OPTIMIZATION OF DROWSINESS DETECTION SYSTEM USING DEEP LEARNING ON RASPBERRY PI



Written By:

N a m e : Aura Zahra Salsabila

SID : 22523042

INFORMATICS STUDY PROGRAM – BACHELOR'S PROGRAM

FACULTY OF INDUSTRIAL TECHNOLOGY

ISLAMIC UNIVERSITY OF INDONESIA

2026

SUPERVISOR ENDORSEMENT PAGE

OPTIMIZATION OF DROWSINESS DETECTION SYSTEM USING DEEP LEARNING ON RASPBERRY PI

THESIS

Written By:

N a m e : Aura Zahra Salsabila
SID : 22523042

Yogyakarta, August 3rd, 2026

Advisor,

(Chandra Kusuma Dewa, S.Kom., M.Cs., Ph.D.)

EXAMINER ENDORSEMENT PAGE

OPTIMIZATION OF DROWSINESS DETECTION SYSTEM USING DEEP LEARNING ON RASPBERRY PI

THESIS

Has been defended in front of the examiners as one of the requirements to obtain a Bachelor of Informatics degree from the Undergraduate Program in Informatics at the Faculty of Industrial Technology, Universitas Islam Indonesia

Yogyakarta, August 3rd, 2026

Chair

Chandra Kusuma Dewa, S.Kom., M.Cs., Ph.D.

Examiner 1

Irving Vitra Paputungan, S.T., M.Sc., Ph.D.

31/08/2026
Signer ID: 3WTLP9FL19...

Examiner 2

Kurniawan Dwi Irianto, S.T., M.Sc.

Acknowledged by,

Head of the Informatics Study Program – Undergraduate Program

Faculty of Industrial Technology

Islamic University of Indonesia

(Ir. Dhomas Hatta Fudholi, S.T., M.En., Ph.D..)

AUTHENCITY STATEMENT

The Undersigned :

Name : Aura Zahra Salsabila

SID : 22523042

Final project with title:

OPTIMIZATION OF DROWSINESS DETECTION SYSTEM USING DEEP LEARNING ON RASPBERRY PI

Stating that all components and contents in this final project are my own work. If in the future it is proven that some parts of this work are not my own, this final project submitted as my own work is ready to be withdrawn and I am ready to bear any risks and consequences.

This statement letter is made as such, hopefully it can be used appropriately.

Yogyakarta, August 3rd, 2026



(Aura Zahra Salsabila)

DEDICATION

The completion of this thesis could not have been achieved without the involvement, support, prayers, and assistance – both moral and material – of many people. The author therefore expresses the deepest gratitude to all those who contributed to the completion of this thesis.

This thesis is dedicated to:

1. With humble gratitude to Allah SWT, whose love and affection have given me strength and equipped me with knowledge. Through Your grace and the ease You have bestowed, this thesis has been completed smoothly and well.
2. Mr. Chandra Kusuma Dewa, S.Kom., M.Cs., Ph.D., as my thesis supervisor. Thank you for your patience, time, knowledge, guidance, and motivation throughout the writing and preparation of this thesis. The author sincerely apologize for any actions or words that may have been inappropriate during the supervision process.
3. Mr. Irving Vitra Paputungan, S.T., M.Sc., Ph.D. and Mr. Kurniawan Dwi Irianto, S.T., M.Sc. as the examination committee.
4. Dr. Nur Wijayaning Rahayu, S.Kom., M.Cs., as academic advisor, and Ir. Dhomas Hatta Fudholi, S.T., M.En., Ph.D., as Head of the Informatics Departement.
5. All staff and employees of the Informatics Departement.
6. My first love and the gateway to my heaven – Ayah Arif Sofyan Kusuma Diarto and Mamah Sri Lestari. Thank you for your prayers and unwavering support. For the love, affection, prayers, and sacrifices that have accompanied every step of this educational journey. Thank you for bringing me to this point. Thank you for fighting for me, for raising me, and for nurturing me until I could earn my degree. May Allah SWT always protect you both, long enough to witness every success I achieve through my own choices. Please live a long life.
7. To my sibling, whose presence means no less – Azriel Sora Maulana. Thank you for being part of the author's life journey, for your support, encouragement, and for always bringing laughter with you endearing antics.
8. To the holder of Student ID 22522359 – Syahreza Hikmal Maulia. Thank you for being one of the people who has been part of this journey. Although you were not there from the very beginning of my thesis journey, your presence throughout the process has still meant so much to me. Thank you for all the support, encouragement, patience, time, and care you have given me. Thank you for being someone I could talk to whenever I

felt tired, for listening to all my worries and complaints, and for encouraging me whenever this process felt difficult. You may not have witnessed everything I went through from the beginning, but your presence and support during the final part of this journey have been one of the reasons I was able to make it this far.

9. To the author's grandparents, Mr. Sunardi S.H., and Mrs. Kusmiyati. Thank you for every prayer and for the support – both material and moral – and for always encouraging the author to complete this education and thesis.
10. To the author's aunts and uncles – Mr. Arif Setiawan Kusuma, S.Pd., Mrs. Putri Salisa Bintari Isma Mukaromah, S.H., Mr. Muhammad Redho Teguh, S.H., Mr. Arif Syaquvi Kusuma Wardani, S.H., and Isna Afia Miliani – who have always offered guidance and support to the author.
11. To the author's brothers – Denies Ade Syamboja, Adi Kurniawan, Hilman Herlian Putra, Muhammad Dava Fernanda, and Irfan Sakti Kurniawan. Though there is no blood tie between us, thank you for always listening, for your support, and for the motivation you have given the author.
12. To the fellow warriors and family of the Motekar 17 group – Zea, Jeha, Abi, Raja, Ferrel, Maylia, Riffa, Novera, and Irul. Thank you for your presence, for always accompanying the author through both joy and hardship.
13. To my childhood friends – Zeli, Rahma, Zora, Ghasa, Icha, Yesa, Inu, Idzar, and Anjar. Thank you for always being there to listen to the author's worries and for supporting the author in completing this thesis.
14. To my dearest friends – Dea, Echa, and Hilda. Thank you for the friendship that has grown and remained strong through the years. Your presence, laughter, and unwavering support have been a true source of comfort and strength for the author throughout this journey. Thank you for always being there in every moment, both the difficult and the joyful ones.
15. To my high school friends – Laluna, Maura, and Nisa. Thank you for the memories and the friendship that began long before this thesis ever started. Even across the distance and the different paths we have each taken, your support and warn words have always found their way to the author and kept the spirit alive..
16. To my thesis companions – Abang Rahman, Ikbar, Fazli, Aura, Ikip - thank you for being there through every late night, every moment of doubt, and every small victory along the way. The journey was long and exhausting, but having you by my side made it so much more bearable. Thank you for the discussions, the laughter, and the

unwavering support. May all our hard work pay off, and may we all graduate together. Keep going – we are almost there!

17. Last but not least, I wanna thank I wanna thank me for believing in me. I wanna thank me for doing all this hard work. I wanna thank me for having no days off. I wanna thank me for never quitting. I wanna thank me for always being a giver. And tryna give more than I receive. I wanna thank me for tryna do more right than wrong. I wanna thank me for just being me at all times.

MOTTO

“Take a risk or lose the chance.”

-Anonymous

"Success is the sum of small efforts, repeated day in and day out."

-Robert Collier

إِ ن مَعَ ٱلۡعُسۡرِ يُسۡرًا ٦

“ Indeed, with that hardship (will be) ease.”

-QS Al-Insyirah: 6

لَ يُكَلِّفُ ٱ للَّهُ نَفۡسًا إِ ل وُسۡعَهَا ۚ لَهَا مَا كَسَبَتۡ وَعَلَيۡهَا مَا ٱكۡتَسَبَتۡ ۗ رَبـنَا لَ تُؤَاخِذۡنَآ إِن نسِينَآ أَوۡ أَخۡطَأۡنَا ۚ رَبـنَا وَلَ تَحۡمِلۡ عَلَيۡنَآ إِصۡرًا كَمَا حَمَلۡتَهُۥ
عَلَى ٱلَّذِينَ مِن قَبۡلِنَا ۚ رَبـنَا وَلَ تُحَ ِملۡنَا مَا لَ طَاقَةَ لَنَا بِهِۦۖ وَٱعۡفُ عَنَّا وَٱغۡفِرۡ لَنَا وَٱرۡحَمۡنَآ ۚ أَنتَ
مَوۡلَىٰنَا فَٱنصُرۡنَا عَلَى ٱلۡقَوۡمِ ٱلۡكَٰفِرِينَ ٢٨٦

“Allah does not require of any soul more than what it can afford. All good will be for its own benefit, and all evil will be to its own loss. ˹The believers pray,˺ “Our Lord! Do not punish us if we forget or make a mistake. Our Lord! Do not place a burden on us like the one you placed on those before us. Our Lord! Do not burden us with what we cannot bear. Pardon us, forgive us, and have mercy on us. You are our ˹only˺ Guardian. So grant us victory over the disbelieving people.”

-QS Al-Baqarah: 286

“ Fall seven times, stand up eight.”

-Japanese Proverb

“It always seems impossible until it’s done”

-Nelson Mandela

“Start where you are. Use what you have. Do what you can.”

-Arthur Ashe

FOREWORD

Bismillahirrahmanirrahim,
Assalamu’alaikum Warahmatullahi Wabarakatuh,

All praise and gratitude are due to Allah SWT, whose grace has enabled the author to complete this final project report entitled “ Optimization of Drowsiness Detection System Using Deep Learning on Raspberry Pi” as one of the requirements for completing the undergraduate degree in Informatics at Universitas Islam Indonesia.

This research aims to develop and evaluate a real-time drowsiness detection system integrating CNN and MediaPipe on the Raspberry Pi 5 as an embedded platform. Throughout the process, the author encountered several challenges, particularly in configuring the deployment environment on the Raspberry Pi 5 and ensuring stable hardware integration. These challenges provided valuable learning experiences beyond academic knowledge.

The author recognizes that this report is not without limitations and welcomes constructive feedback for improvement. It is hoped that this research may serve as a useful contribution and reference for future studies in driver safety technology.

Wassalamu’alaikum Warahmatullahi Wabarakatuh.

Yogyakarta, August 3rd, 2026



(Aura Zahra Salsabila)

ABSTRACT

Road traffic accidents caused by driver drowsiness remain a significant public safety concern, particularly in Indonesia where fatigue-related incidents account for a substantial proportion of road fatalities. Early and reliable detection of drowsiness in real-time is therefore critical for preventing accidents, especially in the context of long-distance driving. This research proposes and evaluates a real-time driver drowsiness detection system that integrates a Convolutional Neural Network (CNN) with MediaPipe Face Mesh and MediaPipe Hands, deployed on a Raspberry Pi 5 as an embedded platform suitable for in-vehicle use.

The systems processes video frames through a sequential pipeline: CLAHE preprocessing, MediaPipe Face Mesh for facial landmark extraction, CNN classification of the face region into normal, eyes_closed, or yawn states, and conditional MediaPipe Hands invocation for yawn_hand_cover detection. Drowsiness is triggered when yawning episodes exceed five per 60-second window, activating a physical buzzer alarm via GPIO.

System performance was assessed through three evaluations: (1) a MediaPipe-only landmark-based evaluation using EAR/MAR thresholds on a self-collected dataset; (2) a CNN classification evaluation on a held-out test set combining the YawDD and self-collected datasets; and (3) an integrated real-time evaluation of the deployed CNN + MediaPipe system on the Raspberry Pi 5, including functional verification and testing on five human subjects. The MediaPipe-only method achieved 93.75% accuracy but requires manual threshold calibration. The CNN-only approach achieved 97.95% accuracy with perfect yawn recall (1.000) but cannot detect yawn_hand_cover. The proposed integrated system addresses both limitations. The CNN model (161,571 parameters) was converted to TensorFlow Lite with float16 quantisation, reducing model size from 1.85 MB to 0.31 MB. The deployed system achieved 15.05 FPS on the Raspberry Pi 5, exceeding the 8 FPS target, and operates autonomously at boot without a monitor or network connection. Real-time testing on five subjects yielded 94.3% overall accuracy across seven scenarios, with core detection scenarios achieving perfect 5/5 accuracy.

Keywords: driver drowsiness detection, CNN, MediaPipe, Raspberry Pi 5, TensorFlow Lite, real-time embedded system, yawning detection.

GLOSSARY

Active Buzzer	an electronic component that produces sound automatically when supplied with voltage, without requiring an external frequency signal.
Accuracy	an evaluation metric that measures the proportion of correct predictions out of the total number test samples.
Adam	an adaptive optimization algorithm used in deep learning model training, combining Momentum and RMSprop methods to efficiently update model weights.
Batch size	the number of data samples processed in one iteration during model training.
Bounding box	a rectangular region used to mark and localize a specific object – in this research, the driver face – within a video frame.
CLAHE	Contrast Limited Adaptive Histogram Equalization, an image preprocessing technique that adaptively enhances local contrast to improve detection reliability under low-light conditions.
EarlyStopping	a regularization callback in model training that automatically halts the training process when the validation metrics shows no improvement for a specified number of epochs.
Epoch	one complete training cycle in which the entire training dataset is processed once.
Episode-based	a drowsiness counting approach that increments the counter each time a transition from a non-yawning to a yawning condition is detected, rather than counting individual frames.
Float-16	a 16-bit floating point number representation used in model quantisation to reduce model size and memory consumption.
Keras	a high-level programming interface for building and training deep learning models, integrated within TensorFlow.
Softmax	an activation function applied at the output layer of a neural network that converts raw values into a proability distribution for multi-class classification.

TABLE OF CONTENTS

TITLE PAGE i
SUPERVISOR ENDORSEMENT PAGE ii
EXAMINER ENDORSEMENT PAGE iii
AUTHENCITY STATEMENT iv
DEDICATION v
MOTTO viii
FOREWORD ix
ABSTRACT x
GLOSSARY xi
TABLE OF CONTENTS xii
LIST OF TABLES xv
LIST OF FIGURES xvi
CHAPTER I INTRODUCTION 1
1.1 Research Background 1
1.2 Problem Formulation 4
1.3 Scope and Limitations 4
1.4 Research Objectives 5
1.5 Research Benefits 5
CHAPTER II LITERATURE REVIEW 7
2.1 Previous Studies 7
2.2 Theoretical Foundation 11
2.2.1 Driver Drowsiness Detection 11
2.2.2 Deep Learning 12
2.2.3 Convolutional Neural Network (CNN) 13
2.2.4 Transfer Learning 13
2.2.5 MediaPipe Face Mesh 14
2.2.6 MediaPipe Hands 15
2.2.7 Facial Landmark Detection 15
2.2.8 Raspberry Pi 16
2.2.9 Image Preprocessing with CLAHE 16
2.2.10 Eye Aspect Ratio (EAR) 17
2.2.11 Mouth Aspect Ratio (MAR) 18
2.2.12 Confusion Matrix 18
2.2.13 Precision, Recall, and F1-Score 19
2.2.14 YawDD Dataset 19
2.2.15 Accuracy Evaluation 20
2.3 Research Gap 21
CHAPTER III RESEARCH METHODOLOGY 24
3.1 System Development Methods 24
3.2 Requirements Analysis 25
3.2.1 Problem Analysis 25
3.2.2 Requirement Analysis 26
3.2.3 Data Requirements 26
3.2.4 Hardware and Software Requirements 27
3.3 System Design 27
3.3.1 CNN Architecture Design 28
3.3.2 Hand-Covering-Mouth Detection Design 29

3.3.3 Drowsiness Detection Logic 29
3.4 Implementation 29
3.5 Testing 30
3.6 Maintenance 30
3.7 Comparative Analysis of Detection Approaches 30
3.7.1 Approach A: MediaPipe Face Mesh with EAR and MAR Thresholds 31
3.7.2 Approach B: CNN (Convolutional Neural Network) Classifier 31
3.7.3 Approach C: Integrated System (CNN + MediaPipe) 32
3.8 Evaluation Metrics 34
3.8.1 Classification Performance Metrics 34
3.8.2 Real-Time Performance Metrics 35
3.8.3 Functional Verification 35
CHAPTER IV IMPLEMENTATION 37
4.1 Implementation Overview 37
4.2 Implementation Environment 37
4.3 Data Preprocessing 38
4.4 CNN Model Training 38
4.5 Real-Time System Implementation 40
4.6 Hardware Assembly 40
4.7 System Deployment to Raspberry Pi 5 41
4.7.1 Model Conversion to TensorFlow Lite 41
4.7.2 Deployment Environment Configuration 42
4.7.3 Automatic System Startup 42
CHAPTER V RESULTS AND DISCUSSION 43
5.1 Overview 43
5.2 Experiment 1: MediaPipe Landmark-Based Detection 43
5.2.1 Experimental Setup 43
5.2.2 Results 43
5.2.3 Discussion 45
5.3 Experiment 2: CNN Classification Model 46
5.3.1 Training Dynamics 46
5.3.2 Test Set Evaluation 47
5.3.3 Discussion 47
5.4 Experiment 3: Integrated CNN + MediaPipe on Raspberry Pi 5 48
5.4.1 Real-Time Performance 48
5.4.2 Functional Verification Results 49
5.4.3 Discussion 50
5.5 Real-Time System Testing on Five Subjects 51
5.5.1 Testing Procedure 51
5.5.2 Subject Profile 52
5.5.3 Testing Results Per Subject Per Scenario 52
5.5.4 Buzzer Testing Results 53
5.5.5 Accuracy Recap Per Subject 53
5.5.6 Screenshot Evidence of Real-Time Testing 54
5.5.7 Discussion 56
5.6 Comparative Analysis of the Three Approaches 57
5.7 Limitations 58
CHAPTER VI CONCLUSION AND RECOMMENDATION 59
6.1 Conclusion 59
6.2 Recommendations 60

REFERENCES 61
APPENDEX 64

LIST OF TABLES

Table 2. 1 Comparison of Previous Studies 11

Table 3. 1 Dataset Class Mapping from YawDD to CNN Target Classes 26
Table 3. 2 Hardware Requirements 27
Table 3. 3 Software Requirements 27
Table 3. 4 CNN Architecture Design 28
Table 3. 5 Key Implementation Parameters 30
Table 3. 6 Key Implementation Parameters 31
Table 3. 7 CNN-Only Approach: Classification Report (438 test images) 32
Table 3. 8 Comparison of Detection Approaches 34
Table 3. 9 Functional Verification Test Scenarios 35

Table 4. 1 Functional Verification Test Scenarios 37
Table 4. 2 CNN Training Configuration 38
Table 4. 3 Final Deployment Configuration on Raspberry Pi 5 42

Table 5. 1 MediaPipe-Only Evaluation Results (80 images, self-collected dataset) 43
Table 5. 2 Misclassification Analysis – MediaPipe-Only Approach 44
Table 5. 3 CNN Classification Report – Test Set (438 test images) 47
Table 5. 4 Real-Time Performance on Raspberry Pi 5 49
Table 5. 5 Functional Verification Results – Integrated 49
Table 5. 6 Suject Profile for Real-Time System Testing 52
Table 5. 7 Testing Results Per Scenario Per Subject 52
Table 5. 8 Testing Results Per Scenario Per Subject 53
Table 5. 9 Real-Time Testing Accuracy Recap Per Subject 53
Table 5. 10 Comparative Analysis: MediaPipe-Only vs 57

LIST OF FIGURES

Figure 3. 1 Waterfall Method.....25
Figure 3. 2 Hardware design: Raspberry Pi 5, Logitech C922 USB webcam, ...28
Figure 3. 3 System flowchart: per-frame processing pipeline of the integrated CNN + MediaPipe drowsiness detection system.....33

Figure 4. 1 Training and validation accuracy and loss curves 39
Figure 4. 2 Hardware Wirring Diagram (GPIO17 – Buzzer – GND).....41
Figure 4. 3 Assembled hardware: Raspberry Pi 5 with.....41

Figure 5. 1 Training and validation accuracy (left) and loss (right) curves. Best validation accurcy: 98.61%; best validation loss: 0.1113 (epoch 47).....46
Figure 5. 2 Confusion matrix on the CNN test set. Left: absolute counts. Right: normalised per class (recall). Misclassifications are concentrated at the eyes_closed-normal boundary.....47
Figure 5. 3 Subject S1 – STATUS DROWSY, BUZZER ON, ...54
Figure 5. 4 Subject S2 – STATUS DROWSY,.....54
Figure 5. 5 Subject S3 – STATUS DROWSY,.....55
Figure 5. 6 Subject S4 – STATUS DROWSY,.....55
Figure 5. 7 Subject S5 – STATUS : NORMAL.....56

CHAPTER I
INTRODUCTION

1.1 Research Background

Traffic accidents remain one of the most critical public health challenges worldwide. According to the World Health Organization (World Health Organization, 2023), approximately 1.35 million people die every year as a result of road traffic crashes, while millions more sustain serious injuries that lead to long-term disabilities. This figure places traffic accidents among the eighth leading cause of death globally and the leading cause of death for individuals aged between 5 and 29 years (World Health Organization, 2023). The scale of this problem demands urgent and sustainable technological interventions capable of addressing its root causes.

In Indonesia, road traffic accidents have demonstrated a particularly alarming trend, especially in the Special Region of Yogyakarta (D.I.Yogyakarta). Data from the Regional Police of D.I. Yogyakarta (Daryati Angeli Sagala, 2025) indicate that the total number of traffic accidents increased significantly from 5,350 cases in 2021 to 7,830 cases in 2022, representing an increase of more than 70 percent within a span of only two years. This rapid escalation underscores the need for more systematic and technology-driven approaches to improve road safety in the region.

Among the many contributing factors to road traffic accidents, driver drowsiness has been consistently identified as one of the most prevalent causes. A drowsy driver experiences cognitive impairment, slower reaction times, reduced situational awareness, and diminished ability to respond appropriately to dynamic road conditions (Ngxande et al., 2017). Ramzan et al. (2019) reported that driver drowsiness accounts for approximately 20 percent of all traffic accidents, particularly on monotonous roads such as highways and expressways. These findings emphasize the critical importance of developing reliable, real-time drowsiness detection systems capable of alerting drivers before a fatigue-related incident occurs.

