"AutoWake: driver drowsiness detection system"

A Minor Project Report Submitted to

Rajiv Gandhi Proudyogiki Vishwavidyalaya

Towards Partial Fulfillment for the Award of

Bachelor of Technology in Computer Science and Engineering

Submitted by:

Priyansh Rai (0827CS221202)

Maniya Jeswani (0827CS221157)

Priyal Agrawal (0827CS221198)

Punit Sankhala (0827CS221209)

Guided by

Prof. Krupi Saraf

Computer Science and Engineering

Acropolis Institute of Technology & Research, Indore
Jan-June 2025

EXAMINER APPROVAL

The Minor Project entitled ***"AutoWake: driver drowsiness detection system"*** submitted by **Priyansh Rai** (0827CS221202)**, Priyal Agrawal**(0827CS221198)**, Maniya Jeswani** (0827CS221157)**, Punit Sankhala** (0827CS221209) has been examined and is hereby approved towards partial fulfillment for the award of ***Bachelor of Technology in Computer Science and Engineering*** discipline, for which it has been submitted. It understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed, or conclusion drawn therein, but approve the project only for the purpose for which it has been submitted.

(Internal Examiner) **(External Examiner)**

Date: **Date:**

RECOMMENDATION

This is to certify that the work embodied in this minor project entitled **"*AutoWake: driver drowsiness detection system*"** submitted by **Priyansh Rai** (0827CS221202)**, Priyal Agrawal**(0827CS221198)**, Maniya Jeswani** (0827CS221157)**, Punit Sankhala** (0827CS221209) is a satisfactory account of the bonafide work done under the supervision of ***Prof. Krupi Saraf,*** is recommended towards partial fulfillment for the award of the Bachelor of Technology (Computer Science Engineering) degree by Rajiv Gandhi Proudyogiki Vishwavidhyalaya, Bhopal.

(Project Guide)

(Project Coordinator)

(Dean Academics)

STUDENTS UNDERTAKING

This is to certify that the minor project entitled ***"AutoWake: driver drowsiness detection system*** has developed by us under the supervision of ***Prof. Krupi Saraf***. The whole responsibility of the work done in this project is ours. The sole intension of this work is only for practical learning and research.

We further declare that to the best of our knowledge; this report does not contain any part of any work which has been submitted for the award of any degree either in this University or in any other University / Deemed University without proper citation and if the same work found then we are liable for explanation to this.

Priyansh Rai (0827CS221202)

Priyal Agrawal (0827CS221198)

Manya Jeswani (0827CS221157)

Punit Sankhla (0827CS221209)

Acknowledgement

We thank the almighty Lord for giving me the strength and courage to sail out through the tough and reach on shore safely.

There are number of people without whom this project would not have been feasible. Their high academic standards and personal integrity provided me with continuous guidance and support.

We owe a debt of sincere gratitude, deep sense of reverence and respect to our guide and mentor **Prof. Krupi Saraf**, AITR, Indore for his motivation, sagacious guidance, constant encouragement, vigilant supervision, and valuable critical appreciation throughout this project work, which helped us to successfully complete the project on time.

We express profound gratitude and heartfelt thanks to **Prof. Krupi Saraf**, AITR Indore for his support, suggestion, and inspiration for carrying out this project. I am very much thankful to other faculty and staff members of the department for providing me all support, help and advice during the project. We would be failing in our duty if do not acknowledge the support and guidance received from **Prof. Narendra Pal**, Director, AITR, Indore whenever needed. We take opportunity to convey my regards to the management of Acropolis Institute, Indore for extending academic and administrative support and providing me all necessary facilities for project to achieve our objectives.

We are grateful to **our parent** and **family members** who have always loved and supported us unconditionally. To all of them, we want to say "Thank you", for being the best family that one could ever have and without whom none of this would have been possible.

Priyansh Rai (0827CS221202) **Priyal Agrawal (0827CS221198)**
Manya Jeswani (0827CS221157) **Punit Sankhla (0827CS221209)**

Executive Summary

AutoWake: driver drowsiness detection system

Submitted to Rajiv Gandhi Proudyogiki Vishwavidyalaya, Bhopal (MP), India, for the partial fulfillment of a Bachelor of Technology in Computer Science and Engineering, under the guidance of Prof. Krupi Saraf.

AutoWake is a smart safety system designed to detect if a driver is feeling sleepy or tired while driving. It uses a simple camera to keep track of the driver's face and expressions — without making them wear any extra devices like sensors or bands. This type of setup is called *non-intrusive monitoring*, which means it does not disturb the driver or affect their comfort.

Our system works by identifying signs of drowsiness such as eye closure, yawning, and head movement. When such signs are noticed, the system can alert the driver in real time to prevent accidents. This is especially useful for long-distance drivers or people who drive at night.

The technology behind AutoWake includes a Python-Flask backend that connects with a trained Machine Learning model to process the video feed, and a React-based frontend to make it easy to use. AutoWake makes driving safer, more comfortable, and smarter — all without needing any wearables.

This project is helpful for drivers, fleet companies, and anyone concerned about road safety. It's an easy-to-use, low-cost solution for a very serious problem.

Keywords: Driver Drowsiness Detection, Machine Learning,

AutoWake, Non-Intrusive Monitoring, Road Safety, Smart Driving System

"If your vision is for a year,

plant seeds.

If your vision is for a decade,

plant trees.

If your vision is for a lifetime,

nurture people."

- Oriental Saying

List of Figures

List of Tables

List of Abbreviations

Abbr1: ML – Machine Learning

Abbr2: AI – Artificial Intelligence

Abbr3: UI – User Interface

Abbr4: CNN – Convolutional Neural Network

Abbr5: LSTM – Long Short-Term Memory

Abbr6: JS – JavaScript

Abbr7: HTML – HyperText Markup Language

Abbr8: CSS – Cascading Style Sheets

Abbr9: Flask – A Python Web Framework

Abbr10: CORS – Cross-Origin Resource Sharing

Abbr11: OpenCV – Open Source Computer Vision Library

Abbr12: Dlib – Data Library for Facial Landmark Detection

Table of Contents

Chapter 1. Introduction

AutoWake is a smart and easy-to-use system that helps detect if a driver is feeling sleepy while driving. It does this in a non-intrusive way, which means it doesn't disturb or discomfort the driver. Many road accidents happen because drivers feel drowsy and lose focus. Our project tries to solve this problem using a camera and machine learning. By analyzing the driver's facial expressions—like closed eyes or yawning—our system can give alerts when signs of sleepiness are detected. AutoWake aims to make driving safer by preventing accidents before they happen.

1. Overview

AutoWake is a machine learning-based driver drowsiness detection system. It works without needing the driver to wear any devices like eye trackers or heart rate monitors. Instead, it just uses a camera to keep an eye on the driver's face and checks for signs like eye closure, head tilting, and yawning. The system runs in the background and alerts the driver if it detects any signs of drowsiness. The goal of this project is to improve road safety using a system that is easy to use, reliable, and does not interrupt the driver's comfort.

2. Background and Motivation

Every year, many accidents happen because drivers get sleepy behind the wheel. Even though there are systems out there to monitor drivers, most of them need the person to wear special devices or sensors, which can be uncomfortable or expensive. That's where the idea of **non-intrusive monitoring** comes in.

We wanted to create a system that could check for drowsiness just by watching the driver's face using a regular camera. No need for the driver to wear anything extra. This makes it simpler, cheaper, and more practical for daily use. The motivation

behind AutoWake is to help reduce road accidents by building a low-cost, non-intrusive safety system for drivers.

3. Problem Statement and Objectives

Problem Statement:

Many accidents happen because drivers get drowsy and there's no one to warn them. Current systems that can detect this either require wearing devices or are too expensive and uncomfortable. There's a need for a simple, affordable, and effective way to detect driver drowsiness.

Objectives:

The main goal of AutoWake is to develop a non-intrusive system that can:

- Detect signs of driver drowsiness in real time.
- Use only a camera and machine learning (no wearable devices).
- Alert the driver instantly when signs of drowsiness are detected.
- Make driving safer and prevent accidents.

4. Scope of the Project

AutoWake focuses on detecting drowsiness in drivers using a simple webcam and machine learning model. It is built mainly for private car users, truck drivers, and public transport drivers who need to stay alert on long journeys.

The scope includes:

- Collecting and preparing a dataset of facial expressions (normal and drowsy).
- Training a machine learning model to recognize signs of drowsiness.
- Creating a simple application that takes webcam input and gives alerts.
- Making sure the system works in real-time and is easy to use.

In the future, we can improve this system by adding features like voice alerts, night vision support, and even linking it with vehicle systems to slow down the car if needed.

5. Team Organization

The **AutoWake** project was developed by a team of four dedicated members, each contributing to different aspects of the project based on their strengths and interests. The responsibilities were divided as follows:

1. **Priyansh Rai:** Responsible for designing and implementing the user interface (UI) of th'e application. Ensured that the frontend is interactive, user-friendly, and provides an intuitive experience for car price predictions.
2. **Priyal Agrawal:** Focused on front-end development, ensuring that the web application is responsive and aesthetically appealing. Worked on integrating the frontend with backend services for smooth data exchange.
3. **Manya Jeswani:** Handled backend development, building the API services for data retrieval and model interaction. Additionally, worked on training and testing the machine learning model to predict car prices accurately based on input features.
4. **Punit Sankhala:** Contributed to data preprocessing and model development, ensuring the machine learning model performs optimally. Worked on integrating the model into the backend system and handling server-side logic.

6. Report Structure

This report explains the complete journey of how we created **AutoWake**, starting from a simple idea to a working, real-time driver drowsiness detection system. Each chapter describes a different phase of the project in a clear and easy-to-understand manner.

- **Chapter 1: Introduction** – This chapter gives a clear introduction to the AutoWake project. It explains what our system does, why we chose to work on

drowsiness detection, and what we mean by *non-intrusive monitoring*. We also discuss the problems caused by driver sleepiness and how our system aims to solve them using machine learning and computer vision.

- **Chapter 2: Review of Literature** – In this chapter, we talk about the existing systems and research papers related to driver drowsiness detection. We looked at many methods—some that use sensors and others that require wearable devices. We point out the limitations of those methods and explain how AutoWake is different and more user-friendly, especially because it doesn't need any physical contact or devices.

- **Chapter 3: Proposed System** – This chapter describes how our system works. We explain the technical side of the project, like the machine learning model we used, the features we looked at (like eye movement and yawning), and how we designed the whole system to detect drowsiness in real time. We also mention the tools and technologies we used—such as Python, OpenCV, TensorFlow/Keras (if applicable), and the webcam for video input.

- **Chapter 4: Implementation** – This part of the report shows how we actually built the AutoWake system step by step. We explain how we collected and prepared the data, trained the model, and created the user interface. We also talk about the challenges we faced during development and how we solved them. Screenshots, sample code, and flow diagrams are included to make things clearer and easier to follow.

- **Chapter 5: Conclusion** – The final chapter gives a summary of what we learned from the project and how successful AutoWake was in meeting our goals. We discuss how well the system worked, its accuracy, and the areas where it can be improved. We also share some ideas for future work, like adding voice alerts,

supporting night-time driving, or integrating the system with real vehicles.

