine the covered testing area of the application.
- Defect management should do to manage defect resolving.

CHAPTER 6

RESULTS

The proposed Smart Billboard Advertisement System was successfully implemented and tested in a real-time environment using live video feeds. The system demonstrated efficient vehicle detection capabilities through the YOLOv8 model, accurately identifying multiple vehicle types such as cars, buses, trucks, and motorcycles across various traffic scenarios. The machine learning-based traffic classification model effectively categorized traffic density into low, medium, and high levels based on the detected vehicle count. This classification was then used to select and display advertisements with the highest bid corresponding to the current traffic level, validating the core functionality of intelligent ad targeting.

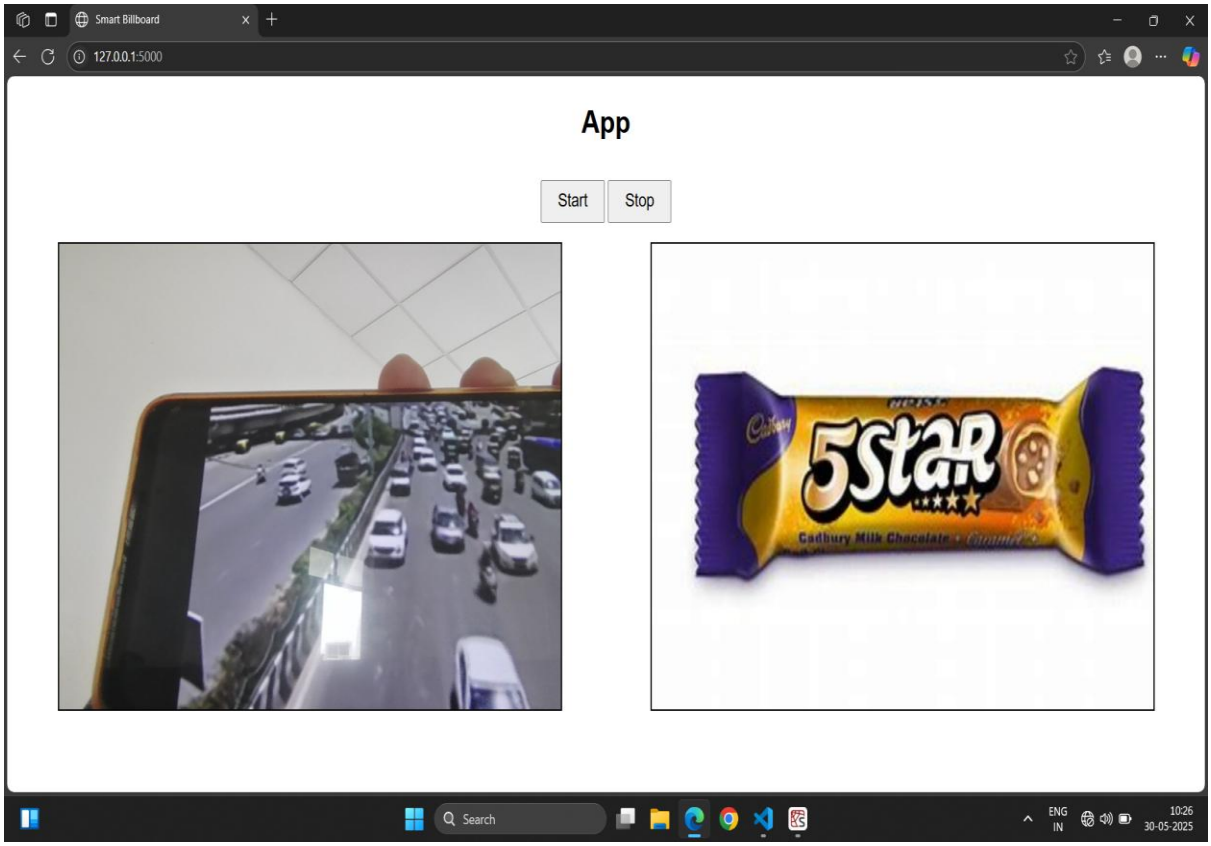

Figure 6: Result

During testing, the system maintained a steady processing speed of approximately 3 frames per second, balancing accuracy and real-time performance. The multithreaded design ensured continuous frame capture and detection without significant delays or frame drops. The web interface provided a seamless user experience, allowing operators to start and stop the system, and view both live camera feeds and the currently selected advertisements simultaneously.