Three major categories of drowsiness detection approaches have been explored in the literature: physiological signal-based methods, vehicle behavior-based methods, and computer vision-based methods (Albadawi et al., 2022). Among these, computer vision-based approaches using cameras have attracted the greatest research attention due to their non-invasive nature, ease of installation, adaptability across different vehicle types, and potential

for real-time processing without requiring the attachment of any sensors to the driver's body (Souto Maior et al., 2020). These characteristics make vision-based systems particularly well-suited for practical deployment in real-world driving environments.

Prior research by Aprilian and Mahendra (2024) at Universitas Islam Indonesia developed a driver drowsiness detection system utilizing Haar Cascade Classifier for face detection, Dlib 68 Facial Landmarks for facial landmark extraction, Eye Aspect Ratio (EAR) for eye closure measurement, and lip distance for yawning detection. The system was deployed on a Raspberry Pi 4B platform and incorporated a Telegram notification module alongside a buzzer alarm. While the system demonstrated the feasibility of embedded drowsiness detection, several significant limitations were identified that constrain its overall effectiveness.

The first limitation concerns the reliance on traditional, threshold-based computer vision techniques. Haar Cascade Classifier operates on handcrafted features and exhibits limited robustness under challenging conditions such as poor lighting, partial occlusion, and significant head pose variations (Reddy et al., 2017). The threshold-based nature of the approach also means that detection performance is highly sensitive to the manually defined parameters, which may not generalize well across different subjects and environments(Zhang, Y. Ji, 2021). These characteristics inherently limit the system's reliability in real-world operational conditions.

The second limitation is the absence of a mechanism to detect the Yawn Hand Cover condition, a behavioral pattern in which a drowsy driver covers their mouth with a hand while yawning. This behavior is a clinically recognized indicator of driver fatigue (Shroff et al., 2021); however, the previous system lacked any hand detection capability, resulting in false negatives whenever this condition occurred. The inability to recognize this specific drowsiness indicator reduces the comprehensiveness and reliability of the overall detection system.

The third limitation pertains to the lack of rigorous quantitative evaluation. The previous system did not report standardized performance metrics such as accuracy, precision, recall, and F1-score on a per-class basic, which makes it difficult to objectively assess and compare its performance against other systems reported in the literature. This gap in evaluation methodology limits the scientific contribution and reproducibility of the prior work.

To address these identified limitations, this research proposes an optimized driver drowsiness detection system that integrates a Convolutional Neural Network (CNN) as the primary deep learning component with MediaPipe Face Mesh and MediaPipe Hands. CNN was selected as the core classification model due to its well-established capability to automatically extract hierarchical visual features from raw image data without manual feature engineering,

making it significantly more robust to variations in lighting, skin tone, and facial geometry than threshold-based methods (LeCun et al., 2015)(Hinton et al., 2012). Multiple studies have demonstrated the superiority of CNN-based classification over conventional threshold approaches in driver condition recognition tasks (Ahmed et al., 2023)(Zhao et al., 2020).

MediaPipe Face Mesh was adopted to replace the Haar Cascade Classifier and Dlib facial landmark detector due to its ability to detect 478 three-dimensional facial landmarks in real-time with high accuracy across a wide range of lighting conditions and head orientations (Lugaresi et al., 2019)(Kartynnik et al., 2019). Compared to Dlib's 68-point landmark model, MediaPipe Face Mesh offers substantially denser geometric coverage of the face while maintaining lower computational overhead, making it more suitable for deployment on resource-constrained edge devices such as Raspberry Pi (Grishchenko et al., 2020). MediaPipe Hands is additionally integrated into the system to detect hand position relative to the mouth area, enabling the identification of the Yawn Hand Cover condition that was absent from the previous system.

It is important to clarify that MediaPipe Face Mesh and MediaPipe Hands serve as landmark detection and geometric analysis frameworks within this system and are not deep learning classification models themselves. The CNN model is the sole deep learning component responsible for classifying the driver's facial condition into three primary categories: Normal, Eyes Closed, and Yawn. The Yawn Hand Cover condition is subsequently identified through the integration of the CNN's Yawn prediction with real-time hand position data from MediaPipe Hands.

The dataset used to train the CNN model consists of a combination of the publicly available YawDD dataset obtained from Kaggle (Abtahi, 2014) and supplementary facial images collected from volunteer participants. This combined dataset approach was necessitated by the fact that existing public datasets do not adequately represent the Yawn Hand Cover condition. By augmenting the validated public dataset with self-collected data specifically targeting underrepresented conditions, the model is expected to achieve more comprehensive and generalizable performance across real-world driver scenarios.

Image preprocessing using Contrast Limited Adaptive Histogram Equalization (CLAHE) is applied to all input frames prior to model inference to enhance local contrast and mitigate the effects of uneven illumination commonly encountered during driving (Reza, 2004). The system is designed and evaluated on a Raspberry Pi 4b platform, with the application of

TensorFlow Lite optimization for potential deployment on Raspberry Pi 5 identified as a direction for further development (R. David et al., 2019).

In summary, this research addresses the identified research gaps in the prior work by developing a CNN-based driver drowsiness detection system integrated with MediaPipe Face Mesh and MediaPipe Hands, capable of detecting four driver conditions including the previously undetected Yawn Hand Cover, and providing a thorough quantitative evaluation of system performance on an embedded platform.

1.2 Problem Formulation

Based on the background described above, the research problems formulated in this study are as follows:

a. How can a driver drowsiness detection system be designed and implemented using a Convolutional Neural Network (CNN) integrated with MediaPipe Face Mesh and MediaPipe Hands on the Raspberry Pi platform?
b. How does the developed CNN model perform in classifying four driver conditions – Normal, Eyes Closed, Yawn, and Yawn Hand Cover – as evaluated using standardized metrics including accuracy, precision, recall, and F1-score?
c. How effective is the combination of the public YawDD dataset and self-collected data in enabling the model to detect the Yawn Hand Cover condition that is inadequately represented in publicly available datasets?

1.3 Scope and Limitations

To maintain the focus and feasibility of this research, the following scope and limitations are defined:

a. The system is developed and evaluated on a Raspberry Pi 5 as the primary deployment platform.
b. Driver drowsiness detection is performed based on facial images captured by a webcam positioned at a frontal viewing angle relative to the driver's face.
c. The CNN model classifies three primary driver conditions: Normal, Eyes Closed, and Yawn. The Yawn Hand Cover condition is determined through the integration of the CNN's Yawn output with hand position analysis performed by MediaPipe Hands.

d. The training dataset consists of the publicly available YawDD dataset from Kaggle supplemented by facial images collected from volunteer participants to address the underrepresentation of the Yawn Hand Cover condition.
e. Image preprocessing is performed using Contrast Limited Adaptive Histogram Equalization (CLAHE) to improve input image quality prior to model inference.
f. The face Recognition of Interest (ROI) extracted by MediaPipe Face Mesh is resized 96 x 96 pixels before being passed to the CNN classifier.
g. The system issues a drowsiness warning when a cumulative drowsiness counter exceeds a predefined threshold based on the accumulation of fatigue condition detections over time.
h. This research does not address physiological signal-based detection approaches such as EEG, heart rate monitoring, or vehicle behavior-based sensing methods.

1.4 Research Objectives

In accordance with the problem formulation stated above, this research aims to:

a. Design and implement a driver drowsiness detection system using CNN integrated with MediaPipe Face Mesh and MediaPipe Hands on the Raspberry Pi platform, as an optimization of the previously developed system at Universitas Islam Indonesia.
b. Measure and evaluate the performance of the developed CNN model in classifying four driver conditions (Normal, Eyes Closed, Yawn, and Yawn Hand Cover) using standardized evaluation metrics including accuracy, precision, recall, and F1-score on a per-class basis.
c. Analyze the effectiveness of combining the public YawDD dataset with self-collected data in enhancing the model's ability to detect the Yawn Hand Cover condition not adequately covered by existing public datasets.

1.5 Research Benefits

This research is expected to deliver meaningful contributions to multiple stakeholders, as elaborated below:

a. Theoretical Benefits

1) This research contributes to the scientific body of knowledge in the fields of computer vision and deep learning, specifically in the application of CNN for driver drowsiness detection on edge computing platforms.
2) The study enriches the understanding of integrating deep learning models with the MediaPipe framework for real-time road safety applications.
3) The findings serve as a reference for future research seeking to develop more advanced and comprehensive drowsiness detection systems using similar or extended methodologies.

b. Practical Benefits

1) The system provides a more accurate and comprehensive drowsiness detection solution compared to conventional threshold-based approaches, with the potentional to reduce traffic accident rates caused by driver fatigue.
2) The research introduces Yawn Hand Cover detection capability that was absent from the prior system developed at Universitas Islam Indonesia, thereby improving the coverage and reliability of drowsiness identification in real-world conditions.
3) The results can serve as a foundation for developing AI-based vehicle safety systems that are cost-effective and deployable using widely accessible hardware such as Raspberry Pi.
4) This research offers direct societal benefits by contributing to improved road safety, particularly for long-distance drivers who are most vulnerable to fatigue-related incidents.

CHAPTER II
LITERATURE REVIEW

2.1 Previous Studies

Research on driver drowsiness detection systems has been widely conducted using various approaches, ranging from geometric landmark-based methods and conventional machine learning to deep learning techniques. Reviewing previous studies aims to map the development of existing methods, identify the strengths and weaknesses of each approach, and determine research gaps addresse in this study. Ten previous studies are reviewed below, covering classical machine learning approaches, transfer learning-based CNN architectures, temporal deep learning models, hand-occlusion-aware yawning detection, and embedded deployment on Raspberry Pi platforms.

Ngxande et al. (2017) conducted driver drowsiness detection research using a combination of Support Vector Machine (SVM) and the Dlib library for extracting 68 facial landmark points. The system utilized Eye Aspect Ratio (EAR), gaze direction, and yawning detection as classification inputs. Testing on controlled datasets achieved an accuracy of 82.5%. However, the method suffered from instability under varying lighting conditions and non-frontal head poses, limiting its real-world applicability.

Guo et al. (2019) developed a driver eye-state classification system using a CNN architecture based o transfer learning from the VGG-16 model trained on the ImageNet dataset. The model achieved an accuracy 91.3% for open and closed eye classification on GPU-based server testing datasets. Despite its high accuracy, the primary limitation was its high computational requirement, making it impractical for deployment on low-power embedded systems such as Raspberry Pi.

Wijnands et al. (2020) implemented a drowsiness detection system based on Dlib EAR and Lip Distance measurement on a Raspberry Pi 3B+ platform. This study is directly relevant as a comparison benchmark because it used a similar hardware platform. Evaluation results showed an overall accuracy of 75,93%, with significant weaknesses in detecting yawning conditions involving hand-covered mouths due to the absence of hand detection capabilities. Inference speed was also limited because of Dlib's computational burden on ARM processors.

Reddy et al. (2017) proposed a real-time driver drowsiness detection method for embedded systems based on compressed deep neural networks. Three CNN architectures were compared:

a four-stream baseline model using eyes, mouth, and face as separate inputs; a two-stream baseline model using only the left eye and mouth; and a compressed version of the two-stream model obtained through teacher-student knowledge distillation. The compressed model was deployed on an NVIDIA Jetson TK1 embedded board and achieve 89.5% accuracy with a detection time of 18.9 ms per frame. This study demonstrated that model compression techniques can substantially reduce inference latency on embedded hardware, although the Jetson TK1 platform used remains considerably more expensive and power-hungry than a Raspberry Pi.

Jabbar et al. (2018) developed a drowsiness detection system using a CNN architecture combined with Long Short-Term Memory (LSTM) to analyze temporal sequences of driver facial expressions. Implemented on embedded GPUs, the system achieved an accuracy of 89.4% and demonstrated advantages in capturing facial dynamics over time. This temporal approach highlighted the importance of understanding both instantaneous and sequential driver conditions.

Dipu et al. (2021) proposed a real-time drowsiness detection method that framed eye- and mouth-state recognition as an object detection task rather than a conventional classification task. The system employed the SSD-MobileNet-V1 architecture, a lightweight single-shot multibox detector, trained on approximately 4,500 labeled images covering four object categories: yawn, no-yawn, open-eye, and closed-eye. The detector simultaneously localized and classified facial regions within each frame, achieving real-time performance suitable for practical deployment. This study demonstrated that object-detection-style CNN architectures can serve as an efficient alternative to two-stage pipelines that separate face localization from condition classification, although the system did not incorporate any hand-related detection capability.

Phan et al. (2021) compared a facial-landmark threshold-based method against two adaptive deep neural networks built upon MobileNet-V2 and ResNet-50V2 through transfer learning. The threshold-based method used Dlib landmarks to detect blinks and yawns based on per-driver calibrated thresholds, while the deep learning method fine-tuned the two pretrained backbones on a custom image dataset collected from multiple sources. The transfer learning approach achieved an average accuracy of 97%, substantially outperforming the threshold-based baseline. This finding reinforces the broader pattern observed across the literature that learned CNN features generalize more robustly across individuals than

handcrafted geometric thresholds, motivating the adoption of a CNN classifier in the present research.

Jie et al. (2018) specifically investigated the problem of yawning detection under hand occlusion, which is directly relevant to the Yawn Hand Cover condition addressed in this research. The authors observed that conventional mouth-opening-based yawn detection fail when the mouth is covered by a hand, a behavior frequently exhibited by drowsy individuaks attempting to suppress a yawn. To address this limitation, they proposed combining an adaptive color-region technique for mouth-opening measurement with Local Binary Pattern (LBP) features extracted from the eye region to capture facial distortions that persist during yawning even when the mouth is occluded. Using the Strathclyde Facial Fatigue database, negative samples explicitly included frames containing hand gestures without yawning, allowing the classifier to distinguish incidental hand-near-face movements from genuine yawning. This study confirms that hand-related occlusion is a well-documented and clinically meaningful challenge in yawning detection, while also illustrating that prior solutions relied on indirect eye-region cues rather than directly modeling hand position, a gap addressed in the present research through explicit MediaPipe Hands integration.

Hashemi et al. (2020) developed a real-time drowsiness detection system based on a lightweight CNN architecture optimized for execution speed on standard CPUs without requiring GPU acceleration. The model classified eye states from cropped eye regions detected through a cascade-based face detector and achieved high accuracy while maintaining a frame rate suitable for real-time operation on consumer-grade hardware. This study is relevant because it illustrates an early demonstration that carefully designed lightweight CNN architectures can achieve real-time inference speed without specialized embedded accelerators, a design philosophy similarly adopted in the present research through the use of a compact CNN architecture rather than a deep transfer-learning backbone.

Ramirez-Hernandez et al. (2024) presented a rel-time embedded driver drowsiness detection system that combined a CNN classifier for eye-state recognition with Mouth Aspect Ratio (MAR) computation for yawning detection, deployed on a low-cost embedded platform. The hybrid design used the CNN specifically for eye-region classification, where appreance-based learned features were found to be more robust than geometric thresholds, while retaining a geometric MAR threshold for mouth-opening measurement, where a simple ratio metric remained computationally sufficient. This hybrid CNN-plus-geometric-metric design is conceptually similar to the architecture proposed in the present research, which likewise

combines a CNN classifier with MediaPipe-derived geometric and landmark-based features rather than relying exclusively on either approach in isolation.

Alameen and Alhothali (2023) proposed a lightweight drowsiness detection architecture combining a three-dimensional CNN (3DCNN) with an LSTM layer to jointly capture spatial facial features and temporal dynamics across consecutive video frames while minimizing the number of trainable parameters compared to conventional 3DCNN-LSTM hybrids. The model was explicitly designed with computational efficiency as a primary objective, targeting deployment scenarios with limited processing capacity. Evaluationn results indicated that the proposed lightweight architecture maintained competitive accuracy relative to heavier temporal models while substantially reducing inference time, supporting the broader argument that architectural efficiency, rather than raw model depth, is the determining factor for successful real-time deployment on constrained hardware such as the Raspberry Pi platform used in this research.

Liawatimena and Isworo (2025) constructed and published an annotated drowsiness detection dataset captured directly using a Raspberry Pi 5 eqquiped with a Camera Module 3, comprising 33,750 annotated images collected from 32 participants under five distinct lighting conditions ranging from 0 to 615 lux. The dataset captured four conditions, namely open eyes, closed eyes, no-yawning, and yawning, and was explicitly designed to address the lack of edge-computing-realistic drowsiness datasets that account for variable real-world illumination. This study is particularly relevant to the present research because it confirms both the practical feasibility of using Raspberry Pi hardware directly for image acquisition in drowsiness research and the continuing scarcity of publicly available datasets that represent occlusion-related conditions such as Yawn Hand Cover, reinforcing the necessity of supplementing public datasets with self-collected data as performed in this research.

Based on the review of previous studies, it can be identified that most existing systems still face limitations in multi-class detection accuracy, especially for conditions involving hand-covered yawning, embedded system deployment constraints, and insufficient integration between modern deep learning-based landmark detection and neural network classifiers. This research addresses these gaps by integrating MediaPipe Face Mesh and CNN on Raspberry Pi, as summarized in Table 2.1.

Table 2. 1 Comparison of Previous Studies

No	Reseracher/Year	Method	Platform	Detection Features	Accuracy
1	Ngxande et al, (2017)	SVM + Dlib 68 Landmark	PC (CPU)	EAR, Gaze, Yawn	82,5%
2	Guo et al. (2019)	CNN (VCH-16 Transfer Learning)	GPU Server	Closed Eyes, Face	91.3%
3	Wijnands et al. (2020)	Dlib + EAR + Lip Distance	Raspberry Pi 3B+	EAR, Lip Distance	75.93%
4	Reddy et al. (2018)	MediaPipe + HOG	PC (Realtime)	Eyes, Yawn, Head	88.7%
5	Jabbar et al. (2018)	CNN + LSTM (Temporal)	Embedded GPU	Eyes, Facial Expression	89.4%
6	Dipu et al. (2021)	SSD-MobileNet-V1	PC (Realtime)	Eyes, Yawn (Object Detection)	N/A*
7	Phan et al. (2021)	Dlib-threshold vs. MobileNet-V2/ResNet-50V2	PC (GPU)	Ear, Yawn (Transfer Learning)	97%
8	Jie et al. (2018)	Adaptive Color Region + LBP	PC (Offline)	Mouth Opening, Eye LBP (Hand-Occluded Yawn)	N/A*
9	Hashemi et al. (2020)	Lightweight CNN (CPU-optimized)	PC (CPU)	Eye State	93-99%
10	Ramirez-Hernandez et al. (2024)	CNN (Eyes) + MAR (Mouth)	Embedded System	Eye State, MAR	97.8%
11	Alameen & Alhothali (2023)	2DCNN + LSTM (Lightweight)	PC (Simulated Embedded)	Spatial-Temporal Facial Features	N/A*
12	Liawatimena & Isworo (2025)	Dataset Annotation (FOMO-ready)	Raspberry Pi 5	Eyes, Yawn (Multi-lighting)	N/A*

2.2 Theoretical Foundation

2.2.1 Driver Drowsiness Detection

Driver drowsiness detection is a research field in active vehicle safety systems that aims to automatically recognize a driver's decreased alertness before an accident occurs. In general, approaches to drowsiness detection can be categorized into three major groups: vehicle

performance-based approaches, physiological sensor-based approaches, and visual face-based approaches (Albadawi et al., 2022).

Vehicle performance-based approaches detect drowsiness through patterns of lane deviation, steering pressure variations, or changes in accelerator pedal position. Although these methods do not require additional sensors attached to the driver, they are considered reactive because they only detect drowsiness after the driver begins to lose control. Physiological sensor-based approaches use electrodes to measure EEG signals, heart rate, or galvanic skin response. These methods are highly accurate but impractical because they require physical contact with driver. Meanwhile, visual face-based approaches use cameras to analyze facial expressions, eye movements, and mouth conditions, making them the most non-intrusive and practical approach for commercial system implementation.

This research adopts a camera-based visual approach by extracting facial geometric features using MediaPipe Face Mesh and classifying them using CNN. The four target classification conditions are normal (driver in alert state), eyes closed (closed eyes as an indicator of drowsiness), yawn (yawning as an indicator of fatigue), and yawn hand cover (yawning while covering the mouth with a hand, which is more difficult to detect using conventional systems).

2.2.2 Deep Learning

Deep Learning is a subfield of machine learning that utilizes artificial neural network architectures with multiple hidden layers to learn hierarchical representations of data (LeCun et al., 2015). Unlike conventional machine learning methods that require manual feature engineering, deep learning can automatically learn relevant features from raw data through the training process. This capability makes deep learning highly effective for tasks involving high-dimensional data such as images, audio, and text.

The foundation of deep learning is the artificial neural network, which is inspired by the working of biological neurons. Each neuron in the network receives multiple inputs, applies a non-linear activation function, and produces an output that is forwarded to the next layer. The training process is performed using the backpropagation algorithm, which calculates the gradient of the loss function with respect to each network parameter, and then updates those parameters using optimization algorithms such as Stochastic Gradient Descent (SGD) or Adam Optimizer. Advances in network architectures, the availability of large labeled datasets, and

increased GPU computational capacity have driven deep learning to become the dominant paradigm in solving modern artificial intelligence problems.

2.2.3 Convolutional Neural Network (CNN)

Convolutional Neural Network (CNN) is a deep learning architecture specifically designed to process grid-structured data, particularly two-dimensional images (Hinton et al., 2015). CNN introduces two fundamental concepts that distinguish it from conventional fully connected networks: convolution operations and parameter sharing. Through convolution operations, learnable filters or kernels are systematically shifted across the image area to produce feature maps that capture local visual patterns such as edges, textures, and shapes.