Chapter 2. Review of Literature

Many researchers have tried to solve the problem of driver drowsiness, but even today, it's still a major cause of accidents. Wearable devices like EEG headbands and heart rate monitors may offer accurate results, but they are uncomfortable and not practical for daily driving. Drivers shouldn't have to feel like lab subjects just to stay safe.

Some systems track biological signals, but again, they demand too much from users and often cost a lot. Camera-based systems seemed like a better option—no contact, no discomfort. But even these solutions often fail in low-light, struggle with different faces, or require powerful hardware.

Despite all the effort, there's still no perfect answer. That's where **AutoWake** steps in—not to be perfect, but to be simple, non-intrusive, and actually usable. We looked at where others struggled and tried to build something better—something that respects both safety and the comfort of the driver.

2.1 Preliminary Investigation

2.1.1 Current System

The dream of creating safer roads has led researchers down many paths, but most solutions for drowsiness detection seem to forget the one person they're trying to help—the driver. Countless studies have explored intrusive methods like EEG headbands, eye-tracking glasses, and heart rate monitors, hoping for precision but ignoring comfort. This section looks at five such systems, each showing potential, but none truly understanding the need for a simple, non-intrusive solution like AutoWake.

1. **Drowsiness Detection Using EEG and Heart Rate Monitors**

Author: Jiaying Gao

Approach: EEG Sensors, Heart Rate Monitoring

Overview:

This system straps sensors to the driver's head and chest, claiming accurate detection of drowsiness through brain and heart activity. But at what cost? Wires and discomfort make it feel more like a medical test than a driving aid. Yes, the data is rich—but no one wants to wear a lab on their face while driving.

2. **Wearable-Based Driver Alertness Monitoring**

Authors: G. SelvaKumar et al.

Approach: Smartwatches, Motion Sensors

Overview:

Another attempt involves tracking hand and body movements using wearables like smartwatches. The idea is good, but again, the burden is on the driver. People don't want another gadget to charge or wear just to stay awake. Real-world use becomes a hassle, and what starts as safety turns into a chore.

3. **Driver Fatigue Detection Using Eye-Tracking Glasses**

Authors: Kafeel Noor, Shahbaz Jan

Approach: Eye-Tracking Wearables

Overview:

Eye-tracking sounds smart—until you have to wear special glasses every time you drive. This system shows high accuracy but completely ignores user comfort. Not everyone wants to feel like they're in a lab experiment on their daily commute. Technology should work *with* us, not *on* us.

4. **Fatigue Detection through Brainwave Monitoring**

Authors: Enis Gegic et al.

Approach: EEG-based Systems

Overview:

The researchers here went deep—literally—into brainwaves. While the system detects fatigue at an early stage, it demands constant contact with the scalp. It's highly technical, but again, highly impractical. Who's going to wear a brainwave cap just to take a road trip? This isn't sci-fi—it's daily life.

5. **Multimodal Detection Using Physiological Signals**

Authors: V. Pattabiraman, M. Ganesh

Approach: Heart Rate, Skin Conductance

Overview:

This approach combines several biometric signals to monitor fatigue, but it turns the car into a science lab. The system's performance is strong, but the experience for the driver? Overwhelming. More wires, more sensors, more things to worry about—when all they want is to stay safe and drive.

2.2 Limitations of Current System

Despite all the advancements, most existing drowsiness detection systems still fall short in practical use. They may look good on paper, but when it comes to real-world driving, they come with several problems that make them difficult to adopt. Below are some major limitations we noticed while studying the current systems—issues that led us to build something better with Auto Wake:

- Uncomfortable Wearables:

Many systems force drivers to wear devices like EEG headsets or heart rate monitors. Honestly, who wants to strap on gear just to drive? It's not only uncomfortable but also distracting—and let's face it, no one likes feeling like a lab

rat behind the wheel.

- Too Much Dependency on Expensive Hardware:

Some solutions require fancy glasses, infrared sensors, or special equipment. This not only increases the cost but also makes the system harder to use for everyday drivers. These aren't made for real people with real cars—they feel more like tech demos than practical tools.

- Invasion of Privacy:

Systems that track everything—from your eye movement to your pulse—might get detailed data, but they also make people uneasy. Drivers want to feel safe, not watched. There's a fine line between monitoring and invading personal space, and many current models cross it.

- No Adaptation to Real Driving Conditions:

A lot of research is done in ideal lab environments, not on chaotic roads. When applied in real-world driving—nighttime glare, bumpy roads, different lighting—their accuracy drops. That makes them unreliable, and honestly, a bit disappointing after all the hype.

- Not User-Friendly:

Most existing systems are hard to set up and operate. They expect the user to be a tech expert, which isn't realistic. Drivers need something that just works, not another complicated device to figure out.

2.3 Requirement Identification and Analysis for Project

For the AutoWake project, our main goal is to create a system that can detect drowsiness in drivers without disturbing them. We wanted something simple, smart, and non-intrusive—a system that just works without needing the driver to wear uncomfortable sensors or gadgets. But before we could even start building it, we

had to figure out exactly what was needed—both in terms of what the system should do, and how it should behave.

Functional Requirements

- Camera-Based Monitoring

The system must use a normal camera to capture the driver's face while driving. No fancy hardware. No headbands or wires. Just a simple webcam that watches for signs of drowsiness like eye closure, yawning, and head-nodding.

- Facial Expression Detection

The core function of AutoWake is to analyze facial features and detect when the driver is feeling sleepy. This means detecting micro expressions and behavioral cues—like slow blinking or drifting gaze. This is where the real challenge lies.

- Real-Time Alerts

Once drowsiness is detected, the system should immediately give a warning—like a beep or on-screen message—to wake the driver. After all, there's no point in detecting sleep if you can't stop it in time.

- Basic UI for Users

There should be a simple interface—nothing flashy—just something that lets users turn the system on, see their status, and get alerts if they're drifting off. It needs to be clean and usable, even by someone who's not great with tech.

Non-Functional Requirements

- Speed (Real-Time Performance)

The system has to work fast. Drivers don't have time to wait for the system to "analyze" while they're nodding off. We want alerts to pop up instantly when sleep is detected.

- Accuracy

If the system keeps giving false alarms or missing real signs of drowsiness, it's useless. So we need the detection to be as accurate as possible—even though working with facial expressions isn't always easy.

- Usability

Most people don't read manuals. So, AutoWake should be super simple to use—plug it in, start driving, and let it do its thing. We don't want drivers fiddling with settings while on the road.

- Scalability

If this system ever gets used in fleets or cabs, it should be able to handle multiple users or long hours of operation without crashing or lagging. We have to think beyond a single laptop setup.

- Privacy & Safety

Since we're using a camera, we also have to make sure that driver privacy is respected. No unnecessary data should be stored or sent anywhere. The system should process everything locally, keeping it secure and personal.

2.3.1 Conclusion

This section looked into various existing methods used for driver drowsiness detection, especially those that involve physical devices like wearables. While these systems work well, they often make the user uncomfortable or are not practical for daily use. Many current approaches require the driver to wear sensors, headbands, or special glasses, which can be annoying and distracting. This is where AutoWake tries to make a difference. Our project focuses on a non-intrusive method using just a camera to monitor the driver's face for signs of drowsiness. It avoids physical contact and makes the system more user-friendly. From what we studied, it's clear

that there's a real need for systems that are both accurate and comfortable to use. AutoWake is designed to fill that gap by combining smart detection with ease of use. The upcoming chapters will explain how we've built and implemented this system, based on the problems

Chapter 3. Proposed System

3.1 The Proposal

AutoWake: Driver Drowsiness Detection System offers a fresh, innovative approach to monitoring driver behavior without interrupting their comfort or normal activities. The idea is simple—using a camera, AutoWake detects facial expressions, eye movements, and other signs of drowsiness to assess whether the driver is becoming tired. The system doesn't require the driver to wear any sensors or devices, which makes it much more convenient than other methods that rely on wearables like heart rate monitors or eye trackers. This non-intrusive approach not only enhances comfort but also makes the system easy to use, as it's all about observing and analyzing the driver's face.

The system has two main parts: the frontend, where drivers can see and interact with the system, and the backend, where the real-time analysis happens. The backend processes the video feed from the camera, detecting drowsiness signs and sending alerts when necessary. The whole setup ensures smooth communication between the driver and the system, providing a safer and more user-friendly experience.

3.2 Benefits of the Proposed System

The AutoWake: Driver Drowsiness Detection System offers several benefits over existing methods for driver monitoring:

1. **Non-Intrusive Monitoring**: AutoWake doesn't require drivers to wear any sensors or devices. It simply uses a camera to observe the driver's face, making the experience more comfortable and less invasive compared to systems that require wearables like heart rate monitors or eye trackers.
2. **Convenience and Comfort**: The system works without interrupting the driver's normal activities, allowing them to drive as usual while still being monitored for

signs of drowsiness. This makes it a lot more user-friendly and less distracting for the driver.

3. **Real-Time Detection**: AutoWake provides real-time analysis of the driver's facial expressions and behavior, allowing it to detect drowsiness and alert the driver instantly. This ensures that potential risks are addressed before they become a serious issue.
4. **Improved Safety**: By detecting drowsiness early, the system can help prevent accidents caused by fatigue. This makes the system a useful tool for keeping drivers safer on the road.
5. **Easy to Use**: Since it doesn't require special equipment or technical know-how, AutoWake is easy for anyone to use. The system automatically detects drowsiness without the need for constant interaction or setup.
6. **Adaptability**: The system is flexible and can work in different driving conditions, such as day or night. It continuously adapts to changes in the driver's face, ensuring that it remains accurate over time.
7. **Cost-Effective**: Because it doesn't require extra sensors or wearables, AutoWake is an affordable solution for improving driver safety without the need for expensive equipment.
8. **More Comfortable Experience**: Unlike other monitoring systems that can be uncomfortable or intrusive, AutoWake provides a seamless experience, allowing drivers to remain focused on the road while being monitored in a non-intrusive way.

