POLITECNICO DI TORINO

MASTER's Degree in Artificial Intelligence and Data Analytics



MASTER's Degree Thesis

An Integrated Vision-Based System for Gesture Interaction and Driver Monitoring in Smart Cockpits

Supervisors

Dott. Davide CALANDRA

Candidate

Francesco GENTILUOMO

October 2026

Summary

The transition towards smart vehicles has driven a significant evolution in active safety systems, shifting the technological focus directly onto the driver. Within next-generation cabins (*smart cockpits*), attention monitoring technologies, known as Driver Monitoring Systems (*DMS*), coupled with the development of intelligent interfaces, have therefore become fundamental. In this context, this thesis proposes an integrated system that enables natural, touchless interaction via dynamic hand gestures while simultaneously monitoring the driver's attention state. The system integrates both functionalities (interaction and monitoring) using a single *RGB-IR* camera positioned at the level of the rearview mirror, thus optimizing hardware resources within the cabin. For gesture recognition, the features extracted via the *Google MediaPipe* framework feed a *bidirectional LSTM* neural network, trained on a custom dataset of 2160 samples (acquired in multiple configurations to increase variety). Concurrently, the Driver Monitoring module leverages facial feature extraction to evaluate the driver's fatigue and gaze direction, operating in synergy with a *YOLO* model dedicated to detecting smartphones as a potential source of distraction. The entire architecture has been optimized and profiled on embedded devices (*Nvidia Jetson Orin NX and Raspberry Pi*) and successfully interfaced with an environment based on Android Automotive OS (*AAOS*). The applicability and usability of the proposed solution are evaluated through a user study protocol, based on a comparison between gestural interaction and traditional touchscreen input. The protocol considers tasks representative of the context of use and analyzes execution times, interaction errors, gesture recognition accuracy, and visual distraction. The evaluation also includes subjective measures of ease of use, cognitive load, and usability, collected through the *SEQ*, *NASA-TLX*, and *SUS* questionnaires. The objective of the validation is to characterize the advantages and limitations of gestural interaction and to identify the types of commands for which it can represent an effective alternative to touch interaction. Indeed, the system allows gestures to be mapped to different interface commands, and this work evaluates its potential in reducing the need to divert the gaze from the road for specific operations.

Table of Contents

List of Tables VIII

List of Figures X

Acronyms XIII

1 Introduction 1
- 1.1 The Evolution Towards the Smart Cockpit and Cognitive Load in Driving . . . 1
- 1.2 The Problem of Visual Distraction and Alternative Interfaces . . . 2
- 1.3 Motivation and Objectives . . . 4
- 1.4 Contributions of the Thesis . . . 5
- 1.5 Thesis Structure . . . 6

2 State of the Art and Enabling Technologies 8
- 2.1 Regulatory Framework and Industry Standards . . . 8
- 2.2 Scientific Literature and Benchmark Datasets . . . 10
  - 2.2.1 Ocular and Facial Indicators for Drowsiness and Distraction 10
  - 2.2.2 Visual Attention Tracking: Head Pose Estimation vs. Direct 3D Gaze . . . 11
  - 2.2.3 Touchless Gestural Interfaces in Automotive HMIs . . . 13
  - 2.2.4 Benchmark Datasets for In-Cabin Gesture Recognition . . . 13
- 2.3 Enabling Technologies and Software Frameworks . . . 15
  - 2.3.1 Industrial DMS Platforms and Near-Infrared Sensing Hardware 16
  - 2.3.2 Google MediaPipe . . . 16
  - 2.3.3 LSTM and Bidirectional LSTM Networks . . . 18
  - 2.3.4 YOLO . . . 19

3 AI Pipeline Design and Dataset Creation 22
- 3.1 Software Architecture, Sensors, and Operational Requirements . . . 22
  - 3.1.1 Real-Time Performance Metrics and Latency Quantification 29

3.2 Spatiotemporal Feature Extraction 30
3.2.1 Spatial Landmark Normalization 30
3.2.2 Spatiotemporal Windowing 31
3.2.3 Tracking Robustness and Temporal Consistency 32
3.3 Design of the Gesture Recognition Model 32
3.3.1 Definition of the Gesture Vocabulary 32
3.3.2 Gesture Vocabulary Design Criteria and HMI Engagement Logic 35
3.3.3 Acquisition of the Custom Multi-View Dataset 35
3.3.4 GestureLSTM Neural Network Architecture 37
3.3.5 State Machine and Activation Logic 38
3.4 Engineering of the Driver Monitoring Module 39
3.4.1 Heuristics for Sleep and Fatigue Detection 40
3.4.2 Heuristics for Facial Distraction and Head Pose Detection . 41
3.4.3 Heuristics for Cognitive Distraction and Smartphone Detection 43
3.4.4 System Calibration and Environmental Robustness 45

4 Hardware Integration and Development on Android Automotive OS 46
4.1 Client-Server Communication via WebSocket Protocol 46
4.1.1 WebSocket Message Protocol and Connection Lifecycle . . . 48
4.2 Development of the Native Infotainment Bridge 49
4.2.1 Accessibility Service and Input-Event Injection 49
4.2.2 Phantom UI Design for DMS Alerts 51
4.2.3 DMS Alarm Logic: Duration, Rate-Limiting, and Prototype Limitations 53
4.3 Deployment on Edge Hardware Architectures 54
4.3.1 Embedded Hardware Specifications and Benchmarking Setup 54
4.3.2 Power Budget Analysis and Overcurrent Throttling Diagnosis 54
4.3.3 Pipeline Rationalization and Software Optimizations 55
4.4 System Interaction Scope, Limitations, and Global Action Roadmap 58
4.4.1 Current Focus Navigation Scope and In-App Limitations . . 58
4.4.2 Future HMI Roadmap: Decoupling Cursors and Mapping Global Actions 59

5 Experimental Validation Protocol 61
5.1 Research Objectives and Hypotheses 61
5.2 Experimental Design and Participant Selection 64
5.2.1 Ethics, Informed Consent, and Data Privacy 65
5.3 Definition of Driving Tasks 65
5.3.1 Task Workflows, Initial UI Focus, and Command Sequences 66

5.4 Instrumentation and Execution Procedure . . . 68
5.4.1 Experimental Apparatus, Sampling Rates, and Data Annotation Protocol . . . 68
5.4.2 Visual Distraction Tracking and Data Annotation Protocol . 69
5.4.3 Execution phases . . . 70
5.5 Objective and Subjective Evaluation Metrics . . . 71
5.5.1 Objective Performance and Visual Distraction Metrics . . . 71
5.5.2 Gesture Error Taxonomy . . . 73
5.5.3 Subjective Usability and Workload Evaluation Questionnaires 74

6 Results Analysis 76
6.1 Cross-Validation and Model Training . . . 76
6.1.1 GroupKFold Cross-Validation Methodology . . . 76
6.1.2 GroupKFold Results . . . 78
6.2 Final Model Training and Validation Performance . . . 82
6.2.1 Training Setup and Evaluation Protocol . . . 82
6.2.2 Learning Curve Analysis . . . 83
6.3 User Learnability and Initial Training Evaluation . . . 86
6.3.1 Training Duration Analysis (H_{1a}) . . . 86
6.3.2 Training Error Frequency (H_{1b}) . . . 87
6.3.3 Qualitative Observations on Gestural Adaptations and Cabin Dynamics . . . 87
6.3.4 Summary of Results . . . 88
6.4 Objective Task Performance and Spatial Precision . . . 88
6.4.1 Task Completion Time Analysis (H_{2a}) . . . 89
6.4.2 Spatial Precision and Error Frequency (H_{2b}) . . . 92
6.4.3 Pipeline Robustness and Overall Error Rate (H_3) . . . 93
6.4.4 Qualitative Observations on Kinematic Dynamics and Visual Attention Management . . . 94
6.4.5 Summary of Objective Task Performance . . . 94
6.5 Visual Distraction Analysis and Glance Behavior . . . 94
6.5.1 Open-Loop Shortcut Commands (H_{4a}: Task 1 Home and Task 2 Back) . . . 96
6.5.2 Closed-Loop Spatial Micro-Navigation Tasks (H_{4b}: Task 3 App Selection and Task 4 Climate Control) . . . 98
6.5.3 Summary of Visual Distraction Results . . . 99
6.6 Subjective Workload, Perceived Ease of Use, and System Usability . 100
6.6.1 Perceived Task Ease Analysis (H_5 – SEQ) . . . 100
6.6.2 Subjective Workload Assessment (H_6 – NASA-TLX) . . . 102
6.6.3 Global Usability Assessment (System Usability Scale) . . . 103
6.7 Qualitative User Feedback and Preference Analysis . . . 103

6.7.1 Overall Modality Preference (Q1) . . . 104
6.7.2 Perceived Benefits and High-Utility Actions (Q2) . . . 104
6.7.3 Usability Friction and Spatial Navigation Bottlenecks (Q3) . 105
6.7.4 Synthesis of Qualitative and Quantitative Findings . . . 105
6.8 Discussion of Results and Edge Case Handling . . . 106
6.8.1 Task-Dependent Workload and Interaction Friction . . . 106
6.8.2 Interpretation of User Preferences and Qualitative Feedback 107
6.8.3 Verification of Experimental Hypotheses . . . 107
6.8.4 Analysis of Edge Cases: Management of the Timeout Event 109

7 Conclusions and Future Work 111
7.1 Summary of Achieved Results . . . 111
7.2 Study Limitations . . . 112
7.3 Future Work . . . 114