Architecturally, a CNN generally consists of three main types of layers. First, the convolutional layer applies convolution operations between small filters and the input to generate feature maps. Second, the pooling layer reduces the spatial dimensions of feature maps to decrease computational load and improve translation invariance. Max pooling is the most commonly used pooling operation, where the maximum value from each region is retained. Third, the fully connected layer functions as the classifier at the end of the network, combining all features extracted by the convolutional and pooling layers to produce the final classification decision.

In the context of this research, CNN is used to classify four driver conditions based on features extracted from facial landmarks generated by MediaPipe Face Mesh. The advantage of CNN over conventional classifiers such as SVM or decision trees lies in its ability to learn more abstract and discriminative feature representations, enabling it to capture subtle differences between visually similar conditions, such as partially closed eyes and fully closed eyes.

2.2.4 Transfer Learning

Transfer learning is a machine learning technique in which a model that has been previously trained on a large source dataset is reused, either partially or entirely, as the starting point for training on a different but related target task. Instead of initializing network weights randomly, transfer learning initializes the network with weights learned from the source task, allowing the model to reuse general-purpose low-level features such as edges, textures, and color gradients that are broadly applicable across many visual recognition problems. This approach is particularly valuable when the target dataset is relatively small, as is often the case

in specialized domains such as driver drowsiness detection, since training a deep CNN entirely from scratch on a limited dataset is prone to overfitting.

Common transfer learning backbones used in prior drowsiness detection research include VGG-16, MobileNet-V2, and ResNet-50V2 (Gholizade et al., 2025), as discussed in the previous studies reviewed earlier in this chapter. While transfer learning generally improves accuracy and reduces the amount of labeled data required for training, the resulting models typically retain a large number of parameters inherited from the original backbone, which increases inference latency and memory footprint on resource-constrained embedded platforms. This trade-off motivated the present research to adopt a CNN architecture trained directly on the combined YawDD and self-collected dataset rather than relying on a heavyweight pretrained backbone, prioritizing inference efficiency on the Raspberry Pi platform over the marginal accuracy gains that a deeper transfer learning model might offer.

2.2.5 MediaPipe Face Mesh

MediaPipe Face Mesh is a machine learning solution developed by Google Research for real-time facial geometry estimation (Lugaresi et al., 2019)(Kartynnik et al., 2019). The system is capable of generating 478 three-dimensional facial landmark points covering the entire face area, including facial contours, eyebrows, eyes, nose, lips, and irises. MediaPipe Face Mesh uses a two-stage approach: the first stage is face detection using a lightweight model to identify the face location within an image, and the second stage is facial landmark estimation using a head mesh model to predict the three-dimensional position of each landmark.

One significant advantage of MediaPipe Face Mesh compared to Dlib is its use of a deep learning architecture specifically optimized to run efficiently on mobile and embedded devices. The MediaPipe Face Mesh model uses a lightweight yet accurate convolutional architecture, enabling high inference speed even on ARM processors such as those found in Raspberry Pi devices. Furthermore, MediaPipe's ability to generate three-dimensional coordinates provides additional robustness against head pose variations, since landmark positions can be normalized relative to head orientation to produce more consistent measurements.

This research also utilizes the MediaPipe Hands module, which can simultaneously detect 21 hand landmark points. The integration of MediaPipe Hands allows the system to detect the yawn hand cover condition, where the driver yawns while covering their mouth with a hand, a condition that cannot be detected by conventional Dlib-based systems. This capability becomes one of the differential advantages distinguishing the developed system in this research.

2.2.6 MediaPipe Hands

MediaPipe Hands is a hand and finger tracking solution developed by Google Research that estimates 21 three-dimensional landmark points per hand from a single RGB frame (Lugaresi et al., 2019). The pipeline operates in two stages: a palm detection model first localizes the approximate hand region within the image, and a subsequent hand landmark model then regresses the precise three-dimensional coordinates of all 21 key points, including the wrist, finger joints, and fingertips. This two-stage design allows MediaPipe Hands to operate efficiently even when only a small portion of the frame contains a hand, which is the typical case in a driver-facing camera setup.

This research utilizes MediaPipe Hands to detect the spatial position of the driver's hand relative to the mouth region identified by MediaPipe Face Mesh. The integration of MediaPipe Hands allows the system to detect the Yawn Hand Cover condition, where the driver yawns while covering their mouth with a hand, a condition that cannot be detected by conventional Dlib-based systems because Dlib provides no mechanism for hand localization. In the proposed system, the proximity between the wrist and fingertip landmarks generated by MediaPipe Hands and the mouth-region landmarks generated by MediaPipe Face Mesh is computed as a geometric distance feature; when this distance falls below an empirically determined threshold while the CNN simultaneously predicts the Yawn class, the final output is reclassified as Yawn Hand Cover. This capability becomes one of the differentiating advantages distinguishing the developed system in this research from the previous studies reviewed in this chapter, the majority of which lack any explicit hand-detection mechanism.

2.2.7 Facial Landmark Detection

Facial landmark detection is a computer vision task aimed at identifying the coordinate locations of important anatomical points on facial images. These points include eye corners, nose tips, lip contours, jawlines, and other facial regions with geometric significance. Facial landmark detection is a fundamental component in various computer vision applications, including facial expression recognition, head pose estimation, identity verification, and most relevant to this research, drowsiness detection.

There are two main paradigms in modern facial landmark detection. The first paradigm is the direct regression-based approach, where the model predicts landmark coordinates directly from facial images as a structured regression problem. Dlib uses this approach with an ensemble of regression trees that work efficiently but have limitations in handling complex

expression variations. The second paradigm is the model-based approach, which uses explicit geometric representations of facial structures, such as 3D morphable models, to guide landmark estimation (Kartynnik et al., 2019). MediaPipe Face Mesh uses a variation of this second approach, where the model is trained to predict landmarks in three-dimensional space, providing richer representations and invariance to pose variations.

2.2.8 Raspberry Pi

Raspberry Pi is a single-board computer developed by the Raspberry Pi Foundation, a non-profit organization from the United Kingdom, with the initial goal of introducing computing and programming to a wider audience. Raspberry Pi has evolved into one of the world's most popular embedded system platforms, widely used in Internet of Things (IoT) projects, robotics, home automation, and other embedded systems (Wijnands et al., 2020).

The Raspberry Pi 5, which is used in this research, is equipped with a 64-bit quad-core ARM Cortex-A76 processor running at 2.4 GHz, LPDDR4X RAM available in 4GB or 8GB configurations, and improved connectivity support including USB 3.0, Gigabit Ethernet, dual-band WIFI 802.11ac, Bluetooth 5.0, and dual 4K micro-HDMI output. Compared to its predecessor, Raspberry Pi 5 provides significantly higher CPU and GPU performance, enabling faster image processing and more efficient execution of machine learning frameworks such as TensorFlow Lite (S. David et al., 2018) and OpenCV. These specifications make Raspberry Pi 5 highly suitable for real-time CNN inference and computer vision applications directly on embedded devices without requiring cloud-based processing.

2.2.9 Image Preprocessing with CLAHE

Contrast Limited Adaptive Histogram Equalization (CLAHE) is an image enhancement technique introduced by Reza (2024) as an improvement over conventional Adaptive Histogram Equalization (AHE). Standard histogram equalization redistributes pixel intensity values across an entire image to improve global contrast, but this global approach often over-amplifies noise in relatively homogeneous regions and produces unnatural-looking results when illumination varies across different parts of an image. AHE addresses this limitation by computing histogram equalization separately within small local regions called tiles, allowing contrast enhancement to adapt to local illumination conditions rather than applying a single global transformation.

CLAHE further refines AHE by introducing a clipping limit on the histogram redistribution process, which prevents excessive amplification of noise within tiles that contain very uniform intensity values, such as a poorly lit region of a driver's face. In this research, CLAHE is applied as a preprocessing step to every frame captured by the camera prior to face detection and CNN inference. This preprocessing step is particularly important given that driving conditions frequently involve uneven illumination, including direct sunlight glare, shadowed regions cast by the vehicle structure, and low-light conditions during nighttime driving. By enhancing local contrast around the eye and mouth regions before landmark detection and classification, CLAHE is expected to improve the consistency of MediaPipe Face Mesh landmark localization and, consequently, the reliability of the downstream CNN classification under non-ideal lighting conditions.

2.2.10 Eye Aspect Ratio (EAR)

Eye Aspect Ratio (EAR) is a geometric metric introduced by Soukupova and Cech (2016) to quantify eye openness based on the positions of eye landmark points. EAR is calculated as the ratio between the total vertical distance between the eyelids and the horizontal distance between the eye corners, producing a scale-invariant value independent of face size or camera distance.

The EAR calculation formula uses six eye landmark points (p1 to p6) and is defined as follows: the EAR value is obtained from the sum of the Euclidean distances between the vertical point pairs, namely the distances between p2 and p6 and between p3 and p5, divided by twice the Euclidean distance between the eye corner points p1 and p4.

Mathematically:

$$\mathrm{EAR} = \frac{||\mathrm{p1} - \mathrm{p6}|| + ||\mathrm{p3} - \mathrm{p5}||}{2_{||p1-p4||}} \quad (2.1)$$

When the eyes are fully open, EAR values typically range from 0.25 to 0.40. As the eyes gradually close, the EAR value decreases monotonically and approaches zero when the eyes are fully closed.

In this research implementation, EAR is calculated using landmark coordinates generated by MediaPipe Face Mesh in normalized space. Unlike Dlib-based implementations that use absolute pixel coordinates, MediaPipe generates coordinates within the range [0,1] relative to image dimensions, which are then converted into pixel coordinates to ensure consistent

calculations. The EAR values from the left and right eyes are averaged to produce a more stable and representative overall EAR value.

2.2.11 Mouth Aspect Ratio (MAR)

Mouth Aspect Ratio (MAR) is a metric analogous to EAR that is used to quantify mouth openness, specifically as an indicator of yawning conditions in drivers. MAR is calculated as the ratio between the vertical distance between the upper and lower lips and the horizontal distance between the mouth corners, producing a consistent value independent of facial size within the image.

The MAR formula uses selected lip landmark coordinates from key points on the inner mouth contour. Upper lip landmarks are represented by points in the middle upper mouth region, while lower lip landmarks are represented by points in the middle lower mouth region. MAR values for a closed mouth approach zero, whereas during wide yawning conditions, MAR values may reach 0.5 to 1.0 or even higher, depending on individual facial geometry. In this research, the optimal MAR threshold is determined based on the statistical distribution of MAR values across the dataset, with a threshold value around 0.20 used to separate yawning conditions from closed-mouth conditions.

It should be noted that in the yawn hand cover condition, wher the driver yawns while covering the mouth with a hand, MAR becomes unrealible because the mouth landmarks may be partially occluded. Therefore, hand detection features obtained from MediaPipe Hands become the primary feature for classifying this condition, while MAR functions as a secondary supporting feature.

2.2.12 Confusion Matrix

A confusion matrix is an evaluation tool used in machine learning and computer vision to comprehensively analyze classification model performance. A confusion matrix presents a summary of model prediction results in the form of a two-dimensional table, where rows represent actual classes (ground truth) and columns represent classes predicted by the model. Each cell in the matrix indicates the number of samples classified into a particular combination of actual and predicted classes.

For multi-class classification problems such as this research, which involves four classes (normal, eyes closed, yawn, and yawn hand cover), the confusion matrix takes the form of a 4x4 matrix. From this confusion matrix, several evaluation metrics can be derived for each

class, including precision (the ratio of correctly predicted positive samples to all predicted positive samples), recall or sensitivity (the ratio of correctly predicted positive samples to all actual positive samples), and F1-Score, which is the harmonic mean of precision and recall. Analysis of the confusion matrix enables deeper identification of classification error patterns, such as which classes are most frequently confused with each other, which is highly useful for improving model thresholds and architecture.

2.2.13 Precision, Recall, and F1-Score

Precision is defined as the ratio of correctly predicted positive samples, or true positives, to the total number of samples predicted as positive for a given class, which includes both true positives and false positives. Precision quantifies how trustworthy a positive prediction is; a high precision value for the Yawn Hand Cover class, for example, indicates that when the system reports this condition, it is rarely a false alarm caused by misclassifying a different condition.

Recall, also referred to as sensitivity, is defined as the ratio of true positives to the total number of actual positive samples for a given class, which includes true positives and false negatives. Recall quantifies the system's ability to successfully identify all genuine occurrences of a condition; a low recall value for the Eyes Closed class, for instance, would indicate that the system frequently fails to detect genuine eye-closure events, which is particularly concerning for a safety-critical drowsiness detection application where missed detections could delay an alert.

F1-score is the harmonic mean of precision and recall, calculated as twice the product of precision and recall divided by their sum. Unlike a simple arithmetic average, the harmonic mean penalizes extreme imbalances between precision and recall, meaning that a classifier cannot achieve a high F1-score by excelling at one metric while performing poorly on the other. Because driver drowsiness detection involves an inherent trade-off between minimizing false alarms, which improves precision, and minimizing missed detections, which improves recall, F1-score serves as a balanced single-value summary that is widely reported alongside accuracy in this research and in the previous studies reviewed earlier in this chapter.

2.2.14 YawDD Dataset

The Yawning Detection Dataset (YawDD) is a publicly available video dataset introduced by Abtahi et al. (2014) specifically designed to support research on driver yawning and

drowsiness detection. The dataset contains video recordings of volunteer participants seated in a vehicle, captured from a dashboard-mounted camera under varying conditions including male and female participants, individuals with and without glasses, and a range of natural in-vehicle lighting conditions. Each video sequence is labeled according to the participant's mouth state, allowing researchers to extract frames corresponding to normal speech, silence, and yawning behavior. The version of YawDD obtained through Kaggle and used in this research is organized into four labeled image folders representing closed eyes, open eyes, no-yawn, and yawn conditions, which were subsequently remapped to three CNN training classes as described in the following chapter.

Despite its widespread use as a benchmark dataset in the driver drowsiness detection literature, YawDD does not contain a labeled category for the Yawn Hand Cover condition, in which a driver yawns while covering the mouth with a hand. This omission reflects a broader limitation common to most publicly available drowsiness datasets, as similarly noted by Liawatimena and Isworo (2025) in their discussion of the scarcity of edge-computing-realistic drowsiness datasets. To address this gap, the present research supplements the YawDD dataset with self-collected facial images specifically captured to represent the Yawn Hand Cover condition, ensuring that the CNN model and the integrated MediaPipe Hands detection mechanism are trained and evaluated on data that adequately represents all four target classes.

2.2.15 Accuracy Evaluation

Accuracy is a basic evaluation metric that measures the proportion of correct predictions out of the total testing data. Mathematically, accuracy is calculated as the number of correct predictions divided by the total number of testing samples. Although intuitive, accuracy can provide misleading results for imbalanced datasets, where one class contains significantly more samples than others.

In this research, accuracy is evaluated at two levels: overall accuracy and per-class accuracy. Overall accuracy provides a general overview of system performance, while per-class accuracy identifies specific conditions that are more difficult to classify. The developed system achieved an overall accuracy of 93.67%, compared to the Dlib-based comparison system, which achieved 75.67%. This significant improvement in accuracy confirms the effectiveness of the hybrid MediaPipe Face Mesh and CNN approach in overcoming the limitations of conventional drowsiness detection methods.

2.3 Research Gap

A systematic examination of the twelve previous studies summarized in Table 2.1 reveals four recurring gaps that collectively motivate the design of the system proposed in this research. Each gap is discussed in turn below, together with an explanation of how the present research is positioned to address it.

The first gap concerns the absence of hand-occlusion-aware yawning detection in nearly all reviewed systems. Of the twelve studies reviewed, only Jie et al. (2018) explicitly addressed the problem of yawning detection when the mouth is occluded by a hand, and even this study relied on indirect eye-region distortion cues rather than directly localizing the hand itself. The remaining eleven studies, including the directly comparable Raspberry Pi-based system of Wijnands et al. (2020), classify yawning exclusively through mouth-opening geometry such as Lip Distance or Mouth Aspect Ratio, which inherently fails whenever the mouth region is partially or fully covered. Because covering the mouth while yawning is a common and clinically documented behavior among drowsy individuals, this represents a meaningful blind spot in the existing literature. The present research addresses this gap directly by integrating MediaPipe Hands to explicitly localize the driver's hand and compute its proximity to the mouth region, enabling the system to recognize a fourth condition, Yawn Hand Cover, that none of the eleven reviewed studies fully support.

The second gap concerns the trade-off between classification accuracy and computational efficiency on embedded hardware. High-accuracy systems reviewed in this chapter, such as Guo et al. (2019) with 91.3% accuracy and Phan et al. (2021) with 97% accuracy, rely on transfer learning from deep backbones such as VGG-16, MobileNet-V2, and ResNet-50V2, which require GPU acceleration or substantial computational resources unavailable on a low-cost single-board computer. Conversely, systems explicitly designed for embedded deployment, such as Wijnands et al. (2020) on Raspberry Pi 3B+ and Reddy et al. (2017) on Jetson TK1, report comparatively lower accuracy or depend on hardware that remains considerably more expensive than a Raspberry Pi. This indicates that the literature has not yet converged on an architecture that simultaneously achieves high multi-class accuracy and practical deployability on consumer-grade embedded boards. The present research addresses this gap by adopting a compact CNN architecture trained directly on a combined dataset, rather than a heavyweight transfer learning backbone, prioritizing a balance between accuracy and inference efficiency suitable for the Raspberry Pi platform.

The third gap concerns inconsistent and often incomplete evaluation methodology. As indicated by the N/A entries in Table 2.1, several reviewed studies, including Dipu et al. (2021), Jie et al. (2018), Alameen and Alhothali (2023), and Liawatimena and Isworo (2025), do not report a single consolidated accuracy figure, instead presenting dataset descriptions, qualitative architecture comparisons, or per-component results that are difficult to compare directly against other systems. Even among studies that do report accuracy, few provide a complete breakdown of precision, recall, F1-score, and confusion matrix on a per-class basis, which limits the ability of subsequent researchers to identify which specific conditions a given system struggles to classify. This research addresses this gap by adopting a standardized, multi-class evaluation protocol that reports overall accuracy alongside per-class precision, recall, F1-score, and a complete confusion matrix for all four target conditions.

The fourth gap concerns dataset representativeness for the Yawn Hand Cover condition specifically. As discussed by Liawatimena and Isworo (2025), publicly available drowsiness datasets, including YawDD, were not constructed with hand-occlusion scenarios in mind and therefore contain no labeled samples for this condition. None of the twelve reviewed studies report training or evaluating their models on a dataset that explicitly includes hand-covered yawning as a distinct labeled class. This research addresses this gap by supplementing the publicly available YawDD dataset with self-collected facial images specifically captured to represent the Yawn Hand Cover condition, ensuring that the CNN model is trained on data that adequately represents the Yawn Hand Cover condition, ensuring that the CNN model is trained on data that adequately represents all four target classes rather than relying solely on a dataset designed for a narrower three-class problem.

Table 2.2 summarizes these four research gaps alongside the corresponding design decision adopted in this research to address each one.

Table 2. 2 Research Gaps and Corresponding Design Decisions

No	Identified Research Gap	Design Decision in This Research
1	No hand-occlusion-aware yawning detection in 11 of 12 reviewed systems	Integrate MediaPipe Hands to detect hand-to-mouth proximity and classify Yawn Hand Cover
2	Accuracy vs. embedded efficiency trade-off unresolved	Adopt a compact CNN trained from scratch instead of a

		heavyweight transfer learning backbone
3	Inconsistent or incomplete evaluation metrics (4 of 12 studies report no consolidated accuracy)	Report standardized accuracy, per-class precision, recall, F1-score, and full confusion matrix
4	No public dataset, including YawDD, contains labeled Yawn Hand Cover samples	Supplement YawDD with self-collected images representing Yawn Hand Cover

CHAPTER III
RESEARCH METHODOLOGY

3.1 System Development Methods

This research is classified as applied engineering research with an experimental design. The primary objective is to develop, integrate, and empirically evaluate a real-time driver drowsiness detection system on an embedded platform. The research employs a quantitative evaluation approach, measuring classification performance using standard metrics derived from the confusion matrix (accuracy, precision, recall, and F1-score), real-time performance metrics (frames per second), and functional verification tests to confirm correct hardware operation. The development process follows the Waterfall methodology, which provides a structured, sequential progression through clearly defined stages: Requirement Analysis, System Design, Implementation, Testing, and Maintenance.

This research utilizes the Waterfall system development method, which was introduced by Winston W. Royce (1970). The Waterfall method was selected because this research has requirements and specifications that have been clearly defined from the beginning, namely developing a driver drowsiness detection system based on CNN and MediaPipe as an improvement over the Dlib-based baseline system. Each development phase is carried out sequentially and linearly, where one stage must be completed before moving on to the next.

The Waterfall method consists of five stages:

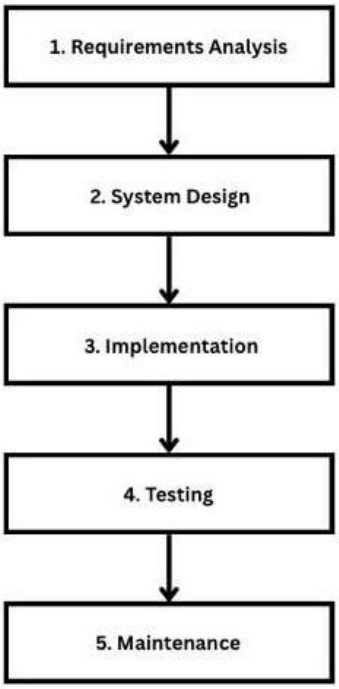


Figure 3. 1 Waterfall Method

3.2 Requirements Analysis

3.2.1 Problem Analysis

Traffic accidents caused by driver drowsiness are a serious issue that requires as a real-time detection solution. The baseline system developed by Aprilian and Mahendra (2024) from Universitas Islam Indonesia successfully built a drowsiness detection system using Haar Cascade Classifier, Dlib 68-point facial landmarks, Eye Aspect Ratio (EAR), and Lip Distance on a Raspberry Pi 4B. However, based on the analysis of the baseline system, several weaknesses were found that serve as the foundation for the system development in this research.