3.3 Block Diagram

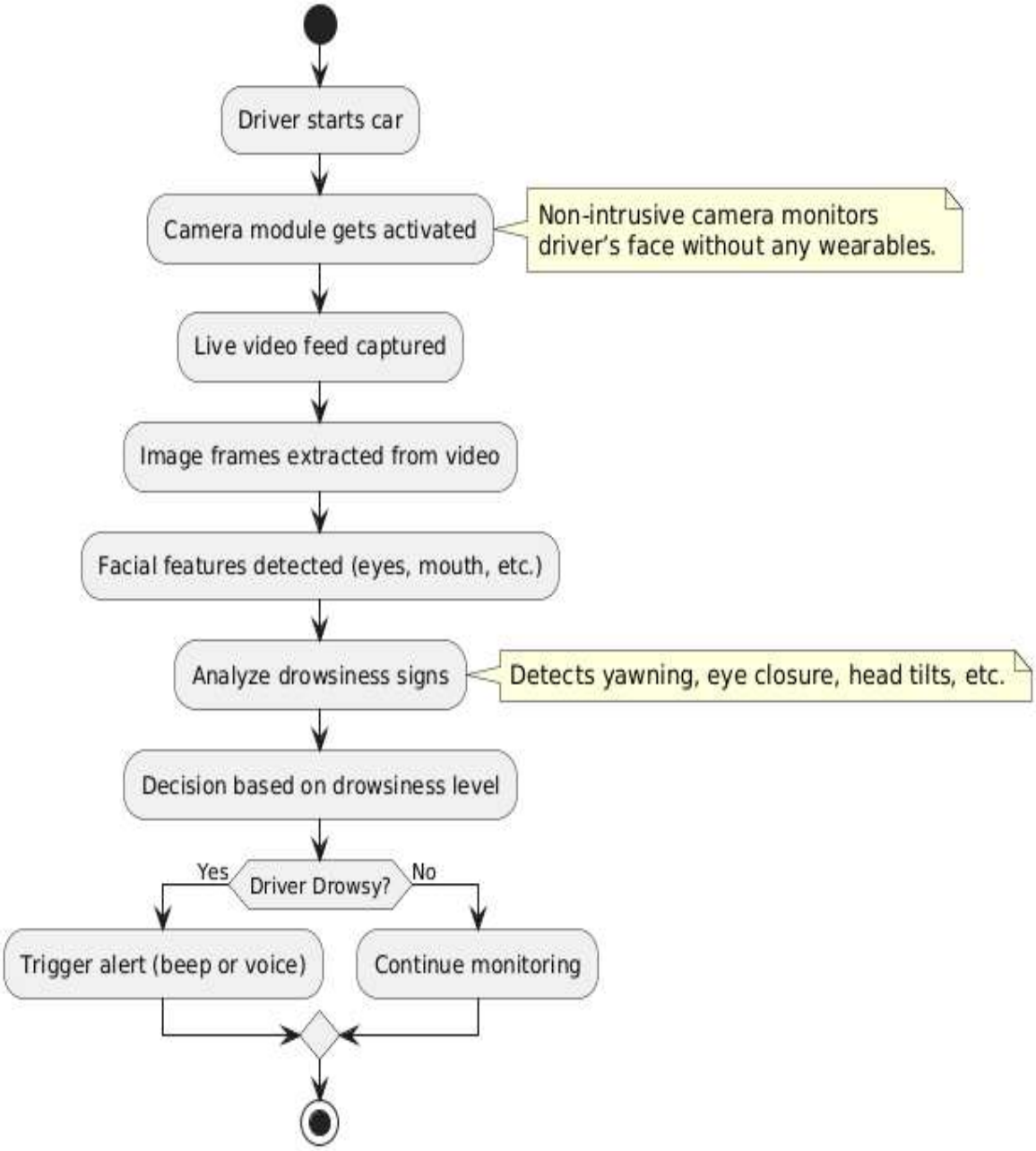


Figure 3.1: Block Diagram

3.4 Feasibility Study

A feasibility study helps us decide whether our system is practical and if it can really be built using the tools and knowledge we currently have. It checks if the system is technically possible, cost-effective, and easy to use in real life. For our project AutoWake, the study helps us make sure that our non-intrusive driver drowsiness detection system is actually useful and implementable.

3.4.1 Technical Feasibility

This part checks if AutoWake can be developed using the technologies and platforms that are easily available to us.

1. **Frontend Development:**

The frontend of **AutoWake** is developed using **Electron JS** along with **React JS** and **Vite**. This setup allows us to build a desktop-like application using web technologies. Electron helps us run the app like regular software on a computer, while React and Vite make the interface fast and responsive.

We used **JavaScript** for the logic, and **HTML5** and **CSS3** for creating and styling the user interface. The combination of these technologies helps make the application clean, interactive, and easy to use for drivers or system users.

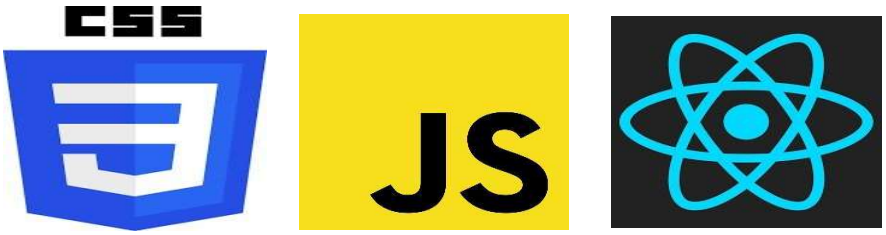


Figure 3.2 – Frontend Languages

2. **Backend Development:**

The backend is built using **Flask**, a lightweight Python web framework, along with **Flask-CORS** for handling cross-origin requests. Flask helps us connect the frontend with the machine learning model running in the backend. It handles the user's requests, processes the video frames, and returns results like drowsiness status.

Since the backend is written in **Python**, it's easy to integrate with ML libraries like **OpenCV**, **dlib**, and **NumPy**, which are used for processing the camera feed and detecting signs of drowsiness such as blinking, yawning, or head movements.

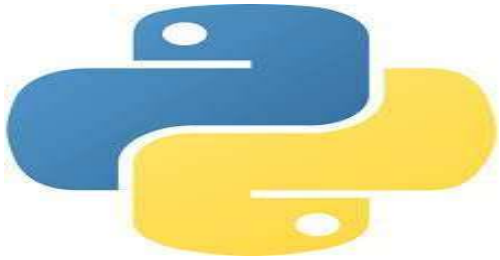


Figure 3.3 – Backend Languages

3. Video & Machine Learning Processing:

To detect drowsiness in a non-intrusive way, we used OpenCV for video capture and frame analysis, dlib for face detection and landmark tracking, and NumPy for numerical operations. These tools allow us to analyze a driver's face without needing any sensors or wearables.

Our model uses Convolutional Neural Networks (CNN) to recognize patterns in facial expressions and Long Short-Term Memory (LSTM) networks to track changes over time. This combination helps in accurately identifying if a driver is feeling sleepy

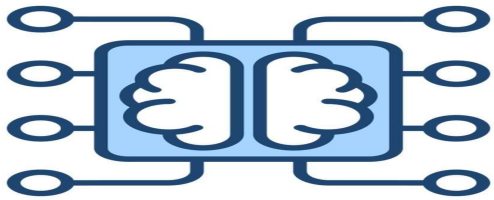


Figure 3.4 – Machine Learning

4. Development Tools & Environment:

We used Python for machine learning and backend scripting, and Node JS to manage packages and development tools. Git was used for version control so we could track our changes and collaborate easily as a team.

These tools helped us develop and maintain a smooth, real-time driver monitoring system.

5. Deployment:

AutoWake can be hosted online using platforms like **Heroku**, **Render**, or **AWS**. This way, anyone can use the system just with a browser and a webcam. No downloads or installations are needed. These platforms also help handle more users and requests if needed in the future.

So technically, the project is very much doable using the tools and knowledge we have. It doesn't require any expensive hardware, and everything runs using basic programming and cloud tools.

3.4.2 Economic Feasibility

This part checks if our project is affordable and worth the money spent.

- **Low Cost Development**: AutoWake is made using free and open-source tools like Python, OpenCV. This means we don't have to buy any expensive software, which saves a lot of money for students like us.
- **No Need for Expensive Hardware**: Unlike other drowsiness systems that need special devices like eye trackers or heart monitors, AutoWake works with just a normal webcam. This reduces hardware costs and makes the system much more budget-friendly.
- **Cheap Hosting Options**: We can host the system on low-cost platforms like Heroku, Render, or even local machines for testing. These platforms offer

free tiers or affordable pricing, so deployment doesn't cost much.

- **Future Earning Possibility**: If we plan to make it public or commercial, we can add premium features like real-time alert systems, fleet monitoring for transport companies, or even partner with car manufacturers. There's potential to earn through subscriptions or partnerships.
- **Efficient Use of Resources**: Since everything is automated and non-intrusive, there's no need for human monitoring. This saves time and reduces the need for extra staff or equipment.

3.4.3 Operational Feasibility

This section checks if the system will work smoothly in real life:

- **Simple to Use**: The system is very easy to operate. The user just needs to sit in front of a webcam. The software automatically detects if the driver is getting sleepy—no need to wear anything or press any buttons.
- **No Disturbance to Driver**: The main goal is to keep the monitoring non-intrusive. AutoWake silently observes the face without bothering or distracting the driver. This helps drivers stay comfortable and focused.
- **Runs Smoothly in Real-Time**: The system processes video in real time and immediately shows alerts if drowsiness is detected. This helps prevent accidents before they happen.
- **Easy to Maintain**: Since we used clean and modular code, future updates like adding new detection features (e.g., yawning detection or head-nodding) can be added easily without changing the whole system.
- **Can Be Used in Real Vehicles**: With a simple setup (just a laptop or small onboard computer and a webcam), AutoWake can be installed in real cars, especially for long-distance drivers, taxi services, or logistics companies.
- **Better Driver Safety**: Overall, AutoWake improves road safety by warning the driver in time. This makes it useful not just for individuals but

also for companies managing many vehicles.

1.5 Design Representation

Frontend Design Representation of our project is:

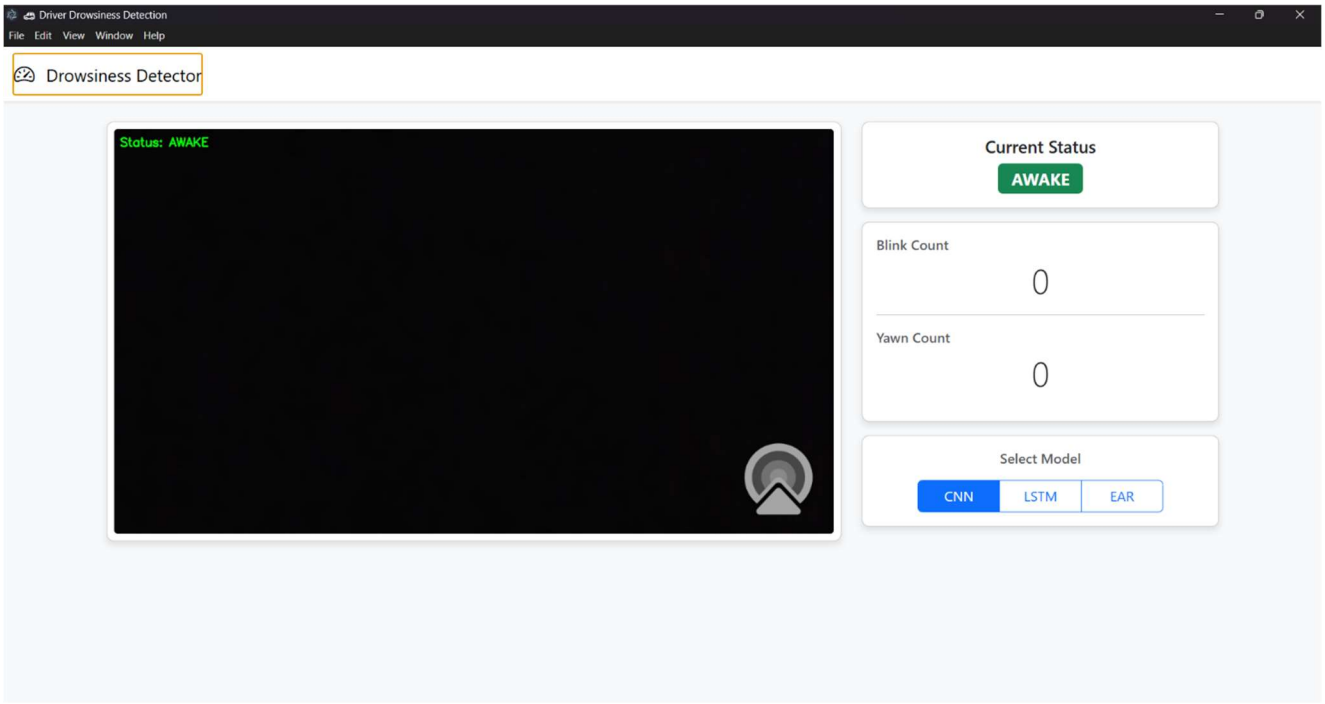


Figure 3.5– Design Representation

- **Activity Diagram**

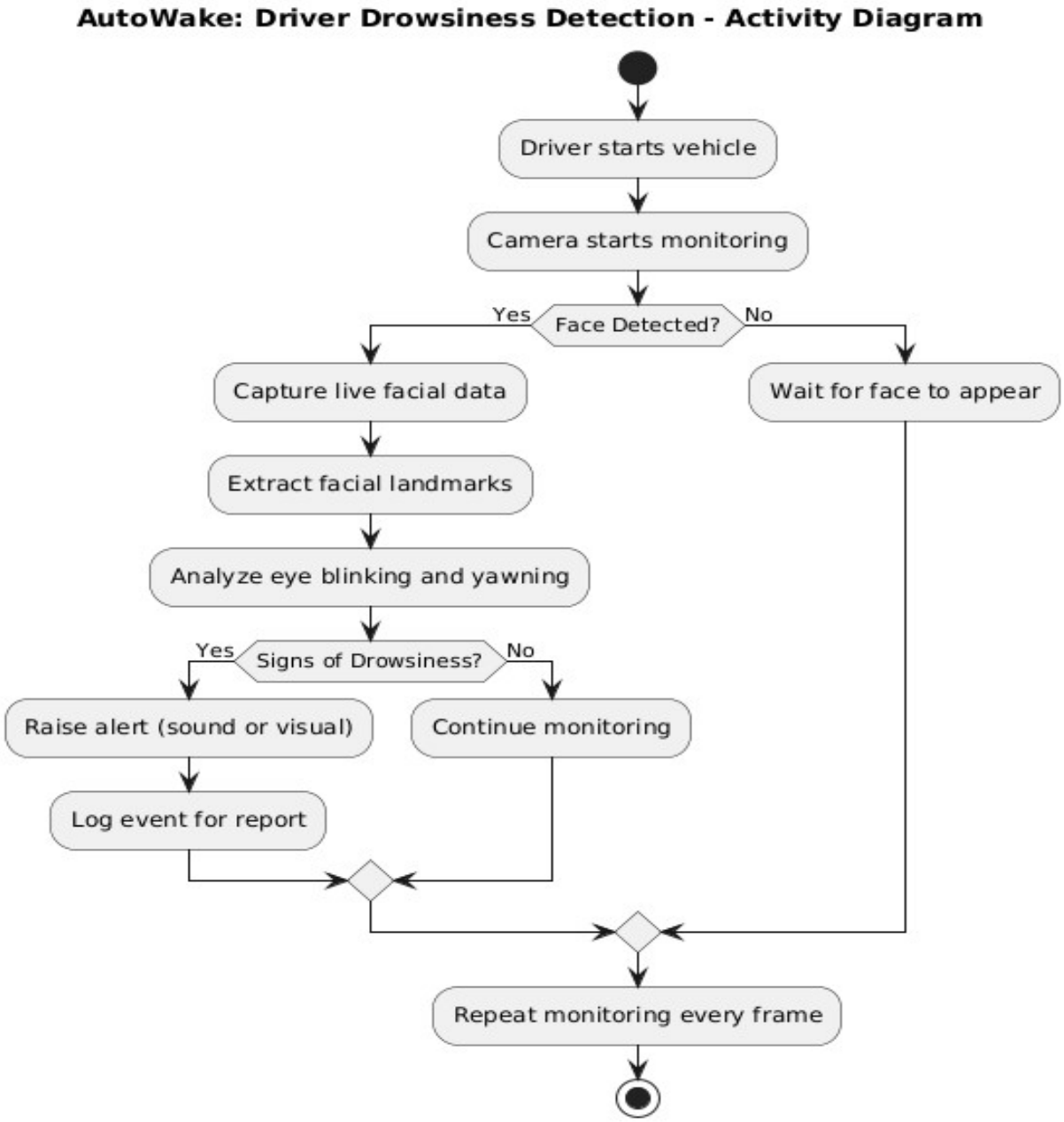


Figure 3.6 : Activity Diagram

- **Use Case Diagram**

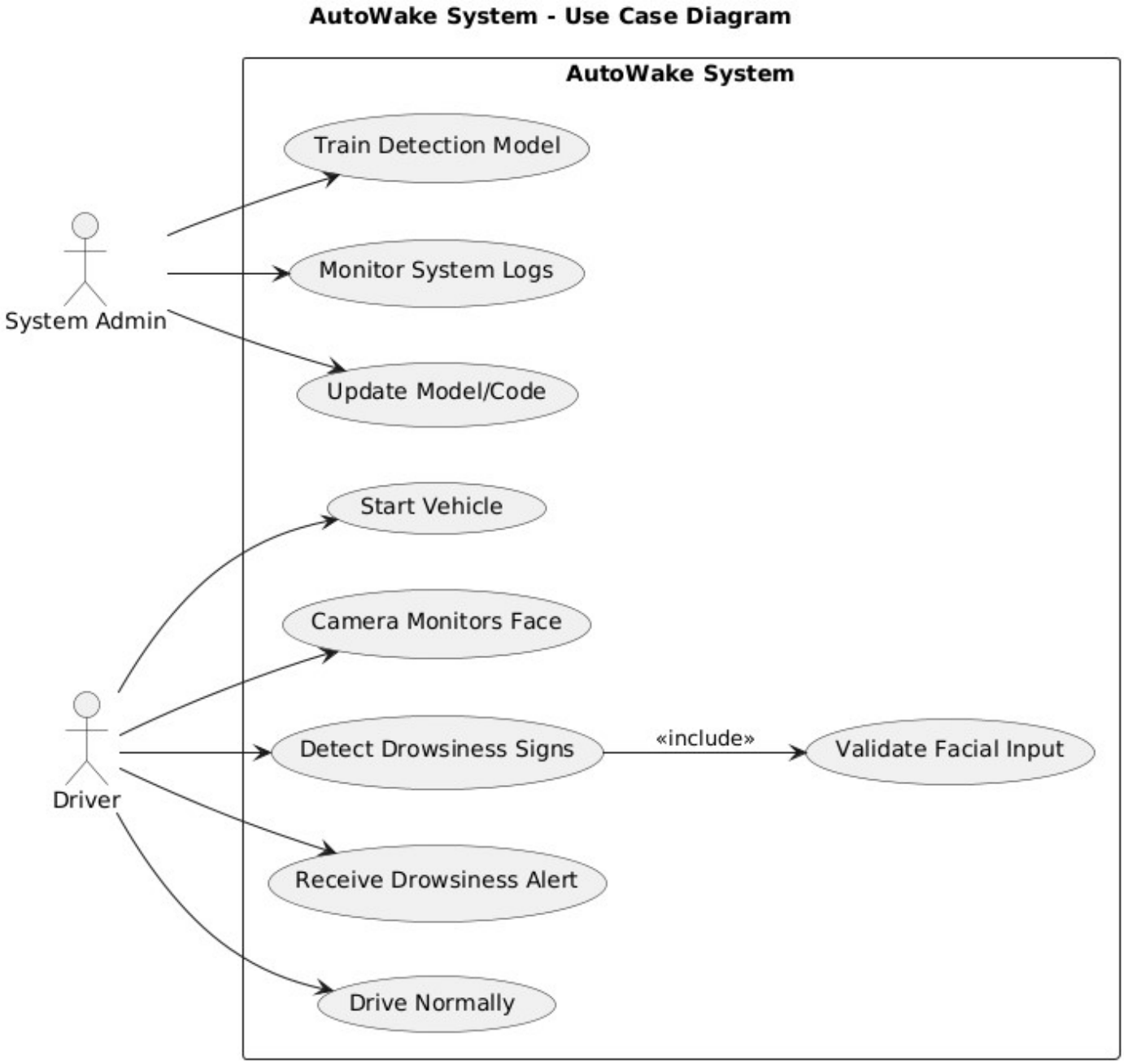


Figure 3.7 : Use Case Diagram

- **ER Diagram**

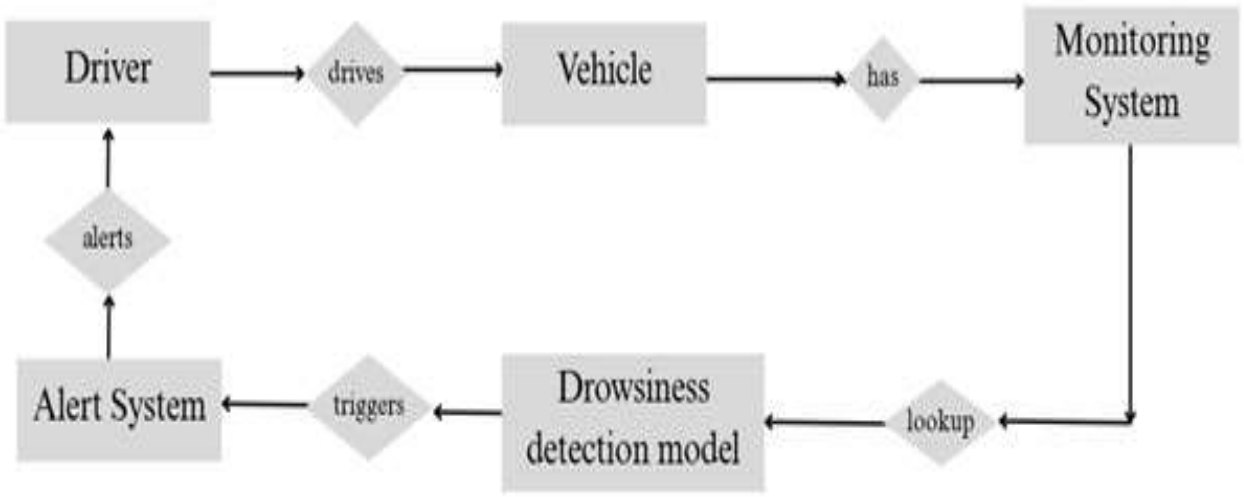


Figure 3.8: ER Diagram

- **Flowchart**

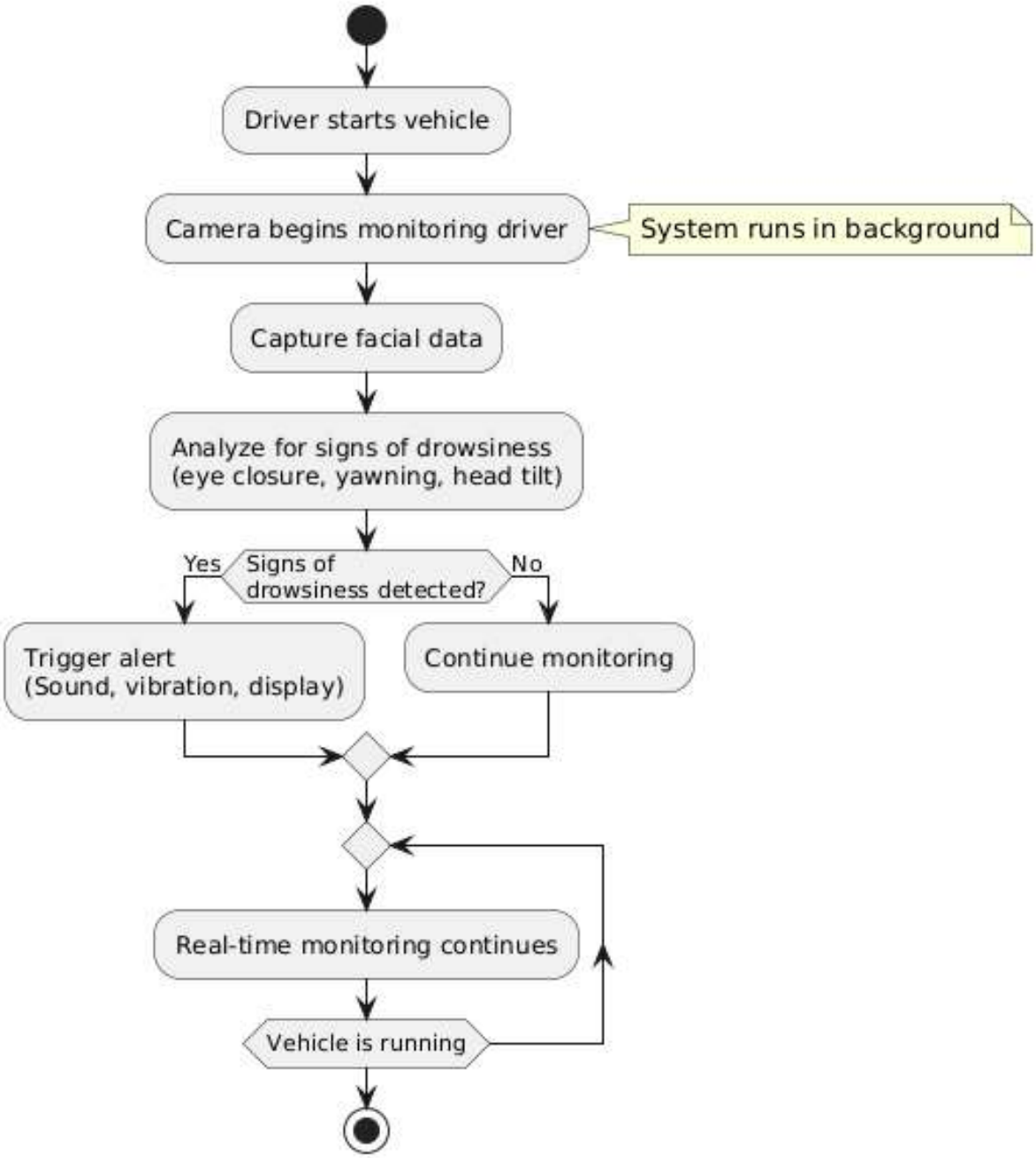


Figure 3.9: Flowchart

- **Sequence Diagram**

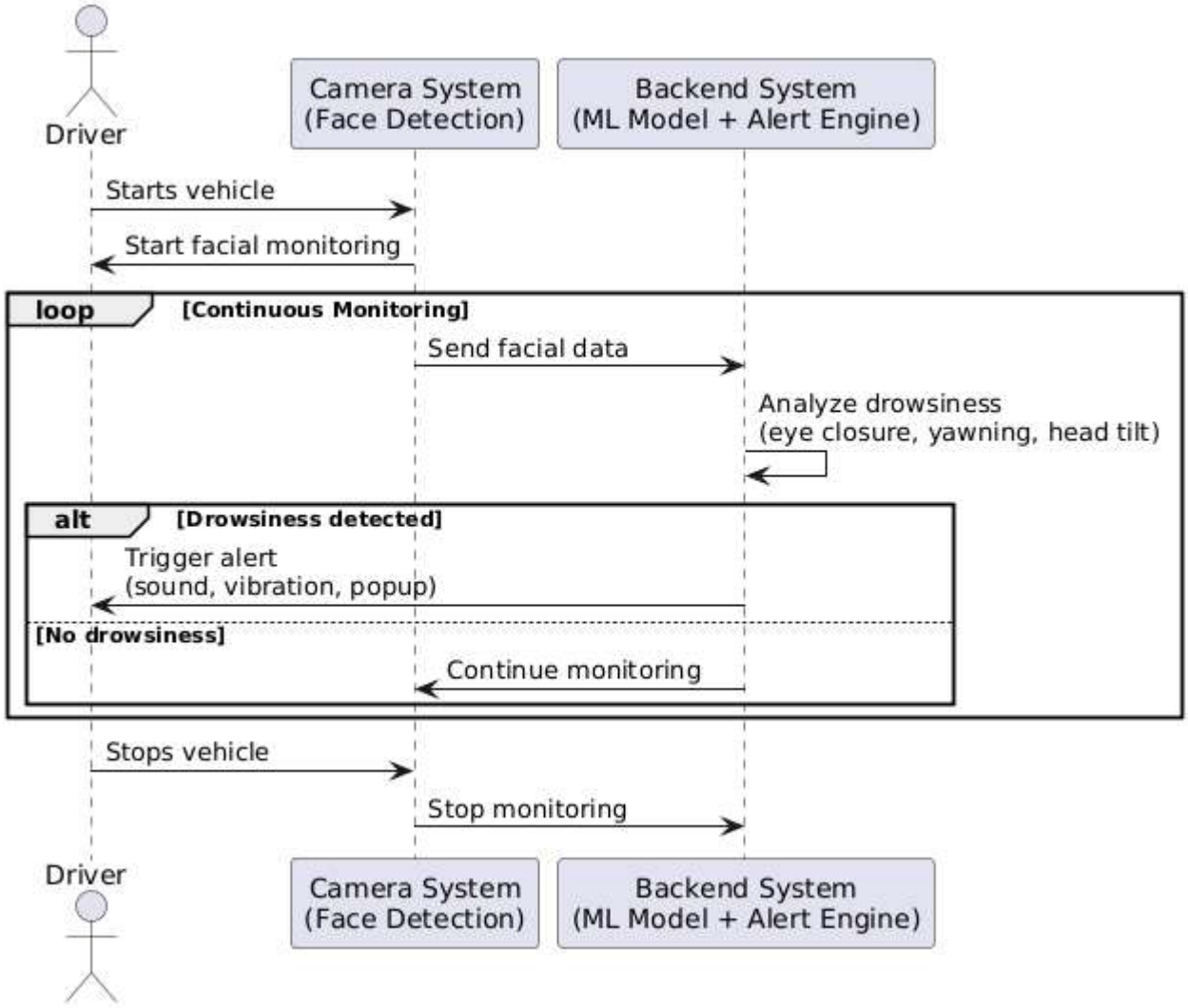


Figure 3.10: Sequence Diagram

3.6 Deployment Requirements

There are various requirements (hardware, software and services) to successfully deploy the system. These are mentioned below

3.6.1 Hardwarw

Component	Specification
Processor	Minimum Intel i5 or AMD Ryzen 5 or above
RAM	Minimum 8 GB (Recommended: 16 GB)
Storage	At least 500 GB HDD or 256 GB SSD
Network	Stable internet connection (for web hosting)

3.6.2 Software

Component	Specification / Tools Used
Operating System	Windows 10/11, Linux (Ubuntu), or macOS
Programming Language	JavaScript (for frontend), Python (for backend and ML logic)
Video & ML Processing	OpenCV (for video frame capture), dlib (for facial landmark detection), NumPy (for data ops)
Alert Mechanism	Python modules like playsound or pygame (to play sound alerts), or simple buzzer (if hardware is involved)