Generative AI Disclosure 116

A Group K-Fold Cross-Validation 117

B Model Training 123

C Real-time inference on Edge Devices 129

Bibliography 144

List of Tables

2.1 Comparative analysis of primary benchmark datasets for hand gesture recognition in automotive environments. 15
2.2 Architectural and Performance Comparison Across YOLO26 Model Scales on the MS COCO Benchmark [32]. 21

3.1 Functional separation and operational metrics across deployment environments. 26
3.2 Hardware resource utilization and empirical latency metrics (30 FPS capture rate). 29

5.1 Summary of Research Hypotheses, Variables, Metrics, and Verification Methods. 63
5.2 Definition of navigation tasks, completion criteria, and error conditions. 67

6.1 Top 5 hyperparameter configurations identified during GroupKFold cross-validation ($K = 2$), ranked by perspective stability and mean accuracy. 79
6.2 Detailed per-class classification metrics evaluated on the validation set (216 samples). 85
6.3 Summary of User Learnability Metrics and Hypothesis Verification Results ($N = 15$). 88
6.4 Three-Way Repeated Measures ANOVA ($2 \times 2 \times 3$) results for Task Completion Time (TCT). 89
6.5 Three-Way Repeated Measures ANOVA ($2 \times 2 \times 3$) results for Interaction Error Frequency. 92
6.6 Summary of Objective Task Performance, Spatial Precision, and Robustness Verification ($N = 15$). 95
6.7 Compact Summary of Visual Distraction Metrics and Hypothesis Verification ($N = 15$). 100
6.8 Two-Way Repeated Measures ANOVA results for Single Ease Question (SEQ) scores. 101

6.9 Single Ease Question (SEQ) scores (Mean ± SD, scale 1–7; higher is better) with unadjusted and Holm-adjusted post-hoc paired t-tests ($N = 15$). 101
6.10 NASA-TLX workload subscale ratings (Mean ± SD, 20-point scale: 1–20) with unadjusted and Holm-adjusted paired t-test comparisons ($N = 15$). 102
6.11 Comparison of Primary Protocol Analysis (excluding timeout) and Sensitivity Analysis (incorporating the 132.0 s observed timeout duration) for Task 3 (T_3). 109

List of Figures

2.1 Mediapipe Hand landmarks in detail 17

3.1 Data processing pipeline of the unified HMI and DMS ecosystem. . 23
3.2 Elp USB RGB-IR Camera 24
3.3 Rapsberry Pi 4 Model B 27
3.4 Nvidia Jetson Orin NX (8GB) 27
3.5 The custom-designed chassis from Reply's Smart Mobility Lab housing both embedded processing units (Raspberry Pi 4 and NVIDIA Jetson Orin NX) into a self-contained test bench. 28
3.6 State transition diagram of the Finite State Machine governing the gesture-based HMI. 39

4.1 Android Automotive Operative System 49
4.2 Detailed deployment architecture of the inference server node executing on the NVIDIA Jetson Orin NX platform. The diagram illustrates the V4L2 video frame acquisition thread (30 FPS), the multi-threaded perception and inference pipeline (incorporating MediaPipe 3D keypoint extraction, Bi-LSTM ONNX engine execution, and geometric DMS monitoring, with the optional YOLO module omitted), and the asynchronous WebSocket network server operating on TCP port 9543. 56
4.3 Deployment architecture of the infotainment client node on Raspberry Pi 4 (AAOS 14), showing the incoming Ethernet WebSocket channel (port 9543), the Kotlin Accessibility Service, and the privileged root-shell engine for touch input injection. 57

6.1 Parallel coordinates visualization of the hyperparameter search space logged in Weights & Biases, illustrating the best run, characterized by `batch_size = 16`, `dropout = 0.2` and `hidden_size = 128` . . 80

6.2 Training and validation loss progression (left) and accuracy trajectory (right) for Fold 1 (Camera 1 → Camera 2) under the optimal hyperparameter configuration. 80
6.3 Training and validation loss progression (left) and accuracy trajectory (right) for Fold 2 (Camera 2 → Camera 1) under the optimal hyperparameter configuration. 81
6.4 Training and validation loss progression for the final GestureLSTM model on the combined dataset, highlighting the global minimum checkpoint at epoch 36 and early stopping termination at epoch 51. 83
6.5 Training and validation accuracy trajectories for the final GestureLSTM model, illustrating learning convergence across training epochs on the single-subject dataset. 84
6.6 9×9 confusion matrix of the final GestureLSTM model evaluated on the validation split (216 samples). 85
6.7 Task Completion Time (TCT) comparison between touchscreen and mid-air gesture interaction across spatial micro-navigation tasks (T_3, T_4, and overall mean). 90
6.8 Motor learning curves illustrating per-action execution speedup in Task 3 and total time reduction in Task 4 across trial repetitions. . 91
6.9 Comparison of Total Off-Road Time (TOT) and Percentage of Glance Time ($\%GT$) in open-loop shortcut tasks (T_1 and T_2), verifying hypothesis H_{4a}. 97
6.10 Longest Glance Duration (LGT) relative to the 2.0 s operational threshold for prolonged glances and Total Off-Road Time (TOT) in spatial navigation tasks (T_3 and T_4), verifying hypothesis H_{4b}. . . . 99

Acronyms

AAOS
Android Automotive OS

ACK
acknowledgment

ADAS
advanced driver-assistance systems

ADDW
Advanced Driver Distraction Warning

AI
Artificial Intelligence

ANOVA
Analysis of Variance

AOIs
Areas of Interest

API
Application Programming Interface

ART-ANOVA
Aligned Rank Transform ANOVA

Bi-LSTM
Bidirectional LSTM

CNN

Convolutional Neural Network

COCO

Common Objects in Context

DDAW

Driver Drowsiness and Attention Warning

DFL

Distribution Focal Loss

DMS

Driver Monitoring Systems

DV

Dependent Variable

EAR

Eye Aspect Ratio

ECUs

Electronic Control Units

FIFO

First-In-First-Out

FPS

Frame Per Second

FSM

Finite State Machine

GDPR

General Data Protection Regulation

GG

Greenhouse-Geisser

gRPC

Google Remote Procedure Call

GSR II

General Safety Regulation II

GT

Glance Time

H

Hypotheses

HMI

Human-Machine Interaction

HPE

Head Pose Estimation

HUD

Head-Up Display

HVAC

Heating, Ventilation, and Air Conditioning

ICR

IR-Cut Filter Removable

IEC

International Electrotechnical Commission

IOD

Inter-Ocular Distance

IoU

Intersection-over-Union

ISO

International Organization for Standardization

IV

Independent Variable

IVI

in-vehicle infotainment

KISS

Keep It Simple, Stupid

KSS

Karolinska Sleepiness Scale

LGT

Longest Glance Duration

LISA

Laboratory for Intelligent and Safe Automobiles

LoS

Line of Sight

LOSO

Leave-One-Subject-Out

LSTM

Long Short-Term Memory

MAR

Mouth Aspect Ratio

NASA

National Aeronautics and Space Administration

NASA-TLX

NASA Task Load Index

Nglance

Glance Frequency

NHTSA

National Highway Traffic Safety Administration

NIR

Near-Infrared

NLLLoss

Negative Log-Likelihood Loss

NMS

Non-Maximum Suppression

NUIs

natural user interfaces

OCSC

Over-Current Safety Circuitry

OEMs

Original Equipment Manufacturer

ONNX

Open Neural Network Exchange

PCCR

Pupil-Center Corneal Reflection

PERCLOS

Percentage of Eye Closure

PMIC

Power Management IC

RM-ANOVA

Repeated-Measures ANOVA

RNNs

Recurrent Neural Networks

ROI

Region of Interest

RQs

Research Questions

SD

Standard Deviation

SDV

Software-Defined Vehicle

SEQ

Single Ease Question

SoC

System-on-Chip

SUS

System Usability Scale

TCT

Task Completion Time

TOT

Total Off-Road Glance Time

UCSD

University of California, San Diego

UNIMORE

Università degli Studi di Modena e Reggio Emilia

V4L2

Video4Linux2

VHAL

Android Vehicle Hardware Abstraction Layer

WSS

WebSocket Secure

YOLO

You Only Look Once

Chapter 1

Introduction

1.1 The Evolution Towards the Smart Cockpit and Cognitive Load in Driving

Over the past decade, automotive interior design has undergone a profound transformation, moving from passive, hardware-bound mechanical cockpits to highly integrated, responsive, and software-driven environments [1]. This structural evolution is largely due to the automotive industry's transition to centralized computing and Software-Defined Vehicle (SDV) architectures. Traditional vehicle architectures relied on dozens of isolated Electronic Control Units (ECUs) communicating via legacy bus systems. In contrast, modern high-performance System-on-Chip (SoC) platforms allow previously separate domains, such as the digital cockpit, in-vehicle infotainment (IVI), advanced driver-assistance systems (ADAS), and body control gateways, to execute concurrently on unified virtualized hardware platforms.

By consolidating these systems into a single smart-cockpit architecture, safety features and cabin comfort functions are increasingly interconnected, leading to the rapid replacement of traditional tactile physical buttons, analog gauges and mechanical switches with large, multi-functional capacitive touchscreen displays [2].

Although these consolidated digital interfaces vastly expand in-vehicle functionality and design flexibility, they introduce a well-documented safety trade-off. In literature, driving is conceptualized as a dual-task activity comprising a primary task, which involves the control of the vehicle, such as steering, accelerating and braking, and various second tasks concerning all non-driving-related activities. Operating complex, nested menu structures on a touchscreen acts as a highly demanding *secondary task* that competes directly for the driver's limited cognitive

and attentional resources [2, 3]. In dynamic driving scenarios, these secondary interactions interfere with the continuous and safety-critical demands of the *primary task*.