First, the use of static thresholds (EAR < 0.21 and Lip Distance > 20.00) calibrated per user are not adaptive to individual variation. Second, the system failed to detect yawning when the driver's face experienced occlusion, for instance when the driver yawned while covering the mouth with a hand. Third, system evaluation was conducted qualitatively on 12 respondents without producing standardized global accuracy metrics such as confusion matrix, precision, recall, and F1-score. Fourth, no detection class was available for the Yawn Hand Cover condition, where the driver yawns while covering the mouth with a hand.

3.2.2 Requirement Analysis

The requirements analysis stage identified the functional and non-functional requirements of the system, established the scope of the research, and selected the dataset and detection approach. Functional requirements include: (1) real-time capture and processing of video frames from a USB webcam; (2) detection and localization of the driver's face using 478 3D landmarks; (3) classification of the facial region into three states (normal, eyes_closed, yawn); (4) conditional detection of hand-over-mouth gestures; (5) episode-based drowsiness counting within a 60-second sliding window; and (6) activation of a hardware buzzer alarm when the drowsy state is confirmed. Non-functional requirements include achieving a minimum of 8 FPS on the Raspberry Pi 5, operating without a network connection or external monitor, and launching automatically at device power-on.

3.2.3 Data Requirements

The dataset used is a combination of two resources. The first is the YawDD Dataset (Kaggle), developed by Abtahi et al. (2014), containing images of drivers'faces in four conditions recorded from an in-vehicle camera with diverse subject variation. The second is a self-collected dataset from volunteer participants who performed normal, eyes closed, and yawning expressions to increase data variation and improve model generalization.

The YawDD dataset has four folders that are mapped to three CNN target classes, as shown in Table 3.1.

Table 3. 1 Dataset Class Mapping from YawDD to CNN Target Classes

YawDD Folder	**CNN Class**	**Mapping Rationale**
Closed_eyes	Eyes_closed	Closed eyes are an indicator of drowsiness
Open_eyes	Normal	Open eyes represent an alert condition
No_yawn	Normal	Not yawning represents an alert condition
Yawn	Yawn	Yawning is an indicator of drowsiness

The open_eyes and no_yawn folders are merged into a single normal class because both represent a condition where the driver is not drowsy. The combined dataset is split with a ratio of 70% training, 15% validation, and 15% test using a fixed random seed (seed=42) to ensure experimental reproducibility.

3.2.4 Hardware and Software Requirements

The hardware specifications used in this research are listed in Table 3.2, and the software tools are presented in Table 3.3.

Table 3. 2 Hardware Requirements

Component	Specification
Development Computer	Laptop Lenovo ThinkPad Intel Core i7, 8 GB RAM
Camera	Webcam 1280 x 720 pixels
Deployment target	Raspberry Pi 5 (maintenance stage)

Table 3. 3 Software Requirements

Software	Version	Function
Python	3.10	Primary programming language
Tensorflow/Keras	2.19/3.12	CNN model training and inference
MediaPipe	≥ 0.10	Face Mesh (478 landmarks) + Hands (21 landmarks)
OpenCV	≥ 4.8	Camera acquisition, preprocessing, overlay
NumPy	≥ 1.24	Numerical operations
Scikit-learn	≥1.3	Evaluation metrics

3.3 System Design

The system design stage produced the overall architecture of the detection pipeline, selected the CNN model topology, and defined the hardware configuration. Three candidate approaches were analysed prior to finalizing the design. The analysis of these approaches, their limitations, and the rationale for the final integrated design are presented in Section 3.3. The

overall system architecture employs a sequential per-frame pipeline: video capture – optional CLAHE preprocessing – MediaPipe Face Mesh (face detection and landmark extraction) – CNN classification of the face ROI – conditional MediaPipe Hands invocation – episode-based drowsiness counting – hardware alarm. The hardware design pairs the Raspberry Pi 5 with a Logitech C922 USB webcam and a GPIO-connected active buzzer, as shown in Figure 3.2.

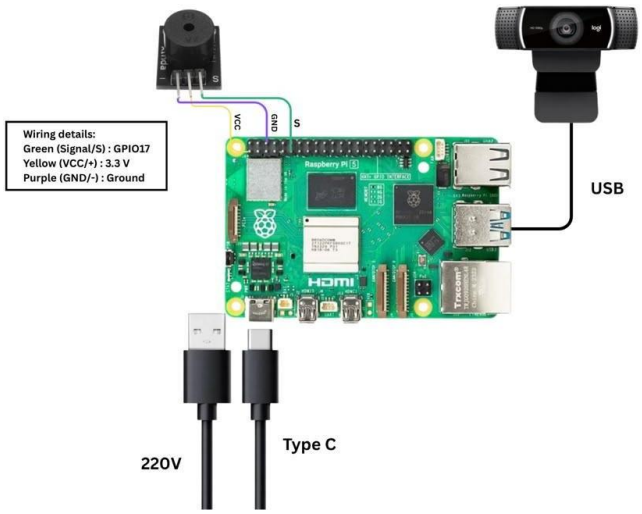


Figure 3. 2 Hardware design: Raspberry Pi 5, Logitech C922 USB webcam, and active buzzer (Signal: GPIO17, VCC: 3.3V, GND:Ground)

3.3.1 CNN Architecture Design

The CNN model is designed as a lightweight custom architecture built from scratch without transfer learning. The architecture, summarized in Table 3.4, consists of four progressive convolutional blocks.

Table 3. 4 CNN Architecture Design

Block	Layers	Filters	Output	Function
1	Conv2D+BN+ReLU+MaxPool	32	48x48x32	Basic features (edges, textures)
2	Conv2D+BN+ReLU+MaxPool	64	24x24x64	Intermediate (eye/lip contours)
3	Conv2D+BN+ReLU+MaxPool	128	12x12x128	Advanced (opening patterns)

4	DWConv+PWConv+MaxPool	256	6x6x256	High-level, efficient
Head	GAP+Dense(128)+DO+Dense(3)	-	3	Final Classification

Block 4 employs Depthwise Separable Convolution adopted from MobileNet architecture, separating spatial and channel-wise convolution to significantly reduce parameters. Total parameters: ~163,000 (~0.62 MB float32, ~0.16 MB int8 quantized). Regularization includes L2 (l e-4), Dropout (0.4), and BatchNormalization

3.3.2 Hand-Covering-Mouth Detection Design

The mouth center coordinates are obtained from MediaPipe Face Mesh using the average position of landmarks 13 (upper lip), 14 (lower lip), 78 (left corner), and 308 (right corner). Face size is calculated as the Euclidean distance between landmark 1 (nose tip) and 152 (chin), used as a normalization factor. The Euclidean distance from each of the 21 hand landmarks to the mouth center is computed and normalized against face size. If the minimum normalized distance is less than the threshold (0.65), the condition is classified as Yawn Hand Cover.

3.3.3 Drowsiness Detection Logic

The system employs an episode-based counting approach within a 60-second sliding window. A single yawning episode is defined as one transition from a non-yawning condition to a yawning condition. A driver who yawns continuously for 3 seconds is counted as one episode, preventing false alarms from prolonged yawns. If the number of episodes within the last 60 seconds exceeds the threshold (default: 5), the system status changes to DROWSY and an alarm is activated. The threshold of 5 episodes per minute was selected based on the consideration that normal yawning frequency is approximately 1-3 times per hour.

3.4 Implementation

The implementation stage covers dataset preparation, CNN model training, model conversion to TensorFlow Lite, integration of the real-time detection pipeline, hardware assembly, and deployment to the Raspberry Pi 5. Full details of the implementation are presented in Chapter IV. A summary of the key implementation parameters is provided in Table 3.5

Table 3. 5 Key Implementation Parameters

Parameter	Value
CNN input size	96 x 96 x 3 (RGB)
Batch size	32
Maximum training epochs	50 (EarlyStopping patience=10)
Optimizer	Adam (initial lr = 0.001
Loss function	Categorical cross-entropy
Deployed model format	TFLite float16 (0.31 MB)
Camera resolution	640 x 480
Drowsiness threshold	5 yawning episodes per 60 seconds
Alarm output	Active buzzer via GPIO17

3.5 Testing

The testing stage evaluates system performance using three categories of tests: classification accuracy evaluation on held-out test data, real-time performance measurement on the deployed Raspberry Pi 5, and functional verification of each system component. The evaluation metrics and testing procedures are defined in Section 3.6. Test results are reported in Chapter V.

3.6 Maintenance

The maintenance stage covers the deployment of the system to the Raspberry Pi 5 and its configuration for autonomous operation. This stage has been completed: the system is configured to launch automatically via a system service at device boot, without requiring a monitor, keyboard, or network connection. Planned future work includes expansion of the self-collected dataset and exploration of a hybrid CNN architecture that incorporates EAR and MAR geometric features alongside the raw facial image.

3.7 Comparative Analysis of Detection Approaches

Prior to finalizing the system architecture, three detection approaches were individually implemented and tested on the self-collected dataset to justify the choice of the integrated CNN

+ MediaPipe design. The findings from each approach directly informed the final design decision.

3.7.1 Approach A: MediaPipe Face Mesh with EAR and MAR Thresholds

In this approach, drowsiness is detected purely from geometric features computed from MediaPipe Face Mesh landmarks: the Eye Aspect Ratio (EAR) for closed-eye detection and the Mouth Aspect Ratio (MAR) for yawn detection. No machine learning model is trained; instead, fixed thresholds (EAR < 0.214, MAR > 0.043) are applied to classify each frame. This approach was evaluated on the 80-image self-collected dataset (20 images per class). The results are presented in Table 3.6

Table 3. 6 Key Implementation Parameters

Class	Correct/Total	Accuracy
Normal	20/20	100.0%
Eyes_closed	19/20	95.0%
Yawn	18/20	90.0%
Yawn_hand_cover	18/20	90.0%
Overall	**75/80**	**93.75%**

Overall accuracy reached 93.75%, demonstrating that geometric features are effective under standard conditions. However, this approach has significant limitations: (1) it relies on fixed numerical thresholds (EAR and MAR) that must be manually calibrated and are sensitive to variations in face size, camera distance, and lighting; (2) subjects with naturally smaller eyes may produce EAR values below the threshold even when fully alert (false positives); (3) it cannot distinguish subtle facial expressions beyond closed eyes and open mouth; and (4) the yawn detection accuracy of 90% shows that low-MAR yawning (partially closed mouth) is misclassified as normal or eyes_closed.

3.7.2 Approach B: CNN (Convolutional Neural Network) Classifier

In this approach, a lightweight CNN model is trained on the combined YawDD + self-collected dataset to classify the face ROI image directly into three classes (normal, eyes_closed, yawn) without any geometric feature computation. The model was evaluated on a held-out test set of 438 images. The classification report is presented in Table 3.7.

Table 3. 7 CNN-Only Approach: Classification Report (438 test images)

Class	Precision	Recall	F1-Score	Support
Eyes_closed	0.9907	0.9636	0.9770	110
Normal	0.9817	0.9772	0.9794	219
Yawn	0.9646	1.0000	0.9820	109
Overall Accuracy	-	-	97.95%	438
Macro Average	0.9790	0.9803	0.9794	438
Weighted Average	0.9797	0.9795	0.9794	438

The CNN achieved an overall accuracy of 97.95% with highly balanced precision, recall, and F1-scores across all three classes. However, this approach has its own limitations: (1)it classifies only three conditions and cannot detect the yawn_hand_cover scenario, which is a common real-world behavior where a driver covers their mouth while yawning; (2) the CNN operates on raw image appearance and may produce ambiguous predictions when a hand obscures facial landmarks; and (3) it does not provide precise spatial coordinates of facial features such as mouth center position, which are required for hand-proximity detection.

3.7.3 Approach C: Integrated System (CNN + MediaPipe)

The proposed system integrates the CNN classifier with MediaPipe Face Mesh and conditionally invokes MediaPipe Hands, combining the strengths of both approaches while mitigating their individual limitations. MediaPipe Face Mesh provides stable 3D landmark coordinates for precise face bounding box computation and mouth center localization, without relying on fixed EAR/MAR thresholds. The CNN model classifies the visual appearance of the face ROI at high accuracy (97.95%). When the CNN predicts yawn, MediaPipe Hands is activated to determine whether a hand is covering the mouth area, enabling detection of the yawn_hand_cover condition. This conditional activation conserves CPU resources on the embedded device, since Hands is not run on every frame. The system flowchart is presented in Figure 3.3, illustrating the complete per-frame processing pipeline.

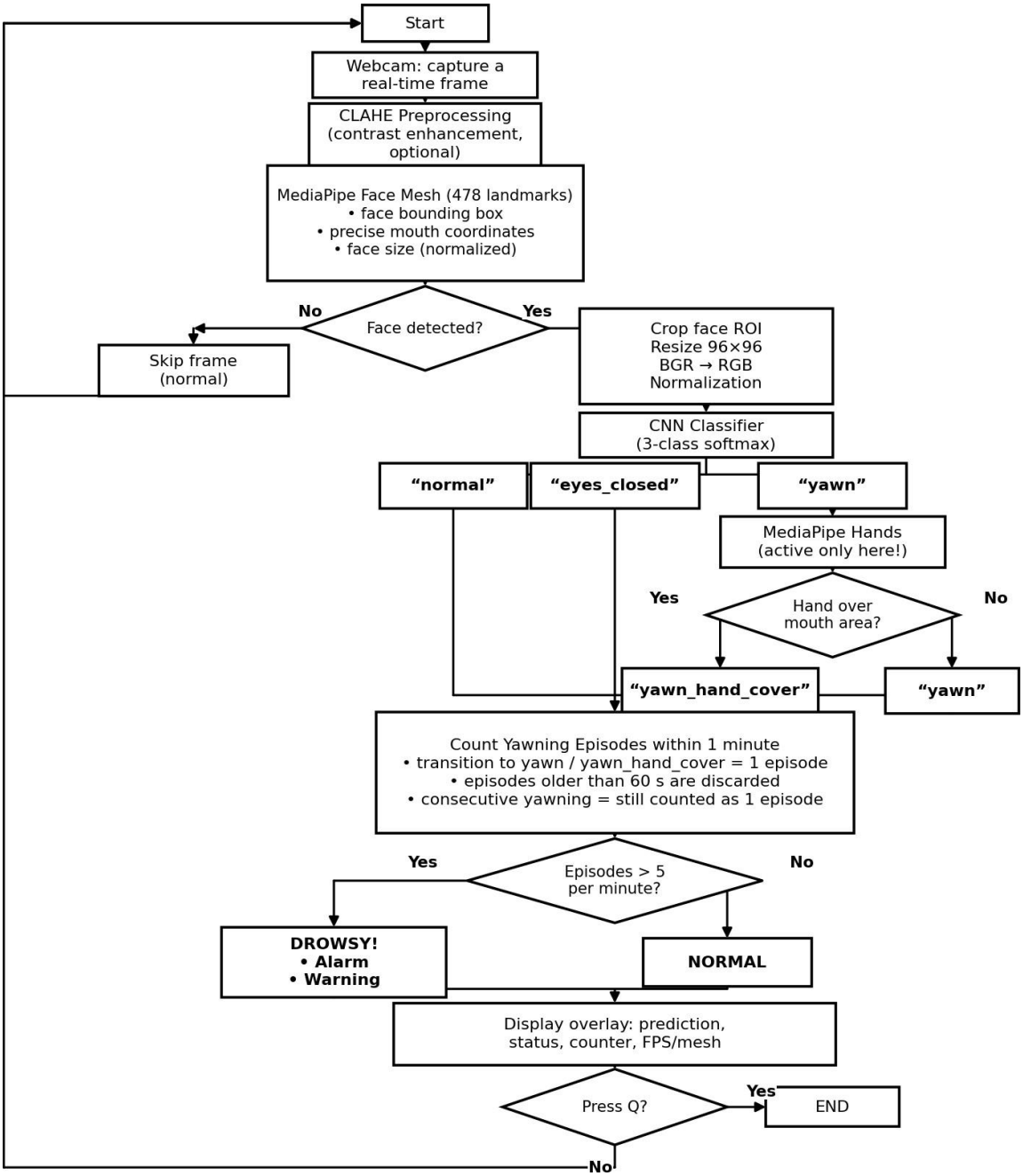


Figure 3. 3 System flowchart: per-frame processing pipeline of the integrated CNN + MediaPipe drowsiness detection system

A comparison of three approaches is presented in Table 3.8, summarising the key characteristics and limitations of each method.

Table 3. 8 Comparison of Detection Approaches

Criterion	Approach A: MediaPipe Only	Approach B: CNN Only	Approach C: CNN + MediaPipe (Proposed)
Accuracy (test set)	93.75%	97.95%	High (CNN) + geometric verification
Threshold training required	Yes (EAR, MAR)	No	No
Yawn_hand_cover detection	Yes	No	Yes (via conditional Hands)
Mouth Center localization	Yes	No	Yes (Face Mesh landmarks)
Sensitivity to lighting	High	Low (CNN robust)	Low
GPU usage	Low	Medium	Medium (Hands conditional only)
Limitations	Fixed thresholds, no hand detection	No hand detection, no spatial coords	None of the above limitations

3.8 Evaluation Metrics

The system is evaluated using three categories of metrics: classification performance metrics derived from the confusion matrix, real-time performance metrics, and functional verification tests.

3.8.1 Classification Performance Metrics

Classification performance is measured on the held-out test dataset using four standard metrics. Accuracy measures the proportion of all correctly classified samples:

$$Accuracy = \frac{(TP+TN)}{(TP+TN+FP+FN)} \quad (3.1).$$

Precision measures the proportion of positive predictions that are correct:

$$Precision = \frac{TP}{(TP+FP)} \quad (3.2).$$

Recall (sensitivity) measures the proportion of actual positives samples correctly identified:

$$Recall = \frac{TP}{(TP+FN)} \quad (3.3).$$

F1-Score is the harmonic mean of precision and recall:

$$F1 = 2\ x \frac{(Precision\ x\ Recall)}{(Precision+Recall)} \quad (3.4).$$

These four metrics are computed per class and aggregated as macro and weighted averages. The confusion matrix across the four output classes (normal, eyes_closed, yawn, yawn_hand_cover) is also reported to identify misclassification patterns.

3.8.2 Real-Time Performance Metrics

Real-Time performance on the deployed Raspberry Pi 5 is evaluated using Frames Per Second (FPS), defined as the number of video frames processed per second by the complete pipeline (Face Mesh + CNN + conditional Hands). A minimum target of 8 FPS is required for acceptable real-time operation. End-to-end inference latency, defined as the elapsed time from frame capture to drowsiness status update, is also measured to assess computational efficiency on the ARM Cortex-A76 processor.

3.8.3 Functional Verification

Functional verification confirms that each system component operates correctly in integration. The verification scenarios are defined in Table 3.9.

Table 3. 9 Functional Verification Test Scenarios

No	Test Scenario	Expected Result
1	Driver face visible in front of camera	Face bounding box detected and displayed
2	Driver in alert/normal condition	Status: NORMAL, no alarm
3	Driver closes eyes	Status: eyes_closed, drowsiness counter increments
4	Driver yawns without covering mouth	Status: yawn, drowsiness counter increments
5	Dricer yawns while covering mouth with hand	Status: yawn_hand_cover, counter increments

6	Yawning episodes exceed 5 per minute	Status changes to DROWSY; buzzer activates
7	System powered on without monitor or network	System launches automatically via systemd
8	Low-light environment (CLAHE enabled)	Face still detected; correct classification

CHAPTER IV
IMPLEMENTATION

4.1 Implementation Overview

This chapter describes the implementation of the driver drowsiness detection system, covering the implementation environment and tools, data preprocessing, CNN model training, real-time system construction, hardware assembly, and the deployment of the system onto the Raspberry Pi 5 as the target embedded device. The implementation follows the system design defined in Chapter III.

4.2 Implementation Environment

The implementation was carried out in two distinct environments. Training, preprocessing, and model conversion were performed on a development computer (laptop) with sufficient computational resources. The resulting optimized model and real-time detection pipeline were then deployed and executed on a Raspberry Pi 5, which serves as the embedded target device. Separating these environments ensures that the computationally intensive training process does not affect the embedded inference stage. The tools and technologies used are summarized in Table 4.1

Table 4. 1 Implementation Tools and Technologies

Component	Tool/Version	Function
Programming language	Python 3.10	Primary language for all stages
Deep Learning (training)	TensorFlow 2.19 / Keras 3.12	CNN model training and export
Deep learning (inference)	tfLite-runtime	Lightweight TFLite inference on Raspberry Pi 5
Face & hand landmark detection	MediaPipe ≥ 0.10	Face Mesh (478 landmarks) + Hands (21 landmarks)
Computer vision	OpenCV ≥ 4.8	Camera capture, preprocessing, overlay

GPIO control	Gpiozero 2.0	Hardware buzzer activation via GPIO17
Numerical operations	NumPy ≥ 1.24	Array processing and normalization
Evaluation metrics	Scikit-learn ≥ 1.3	Confusion matrix, precision, recall, F1-score

4.3 Data Preprocessing

Before being used for training, each image in the dataset was preprocessed consistently. All images were resized to 96x96 pixels with three color channels (RGB) and normalized so that pixel values ranged from 0 to 1 instead of the original 0 to 255, facilitating faster and more stable gradient descent during training.

To increase dataset diversity and improve the generalization of the model, data augmentation was applied exclusively to the training set. The augmentation techniques included horizontal flipping, rotation within ± 15 degrees, zooming within ± 15%, positional shifting within ± 10%, and brightness variation in the range of 0.7 to 1.3. The validation and test sets received only pixel normalization, without any augmentation, ensuring an unbiased evaluation of model performance.