Frontend Technologies	Electron JS, React JS (with Vite), JavaScript, HTML5, CSS3 – for building the user interface
Machine Learning	CNN (for spatial analysis), EAR(for Aspect measurement)
IDE/Code Editor	VS Code, Jupyter Notebook – used for development
Testing Tools	Webcam (for live drowsiness detection testing), Python console
Deployment Platform	Localhost (during development), can be deployed to Heroku or Render if needed later

Chapter 4. Implementation

This chapter explains how we built the **AutoWake** system step by step. From setting up the frontend and backend to integrating machine learning with real-time video processing, all the components were connected to detect driver drowsiness in a non-intrusive way.

4.1 Technique Used

4.1.1 React.js + Electron.js for Frontend

We used React JS (with Vite) along with Electron JS to build the user interface. React helped us make a modern and dynamic web-based UI, while Electron allowed us to turn it into a desktop application that runs independently on Windows/Linux.

React & Electron were used for:

Frontend Interface

Designed a simple UI to show the camera feed, drowsiness alerts, and system status.

Component-Based Design

Used reusable React components to manage the layout for video, alert messages, and detection status.

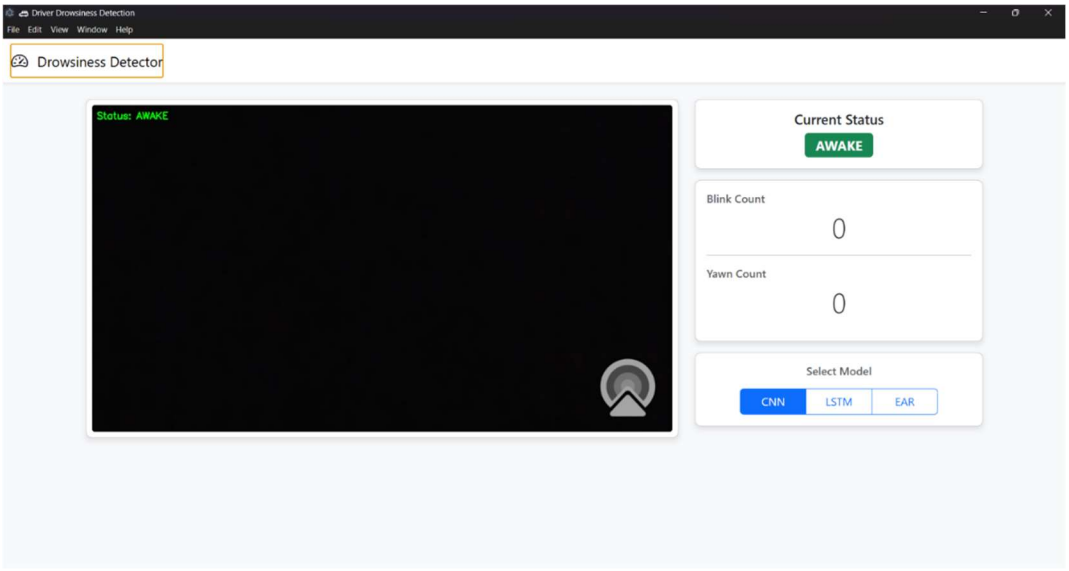
Real-Time Updates

Used React's useState and useEffect to update the UI in real-time when drowsiness is detected.