According to the National Highway Traffic Safety Administration (NHTSA), driver distraction is defined as any activity that diverts a person's attention away from the primary task of operating and controlling the vehicle. In automotive Human-Machine Interaction (HMI) literature, this phenomenon is formally categorized into three distinct attentional domains [4]:

- **Manual Distraction:** Occurs when the driver physically removes one or both hands from the steering wheel, degrading immediate physical vehicle control and reducing preparedness for sudden road hazards.
- **Visual Distraction:** Occurs when the driver diverts their gaze away from the roadway and the peripheral driving environment to focus on internal cockpit displays or peripheral elements.
- **Cognitive Distraction:** Occurs when the driver's mental focus, situational awareness, and vigilance are degraded due to mental workload, sleepiness, or cognitive tunneling arising from secondary activities, even if their eyes are temporarily on the road.

Among these, visual inattention, drowsiness, and fatigue are particularly critical; large-scale naturalistic driving studies demonstrate that driver-related distraction and impairment are present in nearly 90% of crash events, with visual distraction alone contributing to over 68% of injurious collisions [5].

1.2 The Problem of Visual Distraction and Alternative Interfaces

Although on-board digital technology has advanced, not all new features translate into a benefit for road safety. The progressive replacement of physical knobs and buttons, in favor of capacitive touchscreens, allowed the use of complex graphical interfaces and sub-menus, to the detriment of driving attention [2]. Traditional physical buttons allow for a so-called *open-loop* interaction, as they rely on the driver's sense of touch and tactile memory. This enables them to locate and activate the command based on the shape, size, and physical position of the button, without the need for a visual control loop to ensure the successful completion of the action.

Vice versa, with touchscreen interaction, the driver is forced to operate in a *closed-loop* system where they must visually verify that they have placed their finger on the correct virtual button and wait for the screen's visual response to confirm the input, due to the lack of physical controls. This process is further complicated

by the presence of vibrations, bumps, and vehicle stresses, which cause unstable pointing, forcing the driver to repeat the action or make input errors [2, 6].

Visual-motor interaction with a touchscreen can be divided into three sequential phases:

- **Visual search:** The driver must look away from the road to search for and locate the desired icon on the digital interface.

- **Precision pointing:** The driver must remove their hand from the steering wheel and direct their finger to the display to tap the previously identified icon, attempting to compensate for the oscillations of the moving vehicle.

- **Visual verification:** Lacking physical feedback upon tapping, the driver must look at the screen again to ensure that the system has effectively received the command and that the action is correct.

This continuous cycle of searching and pointing causes the duration of individual off-road glances to frequently exceed the critical threshold of 2.0 seconds, a limit beyond which NHTSA guidelines highlight a progressive increase in collision risk [7]. To mitigate this problem, modern automotive research is actively focusing on the development of contactless natural user interfaces (NUIs), such as mid-air gesture control systems [8, 9, 10].

However, gesture interaction cannot be assumed to universally eliminate visual distraction across all secondary tasks. In this architecture, gesture controls operate under two distinct interaction paradigms:

- **Open-Loop Gesture Shortcuts:** Direct global commands (such as Home and Back) allow open-loop execution, requiring minimal to no visual feedback.

- **Closed-Loop Pointer Navigation:** Spatial menu navigation via a virtual pointer operates as a closed-loop system, where moving the focus cursor step-by-step necessitates continuous visual verification from the driver.

Consequently, rather than assuming a universal reduction in visual load, the experimental study in this thesis aims to empirically evaluate whether, and for which specific task categories, touchless gesture interaction offers a genuine benefit compared to capacitive touchscreens.

1.3 Motivation and Objectives: A Unified Ecosystem for Gesture Recognition and Driver Monitoring

Although Driver Monitoring Systems (DMS) and mid-air gesture recognition interfaces have been widely explored in the automotive domain for several years, they are often treated as two functionally separate entities. In traditional architectures, the DMS utilizes a front-facing Near-Infrared (NIR) camera, typically integrated into the instrument cluster or steering column, framing exclusively the driver's face to estimate fatigue and distraction. Conversely, gesture recognition systems generally require supplementary dedicated sensors (such as Time-of-Flight depth cameras or secondary NIR sensors) positioned on the cabin ceiling or center console. This sensor redundancy leads to an increase in production costs, system integration complexity, and overall electronic power consumption.

This thesis aims to address this hardware and software fragmentation by evaluating the feasibility of a unified ecosystem capable of managing both active driver monitoring (DMS) and touchless infotainment interaction (HMI). The proposed architecture relies on a single, cost-effective wide-angle *RGB-IR* camera positioned near the central rearview mirror, simultaneously capturing the driver's facial features and the spatial interaction volume for hand gestures. Geometric facial and hand landmarks are extracted using a single lightweight framework (*Google MediaPipe*) to reduce computational complexity and enable real-time execution on resource-constrained embedded targets.

To address the limitations of purely static or simple dynamic gestures, such as classic swipes that are prone to unintentional triggers during driving maneuvers, a specialized gesture vocabulary was designed and evaluated using a Bidirectional LSTM *(Bi-LSTM)* neural network trained on a custom dataset.

Rather than introducing discrete sequence accumulation delays between gesture attempts, the Bi-LSTM classification module operates continuously over a 30-frame FIFO sliding window (1.0 s of temporal context at 30 FPS). Once the initial buffer is populated, inference is triggered at every incoming frame step ($\Delta t \approx 33.3$ ms), yielding a post-recognition system latency of approximately 66.6 ms ($\mu = 2$ frames).

To evaluate the proposed ecosystem, the software architecture was investigated across two operational scopes: a full-featured workstation prototype, integrating MediaPipe, Bi-LSTM, and a YOLO-based smartphone detector for the user study, and an optimized embedded edge deployment on an *NVIDIA Jetson Orin NX*. On the embedded target, the architecture was rationalized by excluding the YOLO object detection module to maintain a stable, real-time rate of 30 FPS without

thermal or overcurrent throttling, focusing onboard active safety on lightweight geometric facial monitoring.

Building upon these methodological foundations, this research addresses three primary Research Questions (RQs):

- **RQ1 (Single-Camera Unified Feasibility):** Is it technically feasible to unify driver monitoring (DMS) and gesture-based infotainment control (HMI) using a single wide-angle camera without introducing hardware or sensor redundancy?
- **RQ2 (Visual Distraction and Usability):** To what extent does a gesture-based interface affect visual distraction and cognitive load across different secondary task categories compared to traditional touchscreen interaction?
- **RQ3 (Edge Real-Time Execution):** Can the rationalized vision pipeline (excluding the optional YOLO object detection module) maintain a stable, real-time performance of 30 FPS on a resource-constrained embedded platform (NVIDIA Jetson Orin NX)?

1.4 Contributions of the Thesis

To overcome the current fragmentation between driver monitoring and touchless interaction systems, this research provides the following four main original contributions:

1. **Unified Single-Camera Architecture:** Design and implementation of a multi-modal vision pipeline that integrates active Driver Monitoring (DMS) and gesture-based Human-Machine Interaction (HMI) using a single, centrally mounted wide-angle RGB-IR camera, evaluating the feasibility of reducing sensor redundancy in smart cockpit environments.
2. **Gesture Vocabulary and Temporal Normalization:** Definition of a specialized automotive gesture vocabulary designed to mitigate unintentional triggers caused by involuntary hand and body movements while driving. This is coupled with a Bidirectional LSTM (Bi-LSTM) framework that employs an anchor-based spatial normalization scheme, anchored to the initial sequence frame, to retain global trajectory dynamics while ensuring scale invariance.
3. **Human-Factors Empirical Evaluation:** A comparative experimental study conducted with 15 participants evaluating touchless gesture control against a conventional capacitive touchscreen. The study provides empirical quantitative data regarding off-road visual distraction evaluated using ISO 15007 glance

metrics across different task categories, subjective cognitive workload measured via NASA-TLX, and overall usability assessed through SUS and SEQ.

4. **Edge Deployment and Resource Trade-off Analysis:** A practical engineering analysis of the computational and power constraints encountered when transitioning from workstation prototyping to an embedded target (NVIDIA Jetson Orin NX). The work analyzes how pipeline rationalization—specifically module removal of the heavy YOLO object detection network—combined with GPU delegation and ONNX Runtime execution, helps mitigate overcurrent throttling and maintain a stable 30 FPS processing rate under the evaluated edge benchmark configuration.

1.5 Thesis Structure

The remainder of this thesis is organized as follows:

- **Chapter 2 (State of the Art):** Reviews the regulatory framework (GSR II, ADDW, Euro NCAP 2026), computer vision metrics for driver state estimation (EAR, MAR, PERCLOS, Head Pose), and existing literature on gesture recognition interfaces.

- **Chapter 3 (System Architecture and ML Pipeline):** Details the unified single-camera vision pipeline, the custom gesture dataset collection, the Bi-LSTM model architecture, and the geometric heuristics used for driver monitoring.

- **Chapter 4 (Distributed Architecture and Edge Deployment):** Describes the client-server WebSocket communication, the Android Automotive OS Accessibility Service integration, and the optimization techniques (ONNX conversion and GPU delegation) implemented on the NVIDIA Jetson Orin NX.