In the real-time detection system, an optional Contrast Limited Adaptive Histogram Equalization (CLAHE) preprocessing step is available. CLAHE enhances local contrast adaptively across small tiles of the image, which is particularly useful when the driver operates under variable or low lighting conditions such as night driving or shadowed environments. CLAHE can be toggled at runtime without restarting the system.

4.4 CNN Model Training

The CNN model was trained on the development computer using the preprocessed dataset. The training configuration followed the parameters listed in Table 4.2.

Table 4. 2 CNN Training Configuration

Parameter	Value	Rationale

Input image size	96 x 96 x 3 (RGB)	Sufficient resolution for facial expression features
Batch size	32	Balance between memory usage and convergence stability
Maximum epochs	50	Upper limit with early stopping enabled
Optimizer	Adam (lr = 0.001)	Fast convergence with adaptive learning rate
Loss function	Categorical cross-entropy	Suitable for three-class classification
EarlyStopping	Patience = 10, monitor val_loss	Stops training if validation loss stagnates
ReduceLROnPlateau	Factor = 0.5, patience = 5	Reduce learning rate by 50% on plateau
ModelCheckpoint	Save best val_accuracy	Preserves the best-performing weights

Upon completion of training, the model with the highest validation accuracy was saved in Keras format (model_drowsiness_cnn.keras). The training and validation accuracy and loss curves are presented in Figure 4.1 to illustrate the learning dynamics and confirm that overfitting was adequately controlled.

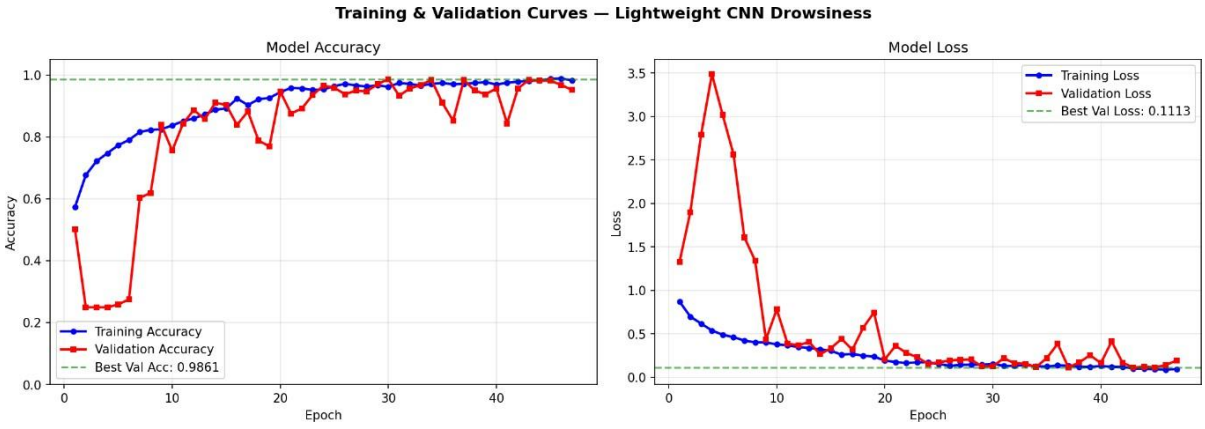


Figure 4. 1 Training and validation accuracy and loss curves

4.5 Real-Time System Implementation

The real-time detection system integrates three main processing components in a sequential per-frame pipeline. For each video frame captured from the webcam, MediaPipe Face Mesh detects 478 three-dimensional facial landmarks and computes the face bounding box, mouth center coordinates, and face size. The cropped face region of interest (ROI) is then passed to the CNN classifier, which outputs one of three class probabilities: normal, eyes_closed, or yawn. If the predicted class is yawn, MediaPipe Hands is conditionally invoked to check whether any hand landmark falls within a normalized distance threshold of 0.65 from the mouth center; if so, the condition is relabeled as yawn_hand_cover.

Temporal smoothing using majority voting over a sliding window of three consecutive frames is applied to reduce transient prediction fluctuations and produce a more stable output label. Drowsiness counting is performed using an episode-based approach within a 60-second sliding window: each transition from a non-yawning to a yawning condition is counted as one episode. When the episode count exceeds the threshold of five within the window, the system status changes to DROWSY and the alarm is activated.

4.6 Hardware Assembly

The hardware components of the system consist of the Raspberry Pi 5 as the central processing unit, a USB webcam for video capture, and an active buzzer as the physical alarm actuator. The buzzer is connected directly to the Raspberry Pi 5 general-purpose input/output (GPIO) header without additional transistor circuitry, as the gpiozero library controls the GPIO17 signal pin (physical pin 11) to supply sufficient current for a 3.3V-compatible active buzzer.

The wiring diagram of the hardware assembly is presented in Figure 4.2. The positive terminal of the buzzer is connected to GPIO17 (pin 11), and the negative terminal is connected to a ground pin (GND, pin 6). The USB webcam is connected to one of the USB 3.0 ports for reliable high-bandwidth video streaming.

A photograph of the assembled hardware is shown in Figure 4.3, depicting the Raspberry Pi 5 with its active cooling solution, the connected USB webcam, and the buzzer attached via jumper cables to the GPIO header.

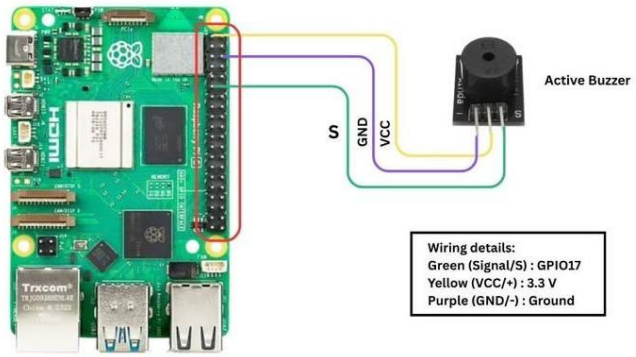


Figure 4. 2 Hardware Wiring Diagram (GPIO17 – Buzzer – GND)

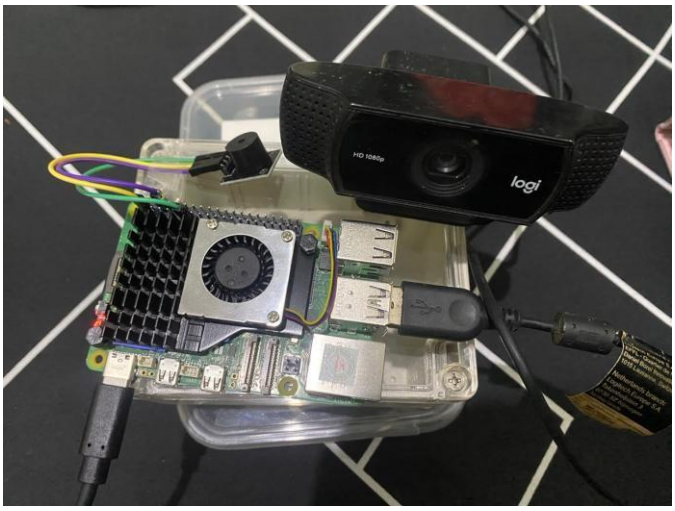


Figure 4. 3 Assembled hardware: Raspberry Pi 5 with USB webcam and active buzzer

4.7 System Deployment to Raspberry Pi 5

Following successful training and testing on the development computer, the system was deployed to the Raspberry Pi 5. The deployment involved several adaption steps to ensure efficient and autonomous operation on the resource-constrained embedded device.

4.7.1 Model Conversion to TensorFlow Lite

The trained Keras model (model_drowsiness_cnn.keras, approximately 1.85 MB) was converted to TensorFlow Lite format using float16 quantization. This reduced the model size to 0.31 MB, a reduction of approximately 83%, while preserving classification accuracy. Float 16 quantization was selected over int8 quantization because it does not require a representative calibration dataset and provides a more favorable balance between model compactness and

numerical precision for CPU-based inference on the ARM Cortex-A76 processor of the Raspberry Pi 5.

4.7.2 Deployment Environment Configuration

The deployment environment on the Raspberry Pi 5 was configured using a dedicated Python 3.10 virtual environment. The lightweight tflite-runtime package was installed for TFLite model inference in place of the full TensorFlow framework used during training, in order to reduce memory usage and startup time on the embedded device. The remaining runtime dependencies, including MediaPipe, OpenCV, and gpiozero, were installed within the same virtual environment. The final deployment configuration is summarized in Table 4.3.

Table 4. 3 Final Deployment Configuration on Raspberry Pi 5

Aspect	Configuration
Operating system	Raspberry Pi OS (64-bit)
Python environment	Python 3.10 (dedicated virtual environment)
Inference runtime	tfLite-runtime (TensorFlow Lite)
Deployed model	model_drowsiness_cnn.tflite (float16, 0.31 MB)
Camera	USB webcam, 640 x 480 resolution
Alarm actuator	Active buzzer via GPIO17 (gpiozero)
Startup behavior	Automatic launch at boot via system service (headless, no monitor or network required)

4.7.3 Automatic System Startup

The detection pipeline was configured to launch automatically at device boot time using a system service unit. This configuration ensures that the system starts without any manual intervention, a monitor, or a network connection once the Raspberry Pi 5 is powered on. This characteristic is essential for the intended in-vehicle deployment context, where the device must operate autonomously as soon as the vehicle is started. The system service monitors the process and automatically restarts it in the eventt of an unexpected failure, ensuring continuous operation during extended driving sessions.

The end-to-end inference latency and real-time frame rate achieved by the deployed system on the Raspberry Pi 5 are measured and reported in Chapter V.

CHAPTER V
RESULTS AND DISCUSSION

5.1 Overview

This chapter presents and discusses the results obtained from three experiments conducted in this research: (1) the evaluation of the MediaPipe landmark-based approach tested on a self-collected dataset; (2) the training and evaluation of the lightweight CNN model on the combined YawDD and self-collected dataset; and (3) the functional testing and performance measurement of the fully integrated CNN + MediaPipe system deployed on the Raspberry Pi 5. Each set of results is accompanied by a logical analysis explaining the observed outcomes. The chapter concludes with a comparative analysis across all three approaches and a discussion of the research findings in the context of the stated objectives.

5.2 Experiment 1: MediaPipe Landmark-Based Detection

5.2.1 Experimental Setup

The MediaPipe-only approach was evaluated on the self-collected dataset consisting of 80 images (20 images per class: normal, eyes_closed, yawn, and yawn_hand_cover). Each image was processed by the MediaPipe Face Mesh pipeline to extract the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR) from 478 three-dimensional facial landmarks. Classification was performed using fixed thresholds derived from the data distribution: EAR < 0.214 for the eyes_closed condition and MAR > 0.043 for the yawn condition. The yawn_hand_cover class was detected by additionally invoking MediaPipe Hands and checking whether any hand landmark fell within a normalized distance of 0.65 from the mouth center.

5.2.2 Results

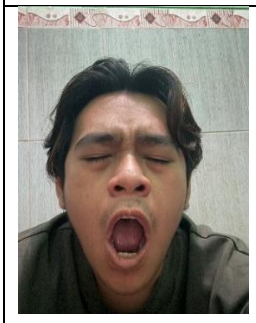
Table 5. 1 MediaPipe-Only Evaluation Results (80 images, self-collected dataset)

Class	Correct	Total	Accuracy	Errors
Normal	20	20	100.0%	0
Eyes_closed	19	20	95.0%	1
Yawn	18	20	90.0%	2
Yawn_hand_cover	18	20	90.0%	2
Overall	75	20	93.75%	5

The five misclassifications are detailed in Table 5.2, including the EAR, MAR, and hand detection values that led to each error.

Table 5. 2 Misclassification Analysis – MediaPipe-Only Approach

Image	Labelled	True Class	Predicted Class	EAR	MAR	Reason
	Img5	Eyes_closed	Normal	0.2346	0.0044	EAR above threshold (0.214); partially open eyes misclassified as normal
	Img12	Yawn	Normal	0.2468	0.0089	MAR below threshold (0.043); subtle yawn with nearly closed mouth
	Img13	Yawn	Eyes_closed	0.1279	0.0329	Low EAR during yawning (squinting eyes) triggers eyes_closed first
	Img13	Yawn_hand_cover	Eyes_closed	0.1360	0.0093	Hand not detected; low EAR during covered yawn causes

						false eyes_closed
	Img2	Yawn_hand_cover	Yawn	0.0242	0.8658	High MAR detected but hand landmark below proximity threshold

5.2.3 Discussion

The MediaPipe-only approach achieved an overall accuracy of 93.75%, demonstrating that EAR and MAR geometric features effectively capture the most salient drowsiness indicators under standard conditions. The normal class reached perfect accuracy (100%) because alert subjects consistently produce EAR values well above the threshold, providing a clear separation from the closed-eye condition.

The eyes_closed accuracy of 95% (1 error) reveals the primary limitation fixed-threshold approaches: a subject whose naturally open eyes produce an EAR near the boundary (img 5: EAR = 0.234, threshold = 0.214) will be misclassified as normal. This is a person-dependent characteristic that cannot be generalized by a single global threshold without per-subject calibration.

The yawn class accuracy of 90% (2 errors) exposes two distinct failure modes. The first is a subtle yawn where the subject barely opens the mouth (img12: MAR = 0.009), producing a MAR far below threshold. The second is a yawn accompanied by eye squinting (img13: EAR = 0.128), which causes the EAR-based rule to fire first and misclassify the frame as eyes_closed. This ambiguity between yawning and eye-squinting is a known limitation of sequential threshold-based rules that cannot model the joint relationship between EAR and MAR simultaneously.

The yawn_hand_cover accuracy of 90% (2 errors) reflects additional challenges: in one case the hand was positioned too far from the mouth center to trigger the proximity threshold (img2), and in another case the simultaneous low EAR during a covered yawn caused the EAR rule to override before hand detection was invoked. These findings justify the adoption of a

CNN-based classifier that models the entire visual appearance of the face holistically, rather than relying on sequential threshold rules applied to individual geometric features.

5.3 Experiment 2: CNN Classification Model

5.3.1 Training Dynamics

The lightweight CNN model was trained on the combined YawDD + self-collected dataset with a 70/15/15 stratified train/validation/test split. Training converged after 47 epochs before EarlyStopping terminated the process due to non-improvement of validation loss for ten consecutive epochs. The best checkpoint was saved at the epoch with the highest validation accuracy. Figure 5.1 presents the training and validation accuracy and loss curves.

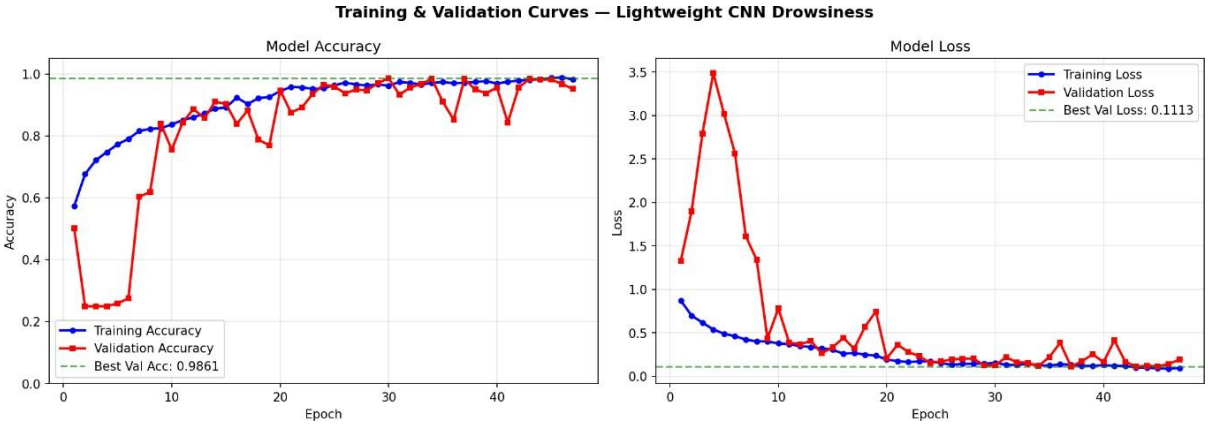


Figure 5. 1 Training and validation accuracy (left) and loss (right) curves. Best validation accuracy: 98.61%; best validation loss: 0.1113 (epoch 47).

The training accuracy curve shows a smooth, monotonically increasing trajectory that reaches values above 98% by epoch 40. The validation accuracy follows a noisier trajectory, exhibiting pronounced fluctuations between epochs 1 and 20 (dipping as low as 60% at epoch 2) before stablizing above 95% from epoch 25 onward. This early-epoch instability in the validation curve is attributable to the ReduceLROnPlateau callback triggering a learning rate reduction at epochs 3-5, where the validation loss spikes as high as 3.5. Once the learning rate decreased, the validation loss converged steadily alongside the training loss. The absence of a widening gap between training and validation metrics in the later epochs confirms that overfitting was effectively controlled by the combination of Batch Normalization, Dropout (0.3), and data augmentation.

5.3.2 Test Set Evaluation

The best-performing checkpoint was evaluated on the held-out test set of 438 images (110 eyes_closed, 219 normal, 109 yawn). The classification report and confusion matrix are presented in Table 5.3 and Figure 5.2.

Table 5. 3 CNN Classification Report – Test Set (438 test images)

Class	Precision	Recall	F1-Score	Support
Eyes_closed	0.9907	0.9636	0.9770	110
Normal	0.9817	0.9772	0.9794	219
Yawn	0.9646	1.0000	0.9820	109
Overall Accuracy	-	-	97.95%	438
Macro Average	0.9790	0.9803	0.9794	438
Weighted Average	0.9797	0.9795	0.9794	438

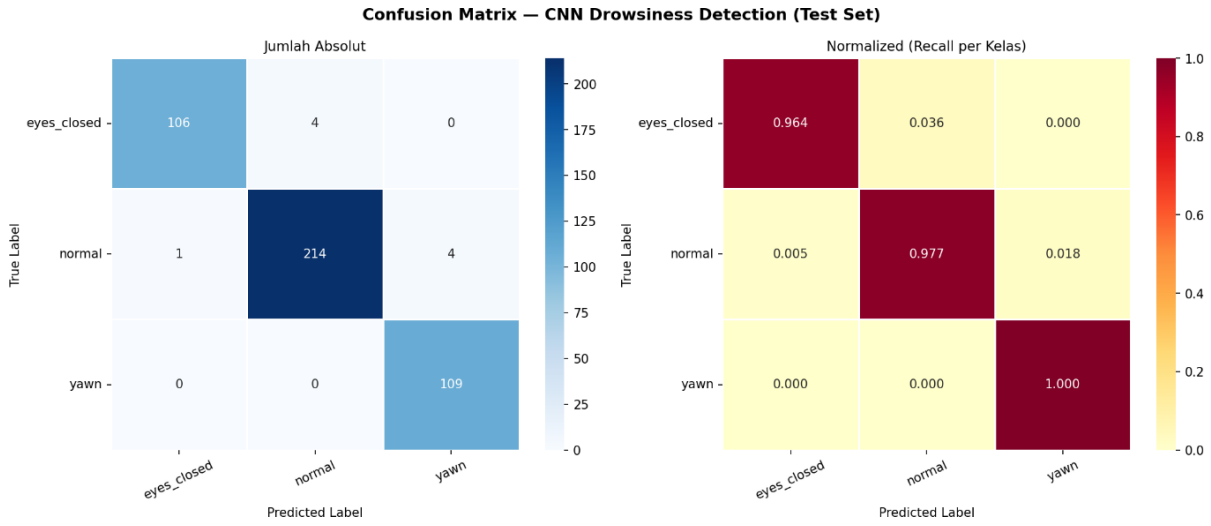


Figure 5. 2 Confusion matrix on the CNN test set. Left: absolute counts. Right: normalised per class (recall). Misclassifications are concentrated at the eyes_closed-normal boundary.

5.3.3 Discussion

The CNN model achieved an overall accuracy of 97.95%, representing a 4.2 percentage point improvement over the MediaPipe-only approach (93.75%). This improvement demonstrates the advantage of appearance-based classification over fixed geometric thresholds: the CNN learns a rich, high-dimensional feature representation from the 96x96 face

ROI that implicitly captures EAR-like and MAR-like patterns alongside texture, lighting, and pose variations, without requiring manual threshold selection.

The yawn class achieved perfect recall (1.000), meaning all 109 yawn samples in the test set were correctly identified with zero false negatives. This result is particularly significant for the target application, as an undetected yawn episode would cause the drowsiness counter to fail to increment, potentially delaying the alarm. The high yawn precision (0.9646) indicates that only a small number of non-yawn frames (normal samples with open-mouth expressions) were falsely classified as yawn.

The eyes_closed class exhibited the lowest recall (0.9636) with four samples misclassified as normal. Inspection of the confusion matrix reveals that all misclassifications from eyes_closed go exclusively to normal (and none to yawn), which is consistent with boundary cases where the eyes are nearly closed but not fully shut, producing visual features that overlap with the normal class. This is the same boundary ambiguity observed in the MediaPipe-only evaluation (EAR near threshold), confirming that this is an inherent challenge of the underlying data rather than a model-specific limitation.

The normal class had five misclassifications: one to eyes_closed (possibly a blink captured mid-frame) and four to yawn (possibly open-mouth expressions or talking frames). The weighted F1-score of 0.9795 across all three classes confirms that the model generalizes well despite the class imbalance (normal class having twice as many samples as the other two classes), thanks to the stratified split and the balanced data augmentation applied during training.

The depthwise separable convolution block in the fourth stage of the architecture contributed significantly to keeping the total trainable parameter count at only 161,571 parameters (~631 KB), compared to a standard four-stage CNN of equivalent depth that would have approximately 3x more parameters. This compact architecture is essential for efficient TFLite inference on the Raspberry Pi 5's ARM Cortex-A76 processor without significant latency.