Electron Integration

Electron JS helped us turn the web app into a cross-platform desktop application that works even without internet.

Home Page:



Result Page :

4.1.2 Python + Flask for Backend

The backend was developed using **Flask**, a lightweight Python web framework. It connects the frontend with the ML model and manages data flow.

- **Flask-API**
  Handles requests from the frontend and processes video frames for prediction.

- **Flask-CORS**
  Used to manage cross-origin requests between the frontend and backend.

How Processing is done:

1. Real-Time Data Collection (Video Frames)

We didn't use any ready-made datasets. Instead, we captured live video directly from the webcam using OpenCV. The system continuously takes frame-by-frame input and analyzes the driver's face in real-time.

This live feed acts like our "dataset" — but it's processed instantly, not stored.

2. Preprocessing the Video Frames

Before feeding frames to the model, we cleaned and prepared them:

- Grayscale Conversion – Video frames are converted to grayscale to reduce computation.
- Face Detection – Using dlib, we detect the face area and extract facial landmarks like eyes and mouth.
- Normalization – Landmarks are normalized so the model can understand patterns regardless of the driver's position or size.
- Frame Sequencing – Multiple frames (about 20-30) are grouped as one input sequence for the LSTM part of the model.
  This helps the model understand motion over time (like eye blinking speed)

3. Feature Extraction (Facial Landmarks)

We used dlib's 68-point facial landmark detector to get key features:

- Eye aspect ratio (EAR)
- Mouth opening ratio (for yawns)
- Head tilt

These values are treated as our input features — instead of using raw pixels.

4. Data Feeding to CNN-LSTM Model

Once we have the landmark data from several frames, we feed it to our **CNN-LSTM hybrid model**:

- **CNN** part analyzes spatial features like the shape and position of eyes/mouth.
- **LSTM** part tracks changes over time (like how long the eyes remain closed).

This combo helps detect both **instant drowsiness** and **gradual sleepiness**.

5. Drowsiness Detection Output

The model returns either:

- 0 (Alert) — if the driver looks fine
- 1 (Drowsy) — if signs of fatigue are found

This output is sent back to the frontend using Flask, and the frontend responds by showing a visual warning and playing a sound alert using JavaScript or Node.js modules (like howler.js or play-sound).

6. Visualization and Alerts

- **Live Feedback** – The Electron frontend shows a red warning when drowsiness is detected.
- **Audio Alerts** – A buzzer sound is played immediately using Node.js or Python's playsound if needed.
- **Optional Logging** – Drowsy events can be logged locally for later review.

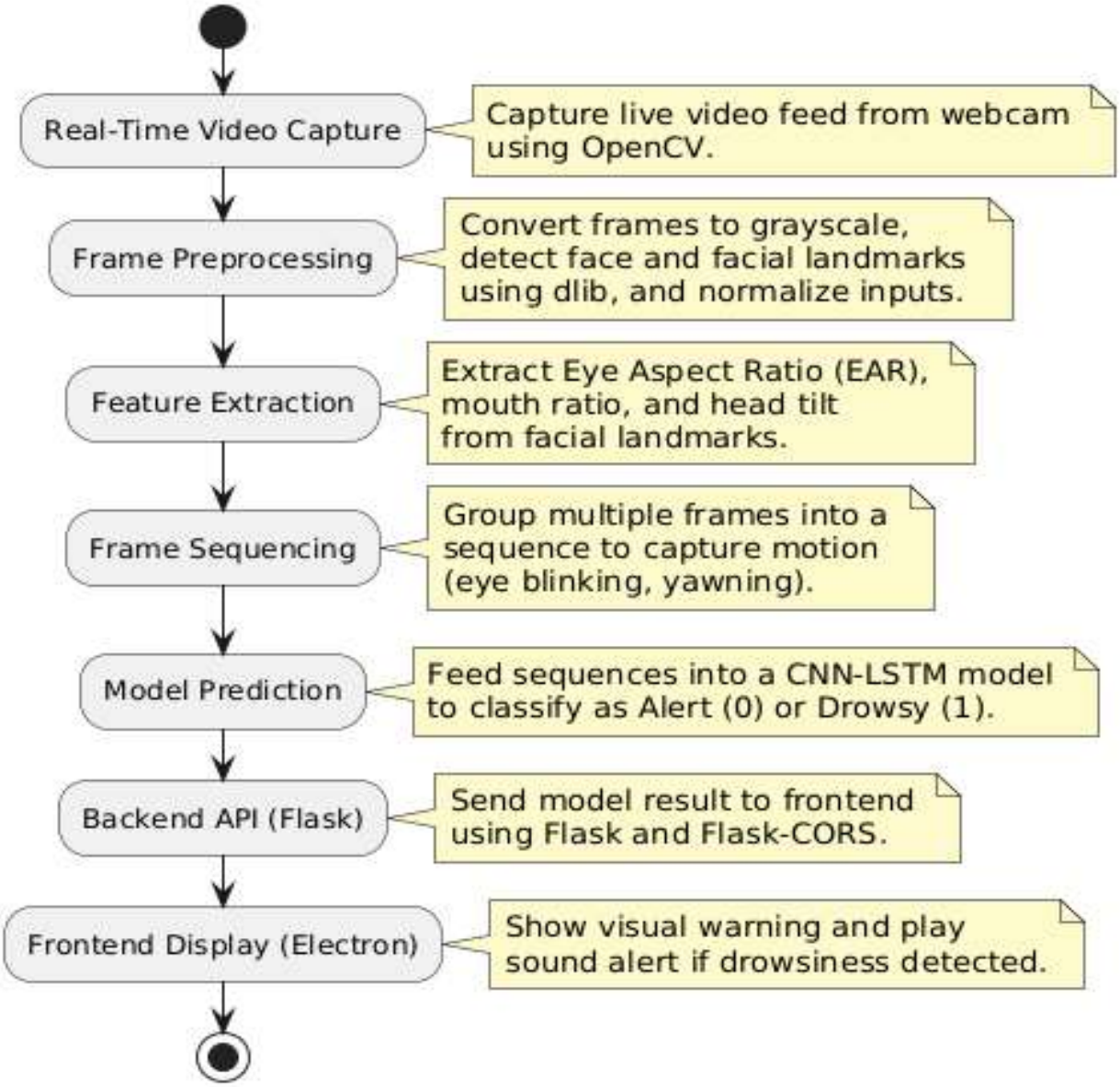


Figure4.4: Processing

4.2 Tools Used

In our project AutoWake, we used a bunch of different tools and technologies to bring everything together. Each tool played an important role in making sure the system works smoothly from detecting drowsiness to showing alerts on the screen. Here's a simple breakdown:

1. Python

Python is the main language we used for all the backend logic. It helped us in:

- Handling image frames captured from the webcam.
- Detecting facial features using libraries like OpenCV and dlib.
- Running the machine learning model that checks if the driver is sleepy. Python is simple to write and has tons of useful libraries, which made our development faster and easier

2. JavaScript + HTML + CSS

We used JavaScript along with HTML5 and CSS3 to handle the frontend part of the project. It gave our app a clean and interactive look.

3. React JS with Vite

React.js (with Vite for faster development) was used to build the user interface of the app. We created reusable components for different screens like home, detection page, and alerts. React made it easier to manage the app's state and update the UI whenever drowsiness is detected.

4. Electron JS

Since we wanted to make a desktop application, we used **Electron JS** to wrap our React frontend and turn it into a native desktop app. This way, the app runs independently without needing a browser.

5. Flask + Flask-CORS

For the backend API, we used **Flask** (a lightweight Python web framework). It helped us:

- Connect the ML model to the frontend.
- Send real-time detection results (like drowsy or not) to the Electron-based UI. We also used **Flask-CORS** to allow smooth communication between frontend and backend.

6. OpenCV, dlib, NumPy

These are the main libraries for video processing and facial detection:

- **OpenCV**: For capturing and processing video frames.
- **dlib**: For detecting facial landmarks like eyes and mouth.
- **NumPy**: For numerical operations on image arrays and data.

7. CNN + LSTM (Machine Learning)

We trained a **CNN-LSTM** model to detect signs of drowsiness from facial features:

- **CNN (Convolutional Neural Network)**: For extracting features from face images.
- **LSTM (Long Short-Term Memory)**: For understanding patterns over time (like blinking and yawning sequences).

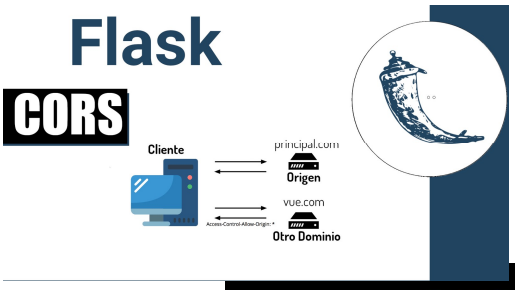
This combo helped us detect drowsiness in a smarter and more reliable way.

8. Node.js

Node.js was used as part of the development environment, especially for running the Vite server and handling package installations (npm).

9. Git (Version Control)

Throughout the project, we used **Git** to track changes in our code. It helped us manage versions, back up work, and collaborate easily.



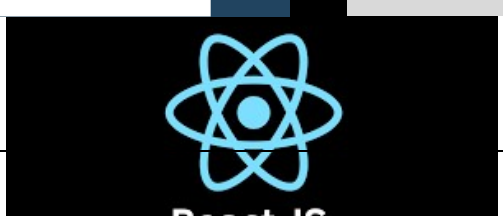


Figure 4.5 – Tool Used (Electron, Node, Flask, Git, React)

4.1 Language Used

The project uses a combination of modern and versatile programming languages to ensure seamless functionality, interactive user experience, and efficient backend operations. Here's a detailed overview:

1. Python

Python was the main language for backend logic and machine learning parts. We used it for:

- Capturing and analyzing real-time video from the webcam.
- Writing our CNN-LSTM model for detecting driver drowsiness.
- Integrating the model with the web server using Flask.

Python's syntax is easy to learn, and it has powerful libraries for both ML and image processing, which made it perfect for our project.

2. JavaScript

JavaScript was mainly used for building the frontend of the application. It helped us:

- Create an interactive user interface using React.js.
- Handle user actions like starting the detection process.
- Communicate with the backend via API calls.

JavaScript made it easy to add interactivity and manage real-time data updates in the UI.

3. HTML5

HTML was used to define the structure of the frontend layout. It helped us create different sections of the user interface and embed visual components easily.

4. CSS3

CSS was used to style the frontend and give it a clean, modern look. We used animations, layouts, and responsiveness to improve the user experience.

5. JSX (JavaScript XML)

Since we used React, JSX was automatically part of our coding. It allowed us to write HTML-like code directly inside JavaScript functions, which made UI development much easier and cleaner.

Figure 4.6 – Languages Used (CSS , JS ,Python)

4.4 Testing

Testing ensures that the system performs as expected. The functionality and reliability of each tool were tested using various test cases.

4.4.1 Testing Approach

To ensure that the AutoWake platform performs accurately and smoothly for users, we'll follow a structured testing approach. Here's the plan:

1. **Unit Testing:**

- **What it is:** We tested each part of the code separately. For example, we checked whether the webcam works, whether the face is detected, and whether the eye-closing detection works as expected.
- **Why it's important:** This helps us catch small issues early before we combine everything together.