- **Chapter 5 (Experimental Methodology):** Presents the within-subjects user study design with 15 participants, defining the four experimental tasks, objective performance metrics, and subjective evaluation protocols (NASA-TLX, SUS, SEQ).

- **Chapter 6 (Results Analysis and Discussion):** Provides a detailed quantitative and qualitative analysis of model training performance, user learning curves, visual distraction metrics, cognitive workload, and system usability.

- **Chapter 7 (Conclusions and Future Work):** Summarizes the primary findings addressing the research questions, outlines the main limitations of the current prototype, and proposes directions for future research.

Chapter 2

State of the Art and Enabling Technologies

2.1 Regulatory Framework and Industry Standards

In the context of human-machine interaction (HMI) applied to the automotive sector, objectively evaluating driver visual distraction is fundamental to assessing road safety. This process relies on a standardized mapping of visual behavior, for which ISO 15007 (International Organization for Standardization) provides the established scientific and methodological framework for glance measurement and data reduction [11].

Within this methodological framework, transient visual deviations lasting less than 120 milliseconds are defined as *fly-throughs* artefacts of saccadic movements across intermediate Areas of Interest (AOIs) rather than deliberate visual fixations. Consequently, when processing driver visual behavior data, glance durations are operationalized above this 120 ms threshold to separate sustained off-road glances from involuntary saccades and visual transition noise [11].

Starting from this rigorous definition, the assessment of visual load is structured around three fundamental temporal and frequency metrics:

- **Glance Frequency (F)**: quantifies the number of glances directed towards a secondary target, such as the infotainment display, per unit of time or during the execution of a single driving task.
- **Single Glance Duration (t_g)**: represents the temporal persistence of a single visual deviation off the road. According to the guidelines of the NHTSA (the

National Highway Traffic Safety Administration), a duration of 2.0 seconds as an operational threshold established for identifying prolonged off-road glances, beyond which empirical naturalistic driving studies demonstrate a statistically significant increase in near-crash and crash risk [7].

- **Total Off-Road Glance Time (T_{total})**: defines the total accumulation of individual glances required to complete a secondary action. It is mathematically calculated as the sum of all individual glances made to complete that specific task:

$$T_{\text{total}} = t_{g,1} + t_{g,2} + \cdots + t_{g,n} \tag{2.1}$$

Alongside measurement standards, the adoption of these technologies is experiencing a strong acceleration due to an extremely stringent international legislative framework. In Europe, EU Regulation 2019/2144, known as the General Safety Regulation II (GSR II), imposes on car manufacturers the obligation of driver state monitoring through distinct regulatory mandates [12]:

- **DDAW (Driver Drowsiness and Attention Warning):** mandatory for new type-approvals since July 2022 and for all new vehicle registrations since July 2024 [12]. These systems operate predominantly through passive and indirect monitoring, assessing driver fatigue and drowsiness via vehicle dynamic metrics, such as steering wheel movement analysis or involuntary lane departures.
- **ADDW (Advanced Driver Distraction Warning):** mandatory for new type-approvals starting from July 2024 and for all new vehicle registrations from July 2026 [13]. Unlike indirect DDAW modules, ADDW mandates the active, real-time detection of visual distraction. Although the regulation is technology-neutral, compliance in industry practice relies on driver-facing cameras to track gaze orientation, identify prolonged off-road glances, and detect high-risk behaviors or cognitive inattention.

A further industrial incentive for the introduction of such technologies is represented by the Euro NCAP assessment protocols [14]. Under these assessment frameworks, equipped vehicles earn points in the *Occupant Status Monitoring* domain, where camera-based direct monitoring is established as a key requirement to achieve a five-star safety rating.

Specifically, the protocol requires the system to map the driver's visual focus across discrete spatial *Areas of Interest* (AOIs) within the cabin, distinguishing between primary roadway vision, side/rearview mirrors, instrument cluster, and the central *in-vehicle infotainment* (IVI) display. To ensure effective operation, driver monitoring algorithms must incorporate spatial and angular tolerances that

accommodate brief, necessary glances for secondary task coordination, thereby preventing intrusive acoustic or visual false alarms that would compromise user acceptance.

2.2 Scientific Literature and Benchmark Datasets

2.2.1 Ocular and Facial Indicators for Drowsiness and Distraction

In the context of active vehicle safety, computer vision-based driver monitoring systems rely heavily on extracting facial landmarks to evaluate physiological driver states, such as fatigue, drowsiness, and yawning. To translate raw facial landmark coordinates into robust indicators of driver alertness, literature establishes three fundamental geometric and temporal metrics: *Eye Aspect Ratio* (EAR), *Mouth Aspect Ratio* (MAR), and *Percentage of Eye Closure* (PERCLOS).

The foundational metric for ocular open-close estimation is the Eye Aspect Ratio (EAR), introduced by Soukupová and Čech [15]. EAR is a two-dimensional scalar value that quantifies the eye's opening degree by calculating the ratio between vertical eyelid distances and the horizontal eye width:

$$\mathrm{EAR} = \frac{\|p_2 - p_6\| + \|p_3 - p_5\|}{2 \cdot \|p_1 - p_4\|} \tag{2.2}$$

where p_1 and p_4 denote the coordinates of the outer and inner canthi (horizontal extremities of the eye), while p_2, p_3, p_5, and p_6 represent the vertical anatomical contours of the upper and lower eyelids. When the eye is open, EAR maintains a relatively high value; as the eyelid closes during blinking or sleep onset, EAR drops sharply towards zero, providing a scale-invariant threshold for blink and eye-closure detection.

Analogous to EAR, the Mouth Aspect Ratio (MAR) measures vertical jaw drop and lip deformation to identify yawning as an early behavioral symptom of fatigue [16]. MAR is computed using keypoints along the mouth's inner and outer contours:

$$\mathrm{MAR} = \frac{\|p_2 - p_8\| + \|p_3 - p_7\| + \|p_4 - p_6\|}{2 \cdot \|p_1 - p_5\|} \tag{2.3}$$

where p_1 and p_5 bound the outer commissures (corners) of the lips, while $p_2, \ldots, p_8$ define the internal vertical profile of the upper and lower lips.

While EAR and MAR evaluate instantaneous or short-duration events (such as individual blinks or discrete yawns), progressive drowsiness over extended periods is evaluated via PERCLOS (Percentage of Eye Closure), originally established by

Wierwille et al. [17] and Dinges et al. [18]. PERCLOS measures the proportion of time within a fixed observation window (typically 60 seconds) during which the driver's eyes are closed by 80% or more:

$$\text{PERCLOS} = \frac{T_{\text{closed}}}{T_{\text{total}}} \times 100 \tag{2.4}$$

where T_{closed} represents the cumulative duration (or frame count) in which EAR falls below the defined closure threshold, and T_{total} is the total window duration.

In driver monitoring literature, PERCLOS is recognized as a primary behavioral proxy for fatigue, often evaluated alongside subjective alertness scales such as the *Karolinska Sleepiness Scale* (KSS) introduced by Åkerstedt and Gillberg [19]. While empirical studies demonstrate a statistical correlation between elevated PERCLOS values (e.g., thresholds between 7% and 12%) and high self-reported sleepiness (KSS $\geq$ 7), PERCLOS represents an empirical heuristic rather than a deterministic physiological equivalence. Factors such as individual anatomical differences, circadian variations, ambient lighting shifts, and eye-smoothing filters necessitate adaptive thresholding or multi-feature fusion in practical automotive DMS implementations.

2.2.2 Visual Attention Tracking: Head Pose Estimation vs. Direct 3D Gaze

While physiological state monitoring aims to detect a decline in driver alertness, active collision prevention also requires tracking where the driver directs their visual attention moment by moment. In computer vision-based driver monitoring literature, evaluating visual attention relies primarily on two estimation techniques: *Head Pose Estimation* (HPE) and *Direct 3D Gaze Estimation*.

From a biomechanical and functional standpoint, head pose estimation and gaze tracking represent distinct estimation paradigms that cannot be conflated. Head pose estimation quantifies the 3D spatial orientation of the driver's head, typically expressed through *Euler* angles (*Yaw, Pitch, Roll*) relative to the camera frame. However, head pose serves only as an indirect proxy for visual attention. Drivers frequently exhibit decoupled head-eye coordination: for example, a driver may rotate their head slightly while maintaining ocular fixation on the primary roadway, or conversely, keep their face oriented straight ahead while laterally deflecting their eyes toward secondary displays or a handheld smartphone [20]. Relying solely on head orientation can therefore fail to capture active visual distraction caused by ocular deviation.

To overcome the limitations of head pose proxies, direct 3D gaze estimation computes the three-dimensional gaze vector representing the true *Line of Sight*

(LoS). Classical feature-based gaze estimation methods rely on geometric eye models, extracting local anatomical features such as pupil centers, iris contours, and corneal reflections (glints) produced by active *Near-Infrared* (NIR) illuminators.

While effective in controlled laboratory environments, feature-based methods degrade significantly in automotive cabins due to uncontrolled ambient lighting shifts, partial eyelid occlusions, and severe perspective distortions when recorded from off-center camera mounting positions. Furthermore, while NIR illumination mitigates low-light conditions, the capability to track gaze through sunglasses is not universal and depends heavily on the specific optical transmittance properties of the lenses in the NIR spectrum.