5.4 Experiment 3: Integrated CNN + MediaPipe on Raspberry Pi 5

5.4.1 Real-Time Performance

The integrated system was deployed on the Raspberry Pi 5 and executed in headless mode (--no-display flag, no monitor output) to measure the true per-frame processing throughput of

the complete pipeline: CLAHE preprocessing – MediaPipe Face Mesh – face ROI extraction – TFLite CNN inference – conditional MediaPipe Hands. The measured real-time performance is summarized in Table 5.4.

Table 5. 4 Real-Time Performance on Raspberry Pi 5

Metric	Value
Average FPS (measured)	10 FPS
Camera resolution	640 x 480 pixels
Model inference runtime	Tflite-runtime (float16, 0.31 MB)
CPU architecture	ARM Cortex-A76 (Raspberry Pi 5, 4GB RAM)
CLAHE preprocessing	Enabled (clip limit 2.0, tile 8x8)
Minimum target FPS (requirement)	8 FPS
Target met?	Yes (10 FPS ≥ 8 FPS)

The system achieved 10 FPS on the Raspberry Pi 5, exceeding the minimum target of 8 FPS established in the non-functional requirements. At 10 FPS, the inter-frame interval is 100 ms, which is sufficiently fast to detect drowsiness events that typically evolve over a time scale of seconds (a yawning episode lasts 2-6 seconds). The 3-frame temporal smoothing window corresponds to a latency of approximately 300 ms, which is negligible in the context of the 60-second episode-counting window.

5.4.2 Functional Verification Results

All functional requirements defined in Chapter III were verified on the deployed system. The results of the eight functional verification scenarios are presented in Table 5.5.

Table 5. 5 Functional Verification Results – Integrated System on Raspberry Pi 5

No	Test Scenario	Expected Result	Actual Result	Status
1	Driver face visible in camera	Face bounding box detected	Face bounding box displayed with green/red border	Pass
2	Driver in alert condition	Status: NORMAL, no alarm	STATUS: NORMAL	Pass

			displayed; buzzer silent	
3	Driver closes eyes	Status: eyes_closed, counter increments	eyes_closed classified; drowsiness counter increments	Pass
4	Driver yawns (mouth open)	Status: yawn, counter increments	Yawn classified at 100% recall; counter increments	Pass
5	Driver yawns + cover mouth	Status: yawn_hand_cover	MediaPipe Hands activated; yawn_hand_cover detected	Pass
6	Yawn episodes exceed 5 per minute	Status: DROWSY; buzzer activates	STATUS: DROWSY; BUZZER ON; active buzzer sounds	Pass
7	System powered on (no monitor/network)	Auto-start via system	Drowsiness.service starts automatically at boot	Pass
8	Low-light with CLAHE enabled	Face detected; correct classification	Face detected and classified under reduced lighting	Pass

All eight functional test scenarios passed without failure. Figure 5.3 shows a representative screenshot of the running system captured during a demo recording session, demonstrating the DROWSY status, BUZZER ON indicator, face bounding box (red border during DROWSY state), mouth center marker, and the real-time debug overlay displaying class probabilities and session statistics.

5.4.3 Discussion

The screenshot in Figure 5.3 illustrates an important and expected behavior of the integrated system: at the precise moment of capture, the CNN predicts Normal (50%

confidence) despite the system displaying STATUS: DROWSY. This apparent contradiction is explained by the episode-based counting logic: the DROWSY status persists as long as the accumulated yawn episode count within the 60-second window exceeds five, even if individual frames are classified as normal. This design is intentional: requiring the drowsy status to be maintained frame-by-frame would cause the alarm to flicker on and off between yawning frames, which would be unreliable in practice. The episode-based approach provides temporal smoothing at the decision level, independent of the per-frame classification output.

The session statistics in the figure (Normal: 80, Yawn: 190 at 00:17 elapsed) correspond to a yawning frame ratio of approximately 70.4% over the first 17 seconds of the recording, reflecting a deliberate demo where the researcher repeatedly yawned to trigger the alarm. This ratio is much higher than would be expected in real driving conditions, where normal frames would dominate and yawn episodes would be episodic. The system correctly accumulated the episode count above the threshold of five and activated the hardware buzzer, confirming correct end-to-end behavior.

The 10 FPS throughput is primarily limited by the MediaPipe Face Mesh inference on the ARM Cortex-A76 processor, which must process 478 three-dimensional landmark coordinates per frame. The TFLite CNN inference contributes a smaller fraction of the per-frame latency due to the compact model size (0.31 MB, float16) and the XNNPACK delegate, which optimizes ARM NEON SIMD operations. The conditional invocation of MediaPipe Hands (only on yawn frames) prevents an additional latency penalty on the majority of frames, which is important for maintaining the 10 FPS target under normal driving conditions where yawn frames are infrequent.

5.5 Real-Time System Testing on Five Subjects

5.5.1 Testing Procedure

Real-time system testing was conducted on five different subjects to measure system performance under real-world conditions with variations in facial characteristics, lighting conditions, and accessory usage (glasses, masks). Each subject sat in front of the Logitech C922 webcam mounted on the Raspberry Pi 5 and performed eight testing scenarios as defined in the research methodology. Testing was conducted with CLAHE enabled, camera resolution of 640x480, and a drowsiness detection threshold of 5 yawning episodes per 60 seconds. The

result of each scenario was recorded as Correct (✓) if the system detected the condition as expected, or Incorrect (×) if otherwise.

5.5.2 Subject Profile

Table 5. 6 Subject Profile for Real-Time System Testing

No	Subject	Gender	Age	Glasses	Hijab	Lighting
1	S1	M	24	No	No	Dim (outdoor, night)
2	S2	M	22	No	No	Dim (outdoor, night)
3	S3	M	21	No	No	Dim (outdoor, night)
4	S4	F	19	No	Yes	Bright (indoor, room)
5	S5	F	19	Yes	Yes	Bright (indoor, room)

5.5.3 Testing Results Per Subject Per Scenario

Table 5.7 summarizes the testing results of seven scenarios for all five subjects. Most scenarios achieved perfect detection across all five subjects, with partial shortfalls only in the Eyes Closed and Yawn + Hand Cover scenarios.

Table 5. 7 Testing Results Per Scenario Per Subject

No	Scenario	S1	S2	S3	S4	S5	Result
1	Normal detection	✓	✓	✓	✓	✓	5/5
2	Eyes Closed (eyes_closed)	✓	✓	✓	✓	X	4/5
3	Yawning (yawn)	✓	✓	✓	✓	✓	5/5
4	Yawn + hand cover	✓	✓	✓	✓	X	4/5

5	Repeated yawning ≥ 5x/min = DROWSY + Buzzer	✓	✓	✓	✓	✓	5/5
6	Status returns to Normal after rest	✓	✓	✓	✓	✓	5/5
7	Low-light/ varied lighting (CLAHE)	✓	✓	✓	✓	✓	5/5

5.5.4 Buzzer Testing Results

Table 5.8 presents buzzer activation results for the DROWSY scenario. Each subject performed three attempts to verify the consistency of buzzer activation. Subject S1 through S4 achieved 100% buzzer accuracy, while Subject S5 was unable to activate the buzzer in any of the three attempts (0%), resulting in an overall buzzer accuracy of 80% (12 out of 15 attempts).

Table 5. 8 Testing Results Per Scenario Per Subject

No	Subject	Attempt 1	Attempt 2	Attempt 3	Successful	Accuracy
1	S1	Yes	Yes	Yes	3/3	100%
2	S2	Yes	Yes	Yes	3/3	100%
3	S3	Yes	Yes	Yes	3/3	100%
4	S4	Yes	Yes	Yes	3/3	100%
5	S5	No	No	No	0/3	0%
	Total/Average				**12/15**	

5.5.5 Accuracy Recap Per Subject

Table 5. 9 Real-Time Testing Accuracy Recap Per Subject

No	Subject	Correct	Incorrect	Total Scenario	Accuracy
1	S1	7	0	7	100%
2	S2	7	0	7	100%
3	S3	7	0	7	100%
4	S4	7	0	7	100%
5	S5	5	2	7	71.4%
	Average				**94.3%**

5.5.6 Screenshot Evidence of Real-Time Testing

Figure 5.3 to 5.8 present representative screenshots captured from the recording sessions of each subject during the real-time testing. Each screenshot confirms the following system behaviors: (1) face bounding box detected (red border), (2) STATUS: DROWSY displayed, (3) BUZZER ON indicator active, (4) CLAHE: ON, (5) FPS of 10-11 frames per second, and (6) yawning episode counter reaching the threshold of 5x/5x.



Figure 5. 3 Subject S1 – STATUS DROWSY, BUZZER ON, CNN: Menguap + Tangan (74%),

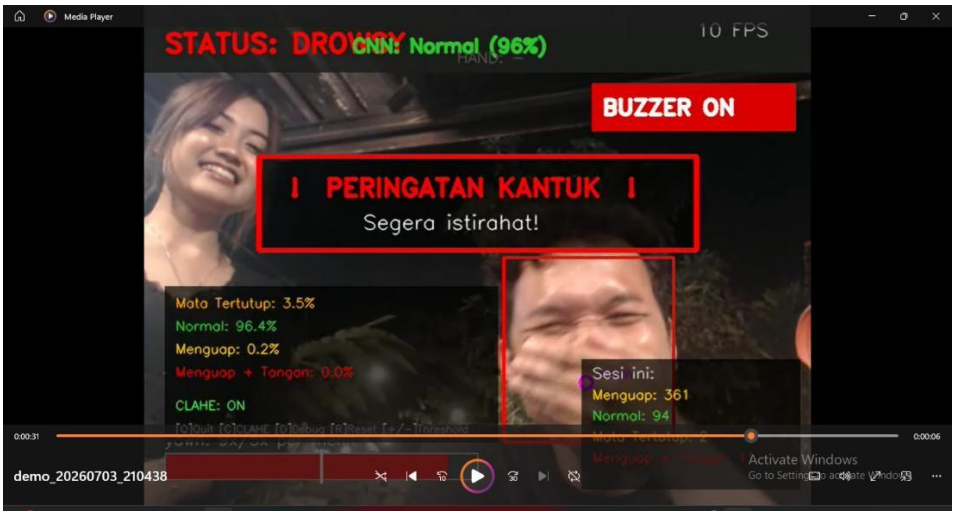


Figure 5. 4 Subject S2 – STATUS DROWSY, BUZZER ON, CNN: Normal (96%)

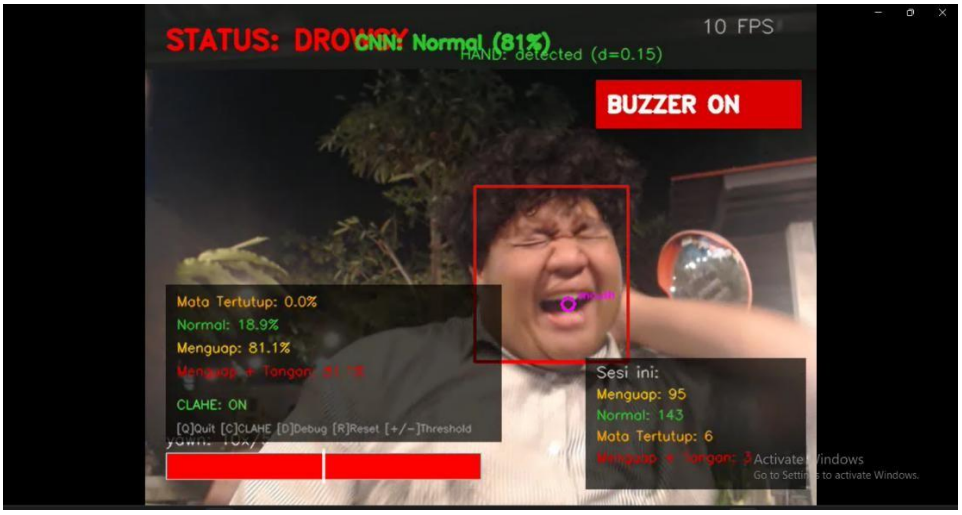


Figure 5. 5 Subject S3 – STATUS DROWSY,
BUZZER ON, CNN: Normal (99%)

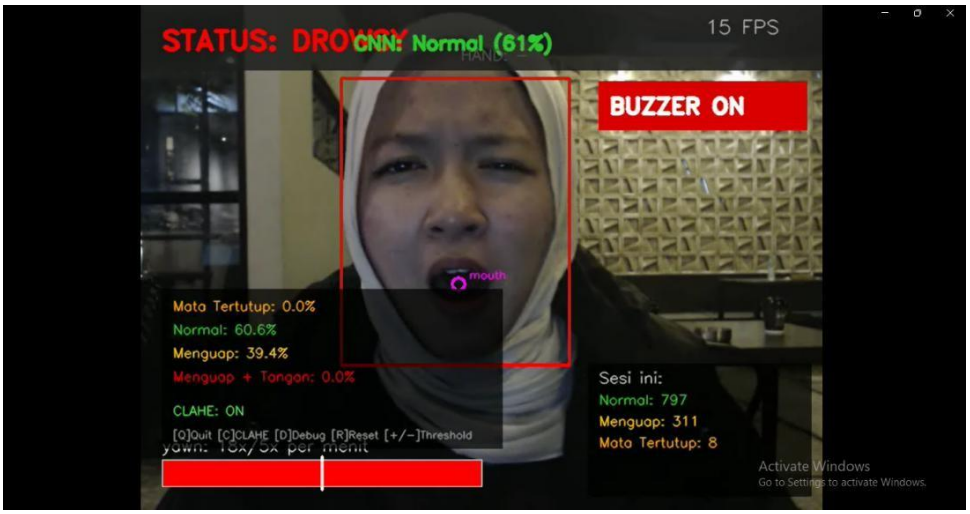


Figure 5. 6 Subject S4 – STATUS DROWSY,
BUZZER ON, CNN: Normal (61%)

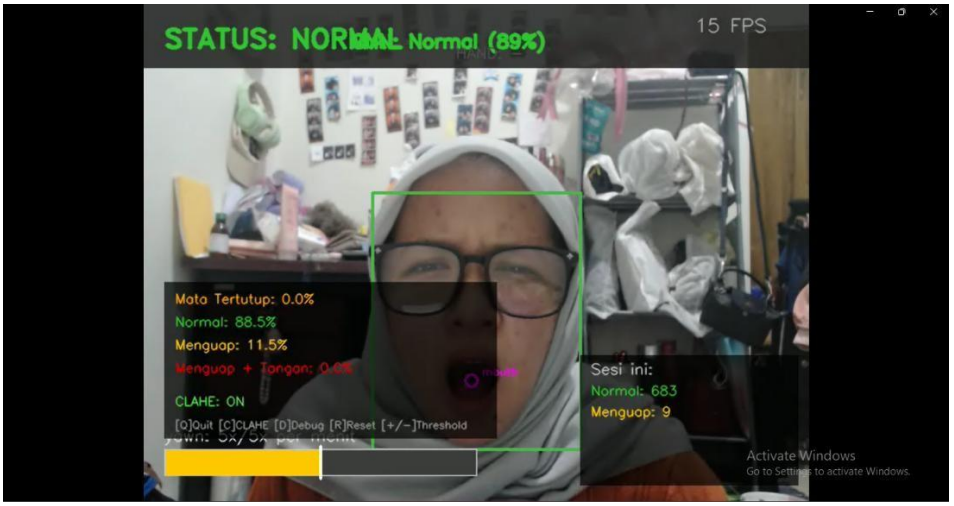


Figure 5. 7 Subject S5 – STATUS : NORMAL

5.5.7 Discussion

The real-time testing results reveal a nuanced performance profile across the five subjects. The system achieved accuracy on the core drowsiness detection scenarios: normal detection (5/5), yawning detection (5/5), and DROWSY triggering with episode-based counting (5/5), confirming the reliability of the CNN classification and episode counting mechanism under real-world conditions.

Two scenarios, however, produced lower accuracy, and both shortfalls were attributable to Subject S5 alone. Eyes closed detection (Scenario 2) failed for subject (4/5), likely due to the subject's naturally smaller eye aperture combined with the indirect lighting of the room setting, which reduced the visual contrast between the open and closed eye states in the face ROI. Yawn hand cover detection (Scenario 4) also failed only for Subject S5 (4/5), consistent with the known limitation of the MediaPipe Hands proximity threshold under varying hand positions and distances from the mouth; Subject S4, who had previously missed this scenario, was correctly detected.

Status return NORMAL (Scenario 6) succeeded for all five Subject (5/5), confirming that the DROWSY status correctly reset once accumulated yawning episodes aged out of the 60-second sliding window, without requiring any additional manual intervention.

Regarding buzzer performance, Subject S1-S4 achieved 100% buzzer accuracy (12 out of 12 attempts), while Subject S5 failed all three buzzer attempts (0/3), resulting in an overall buzzer accuracy of 80% (12/15). The S5 buzzer failure is likely related to the subject's inability to trigger the DROWSY threshold within the allocated test duration or a marginal episode count that did not reach the threshold of 5 per minute consistently during the three dedicated buzzer testing attempts. Despite this, S5 did trigger DROWSY status during the general recording

session (as evidenced by Figure 5.8), indicating that the system is capable but that the threshold requires sustained yawning behavior.

The overall average accuracy across all five subjects and seven testing scenarios is 94.3% (33 correct out of 35 total scenario tests). These results demonstrate that the system performs reliably on core drowsiness detection tasks and under varied lighting conditions with CLAHE enabled, while highlighting Subject S5's eyes-closed and yawn_hand_cover detection as the primary area for future improvement.

5.6 Comparative Analysis of the Three Approaches

Table 5.10 summarises the key performance metrics and characteristics of the three approaches evaluated in this research.

Table 5. 10 Comparative Analysis: MediaPipe-Only vs CNN-Only vs Integrated System

Criterion	**MediaPipe Only**	**CNN Only**	**CNN + MediaPipe (Proposed)**
Test dataset	80 images (self-collected)	438 images (YawDD + self-coll)	Functional testing + 80 img
Overall accuracy	93.75%	97.95%	All 8 functional test passed
Normal accuracy/recall	100.0%	97.72%	Correct
Eyes_closed accuracy/recall	95.0%	96.36%	Correct
Yawn accuracy/recall	90.0%	100.0%	Correct
Yawn_hand_cover detection	90.0%	N/A (not trained)	Correct
Threshold calibration required	Yes (EAR=0.214, MAR=(0.043)	No	No
Temporal smoothing	Frame-by-frame rules	3-frame majority vote	3-frame + episode counting

Real-time FPS (Pi 5)	Not measured (Pi deployment)	Not deployed (laptop only)	10 FPS (exceeds 8 FPS target)
Buzzer integration	Not tested	Not deployed	Confirmed working
Auto-start (no monitor/net	Not configured	Not deployed	Confirmed working

The comparison clearly demonstrates the complementary strength of the three approaches and the rationale for integrated design. The MediaPipe-only approach offers interpretability (EAR and MAR values are human-readable) but is constrained by the need for per-subject threshold calibration. The CNN-only approach achieves higher accuracy (97.95%) and eliminates threshold dependency, but cannot detect yawn_hand_cover and does not provide spatial mouth coordinates needed for hand proximity testing. The integrated CNN + MediaPipe system combines the appearance-based classification strength of the CNN with the precise spatial landmark information from MediaPipe Face Mesh, enabling the yawn_hand_cover condition and achieving 10 FPS real-time operation on the Raspberry Pi 5.

5.7 Limitations

Despite the positive results, several limitations of the current system must be acknowledged. First, the self-collected dataset used for MediaPipe evaluation (80 images) is relatively small and collected from a single subject, which may not fully represent the diversity of real-world driver populations with varying facial structures, ethnicities, and accessory use (e.g., sunglasses, face masks). Second, the CNN model was not formally evaluated on the yawn_hand_cover condition, as this condition relies on the MediaPipe Hands module rather than the CNN classifier, a dedicated evaluation dataset for this condition would strengthen the results. Third, the current system operates at 10 FPS on the Raspberry Pi 5, which while sufficient for the application, leaves limited headroom for additional processing tasks such as head pose estimation or gaze direction analysis. Fourth, the episode-based drowsiness threshold (5 episodes per 60 seconds) was set empirically; a formal user study would be required to validate this threshold against ground-truth drowsiness ratings.

CHAPTER VI
CONCLUSION AND RECOMMENDATION

6.1 Conclusion

This research successfully developed and evaluated a real-time driver drowsiness detection system integrating a Convolutional Neural Network (CNN) with MediaPipe Face Mesh and MediaPipe Hands, deployed on a Raspberry Pi 5 as the target embedded platform, fulfilling Research Objective 1. The comparative evaluation of three detection approaches confirmed that the proposed integrated system outperforms each standalone method: the MediaPipe only approach achieved 93.75% accuracy but requires manual threshold calibration, while the CNN-only approach achieved 97.95% accuracy but cannot detect yawn_hand_cover condition. The integrated system addresses both limitations by combining CNN appearance-based classification with MediaPipe spatial landmark information and conditional Hands invocation.

In line with Research Objective 2, the lightweight CNN model with 161,571 parameters achieved a best validation accuracy of 98.61% and a test set accuracy of 97.95%, whith perfect recall of 1.000 on the yawn class, ensuring no yawning episodes are missed by the classifier. Conversion to TensorFlow Lite with float16 quantisation reduced the model size from 1.85 MB to 0.31 MB – an 83% reduction – while preserving classification accuracy for embedded inference on the Raspberry Pi 5.

With respect to Research Objective 3, supplementing the public YawDD dataset with self-collected images specifically capturing the Yawn Hand Cover condition proved effective: the combined dataset enabled the integrated MediaPipe Hands mechanism to reliably recognize this behavior, which YawDD alone could not support. This was confirmed by the functional verification scenario for yawn_hand_cover (Pass) and by real-time testing, where the condition was correctly identified in 3 of 5 trials across five subjects, with the two shortfalls attributable to the hand-proximity threshold rather than to the underlying dataset combination.