2. **Integration Testing:**

- **What it is:** We checked how all modules work together — like how camera input flows to the ML model and then triggers an alert if needed.
- **Why it's important:** It makes sure all the parts (camera → detection → alert) are well-connected and the full system works smoothly.

3. User Testing:

- **What it is:** We sat in front of the system and acted like a driver — blinking slowly, yawning, or tilting our heads — to see how well AutoWake responds.
- **Why it's important:** This helped us understand how the system behaves in real conditions and if it's accurate at catching signs of sleepiness.

4. Performance Testing:

- **What it is:** We checked how fast the system responds when the webcam is running and the model is analyzing facial expressions in real time.
- **Why it's important:** Since drivers need alerts instantly, the system must work quickly and not lag.

5. False Detection Testing

- **What it is:**
  We tested scenarios where the driver is not sleepy (e.g., wearing glasses, talking, or looking aside) to make sure the system doesn't give false alarms.
- **Why it's important:**
  A good system should avoid unnecessary alerts and only trigger when there's real drowsiness.

6. Bug Fixing and Retesting:

- **What it is:** After discovering any bugs, we'll fix them and retest the full workflow — from user input to report download — to make sure everything works correctly.
- **Why it's important:** Fixes don't cause new issues, and the overall system remains stable.

4.4.2 Test Cases

Test Case 1: Detecting Eye Closure:

Test Case ID	TC001
Test Case	Check if the system detects when the driver closes their eyes for a long time.
Test Procedure	Sit in front of the camera and close your eyes for more than 2 seconds. Observe if the alert is triggered.
Expected Result	System should identify drowsiness and trigger a sound or visual alert.
Actual Result	Alert was successfully triggered after eyes were closed for a while.
Status	Pass

Test Case 2: Yawning Detection:

Test Case ID	TC002
Test Case	Check if the system detects a yawn and gives an alert.
Test Procedure	Look at the camera and perform a visible yawn. Watch for alert trigger.
Expected Result	System should recognize yawning and alert the driver.
Actual Result	System detected the yawn and gave an alert.
Status	Pass

Test Case Outputs

Test Case Output 1: Detecting Eye Closure:

Figure 4.7 – Detecting Eye Closure

Test Case Output 2: Yawning Detection

Figure 4.8 – Yawning Detection

Chapter 5.Conclusion

5.1 Conclusion

To sum up, our project **AutoWake** was designed to detect driver drowsiness using a non-intrusive method. Instead of asking the driver to wear any device or sensor, we simply used a camera to observe their face and detect signs like eye closure, yawning, and head tilting. This makes the system more comfortable and user-friendly.

By using computer vision and basic machine learning techniques, we built a working prototype that can alert a sleepy driver in real-time. This can help reduce accidents caused by drowsiness and improve road safety.

Though our system works well in basic scenarios, there's still room for improvement. But overall, this project helped us learn how AI and tech can be used for real-world safety problems.

5.2 Limitations of the Work

1. **Lighting Conditions:**
   - The camera-based system doesn't work well in low-light or very bright conditions. If the driver's face is not clearly visible, it may not detect drowsiness properly.
2. **Single Face Detection:**
   - Our system works best when only one person (the driver) is in front of the camera. It may get confused if there are multiple faces in the frame.
3. **Basic Facial Expressions Only:**

- We focused on basic signs like eye closure and yawning. Other advanced signals like blinking speed or facial muscle fatigue are not considered yet.

4. **Limited Dataset:**

- The accuracy of detection depends on the dataset used to train the model. We used a small sample set, so it might not perform well for all types of faces or behaviors.

5. **No Real-Time Adaptation:**

- The system doesn't learn from the driver over time. It treats every session as new and doesn't adjust based on past behavior.

5.3 Suggestion and Recommendations for Future Work

1. **Improve Lighting Adaptability**

- One of the main challenges in detecting drowsiness is poor lighting, especially during night drives. In future upgrades, we can use **infrared (IR) cameras** or **night vision** technology to capture the driver's face clearly even in low-light or dark environments. This would make the system more reliable and work 24/7 regardless of the time of day or weather conditions.

2. **More Facial Features for Detection**

- Currently, our system focuses on basic facial movements like eye closure and yawning. But for better accuracy, we can add detection for more detailed signs such as **blinking rate, duration of eye closure, mouth movement speed**, and **head tilting or nodding patterns**. These additional cues can help identify drowsiness earlier and reduce false alerts.

3. **Use a Bigger and More Diverse Dataset**

- Our model was trained on a limited dataset which might not cover all types of faces or expressions. In the future, we should **train on a larger dataset** that includes people of different **genders, age groups, skin tones, facial hair**, and expressions, recorded under various **lighting and camera angles**. This would make the system more accurate and generalizable for real-world

use.

4. **Add Real-Time Learning Capabilities**

- Each driver is different. Some may naturally blink slower or faster. If the system could **learn and adapt to the regular behavior of a specific driver**, it could detect drowsiness more accurately by recognizing sudden changes. This kind of real-time learning or **personalization** would reduce false positives and improve user experience.

5. **Mobile App or Embedded System Version**

- Right now, the system runs on a laptop/PC. In the future, we can develop a **mobile app** or deploy the system on small devices like a **Raspberry Pi** or **Jetson Nano**. This would make AutoWake compact, affordable, and easy to integrate directly into cars without needing a full computer setup.

6. **Advanced Alert System with Voice and Vibration**

- At present, the alert is just visual or basic audio. For better effectiveness, we can include **voice alerts** like "Please stay awake!" or **seat vibration alerts**, which can physically jolt the driver awake. These kinds of multi-modal alerts are more effective in grabbing attention, especially in emergency situations.

Bibliography

- **Dong, Y., Hu, Z., Uchimura, K., & Murayama, N. (2011).** Driver inattention monitoring system for intelligent vehicles: A review. *IEEE Transactions on Intelligent Transportation Systems*,**12**(2),596–614.

  ➤ This paper gave us an overview of how driver inattention can be detected using camera-based systems. It helped us understand the background of non-intrusive monitoring, which is the base idea behind AutoWake.
- **Sahayadhas, A., Sundaraj, K., & Murugappan, M. (2012).** Detecting driver drowsiness based on sensors: A review. *Sensors*, **12**(12), 16937–16953.

  ➤ Although this research focused more on sensor-based methods like EEG and heart rate monitors, it helped us compare why our camera-only method is more comfortable and easy to use for drivers.
- **Abtahi, S., Omidyeganeh, M., Shirmohammadi, S., & Hariri, B. (2014).** YawDD: A yawning detection dataset. *Proceedings of the 5th ACM Multimedia Systems Conference*, 24–28.

  ➤ This paper introduced a public dataset for yawning detection using facial features. It supported us in understanding how to collect and label image data for training our model to recognize drowsy behavior like yawning.
- **Lv, C., Zhang, J., Liu, Y., & Han, Y. (2020).** Driver drowsiness detection based on deep learning and facial landmarks. *Sensors*, **20**(2), 328.

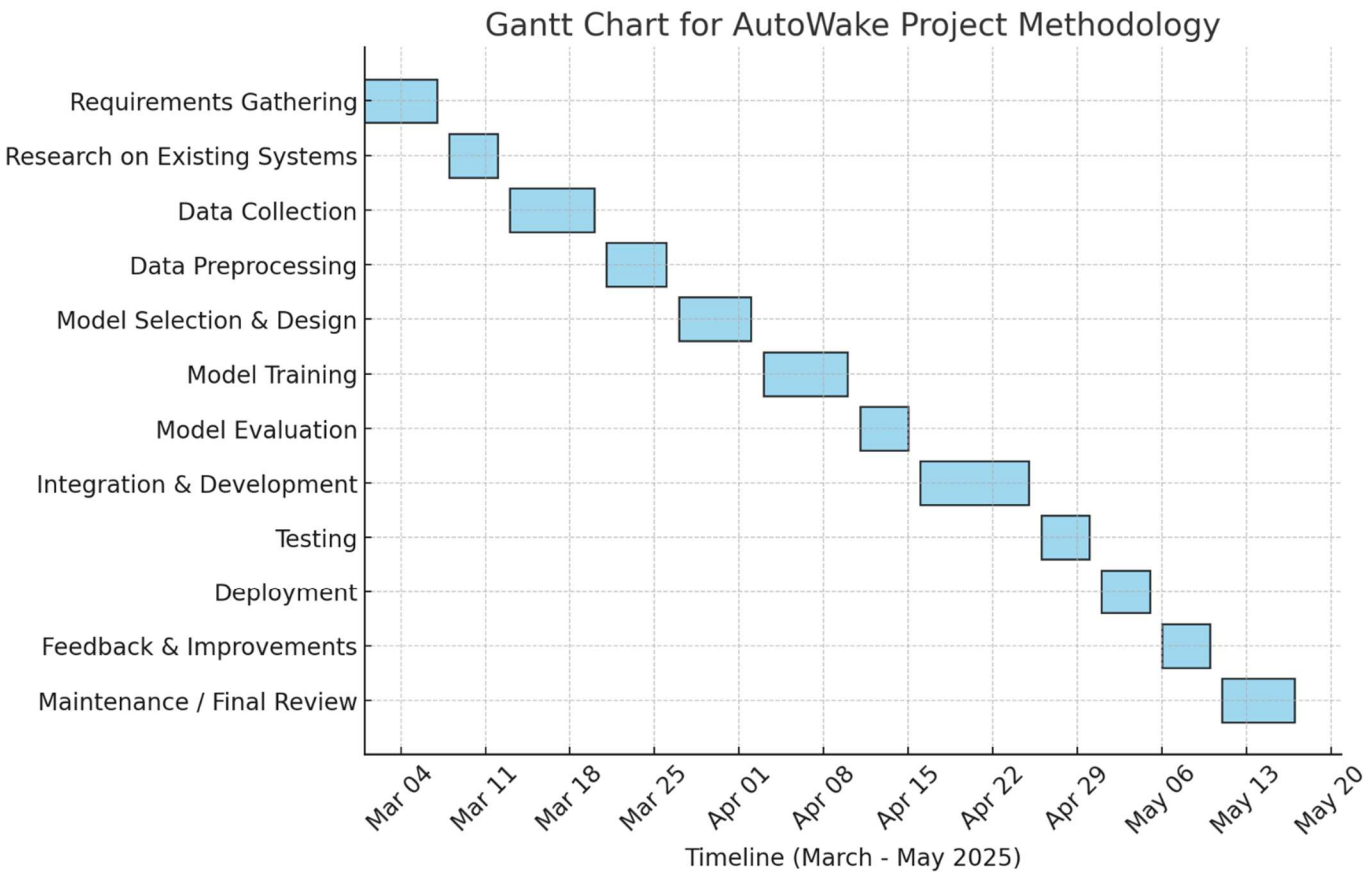
  ➤ This study used deep learning to detect fatigue based on facial expressions like eye closure and blinking rate, which is very similar to our approach in AutoWake.
- **Khunpisuth, P., Choksuriwong, A., & Rattanabannakit, C. (2016).** Driver drowsiness detection using eye-closing ratio. *2016 12th International Conference*

on Signal-Image Technology & Internet-Based Systems (SITIS), 661–666.