To achieve robust performance in unconstrained automotive environments, contemporary research emphasizes appearance-based deep learning frameworks, which regress 3D gaze vectors directly from full-face or eye-region RGB/IR pixels without explicit feature segmentation. Within recent appearance-based gaze estimation literature, two prominent convolutional neural network models are frequently examined:

- **Gaze360**: Proposed by Kellnhofer et al. [21], this architecture addresses unconstrained 3D gaze estimation by leveraging a pinhole and panoramic camera dataset featuring over 130,000 images captured across diverse indoor and outdoor settings. Incorporating temporal context via a Recurrent Neural Network (LSTM), Gaze360 exhibits high robustness across wide head rotation angles and extreme ocular gaze vectors.

- **L2CS-Net**: Introduced by Abdelrahman et al. [22], this model utilizes a ResNet-50 backbone to perform fine-grained gaze estimation in unconstrained settings. Rather than regressing continuous gaze angles directly, L2CS-Net frames yaw and pitch estimation as a joint classification and regression task, discretizing angular ranges into discrete bins before applying a continuous expectation operator and a combined Cross-Entropy and L2 loss function.

Starting from continuous 3D gaze vectors or head orientation angles, driver monitoring systems perform *Gaze Zone Mapping* [23]. This processing stage projects continuous angular estimates onto discrete spatial Areas of Interest (AOIs) within the cabin geometry, such as the forward windshield, rearview mirror, side-view mirrors, instrument cluster, and central infotainment display. This classification allows the system to discriminate between legitimate visual glances required for driving maneuvers and sustained deviations into off-road distraction zones.

2.2.3 Touchless Gestural Interfaces in Automotive HMIs

In modern automotive design, contactless gestural interaction originated as a *Natural User Interface* (NUI) aimed at overcoming the intrinsic visual and manual limitations of physical buttons and capacitive touchscreens [8]. Interacting with touchscreens forces the driver to take their eyes off the road to visually locate virtual controls, while simultaneously requiring them to temporarily remove at least one hand from the steering wheel. In contrast, touchless gestural interfaces provide an alternative input modality for executing secondary infotainment commands, such as adjusting the volume, skipping audio tracks, or navigating submenu, with a reduced dependency on closed-loop visual feedback.

In computer vision literature, hand gestures in the cabin are broadly categorized into two macro-categories:

- **Static Gestures (Hand Postures)**: Defined by a fixed spatial geometric configuration of the hand captured within a single temporal frame, lacking intrinsic motion (e.g., a closed fist, an "OK" sign, or a "Thumbs Up"). While computationally lightweight, static postures evaluated frame-by-frame are prone to unintended activations when drivers perform ambient hand movements, adjust their grip on the steering wheel, or gesture naturally during a conversation.
- **Dynamic Gestures (Dynamic Hand Gestures)**: Defined as spatiotemporal sequences composed of a continuous series of frames where both the configuration and spatial trajectory of the hand evolve over time (e.g., horizontal or vertical swipes, rotations, or pinch-to-zoom maneuvers).

While incorporating temporal motion dynamics provides richer feature representations that help differentiate deliberate control inputs from stationary hand poses, dynamic gestures in unconstrained cabin environments are not inherently immune to false positives. Driving maneuvers, gear shifts, and conversational gesticulation can still lead to accidental triggers. Consequently, automotive gesture recognition frameworks rely on temporal sequence modeling (such as Recurrent Neural Networks, 3D Convolutional Networks, or temporal sliding windows) and structured gesture vocabularies to reliably isolate intentional control inputs from background cabin activity.

2.2.4 Benchmark Datasets for In-Cabin Gesture Recognition

To train and benchmark computer vision models for gesture recognition in automotive environments, public datasets provide standardized ground-truth annotations

across varied lighting conditions, subject demographics, and camera viewpoints. Within automotive gesture recognition literature, three primary benchmark datasets are widely analyzed [8, 24, 10]:

1. **VIVA Hand Gesture Dataset (2014 - UCSD):** Collected by the Laboratory for Intelligent and Safe Automobiles (LISA) at the University of California, San Diego, this dataset was designed to study gestural activities in real driving conditions, characterized by strong lighting artifacts and frequent occlusions. It includes 19 dynamic gesture classes performed by 8 subjects (each recorded twice, once as a driver and once as a passenger). The acquisition was carried out in the car's center console area using the Microsoft Kinect sensor, providing RGB and depth maps at a 640x480 pixel resolution, for a total of 886 gesture samples. To facilitate classical vision algorithms, the subjects performed the gestures while resting their hand on a flat green surface [8].

2. **NVIDIA Dynamic Hand Gesture Dataset (NVGesture - 2016):** This dataset gathers 25 classes of dynamic gestures performed by 20 subjects inside an indoor car simulator. The recordings were made from two viewpoints (frontal and mounted on the car ceiling) using multiple sensors, specifically the SoftKinetic DS235 depth camera and the DUO 3D stereo camera. The dataset releases RGB channels, depth maps, and the precomputed optical flow of the sequences, representing a classic benchmark for multimodal analysis [24].

3. **Briareo Dataset (2019 - UNIMORE):** Developed by the University of Modena and Reggio Emilia, this dataset introduces an alternative upward-facing and strategic viewpoint: the acquisition sensors are positioned in the central tunnel of the cabin, between the driver and passenger seats, pointing upwards. This innovative placement significantly reduces occlusions caused by the driver's body and protects the sensors from direct sunlight. Briareo includes 12 dynamic gesture classes performed by 40 subjects (33 men and 7 women), each of whom repeats every gesture 3 times, for a total of 1440 sequences (each sequence lasts at least 40 frames). The released data includes standard camera RGB images, depth maps and infrared intensity extracted from the Pico Flexx ToF sensor, raw and rectified infrared images from a Leap Motion device, and the 3D coordinates of the hand joints extracted via its SDK [10].

To provide a structured comparative synthesis of these benchmark datasets, Table 2.1 summarizes their key structural parameters, sensor setups, camera framing, and primary operational limitations.

Table 2.1: Comparative analysis of primary benchmark datasets for hand gesture recognition in automotive environments.

Dataset	Year	Subj.	Class.	Samp.	Sensors	Camera	Primary Limitations
VIVA	2014	8	19	886	RGB, Depth	Center Console	Limited subject pool; hand constrained near console surface; sensitive to direct glare.
NVGesture	2016	20	25	1,532	RGB, Depth, IR	Frontal & Overhead	Recorded inside an indoor car simulator rather than a real moving vehicle.
Briareo	2019	40	12	1,440	RGB, Depth, IR, 3D Joints	Center Tunnel	Upward perspective frames ulnar edge, creating self-occlusions for overhead setups.

Starting from these benchmark databases, classical approaches to gesture recognition relied primarily on deep convolutional networks (such as 3D-CNNs or two-stream RGB-D architectures) that processed full video streams frame-by-frame at the pixel level. Although effective, these dense appearance-based models require considerable computational resources and introduce high inference latency, complicating real-time deployment on resource-constrained onboard hardware.

To address these computational barriers, contemporary research increasingly adopts a skeleton-based paradigm. By leveraging lightweight landmark extraction frameworks, systems isolate the hand by extracting exclusively 21 three-dimensional spatial coordinates corresponding to the anatomical joints of the fingers and palm. Processing vector coordinate sequences rather than full RGB frames significantly reduces spatial feature dimensionality and memory footprint, facilitating high-frame-rate sequence modeling on embedded Edge platforms [25, 26]. However, skeleton extraction modules remain upstream vision components; consequently, landmark detection accuracy can still be affected by severe in-cabin lighting shifts, rapid motion blur, and hand occlusions.

2.3 Enabling Technologies and Software Frameworks

To implement Driver Monitoring and touchless gesture recognition systems capable of operating in real-time on resource-constrained embedded platforms, contemporary research and industrial implementations rely on a synergistic set of enabling technologies. Specifically, the technological pillars of this scenario comprise four primary domains: active optical sensing hardware for all-weather cabin acquisition, lightweight frameworks for real-time landmark extraction (such as Google MediaPipe), real-time object detection architectures for identifying physical distraction elements (such as YOLO models), and dedicated Edge AI runtime acceleration

engines.

2.3.1 Industrial DMS Platforms and Near-Infrared Sensing Hardware

Beyond academic algorithmic models, commercial Driver Monitoring Systems (DMS) rely on specialized hardware architectures and integrated software frameworks to meet stringent automotive reliability standards. A representative commercial implementation is the *AIS+* system developed by Smart Eye [27], an embedded driver and occupancy monitoring platform that integrates real-time gaze tracking, facial expression analysis, and drowsiness detection. To notify the driver during critical distraction or microsleep events without generating intrusive false alarms for passengers, such industrial frameworks employ multimodal warning strategies, combining visual dashboard alerts with haptic feedback mechanisms, such as steering wheel or seat vibrations.

The operational robustness of these vision-based monitoring platforms across all driving conditions depends fundamentally on active Near-Infrared (NIR) illumination hardware. Standard RGB sensors are severely limited by low ambient light and night driving conditions. Consequently, automotive-grade DMS setups utilize active NIR camera modules operating within the 850–940 nm wavelength band [28]. Active NIR illuminators provide high-contrast ocular and facial imagery in complete darkness while remaining entirely invisible to the human eye. Furthermore, active NIR illumination enables pupil tracking through many commercial polarized lenses and sunglasses, provided the lens material exhibits sufficient optical transmittance within the NIR spectrum.

2.3.2 Google MediaPipe

Developed by Google Research, *MediaPipe* is an open-source framework designed for building lightweight, low-latency multimodal computer vision pipelines in resource-constrained environments [25]. Unlike classical bottom-up approaches, which detect joint pixels across the full image before assembling individual skeletons, MediaPipe adopts a hierarchical top-down pipeline.