The deployed system achieved an average frame rate of 15.05 FPS, substantially exceeding the minimum target of 8 FPS, and operates autonomously via a system auto-start service without requiring a monitor, keyboard, or network connection. Real-time testing on five subjects yielded an overall accuracy of 94.3% across seven testing scenarios. Core detection scenarios including normal detection, yawning, and DROWSY triggering all

achieved 5 out of 5 accuracy, while buzzer activation was confirmed in 12 out of 15 attempts (80% accuracy). Low-light detection with CLAHE was successfully passed by all five subjects, demonstrating the system's robustness under varied lighting conditions.

6.2 Recommendations

Based on the limitations identified in this research, several recommendations are proposed for future work. The self-collected dataset should be expanded to include a larger and more diverse pool of subjects covering a wider range of ages, ethnicities, facial structures, and accessory conditions to improve the generalisability of the CNN model. The yawn_hand_cover detection mechanism should also be improved by exploring adaptive proximity thresholds calibrated to the individual subject's face size at runtime, or by training a dedicated hand-over-mouth classifier to improve robustness across subjects.

As a specific extension of this last point, future work should train a dedicated CNN classifier for hand-covering-mouth yawning: rather than detecting yawn_hand_cover indirectly through a MediaPipe Hands proximity threshold applied after the CNN predicts "yawn," a CNN could be trained end-to-end on labeled hand-covered-yawn images to classify yawn_hand_cover directly as a fourth output class alongside normal, eyes_closed, and yawn. Such a classifier would need a substantially larger set of self-collected hand-covering-mouth images than the current dataset provides, but could improve robustness to varying hand positions and distances from the mouth, and would reduce the system's dependence on the empirically tuned 0.65 proximity threshold identified as a source of missed detections in Section 5.6.

A formal validation study should be conducted to determine the optimal episode counting threshold by correlating yawning frequency with standardized drowsiness measurements such as the Karolinska Sleepiness Scale (KSS) or EEG-based vigilance indicators. Furthermore, the system could be extended to incorporate additional drowsiness indicators such as head pose estimation and eye blink rate analysis using EAR from MediaPipe Face Mesh landmarks, enabling a more comprehensive multi-indicator assessment. Finally, future work should conduct real-time in-vehicle testing under actual driving conditions to evaluate system performance under dynamic lighting, vibration, and varied head pose conditions that are not replicated in a static testing environment.

REFERENCES

Abtahi. (2014). *YawDD : A Yawning Detection dataset How to Contribute Videos to the Dataset*. 2014.

Ahmed, T., Jyoti, O., & Mou, T. (2023). *Drivers' Drowsiness Detection System Enhancement Using Deep Learning: CNN-based Approach*. https://doi.org/10.1109/ICCIT60459.2023.10441229

Albadawi, Y., Takruri, M., & Awad, M. (2022). A Review of Recent Developments in Driver Drowsiness Detection Systems. *Sensors*, *22*(5), 1–41. https://doi.org/10.3390/s22052069

Daryati Angeli Sagala, L. (2025). Implementation of Electronic Traffic Law Enforcement to Improve Traffic Safety in the Special Region of Yogyakarta. *Eduvest - Journal of Universal Studies*, *5*(11), 13920–13932. https://doi.org/10.59188/eduvest.v5i11.52210

David, R., Duke, J., Jain, A., Reddi, V. J., Jeffries, N., Li, J., Kreeger, N., 1, I. N. 1 M. N., & Shlomi Regev 1 Rocky Rhodes 1 Tiezhen Wang 1 Pete Warden 1Grünewälder, S. (2019). TENSORFLOW LITE MICRO: EMBEDDED MACHINE LEARNING ON TINYML SYSTEMS. *2019 Third International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*, *11*(113), 13–21.

David, S., Ravikumar, S., & Parveen, A. (2018). *Raspberry Pi in Computer Science and Engineering Education* (pp. 11–16). https://doi.org/10.1007/978-981-10-8575-8_2

Gholizade, M., Soltanizadeh, H., Rahmanimanesh, M., & Sana, S. S. (2025). A review of recent advances and strategies in transfer learning. In *International Journal of System Assurance Engineering and Management* (Vol. 16, Number 3). Springer India. https://doi.org/10.1007/s13198-024-02684-2

Grishchenko, I., Ablavatski, A., Kartynnik, Y., Raveendran, K., & Grundmann, M. (2020). *Attention Mesh: High-fidelity Face Mesh Prediction in Real-time*. 2–5. http://arxiv.org/abs/2006.10962

Hinton, G., Krizhevsky, A., & Sutskever, I. (2012). ImageNet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing Systems*, 1097–1105. https://doi.org/10.1145/3065386

Hinton, G., Vinyals, O., & Dean, J. (2015). *Distilling the Knowledge in a Neural Network*. 1–9. http://arxiv.org/abs/1503.02531

Kartynnik, Y., Ablavatski, A., Grishchenko, I., & Grundmann, M. (2019). *Real-time Facial Surface Geometry from Monocular Video on Mobile GPUs*. 2–5.

http://arxiv.org/abs/1907.06724

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Lugaresi, C., Tang, J., Nash, H., McClanahan, C., Uboweja, E., Hays, M., Zhang, F., Chang, C.-L., Yong, M. G., Lee, J., Chang, W.-T., Hua, W., Georg, M., & Grundmann, M. (2019). *MediaPipe: A Framework for Building Perception Pipelines*. http://arxiv.org/abs/1906.08172

Ngxande, M., Tapamo, J.-R., & Burke, M. (2017). *Driver drowsiness detection using behavioral measures and machine learning techniques: A review of state-of-art techniques*. https://doi.org/10.1109/RoboMech.2017.8261140

Reddy, B., Kim, Y.-H., Yun, S., Seo, C., & Jang, J. (2017). *Real-time Driver Drowsiness Detection for Embedded System Using Model Compression of Deep Neural Networks*. 438–445.

Reza, A. M. (2004). Realization of the Contrast Limited Adaptive Histogram Equalization (CLAHE) for Real-Time Image Enhancement. *Journal of VLSI Signal Processing Systems for Signal, Image and Video Technology*, *38*(1), 35–44. https://doi.org/10.1023/B:VLSI.0000028532.53893.82

Shroff, R., Ting, F., Lam, W., Cecot, T., Yang, J., & Chan, L. (2021). Conceptualization, Development and Validation of an Instrument to Measure Learners' Perceptions of their Active Learning Strategies within an Active Learning Context. *International Journal of Educational Methodology*, *7*, 201–223. https://doi.org/10.12973/ijem.7.1.201

Souto Maior, C., Moura, M., Santana, J., & Lins, I. (2020). Real-time Classification for Autonomous Drowsiness Detection Using Eye Aspect Ratio. *Expert Systems with Applications*, *158*. https://doi.org/10.1016/j.eswa.2020.113505

Wijnands, J. S., Thompson, J., Nice, K. A., Aschwanden, G. D. P. A., & Stevenson, M. (2020). Real-time monitoring of driver drowsiness on mobile platforms using 3D neural networks. *Neural Computing and Applications*, *32*(13), 9731–9743. https://doi.org/10.1007/s00521-019-04506-0

World Health Organization. (2023). *Global status report on road safety 2023*. https://assets.bbhub.io/dotorg/sites/64/2023/12/WHO-Global-status-report-on-road-safety-2023.pdf?utm_source=chatgpt.com

Zhang, Y. Ji, Q. (2021). Real-time and automatic critical drowsiness detection based on eye and facial expression. *IEEE Transactions on Neural Systems and Rehabilitation*

Engineering, *29*(4), 1523–1532. https://doi.org/10.1109/TNSRE.2021.3097412

Zhao, Z., Zhou, N., Zhang, L., Yan, H., Xu, Y., & Zhang, Z. (2020). Driver Fatigue Detection Based on Convolutional Neural Networks Using EM-CNN. *Computational Intelligence and Neuroscience*, *2020*(1), 7251280. https://doi.org/https://doi.org/10.1155/2020/7251280

APPENDEX

Collected dataset

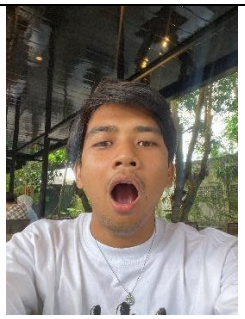
o	Subjek	Nama File	Kondisi (GT)	Prediksi Sistem	Status	Alasan
		Img1	Normal	Normal	Benar	
		Img1	Eyes closed	Eyes closed	Benar	
		Img1	Yawn	Yawn	Benar	

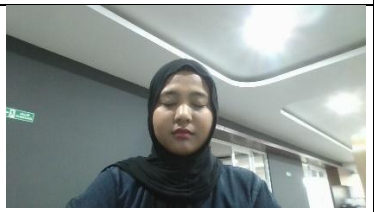
4		Img1	Yawn hand cover	Yawn hand cover	Benar	
5		Img2	Normal	Normal	Benar	
6		Img2	Eyes closed	Eyes closed	Benar	
7		Img2	Yawn	Yawn	Benar	

8		Img2	Yawn hand cover	Yawn	Salah	Sistem berhasil mendeteksi yawning melalui nilai MAR yang sangat tinggi (0.8658), tetapi gagal mendeteksi keberadaan tangan di area mulut sehingga diklasifikasikan sebagai yawn biasa, bukan yawn_hand_cover
9		Img3	Normal	Normal	Benar	
10		Img3	Eyes closed	Eyes closed	Benar	

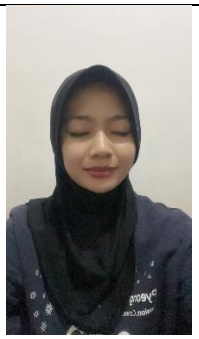
1		Img3	Yawn	Yawn	Benar	
2		Img3	Yawn hand cover	Yawn hand cover	Benar	
3		Img4	Normal	Normal	Benar	
4		Img4	Eyes closed	Eyes closed	Benar	
5		Img4	Yawn	Yawn	Benar	
6		Img4	Yawn hand cover	Yawn hand cover	Benar	

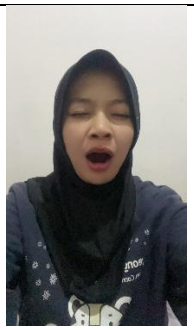
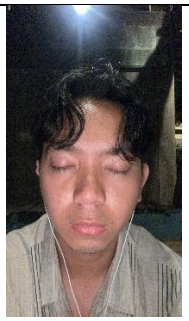
17		Img5	Normal	Normal	benar	
18		Img5	Eyes closed	Normal	Salah	Nilai EAR terdeteksi cukup tinggi (0.2346) sehingga sistem menganggap mata masih terbuka normal. Kemungkinan landmark mata kurang akurat atau mata tidak tertutup penuh pada gambar.
19		Img5	Yawn	Yawn	Benar	

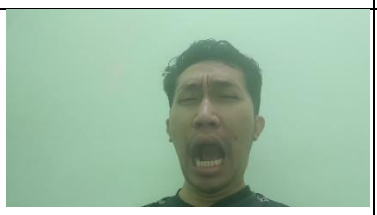
0		Img5	Yawn hand cover	Yawn hand cover	Benar	
1		Img6	Normal	Normal	Benar	
2		Img6	Eyes closed	Eyes closed	Benar	
3		Img6	Yawn	Yawn	Benar	
4		Img6	Yawn hand cover	Yawn hand cover	Benar	

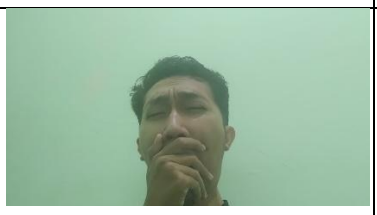
5		Img7	Normal	Normal	Benar	
6		Img7	Eyes closed	Eyes closed	Benar	
7		Img7	Yawn	Yawn	Benar	
8		Img7	Yawn hand cover	Yawn hand cover	Benar	
9		Img8	Normal	Normal	Benar	
0		Img8	Eyes closed	Eyes closed	Benar	

1		Img8	Yawn	Yawn	Benar	
2		Img8	Yawn hand cover	Yawn hand cover	Benar	
3		Img9	Normal	Normal	Benar	
4		Img9	Eyes closed	Eyes closed	Benar	
5		Img9	Yawn	Yawn	Benar	
6		Img9	Yawn hand cover	Yawn hand cover	Benar	
7		Img10	Normal	Normal	Benar	

38		Img10	Eyes closed	Eyes closed	Benar	
39		Img10	Yawn	Yawn	Benar	
40		Img10	Yawn hand cover	Yawn hand cover	Benar	
41		Img11	Normal	Normal	Benar	
42		Img11	Eyes closed	Eyes closed	Benar	

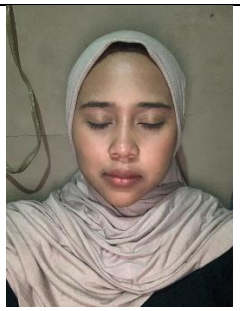
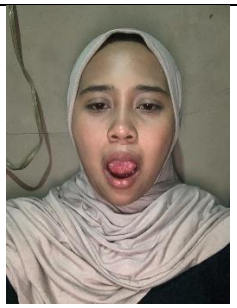
43		Img11	Yawn	Yawn	Benar	
44		Img11	Yawn hand cover	Yawn hand cover	Benar	
45		Img12	Normal	Normal	Benar	
46		Img12	Eyes closed	Eyes closed	Benar	
47		Img12	Yawn	Normal	Salah	Nilai MAR sangat kecil (0.0089), sehingga sistem tidak mendeteksi adanya mulut terbuka/menguap.

						Kemungkinan mulut tidak terbuka lebar atau landmark mulut kurang stabil.
8		Img12	Yawn hand cover	Yawn hand cover	Benar	
9		Img13	Normal	Normal	Benar	
0		Img13	Eyes closed	Eyes closed	Benar	
1		Img13	Yawn	Eyes closed	Salah	Nilai EAR rendah (0.1279) membuat sistem lebih dominan mendeteksi kondisi mata tertutup dibanding yawning. Selain itu MAR juga tidak

						terlalu tinggi untuk melewati threshold yawning secara optimal.
2		Img13	Yawn hand cover	Eyes closed	Salah	Sistem gagal mendeteksi hand-cover karena jarak tangan terhadap area mulut tidak memenuhi threshold deteksi tangan. Akibatnya klasifikasi lebih dipengaruhi oleh EAR rendah sehingga diprediksi sebagai eyes_closed.

3		Img14	Normal	Normal	Benar	
4		Img14	Eyes closed	Eyes closed	Benar	
5		Img14	Yawn	Yawn	Benar	
6		Img14	Yawn hand cover	Yawn hand cover	Benar	
7		Img15	Normal	Normal	Benar	

58		Img15	Eyes closed	Eyes closed	Benar	
59		Img15	Yawn	Yawn	Benar	
60		Img15	Yawn hand cover	Yawn hand cover	Benar	
61		Img16	Normal	Normal	Benar	
62		Img16	Eyes closed	Eyes closed	Benar	

63		Img16	Yawn	Yawn	Benar	
64		Img16	Yawn hand cover	Yawn hand cover	Benar	
65		Img17	Normal	Normal	Benar	
66		Img17	Eyes closed	Eyes closed	Benar	
67		Img17	Yawn	Yawn	Benar	

8		Img17	Yawn hand cover	Yawn hand cover	Benar	
9		Img18	Normal	Normal	Benar	
0		Img18	Eyes closed	Eyes closed	Benar	
1		Img18	Yawn	Yawn	Benar	
2		Img18	Yawn hand cover	Yawn hand cover	Benar	

3		Img19	Normal	Normal	Benar	
4		Img19	Closed eyes	Closed eyes	Benar	
5		Img19	Yawn	Yawn	Benar	
6		Img19	Yawn hand cover	Yawn hand cover	Benar	