The attached screenshots illustrate key stages of the system in operation. These include the live video feed with detected vehicles, the classification results, and the dynamically updated advertisements shown based on traffic conditions. The results confirm that the system can adapt advertisement content in real-time, potentially increasing advertisement effectiveness and revenue. The system's modular design also enables easy scalability and future enhancements, such as incorporating more sophisticated traffic analysis or expanding the advertisement database.

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

The development of the Smart Billboard Advertisement System marks a significant step toward integrating real-time traffic analysis with dynamic advertisement delivery. Leveraging the capabilities of the YOLOv8 object detection model and a machine learning-based traffic classification algorithm, the system effectively detects vehicles in live video streams and selects advertisements tailored to current traffic conditions. This approach not only optimizes advertisement targeting but also maximizes potential ad revenue by adapting to real-world scenarios dynamically. The successful implementation demonstrates robust performance, with accurate vehicle detection and seamless advertisement updates in near real-time, facilitated by an efficient multithreaded architecture and intuitive web interface.

Looking forward, the system offers considerable potential for enhancements. Future work could focus on expanding the classification granularity by incorporating additional traffic parameters such as vehicle speed, time of day, or weather conditions to further refine advertisement targeting. Integration of more advanced deep learning models for multi-object tracking and behaviour analysis could improve detection accuracy and system responsiveness. Additionally, scaling the system to support multiple camera inputs and deploying it in diverse environments would test its robustness and generalizability. Further, incorporating user feedback mechanisms and analytics could provide valuable insights into advertisement effectiveness, enabling continuous optimization. Overall, this project lays a strong foundation for intelligent, context-aware advertising systems that can evolve with advancing technology and market demands.

CHAPTER 8

REFERENCES

[1] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," *arXiv preprint arXiv:2004.10934*, Apr. 2020. [Online]. Available: https://arxiv.org/abs/2004.10934

[2] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, Apr. 2018. [Online]. Available: https://arxiv.org/abs/1804.02767

[3] V. N. Ribeiro and N. S. T. Hirata, "Combining YOLO and Visual Rhythm for Vehicle Counting," *arXiv preprint arXiv:2501.04534*, Jan. 2025. [Online]. Available: https://arxiv.org/abs/2501.04534

[4] Y. Liu, H. Shen, T. Wang, and G. Bai, "Vehicle Counting in Drone Images: An Adaptive Method with Spatial Attention and Multiscale Receptive Fields," *ETRI Journal*, vol. 45, no. 2, pp. 234–244, 2023. [Online]. Available: https://onlinelibrary.wiley.com/doi/10.4218/etrij.2023-0426

[5] K. A. Redmill *et al*., "Automated Traffic Surveillance Using Existing Cameras on Transit Buses," *Sensors*, vol. 23, no. 11, p. 5086, 2023. [Online]. Available: https://www.mdpi.com/1424-8220/23/11/5086

[6] Z. Černeková *et al*., "Evaluating the Significance of Outdoor Advertising from Driver's Perspective Using Computer Vision," *arXiv preprint arXiv:2311.07390*, Nov. 2023. [Online]. Available: https://arxiv.org/abs/2311.07390

[7] M. Everingham *et al*., "The PASCAL Visual Object Classes (VOC) Challenge," *International Journal of Computer Vision*, vol. 88, no. 2, pp. 303–338, Jun. 2010. [Online]. Available: https://link.springer.com/article/10.1007/s11263-009-0275-4

[8] T.-Y. Lin *et al*., "Microsoft COCO: Common Objects in Context," in *European Conference on Computer Vision (ECCV)*, 2014, pp. 740–755. [Online]. Available: https://arxiv.org/abs/1405.0312

[9] M. Tan and Q. Le, "EfficientNet: Rethinking model scaling for convolutional neural networks," in *Proc. ICML*, 2019. [Online]. Available: https://arxiv.org/abs/1905.11946

[10] H. Wang *et al*., "Deep Learning for Object Detection: A Comprehensive Review," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 9, pp. 4031–4052, Sept. 2021. [Online]. Available: https://ieeexplore.ieee.org/document/9390379