The framework first identifies a macro *Region of Interest* (ROI) containing the target using a dedicated single-shot detector. Subsequently, local regression models estimate the precise spatial coordinates of the landmarks exclusively within the cropped bounding box. This approach significantly reduces the computation area for the keypoints, facilitating low-latency inference on commercial embedded platforms [29].

In the context of driver monitoring and gestural interfaces, MediaPipe provides two core perception solutions that operate independently or within a unified

execution graph.

MediaPipe Hand Landmarker

The hand tracking pipeline is structured as a two-stage flow:

1. **Palm Detection (BlazePalm):** The system utilizes an optimized single-shot detector to identify the palm region rather than directly detecting the full articulated hand. The model returns an oriented bounding box centered on the palm with high spatial precision [29].

2. **Hand Landmark Model**: On the cropped ROI provided by BlazePalm, a neural regression model calculates the spatial coordinates of 21 three-dimensional landmarks (x, y, z) corresponding to the anatomical joints of the fingers and palm (Figure 2.1). The x and y components represent the spatial position in the frame normalized within the $[0, 1]$ interval, while the z coordinate describes relative depth centered at the wrist joint (selected as the origin at depth zero). Operating on standard RGB cameras without depth hardware, the z coordinate represents an estimated depth relative to the wrist rather than a metric depth measurement. In addition to joint geometry, the model extracts semantic parameters, including tracking confidence scores and handedness.

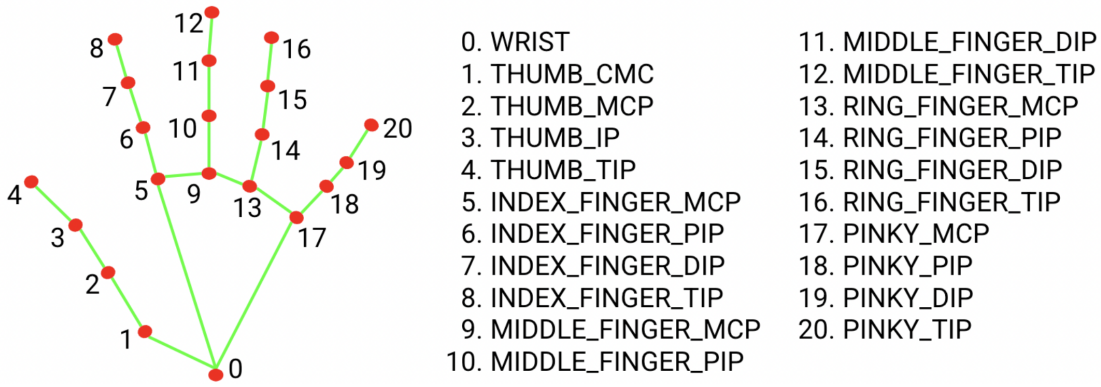


Figure 2.1: Mediapipe Hand landmarks in detail

To optimize runtime performance, full palm detection runs primarily during pipeline initialization or tracking loss; in subsequent consecutive frames, ROI cropping is derived directly from the landmark bounding box of the preceding frame as long as the hand remains in the field of view, reducing computational overhead.

MediaPipe Face Landmarker

Analogous to MediaPipe Hands, the facial tracking pipeline employs a three-stage hierarchical architecture:

1. **Face Detection (BlazeFace)**: A lightweight single-shot detector localizes the facial region across the full image frame and extracts a reference bounding box.

2. **Face Mesh Model**: On the cropped face ROI, a 3D mesh regression model estimates the spatial coordinates of 478 three-dimensional facial landmarks (x, y, z). The (x, y) coordinates are normalized relative to frame width and height, whereas z represents relative depth centered at the head origin. These precise 3D landmark coordinates allow for the direct geometric computation of traditional fatigue indicators, such as the Eye Aspect Ratio (EAR) and Mouth Aspect Ratio (MAR), via Euclidean distances between specific eyelid and lip contours.

3. **Face Blendshapes**: In parallel with geometric mesh regression, MediaPipe offers a neural blendshape classification model that outputs 52 normalized coefficients ranging from 0 to 1. These blendshape scores (e.g., `eyeBlinkLeft`, `eyeBlinkRight`, and `jawOpen`) quantify localized facial muscle contractions. Rather than deriving or calculating geometric EAR and MAR ratios, blendshapes serve as an alternative, data-driven facial expression score to complement rule-based ocular metrics for driver monitoring.

The same tracking loop optimization applies to the facial pipeline: face detection is executed only at initialization or upon tracking loss, while subsequent frames derive the crop region from the preceding landmark coordinates, reducing CPU and GPU computational load.

2.3.3 LSTM and Bidirectional LSTM Networks

To train a neural network on sequences of temporal data, historically, *Recurrent Neural Networks* (RNNs) were used, which allow information to propagate along successive time steps thanks to feedback connections. However, when the need arises to train a model on moderately long time series (as in the case of tracking dynamic gestures composed of several frames), traditional RNNs fail due to the well-known vanishing or exploding gradient problem. This prevents the network weights from updating correctly during the backpropagation phase, thus precluding the learning of long-term temporal dependencies.

To overcome this structural limitation, scientific literature introduced the *LSTM (Long Short-Term Memory)* architecture, specifically designed to preserve information over extended temporal intervals [30]. The operating principle of an LSTM cell is based on the presence of a "*conveyor belt*", namely the Cell State. This element represents a priority connection that runs through the entire computational chain, allowing information to flow along it almost unaltered and thereby preserving long-term dependencies [31].

The LSTM architecture regulates the retention, modification, and transmission of information through three specialized gating mechanisms that dynamically update the Cell State:

- **Forget Gate:** This component determines which historical information is no longer relevant and should be discarded from the Cell State. For instance, in natural language processing, when the model encounters a new subject, the forget gate may discard the gender of the previous subject to optimize the internal memory.

- **Input Gate:** This gate evaluates the incoming data from the current time step, identifying which new information is significant enough to be integrated and stored within the Cell State.

- **Output Gate:** This mechanism controls which portions of the updated Cell State are transmitted to the subsequent Hidden State, thereby determining the final activation used for the immediate prediction.

A natural and particularly effective evolution of this type of architecture is represented by *Bi-LSTM (Bidirectional LSTM)* networks. Unlike standard unidirectional models, Bi-LSTMs process the input sequence not only in chronological order (from beginning to end) but also in reverse order (from end to beginning) through the use of two independent recurrent layers operating in parallel, subsequently concatenating the outputs extracted from both layers. This approach produces a more robust data representation, capable of globally and symmetrically understanding the context of the hand's spatial movement (allowing, for example, the accurate discrimination of a swipe gesture's direction based on the entire temporal trajectory).

2.3.4 YOLO

Monitoring physical distraction inside the vehicle cabin, such as identifying a handheld smartphone or a phone-to-ear call gesture, requires a real-time object detection architecture capable of continuous inference with minimal latency. Within this domain, the *You Only Look Once (YOLO)* framework represents a widely adopted

paradigm for single-stage vision models, performing bounding box localization and class probability estimation in a single forward pass.

Modern iterations of the YOLO family depart significantly from early anchor-based formulations, adopting an anchor-free design with a decoupled detection head that separates classification and spatial bounding box regression.

For the Driver Monitoring System (DMS) proposed in this work, the active object detection pipeline integrates the recent *YOLO26* architecture developed by *Ultralytics* [32]. Designed specifically to address the deployment bottlenecks of real-time vision models on edge platforms, YOLO26 introduces two fundamental architectural optimizations:

- **Native End-to-End NMS-Free Inference**: Traditional object detectors generate multiple overlapping candidate bounding boxes per target and rely on *Non-Maximum Suppression* (NMS) post-processing to filter duplicates based on *Intersection-over-Union* (IoU) thresholds. Because NMS execution time scales dynamically with the number of candidate boxes and object density in the scene, it introduces non-deterministic latency spikes that disrupt synchronous execution streams on embedded hardware. YOLO26 enforces a one-to-one label assignment strategy during training, enabling the network to emit non-redundant predictions directly at inference time. This completely eliminates post-processing NMS, guaranteeing deterministic execution latency regardless of the complexity of the cabin scene [32].
- **Direct Regression Head (DFL-Free)**: Earlier anchor-free detectors model bounding box coordinates as continuous probability distributions via *Distribution Focal Loss* (DFL). While effective for fine spatial localization, resolving DFL distributions requires non-linear Softmax operators during post-processing, creating computational bottlenecks and quantization friction when exporting models to low-bitwidth integer formats (such as INT8) on neural hardware accelerators. YOLO26 removes DFL in favor of direct coordinate regression, yielding a lighter detection head and streamlining compilation for runtime engines like *ONNX Runtime* and *TensorRT* [32].

To accommodate diverse hardware and deployment constraints, the YOLO26 family provides five scaled capacities: Nano (n), Small (s), Medium (m), Large (l), and Extra-Large (x). To determine the optimal trade-off between detection accuracy and computational overhead for identifying distraction objects (such as a handheld smartphone, corresponding to Class 67 of the MS COCO taxonomy), these model scales were evaluated along the Pareto frontier relative to the system's execution budget [32]. Table 2.2 summarizes the architectural specifications and baseline evaluation metrics on the MS COCO benchmark.

Table 2.2: Architectural and Performance Comparison Across YOLO26 Model Scales on the MS COCO Benchmark [32].