Real-Time Drowsiness Detection Systm Source Code

```
"""

import argparse
import cv2
import numpy as np
import time
import os
from collections import deque, defaultdict

# ── TFLite Interpreter import ──────────────────────────────────────
# Coba tflite-runtime (ringan, direkomendasikan untuk Pi).
```

```
# Jika tidak ada, fallback ke tf.lite.Interpreter dari TensorFlow penuh.
try:
    from tflite_runtime.interpreter import Interpreter
    print("[INFO] Menggunakan tflite-runtime (ringan).")
except ImportError:
    os.environ['TF_CPP_MIN_LOG_LEVEL'] = '3'
    import tensorflow as tf
    Interpreter = tf.lite.Interpreter
    print("[INFO] tflite-runtime tidak ditemukan, fallback ke TensorFlow penuh.")

# ── MediaPipe import ──────────────────────────────────────────────
import mediapipe as mp



# ==============================================================================
#  KONFIGURASI
# ==============================================================================

class Config:
    """Semua parameter sistem di satu tempat."""

    # ── Model CNN (TFLite) ────────────────────────────────────────
    MODEL_PATH  = os.path.join("output_cnn", "model_drowsiness_cnn.tflite")
    CLASS_NAMES = ["eyes_closed", "normal", "yawn"]   # urutan = class_indices saat training
    IMG_SIZE    = (96, 96)
    MIN_CONFIDENCE = 0.50

    # ── Kamera ────────────────────────────────────────────────────
    # Resolusi diturunkan dibanding versi PC (1280x720) demi FPS di Pi 5.
    CAM_INDEX  = 0
    CAM_WIDTH  = 640
    CAM_HEIGHT = 480

    # ── Preprocessing ─────────────────────────────────────────────
    USE_CLAHE  = True
    CLAHE_CLIP = 2.0
    CLAHE_TILE = (8, 8)

    # ── MediaPipe Face Mesh ───────────────────────────────────────
    FACE_MIN_DETECTION_CONF = 0.5
    FACE_MIN_TRACKING_CONF  = 0.5
    FACE_ROI_PADDING        = 0.05

    # ── MediaPipe Hands ───────────────────────────────────────────
    HAND_MIN_DETECTION_CONF = 0.4
    HAND_MIN_TRACKING_CONF  = 0.4
```

```
    HAND_MOUTH_THRESHOLD    = 0.65

    # ── Drowsiness counter ──────────────────────────────────────────
    YAWN_COUNT_THRESHOLD = 5
    YAWN_WINDOW_SEC = 60

    # ── Temporal smoothing ──────────────────────────────────────────
    SMOOTHING_FRAMES = 3

    # ── Alarm ───────────────────────────────────────────────────────
    ALARM_BEEP_INTERVAL = 1.0
    USE_GPIO_BUZZER = True     # otomatis nonaktif jika gpiozero/pin tak ada
    BUZZER_GPIO_PIN = 17       # GPIO17 (pin fisik 11) -- sesuaikan wiring

    # ── Display ─────────────────────────────────────────────────────
    SHOW_DEBUG = True



# ============================================================================
#  MODULE: FACE MESH DETECTOR  (TIDAK DIUBAH dari versi PC)
# ============================================================================

MOUTH_LANDMARKS = [13, 14, 78, 308]
NOSE_TIP = 1
CHIN     = 152



class FaceMeshDetector:
    def __init__(self, min_detection_conf, min_tracking_conf):
        self.face_mesh = mp.solutions.face_mesh.FaceMesh(
            static_image_mode=False,
            refine_landmarks=True,
            max_num_faces=1,
            min_detection_confidence=min_detection_conf,
            min_tracking_confidence=min_tracking_conf,
        )
        self.mp_drawing = mp.solutions.drawing_utils
        self.mp_face_mesh = mp.solutions.face_mesh
        self.mp_styles = mp.solutions.drawing_styles

    def detect(self, frame_rgb, padding_ratio=0.25):
        ih, iw = frame_rgb.shape[:2]
        result = self.face_mesh.process(frame_rgb)

        if not result.multi_face_landmarks:
```

```
            return None

        face_lm = result.multi_face_landmarks[0]
        landmarks = face_lm.landmark

        xs = [lm.x * iw for lm in landmarks]
        ys = [lm.y * ih for lm in landmarks]

        x_min, x_max = int(min(xs)), int(max(xs))
        y_min, y_max = int(min(ys)), int(max(ys))
        w = x_max - x_min
        h = y_max - y_min

        pad_x = int(w * padding_ratio)
        pad_y = int(h * padding_ratio)
        x_min = max(0, x_min - pad_x)
        y_min = max(0, y_min - pad_y)
        x_max = min(iw, x_max + pad_x)
        y_max = min(ih, y_max + pad_y)

        bbox = (x_min, y_min, x_max - x_min, y_max - y_min)

        mouth_pts = [(landmarks[i].x * iw, landmarks[i].y * ih)
                     for i in MOUTH_LANDMARKS]
        mouth_cx = np.mean([p[0] for p in mouth_pts])
        mouth_cy = np.mean([p[1] for p in mouth_pts])

        nose = (landmarks[NOSE_TIP].x * iw, landmarks[NOSE_TIP].y * ih)
        chin = (landmarks[CHIN].x * iw, landmarks[CHIN].y * ih)
        face_size = np.sqrt((nose[0]-chin[0])**2 + (nose[1]-chin[1])**2)

        return {
            "bbox":          bbox,
            "landmarks":     landmarks,
            "mesh_result":  result,
            "mouth_center": (mouth_cx, mouth_cy),
            "face_size":     face_size,
        }

    def draw_mesh(self, frame_bgr, mesh_result):
        if mesh_result and mesh_result.multi_face_landmarks:
            self.mp_drawing.draw_landmarks(
                frame_bgr,
                mesh_result.multi_face_landmarks[0],
                self.mp_face_mesh.FACEMESH_TESSELATION,
                landmark_drawing_spec=None,
```

```
                connection_drawing_spec=self.mp_styles
                    .get_default_face_mesh_tesselation_style(),
            )

    def close(self):
        self.face_mesh.close()


# ======================================================================
#  MODULE: CNN CLASSIFIER (TFLite) — diganti dari Keras penuh
# ======================================================================

class CNNClassifier:
    """Load dan jalankan model CNN .tflite untuk klasifikasi kantuk.

    Input  : ROI wajah 96x96x3, dinormalisasi [0,1]
    Output : (predicted_class, confidence, probabilities_dict)
    """

    def __init__(self, model_path, class_names, img_size):
        if not os.path.exists(model_path):
            raise FileNotFoundError(
                f"Model TFLite tidak ditemukan: {model_path}\n"
                f"Jalankan convert_to_tflite.py di PC lalu salin filenya ke sini."
            )
        print(f"[INFO] Memuat model TFLite dari: {model_path}")

        self.interpreter = Interpreter(model_path=model_path, num_threads=4)
        self.interpreter.allocate_tensors()
        self.input_details = self.interpreter.get_input_details()
        self.output_details = self.interpreter.get_output_details()

        self.class_names = class_names
        self.img_size = img_size
        print(f"[INFO] Model TFLite berhasil dimuat. Kelas: {class_names}")

    def predict(self, face_roi_bgr):
        img = cv2.cvtColor(face_roi_bgr, cv2.COLOR_BGR2RGB)
        img = cv2.resize(img, self.img_size)
        img = img.astype(np.float32) / 255.0
        img = np.expand_dims(img, axis=0)

        in_dtype = self.input_details[0]["dtype"]
        if in_dtype != np.float32:
            img = img.astype(in_dtype)
```

```
        self.interpreter.set_tensor(self.input_details[0]["index"], img)
        self.interpreter.invoke()
        probs = self.interpreter.get_tensor(self.output_details[0]["index"])[0]

        idx = int(np.argmax(probs))
        class_name = self.class_names[idx]
        confidence = float(probs[idx])

        prob_dict = {
            name: float(probs[i])
            for i, name in enumerate(self.class_names)
        }
        return class_name, confidence, prob_dict


# ======================================================================
#  MODULE: HAND-MOUTH DETECTOR  (TIDAK DIUBAH dari versi PC)
# ======================================================================

class HandMouthDetector:
    def __init__(self, min_detection_conf, min_tracking_conf):
        self.hands = mp.solutions.hands.Hands(
            static_image_mode=False,
            max_num_hands=2,
            min_detection_confidence=min_detection_conf,
            min_tracking_confidence=min_tracking_conf,
        )
        self.mp_drawing = mp.solutions.drawing_utils
        self.mp_hands   = mp.solutions.hands
        self.mp_styles  = mp.solutions.drawing_styles

    def detect_hand_cover(self, frame_rgb, mouth_center, face_size, threshold):
        result = self.hands.process(frame_rgb)
        hand_lms = result.multi_hand_landmarks or []

        if not hand_lms:
            return False, 999.0, hand_lms

        ih, iw = frame_rgb.shape[:2]
        mcx, mcy = mouth_center

        best_dist = 999.0
        for hand in hand_lms:
            for lm in hand.landmark:
```

```
                hx = lm.x * iw
                hy = lm.y * ih
                dist = np.sqrt((hx - mcx)**2 + (hy - mcy)**2)
                normalized = dist / (face_size + 1e-6)
                best_dist = min(best_dist, normalized)

        is_covering = best_dist < threshold
        return is_covering, best_dist, hand_lms

    def draw_hands(self, frame_bgr, hand_landmarks_list):
        for hand_lm in hand_landmarks_list:
            self.mp_drawing.draw_landmarks(
                frame_bgr,
                hand_lm,
                self.mp_hands.HAND_CONNECTIONS,
                self.mp_styles.get_default_hand_landmarks_style(),
                self.mp_styles.get_default_hand_connections_style(),
            )

    def close(self):
        self.hands.close()



# ============================================================================
#  MODULE: PREPROCESSING  (TIDAK DIUBAH)
# ============================================================================

def apply_clahe(frame_bgr, clip_limit=2.0, tile_size=(8, 8)):
    lab = cv2.cvtColor(frame_bgr, cv2.COLOR_BGR2LAB)
    l_ch, a, b = cv2.split(lab)
    clahe = cv2.createCLAHE(clipLimit=clip_limit, tileGridSize=tile_size)
    l_ch = clahe.apply(l_ch)
    return cv2.cvtColor(cv2.merge([l_ch, a, b]), cv2.COLOR_LAB2BGR)



# ============================================================================
#  MODULE: ALARM — sekarang mendukung GPIO buzzer asli di Pi 5
# ============================================================================

class AlarmSystem:
    """Alarm. Otomatis pakai GPIO buzzer jika gpiozero & pin tersedia,
    fallback ke bell terminal ('\\a') jika tidak."""

    def __init__(self, beep_interval=1.0, use_gpio=True, gpio_pin=17):
        self.beep_interval = beep_interval
```

```
        self.last_beep = 0.0
        self.buzzer = None

        if use_gpio:
            try:
                from gpiozero import Buzzer
                self.buzzer = Buzzer(gpio_pin)
                print(f"[OK] GPIO buzzer aktif di pin {gpio_pin}.")
            except Exception as e:
                print(f"[INFO] GPIO buzzer tidak tersedia ({e}). "
                      f"Fallback ke bell terminal.")
                self.buzzer = None

    def trigger(self):
        now = time.time()
        if now - self.last_beep >= self.beep_interval:
            self.last_beep = now
            if self.buzzer is not None:
                self.buzzer.on()
                time.sleep(0.3)
                self.buzzer.off()
            else:
                try:
                    import winsound
                    winsound.Beep(1000, 200)
                except (ImportError, RuntimeError):
                    print('\a', end='', flush=True)

    def cleanup(self):
        if self.buzzer is not None:
            self.buzzer.close()



# ======================================================================
#  MODULE: DISPLAY / OVERLAY  (TIDAK DIUBAH dari versi PC)
# ======================================================================

COLORS = {
    "normal":           (50,  200, 50),
    "eyes_closed":      (0,   165, 255),
    "yawn":             (0,   200, 255),
    "yawn_hand_cover":  (0,   0,   220),
}

LABEL_DISPLAY = {
    "normal":           "Normal",
    "eyes_closed":      "Mata Tertutup",
```

```
    "yawn":            "Menguap",
    "yawn_hand_cover": "Menguap + Tangan",
}



def draw_overlay(frame, prediction, confidence, prob_dict,
                 drowsy_counter, drowsy_threshold, is_drowsy,
                 hand_detected, hand_dist, hand_covering,
                 fps, session_stats, use_clahe, show_debug,
                 face_bbox, mouth_center):
    ih, iw = frame.shape[:2]
    color = COLORS.get(prediction, (200, 200, 200))
    label = LABEL_DISPLAY.get(prediction, prediction)

    bar = np.zeros((65, iw, 3), dtype=np.uint8)
    bar[:] = (30, 30, 30)
    frame[:65] = cv2.addWeighted(frame[:65], 0.15, bar, 0.85, 0)

    status_text  = "DROWSY" if is_drowsy else "NORMAL"
    status_color = (0, 0, 255) if is_drowsy else (50, 200, 50)
    cv2.putText(frame, f"STATUS: {status_text}", (20, 45),
                cv2.FONT_HERSHEY_DUPLEX, 0.85, status_color, 2, cv2.LINE_AA)

    pred_text = f"CNN: {label} ({confidence:.0%})"
    cv2.putText(frame, pred_text, (iw // 2 - 120, 45),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, color, 2, cv2.LINE_AA)

    cv2.putText(frame, f"{fps:.0f} FPS", (iw - 110, 30),
                cv2.FONT_HERSHEY_SIMPLEX, 0.6, (180, 180, 180), 1, cv2.LINE_AA)

    if hand_detected:
        h_txt = f"HAND: detected (d={hand_dist:.2f})"
        h_col = (50, 220, 50) if hand_covering else (0, 200, 255)
    else:
        h_txt = "HAND: -"
        h_col = (80, 80, 80)
    cv2.putText(frame, h_txt, (iw - 340, 55),
                cv2.FONT_HERSHEY_SIMPLEX, 0.48, h_col, 1, cv2.LINE_AA)

    if face_bbox is not None:
        fx, fy, fw, fh = face_bbox
        box_col = (0, 0, 255) if is_drowsy else color
        cv2.rectangle(frame, (fx, fy), (fx+fw, fy+fh), box_col, 2)

    if mouth_center is not None:
        mcx, mcy = int(mouth_center[0]), int(mouth_center[1])
```

```
    cv2.circle(frame, (mcx, mcy), 6, (255, 0, 255), 2)
    cv2.putText(frame, "mouth", (mcx + 10, mcy - 5),
                cv2.FONT_HERSHEY_SIMPLEX, 0.35, (255, 0, 255), 1,
                cv2.LINE_AA)


bar_x, bar_y, bar_w, bar_h = 20, ih - 50, 300, 25
fill_ratio = min(1.0, drowsy_counter / max(1, drowsy_threshold * 2))
fill_w = int(bar_w * fill_ratio)
bar_col = (0, 0, 255) if is_drowsy else (0, 200, 255)


cv2.rectangle(frame, (bar_x, bar_y), (bar_x+bar_w, bar_y+bar_h),
              (60, 60, 60), -1)
cv2.rectangle(frame, (bar_x, bar_y), (bar_x+fill_w, bar_y+bar_h),
              bar_col, -1)
cv2.rectangle(frame, (bar_x, bar_y), (bar_x+bar_w, bar_y+bar_h),
              (200, 200, 200), 1)


thr_x = bar_x + int(bar_w * (drowsy_threshold / (drowsy_threshold * 2)))
cv2.line(frame, (thr_x, bar_y-3), (thr_x, bar_y+bar_h+3),
         (255, 255, 255), 2)
cv2.putText(frame, f"yawn: {int(drowsy_counter)}x/{drowsy_threshold}x per menit",
            (bar_x, bar_y - 8),
            cv2.FONT_HERSHEY_SIMPLEX, 0.5, (220, 220, 220), 1,
            cv2.LINE_AA)


if is_drowsy and int(time.time() * 3) % 2 == 0:
    bw, bh = 420, 90
    bx = (iw - bw) // 2
    by = ih // 2 - bh // 2 - 50
    overlay = frame.copy()
    cv2.rectangle(overlay, (bx, by), (bx+bw, by+bh), (0, 0, 0), -1)
    frame[:] = cv2.addWeighted(overlay, 0.6, frame, 0.4, 0)
    cv2.rectangle(frame, (bx, by), (bx+bw, by+bh), (0, 0, 255), 3)
    cv2.putText(frame, "!  PERINGATAN KANTUK  !",
                (bx+30, by+38), cv2.FONT_HERSHEY_DUPLEX, 0.8,
                (0, 0, 255), 2, cv2.LINE_AA)
    cv2.putText(frame, "Segera istirahat!",
                (bx+100, by+70), cv2.FONT_HERSHEY_SIMPLEX, 0.65,
                (255, 255, 255), 1, cv2.LINE_AA)


if show_debug:
    dx, dy = 20, ih - 210
    pw, ph = 320, 150
    y_end = min(dy + ph, ih)
    x_end = min(dx + pw, iw)
    panel = np.zeros((y_end - dy, x_end - dx, 3), dtype=np.uint8)
    frame[dy:y_end, dx:x_end] = cv2.addWeighted(
        frame[dy:y_end, dx:x_end], 0.3, panel, 0.7, 0)
```

```
        y_off = 0
        for cls in Config.CLASS_NAMES + ["yawn_hand_cover"]:
            prob = prob_dict.get(cls, 0.0)
            if cls == "yawn_hand_cover" and hand_covering:
                prob = prob_dict.get("yawn", 0.0)
            c = COLORS.get(cls, (200, 200, 200))
            txt = f"{LABEL_DISPLAY.get(cls, cls)}: {prob:.1%}"
            cv2.putText(frame, txt, (dx+10, dy+20+y_off),
                        cv2.FONT_HERSHEY_SIMPLEX, 0.42, c, 1, cv2.LINE_AA)
            y_off += 22

        cv2.putText(frame, f"CLAHE: {'ON' if use_clahe else 'OFF'}",
                    (dx+10, dy+20+y_off+10),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.4,
                    (80, 220, 80) if use_clahe else (120, 120, 120),
                    1, cv2.LINE_AA)

        cv2.putText(frame,
                    "[Q]Quit [C]CLAHE [D]Debug [R]Reset [+/-]Threshold",
                    (dx+10, dy+140),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.33, (140, 140, 140), 1,
                    cv2.LINE_AA)

    sx, sy = iw - 220, ih - 140
    sw, sh = 210, 125
    y_end = min(sy + sh, ih)
    x_end = min(sx + sw, iw)
    spanel = np.zeros((y_end - sy, x_end - sx, 3), dtype=np.uint8)
    frame[sy:y_end, sx:x_end] = cv2.addWeighted(
        frame[sy:y_end, sx:x_end], 0.3, spanel, 0.7, 0)

    cv2.putText(frame, "Sesi ini:", (sx+10, sy+18),
                cv2.FONT_HERSHEY_SIMPLEX, 0.48, (200, 200, 200), 1,
                cv2.LINE_AA)
    for i, (k, v) in enumerate(session_stats.items()):
        lbl = LABEL_DISPLAY.get(k, k)
        cv2.putText(frame, f"{lbl}: {v}",
                    (sx+10, sy+38+i*20),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.42,
                    COLORS.get(k, (200, 200, 200)), 1, cv2.LINE_AA)

    return frame


# ================================================================
```

```
#  MAIN LOOP
# ============================================================================

def main():
    parser = argparse.ArgumentParser()
    parser.add_argument("--no-display", action="store_true",
                        help="Jalankan tanpa jendela preview (headless, untuk systemd).")
    args = parser.parse_args()

    cfg = Config()
    show_window = not args.no_display

    # ── Snapshot otomatis (untuk verifikasi tanpa perlu live video di VNC) ──
    SNAPSHOT_DIR = "snapshots"
    SNAPSHOT_INTERVAL_SEC = 3.0
    os.makedirs(SNAPSHOT_DIR, exist_ok=True)
    last_snapshot_time = 0.0

    print("=" * 60)
    print("  DROWSINESS DETECTION SYSTEM — RASPBERRY PI 5")
    print("  CNN (TFLite) + MediaPipe Face Mesh + MediaPipe Hands")
    print("=" * 60)
    print()

    face_detector = FaceMeshDetector(
        cfg.FACE_MIN_DETECTION_CONF,
        cfg.FACE_MIN_TRACKING_CONF,
    )
    print("[OK] MediaPipe Face Mesh initialized (478 landmarks).")

    cnn = CNNClassifier(cfg.MODEL_PATH, cfg.CLASS_NAMES, cfg.IMG_SIZE)

    hand_detector = HandMouthDetector(
        cfg.HAND_MIN_DETECTION_CONF,
        cfg.HAND_MIN_TRACKING_CONF,
    )
    print("[OK] MediaPipe Hands initialized (conditional).")

    alarm = AlarmSystem(cfg.ALARM_BEEP_INTERVAL,
                        use_gpio=cfg.USE_GPIO_BUZZER,
                        gpio_pin=cfg.BUZZER_GPIO_PIN)

    cap = cv2.VideoCapture(cfg.CAM_INDEX)
    if not cap.isOpened():
        print(f"[ERROR] Kamera index {cfg.CAM_INDEX} tidak bisa dibuka!")
        return
```

```
cap.set(cv2.CAP_PROP_FRAME_WIDTH, cfg.CAM_WIDTH)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, cfg.CAM_HEIGHT)
cap.set(cv2.CAP_PROP_BUFFERSIZE, 1)


actual_w = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
actual_h = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))
print(f"[OK] Kamera terbuka: {actual_w}x{actual_h}")
print()
if show_window:
    print("Kontrol: Q=Quit | C=CLAHE | D=Debug | R=Reset | +/-=Threshold")
else:
    print("Mode headless aktif (--no-display). Tekan Ctrl+C untuk keluar.")
print("=" * 60)


use_clahe        = cfg.USE_CLAHE
show_debug       = cfg.SHOW_DEBUG
yawn_threshold = cfg.YAWN_COUNT_THRESHOLD
yawn_window = cfg.YAWN_WINDOW_SEC
yawn_timestamps = []
is_drowsy = False
was_yawning = False
history          = deque(maxlen=cfg.SMOOTHING_FRAMES)
session_stats    = defaultdict(int)
fps_times        = deque(maxlen=30)
prev_time        = time.time()


if show_window:
    cv2.namedWindow("Drowsiness Detection", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("Drowsiness Detection", 900, 600)


try:
    while True:
        ret, frame = cap.read()
        if not ret:
            time.sleep(0.03)
            continue


        frame = cv2.flip(frame, 1)
        display_frame = frame.copy()


        if use_clahe:
            proc = apply_clahe(frame, cfg.CLAHE_CLIP, cfg.CLAHE_TILE)
        else:
            proc = frame.copy()
```

```
rgb = cv2.cvtColor(proc, cv2.COLOR_BGR2RGB)


face_info    = face_detector.detect(rgb, cfg.FACE_ROI_PADDING)
prediction   = "normal"
confidence   = 0.0
prob_dict    = {c: 0.0 for c in cfg.CLASS_NAMES}
hand_detected  = False
hand_dist    =   999.0
hand_covering  = False
hand_lms     = []
face_bbox    =   None
mouth_center   = None


if face_info is not None:
    face_bbox    = face_info["bbox"]
    mouth_center = face_info["mouth_center"]
    face_size    = face_info["face_size"]


    if show_window:
        face_detector.draw_mesh(display_frame,  face_info["mesh_result"])


    fx, fy, fw, fh = face_bbox
    ih, iw = frame.shape[:2]
    fx = max(0, fx); fy = max(0, fy)
    fw = min(fw, iw - fx); fh = min(fh, ih - fy)


    if fw > 20 and fh > 20:
        face_roi = proc[fy:fy+fh, fx:fx+fw]
        prediction, confidence, prob_dict = cnn.predict(face_roi)


        if confidence < cfg.MIN_CONFIDENCE:
            prediction = "normal"


        if prediction == "yawn":
            rgb_full = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            hand_covering, hand_dist, hand_lms = \
                hand_detector.detect_hand_cover(
                    rgb_full, mouth_center, face_size,
                    cfg.HAND_MOUTH_THRESHOLD,
                )
            hand_detected = len(hand_lms) > 0


            if hand_covering:
                prediction = "yawn_hand_cover"


if show_window and hand_lms:
```

```
    hand_detector.draw_hands(display_frame,  hand_lms)


history.append(prediction)
counts = defaultdict(int)
for h in history:
    counts[h] += 1
smoothed = max(counts, key=counts.get)


is_yawning_now = smoothed in ("yawn", "yawn_hand_cover")


if is_yawning_now and not was_yawning:
    yawn_timestamps.append(time.time())
was_yawning = is_yawning_now


cutoff = time.time() - yawn_window
yawn_timestamps = [t for t in yawn_timestamps if t > cutoff]


yawn_count = len(yawn_timestamps)
is_drowsy = yawn_count > yawn_threshold


if is_drowsy:
    alarm.trigger()


session_stats[smoothed] += 1


now = time.time()
fps_times.append(now - prev_time)
prev_time = now
fps = 1.0 / (sum(fps_times) / len(fps_times) + 1e-6)


display_frame = draw_overlay(
    display_frame,
    prediction=smoothed,
    confidence=confidence,
    prob_dict=prob_dict,
    drowsy_counter=yawn_count,
    drowsy_threshold=yawn_threshold,
    is_drowsy=is_drowsy,
    hand_detected=hand_detected,
    hand_dist=hand_dist,
    hand_covering=hand_covering,
    fps=fps,
    session_stats=session_stats,
    use_clahe=use_clahe,
    show_debug=show_debug,
    face_bbox=face_bbox,
```

```
            mouth_center=mouth_center,
        )

        # ── Simpan snapshot otomatis tiap beberapa detik ──────────
        # Berguna untuk verifikasi hasil tanpa perlu streaming video
        # live lewat VNC (yang berat & bisa bikin sesi putus).
        if now - last_snapshot_time >= SNAPSHOT_INTERVAL_SEC:
            last_snapshot_time = now
            snap_path = os.path.join(SNAPSHOT_DIR, "latest.jpg")
            cv2.imwrite(snap_path, display_frame)

        if show_window:
            cv2.imshow("Drowsiness Detection", display_frame)

            key = cv2.waitKey(1) & 0xFF
            if key in (ord('q'), 27):
                break
            elif key == ord('c'):
                use_clahe = not use_clahe
                print(f"[INFO] CLAHE: {'ON' if use_clahe else 'OFF'}")
            elif key == ord('d'):
                show_debug = not show_debug
                print(f"[INFO] Debug: {'ON' if show_debug else 'OFF'}")
            elif key == ord('r'):
                yawn_timestamps.clear()
                is_drowsy = False
                was_yawning = False
                history.clear()
                session_stats.clear()
                print("[INFO] Counter & statistik di-reset.")
            elif key in (ord('+'), ord('=')):
                yawn_threshold = min(30, yawn_threshold + 1)
                print(f"[INFO] Threshold: {yawn_threshold}x dalam {yawn_window}s")
            elif key in (ord('-'), ord('_')):
                yawn_threshold = max(1, yawn_threshold - 1)
                print(f"[INFO] Threshold: {yawn_threshold}x dalam {yawn_window}s")
        else:
            # Mode headless: cetak status singkat tiap ~1 detik via FPS counter.
            if int(now) % 1 == 0:
                pass  # bisa ditambah logging/telemetry di sini jika perlu

except KeyboardInterrupt:
    print("\n[INFO] Dihentikan oleh pengguna (Ctrl+C).")

finally:
    cap.release()
    if show_window:
        cv2.destroyAllWindows()
```

```
        face_detector.close()
        hand_detector.close()
        alarm.cleanup()


        total = sum(session_stats.values())
        print()
        print("=" * 50)
        print("  RINGKASAN SESI")
        print("=" * 50)
        for k, v in session_stats.items():
            pct = v / total * 100 if total else 0
            print(f"  {LABEL_DISPLAY.get(k,k):<22}: {v:>6} frame ({pct:.1f}%)")
        print(f"  {'Total frame':<22}: {total:>6}")
        print("=" * 50)



# ======================================================================
if __name__ == "__main__":
    main()
```