➤ We referred to this paper while designing the eye-aspect ratio part of our system. It helped us in calculating how much the eyes need to be closed before classifying it as a drowsy state.

Project Plan

Gantt Chart



Guide Interaction Sheet

Date	Discussion	Action Plan
13/09/2024	Discussed about title of the project.	AutoWake: driver drowsiness detection system was decided as title of the project.
20/09/2024	Discussion on the technology to be for implementing the project.	Finalized **React + Vite + Electron JS** for frontend and **Flask (Python)** for backend.
27/09/2024	Discussion of the creation of synopsis of the project.	Gathering of information for synopsis creation.
04/10/2024	Synopsis was created successfully for the project.	Synopsis was submitted successfully.
18/10/2024	Discussed on presentation to be given of the project.	Presentation was prepared successfully using MS Powerpoint.
25/10/2024	Synopsis Presentation to be given.	Presentation given successfully.
08/10/2024	Discussion on the implementation of the project.	Chose **OpenCV, dlib, NumPy, CNN, LSTM** for drowsiness detection.
15/10/2024	Design phase started (UML, ER Diagram, DFD, Use Case, etc.).	Design diagrams prepared successfully and submitted.
22/11/2024	Implementation Demo and report discussion	Showcased how the face is detected and model gives output using camera feed.
29/11/2024	Discussion On Report of the project	Report containing details of the project was prepared.
30/11/2024	Discussion on Research paper of project and its publishing.	Research paper prepared and published successfully.
06/12/2024	Discussion on Video and Technical Poster.	Video and Technical Poster prepared and shown successfully.
14/12/2024	Discussion on Final Project Submission including presentation, implementation and report.	Submitted final code, report, video, and presentation successfully.

Source Code

Model Implementation:

```
import os import cv2
import numpy as np
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense,Dropout from tensorflow.keras.utils import
to_categorical
from sklearn.model_selection import train_test_split

def load_images(folder, label, img_size=(64, 64)): images = []
labels = []

if not os.path.exists(folder):
raise FileNotFoundError(f"Directory {folder} does not exist.") for filename in os.listdir(folder):
if filename.endswith((".jpg", ".png")): # Ensure valid image files img_path = os.path.join(folder, filename)
img = cv2.imread(img_path) if img is not None:
img = cv2.resize(img, img_size) images.append(img) labels.append(label)
if not images:
raise ValueError(f"No valid images found in {folder}.") return images, labels
os.makedirs("models", exist_ok=True)

base_path = "../datasets/" try:
eye_open_images, eye_open_labels = load_images(base_path + "eyes/open/", 0) # 0 = open eye_closed_images,
eye_closed_labels = load_images(base_path + "eyes/closed/", 1) # 1 = closed mouth_yawn_images, mouth_yawn_labels =
load_images(base_path + "mouth/yawn/", 1) # 1 = yawning mouth_not_yawn_images, mouth_not_yawn_labels =
load_images(base_path + "mouth/no_yawn/", 0) # 0 = not
yawning
except Exception as e:
print(f"Error loading images: {e}") exit()

eye_images = np.array(eye_open_images + eye_closed_images) eye_labels = np.array(eye_open_labels +
eye_closed_labels) mouth_images = np.array(mouth_yawn_images + mouth_not_yawn_images) mouth_labels =
np.array(mouth_yawn_labels + mouth_not_yawn_labels)

eye_images = eye_images / 255.0 mouth_images = mouth_images / 255.0

eye_labels = to_categorical(eye_labels, 2) mouth_labels = to_categorical(mouth_labels, 2)

eye_X_train, eye_X_val, eye_y_train, eye_y_val = train_test_split( eye_images, eye_labels, test_size=0.2,
random_state=42
)
mouth_X_train, mouth_X_val, mouth_y_train, mouth_y_val = train_test_split( mouth_images, mouth_labels,
test_size=0.2, random_state=42
)

def create_model(input_shape=(64, 64, 3)): model = Sequential([
Conv2D(32, (3, 3), activation='relu', input_shape=input_shape),
MaxPooling2D((2, 2)),
Conv2D(64, (3, 3), activation='relu'),
MaxPooling2D((2, 2)),
Conv2D(128, (3, 3), activation='relu'),
MaxPooling2D((2, 2)), Flatten(),
Dense(128, activation='relu'), Dropout(0.5),
Dense(2, activation='softmax') # 2 classes
])
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy']) return model

eye_model = create_model()
```

```
eye_model.fit(eye_X_train, eye_y_train, epochs=10, batch_size=32, validation_data=(eye_X_val, eye_y_val))
eye_model.save("models/eye_model.h5")

mouth_model = create_model()
mouth_model.fit(mouth_X_train, mouth_y_train, epochs=10, batch_size=32, validation_data=(mouth_X_val,
mouth_y_val))
mouth_model.save("models/mouth_model.h5")

eye_loss, eye_accuracy = eye_model.evaluate(eye_X_val, eye_y_val, verbose=0) print(f"Eye Model Accuracy on
Validation Set: {eye_accuracy * 100:.2f}%")

mouth_loss, mouth_accuracy = mouth_model.evaluate(mouth_X_val, mouth_y_val, verbose=0) print(f"Mouth Model
Accuracy on Validation Set: {mouth_accuracy * 100:.2f}%")

print("Models trained and saved!")
```

```
import os
import cv2 import dlib
import numpy as np import time
from tensorflow.keras.models import load_model


class DrowsinessDetector:
def  init (self):        base_dir


=os.path.dirname(os.path.abspath( file ))

predictor_path = os.path.join(base_dir, "models", "shape_predictor_68_face_landmarks.dat")

eye_model_path = os.path.join(base_dir, "models", "eye_model.h5") mouth_model_path = os.path.join(base_dir,
"models", "mouth_model.h5") if not os.path.exists(predictor_path):
raise FileNotFoundError(f"Shape predictor file not found at {predictor_path}") if not os.path.exists(eye_model_path):
raise FileNotFoundError(f"Eye model file not found at {eye_model_path}") if not os.path.exists(mouth_model_path):
raise FileNotFoundError(f"Mouth model file not found at {mouth_model_path}")

self.detector = dlib.get_frontal_face_detector() self.predictor = dlib.shape_predictor(predictor_path)

self.eye_model = load_model(eye_model_path) self.mouth_model = load_model(mouth_model_path)

self.blink_start_time = None self.blink_duration_threshold = 0.6 # in seconds self.blink_count = 0
self.blink_threshold_count = 3 # blinks to trigger drowsiness

self.yawn_count = 0
self.yawn_threshold_count = 3 # yawns to trigger drowsiness self.last_yawn_time = None # Time when the last yawn was
detected

self.drowsy_end_time = 0 # Time when drowsy state ends self.drowsy_duration = 5 # Seconds to keep drowsy state after
detection

self.img_size = (64, 64)

def reset(self): self.blink_start_time = None self.blink_count = 0
self.yawn_count = 0 self.last_yawn_time = None self.drowsy_end_time = 0




def preprocess_region(self, region):
region = cv2.resize(region, self.img_size)

region = region / 255.0
return np.expand_dims(region, axis=0)

def get_bounding_box(self, points, frame): x_coords, y_coords = zip(*points)
x_min, x_max = min(x_coords), max(x_coords) y_min, y_max = min(y_coords), max(y_coords) padding = 10
x_min = max(0, x_min - padding) y_min = max(0, y_min - padding)
x_max = min(frame.shape[1], x_max + padding) y_max = min(frame.shape[0], y_max + padding) return (x_min, y_min,
x_max, y_max)

def detect_drowsiness(self, frame):
gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY) faces = self.detector(gray)

if len(faces) == 0: return False, frame

current_time = time.time()
```

```
for face in faces:
landmarks = self.predictor(gray, face)
left_eye_pts = [(landmarks.part(i).x, landmarks.part(i).y) for i in range(36, 42)] right_eye_pts = [(landmarks.part(i).x,
landmarks.part(i).y) for i in range(42, 48)] mouth_pts = [(landmarks.part(i).x, landmarks.part(i).y) for i in range(48, 68)]
left_eye_coords = self.get_bounding_box(left_eye_pts, frame)
right_eye_coords = self.get_bounding_box(right_eye_pts, frame) mouth_coords = self.get_bounding_box(mouth_pts,
frame) left_eye_region = frame[left_eye_coords[1]:left_eye_coords[3],
left_eye_coords[0]:left_eye_coords[2]]
right_eye_region = frame[right_eye_coords[1]:right_eye_coords[3], right_eye_coords[0]:right_eye_coords[2]]
mouth_region = frame[mouth_coords[1]:mouth_coords[3], mouth_coords[0]:mouth_coords[2]] if left_eye_region.size ==
0 or right_eye_region.size == 0 or mouth_region.size == 0:
continue
left_eye_input = self.preprocess_region(left_eye_region) right_eye_input = self.preprocess_region(right_eye_region)
mouth_input = self.preprocess_region(mouth_region)

left_eye_pred = self.eye_model.predict(left_eye_input) right_eye_pred = self.eye_model.predict(right_eye_input)
mouth_pred = self.mouth_model.predict(mouth_input)

left_eye_closed = np.argmax(left_eye_pred) == 1 right_eye_closed = np.argmax(right_eye_pred) == 1 is_yawning =
np.argmax(mouth_pred) == 1
# Blink Detection

if left_eye_closed and right_eye_closed: if self.blink_start_time is None:
self.blink_start_time = current_time else:
blink_duration = current_time - self.blink_start_time if blink_duration >= self.blink_duration_threshold:
self.blink_count += 1
print(f"[Blink] Duration: {blink_duration:.2f}s, Count:

{self.blink_count}")

else:


self.blink_start_time = None

self.blink_start_time = None

# Yawn Detection
if is_yawning and mouth_region.size > 0: if self.last_yawn_time is None:
self.last_yawn_time = current_time else:
yawn_duration = current_time - self.last_yawn_time if yawn_duration >= 2:
self.yawn_count += 1
print(f"[Yawn] Duration: {yawn_duration:.2f}s, Count: {self.yawn_count}") self.last_yawn_time = None
else:
self.last_yawn_time = None

# Check if thresholds are met
if self.blink_count >= self.blink_threshold_count or self.yawn_count >= self.yawn_threshold_count:
self.drowsy_end_time = current_time + self.drowsy_duration self.blink_count = 0
self.yawn_count = 0
print("Drowsiness detected. Setting drowsy state for 5 seconds.")

# Determine if currently drowsy based on timer is_drowsy = current_time < self.drowsy_end_time


# Draw rectangles
cv2.rectangle(frame, (left_eye_coords[0], left_eye_coords[1]), (left_eye_coords[2], left_eye_coords[3]), (0, 255, 0), 2)
cv2.rectangle(frame, (right_eye_coords[0], right_eye_coords[1]), (right_eye_coords[2], right_eye_coords[3]), (0, 255, 0),
2)
cv2.rectangle(frame, (mouth_coords[0], mouth_coords[1]), (mouth_coords[2], mouth_coords[3]), (0, 255, 0), 2)
return is_drowsy, frame
```