Model Scale	Params (M)	GFLOPs	Size (MB)	mAP50-95 (COCO)	CPU Lat. (ms)	Trade-off Evaluation
YOLO26n (Nano)	2.57	5.81	5.3	40.9%	~ 26.7	Lower capacity for occluded objects
YOLO26s (Small)	**10.01**	**22.58**	**19.5**	**48.2%**	~ 44.3	**Pareto-Optimal**
YOLO26m (Medium)	21.90	74.88	42.2	53.5%	~ 103.0	High latency bottleneck on CPU
YOLO26l (Large)	26.30	93.28	50.8	55.8%	~ 131.7	Exceeds embedded execution budget
YOLO26x (XLarge)	58.99	208.76	113.2	57.5%	~ 219.0	Prohibitive computational footprint

As detailed in Table 2.2, YOLO26s represents the Pareto-optimal configuration for real-time Edge deployment:

- **Trade-off vs. YOLO26n (Nano)**: Although the Nano variant (`yolo26n`) minimizes memory footprint (5.3 MB) and CPU latency, its representation capacity is significantly lower (40.9% mAP). In a driver monitoring scenario, this reduced capacity increases the risk of false negatives when a smartphone is partially occluded by the driver's hand or held at an oblique angle relative to the camera.
- **Trade-off vs. Larger Variants (Medium / Large / XLarge)**: Moving from Small to Medium (`yolo26m`) incurs more than a 230% increase in floating-point operations (from 22.58 to 74.88 GFLOPs) and more than doubles CPU inference latency ($\sim$ 103.0 ms per frame). This latency overhead violates the strict real-time constraint of 30 FPS ($\leq$ 33.3 ms budget) required for continuous HMI and DMS thread execution.

Consequently, deploying the pre-trained YOLO26s weights (`yolo26s.pt`) ensures high localization sensitivity for COCO Class 67 while maintaining bounded execution latency and low memory consumption on the target onboard hardware [32].

Chapter 3

AI Pipeline Design and Dataset Creation

3.1 Software Architecture, Sensors, and Operational Requirements

The development and experimental validation of the unified DMS and HMI system proposed in this thesis were conducted as part of an applied research and experimental development project during an industrial internship at the *Concept Engineering* business unit of the Turin-based company *Reply*. The engineering activity focused on designing a unified architecture capable of recognizing dynamic gestures in real-time while driving for infotainment interaction and, simultaneously, identifying and classifying the driver's potentially distracting behaviors. To meet the strict cost and computational efficiency requirements typical of real onboard systems, the research extended to the development of a lightweight software solution, optimized to run on constrained onboard hardware, such as the NVIDIA Jetson Orin NX embedded platform. The high-resolution camera and embedded target hardware, specifically the NVIDIA Jetson Orin NX and *Raspberry Pi 4* platforms, were provided directly by the company, enabling realistic model deployment and evaluation within an industry-standard hardware setup. The prototype was integrated into the existing experimental infrastructure and hardware setup available at Reply's Smart Mobility Lab, located in Area42 in Turin [33]. The laboratory provided the physical infrastructure and hardware resources supporting the work.

The perception unit of the proposed architecture consists of an ELP USB camera (Fig. 3.2) equipped with a 1.56 mm wide-angle lens, an *OV2710 CMOS* sensor, an integrated ambient light photodiode, and an array of 10 Near-Infrared (NIR) LEDs operating at an 850 nm wavelength. To maintain continuous perception

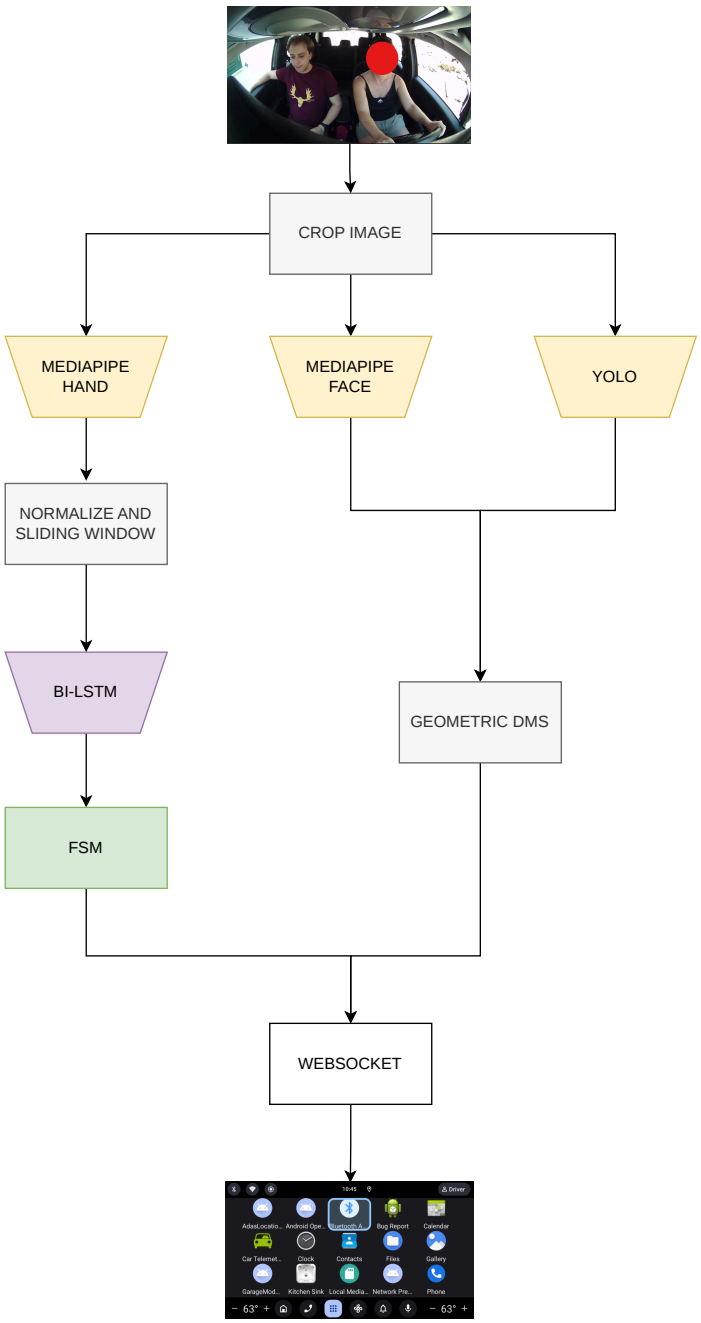


Figure 3.1: Data processing pipeline of the unified HMI and DMS ecosystem.



Figure 3.2: Elp USB RGB-IR Camera

across variable cabin lighting conditions, the camera incorporates an automatic mechanical *IR-Cut Filter Removable* (ICR) mechanism driven by the photodiode, operating under two distinct imaging regimes:

- **Daylight Conditions (Visible Spectrum)**: When ambient light exceeds the operational threshold, the photodiode maintains the IR-Cut filter in front of the CMOS sensor to block infrared interference, outputting a standard full-color RGB video stream (1920×1080 pixels at 30 FPS).
- **Nighttime / Low-Light Conditions (NIR Spectrum)**: When ambient light drops below the threshold, the photodiode triggers the mechanical removal of the IR-Cut filter (ICR switch) and simultaneously activates the 850 nm NIR LED array. The camera automatically switches to outputting a high-contrast monochromatic (grayscale) infrared stream, preserving sharp anatomical contours across facial and hand keypoints for MediaPipe and YOLO regardless of cabin darkness.

To ensure reliable facial and hand keypoint extraction during night driving, active Near-Infrared (NIR) illumination is required. While commercial automotive systems often prefer 940 nm emitters to eliminate the faint red glow visible to the human eye at 850 nm, photobiological ocular safety is fundamentally governed by radiant intensity, exposure duration, and installation geometry rather than wavelength alone.

To ensure reliable keypoint extraction under low-light and night driving conditions, active Near-Infrared (NIR) illumination is required. While commercial

automotive systems often prefer 940 nm emitters to eliminate the faint red glow visible at 850 nm, eye safety is determined by radiant power, exposure time, and installation distance rather than wavelength alone. In the proposed setup, placing the low-power 850 nm LED array at a nominal distance of 50–70 cm from the driver conforms to the *IEC 62471* photobiological safety standard [34], ensuring high-contrast infrared imaging without visual distraction or eye hazard.

To optimize operational efficiency and spatial feature quality, the perception pipeline employs a two-tier Region of Interest (ROI) processing mechanism:

- **Primary Static Crop (**1120×880**)**: A fixed bounding box (1120×880 pixels) is isolated from the native Full HD frame (1920×1080) starting at offset coordinates ($x_{\text{start}} = 400, y_{\text{start}} = 200$). Given the short focal length (1.56 mm) of the wide-angle lens, the uncropped frame captures an overly broad view accompanied by severe peripheral optical distortions (barrel distortion). This primary crop discards the degraded outer boundaries, retaining a clean, geometrically stable representation of the central cabin area.

- **Secondary Configurable Sub-Crop (Driver Isolation Window)**: Since the camera in the experimental prototype was non-permanently mounted, its physical alignment varied slightly across test sessions and vehicle setups. To accommodate these positional shifts without altering model architectures, a secondary configurable sub-ROI was implemented within the primary canvas. During initial system setup, this window is statically adjusted to target strictly the driver's seat and interaction workspace, effectively isolating the driver from passenger movements and external background clutter.

By decoupling the fixed distortion-filtering crop from the configurable sub-crop, the system allows full adaptability to minor camera positional variations, supplying MediaPipe and YOLO with dedicated, light-weight input tensors.

At a logical level, the runtime software architecture is structured according to a distributed client-server paradigm, designed to clearly separate the processing of video streams (AI inference on the server) from the actual HMI interaction (interface control on the client). The inference Server, developed entirely in Python, manages the acquisition of video streams from the camera, executes the artificial intelligence pipelines in real-time, and hosts a local *WebSocket* server. The HMI Client, implemented in Kotlin, runs on a *Raspberry Pi 4 Model B* board (Fig. 3.3), equipped with a custom build of the *Android Automotive OS 14* operating system. The latter acts as a receiver for the commands sent by the server, translating them into specific control actions displayed and applied directly on the vehicle's infotainment interface.

To evaluate the proposed ecosystem under both full-featured functional conditions and resource-constrained hardware scenarios, the inference Server was operated according to two distinct deployment profiles:

1. **Full-Featured Workstation Profile (Experimental User Study Environment)**: Executed on a high-performance host Workstation connected via WebSocket to the AAOS simulator on Raspberry Pi 4, this configuration runs the complete AI perception pipeline concurrently at a stable 30 FPS. It executes MediaPipe Face Landmarker, MediaPipe Hands with the Bi-LSTM gesture classifier, and the YOLO26s object detection model ($\tau_{\text{yolo}} = 0.70$) for secondary device handling. This setup served as the reference platform for the experimental user study with 15 participants.

2. **Embedded Edge Deployment Profile (NVIDIA Jetson Orin NX Benchmark)**: Operating directly on the NVIDIA Jetson Orin NX embedded platform (Fig. 3.4), this setup serves as the target edge portability and hardware benchmark environment. On the Jetson platform, the architecture was evaluated both in a full-pipeline configuration (revealing thermal and power trade-offs under concurrent YOLO execution) and in a rationalized edge profile (MediaPipe GPU delegation + ONNX Bi-LSTM inference) maintaining a deterministic 30 FPS rate focused on geometric DMS and gesture control.

Table 3.1: Functional separation and operational metrics across deployment environments.

Environment	Hardware Setup	Active ML Modules	Target FPS	Supported Features
Workstation (User Study)	Laptop PC (CPU/GPU) + Touch + AAOS Simulator	MediaPipe Face + Hands, Bi-LSTM, YOLO26s	30	Full DMS (Geometric + Smartphone/Call) + Gesture HMI
Jetson Edge (Full Evaluation)	NVIDIA Jetson Orin NX (8GB)	MediaPipe GPU, Bi-LSTM ONNX, YOLO26s	~ 20	Full Pipeline (Exhibits Overcurrent Throttling)
Jetson Edge (Optimized)	NVIDIA Jetson Orin NX (8GB)	MediaPipe GPU, Bi-LSTM ONNX Runtime	30	Geometric DMS (Head Pose, Eyes, Yawn, Face Lost) + Gesture HMI

The software architecture of the Python server is designed as a unified multi-threaded ecosystem, conceived to coordinate and simultaneously execute three

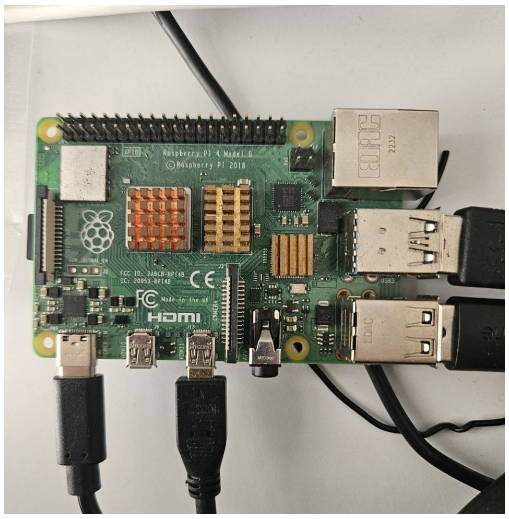


Figure 3.3: Rapsberry Pi 4 Model B

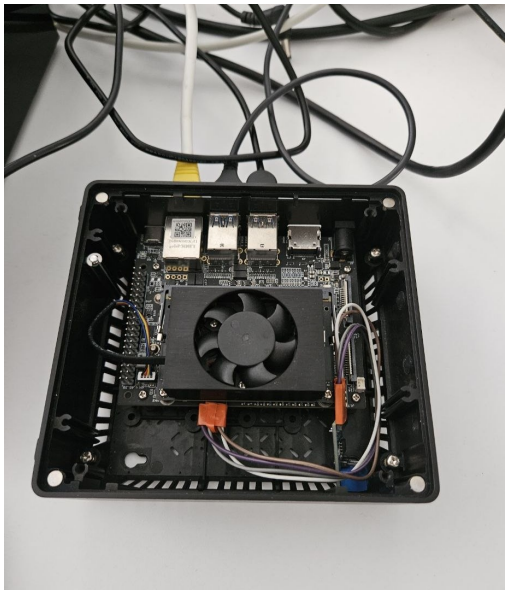


Figure 3.4: Nvidia Jetson Orin NX (8GB)

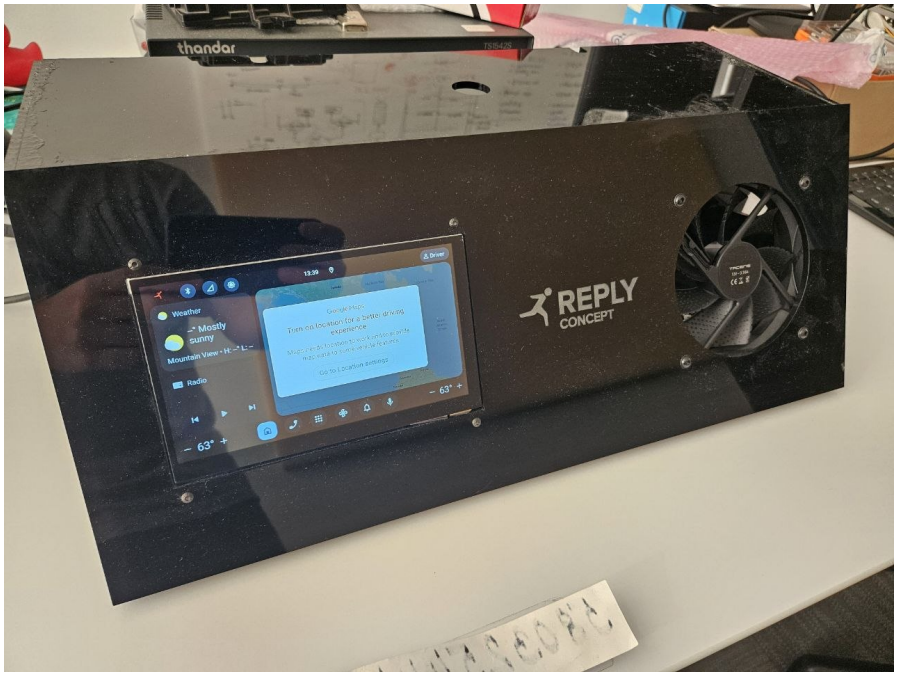


Figure 3.5: The custom-designed chassis from Reply's Smart Mobility Lab housing both embedded processing units (Raspberry Pi 4 and NVIDIA Jetson Orin NX) into a self-contained test bench.

main computer vision tasks: facial mesh tracking, three-dimensional hand keypoint extraction, and active object detection. In order to maximize overall performance and avoid bottlenecks or dropped frames (frame drops), the computational pipeline is divided into parallel execution threads. The first thread leverages the Google MediaPipe framework in the asynchronous $LIVE_STREAM$ mode. Using dedicated callback functions ($face_callback$ and $hand_callback$), the system receives and processes the geometric results without blocking the acquisition and the main execution flow. This module performs simultaneous tracking on two levels: on one hand, it detects the dense 478-point 3D facial mesh, extracting the 52 normalized expressive blendshapes useful for real-time monitoring of the driver's attention and drowsiness states; on the other hand, it tracks the 21 3D landmarks of the right hand necessary for the classification of dynamic gestures via a recurrent Bi-LSTM network. In parallel, the second thread runs the custom YOLO26s model (loaded via the yolo26s.pt weights) with a confidence threshold set to 0.7, constantly monitoring the driver's physical distraction by identifying a smartphone (target class ID 67).

The classified dynamic gestures and active DMS alarms are finally transmitted in real-time to the client using a persistent WebSocket connection, established locally

on port 9543. Upon starting the infotainment system, the Kotlin client application connects to this socket; whenever it receives a data packet from the Python server, a native accessibility service (*Android Accessibility Service*) developed in Kotlin intercepts the command and simulates the interaction on the Android Automotive display.

3.1.1 Real-Time Performance Metrics and Latency Quantification

To evaluate the operational readiness of the vision pipeline for automotive HMI applications, system throughput, memory occupation, and control latency were empirically benchmarked on the target hardware (NVIDIA Jetson Orin NX, 8GB module).

Table 3.2: Hardware resource utilization and empirical latency metrics (30 FPS capture rate).

Metric / Subsystem	**Target Benchmark**	**Empirical Value**
Video Pipeline Throughput	30 FPS	**30** FPS (Constant)
Gesture Onset Recognition	–	500.0–666.7 ms
Post-Recognition Visual Latency	< 100.0 ms	≈ 66.6 ms
System RAM Occupation	< 7.3 GB	$\approx$ **5.0** GB / 7.3 GB
Power Consumption (VDD_IN)	< 25.0 W	**11.0**–**15.0** W

The end-to-end control dynamics and total system latency were empirically quantified via 30 FPS frame-by-frame video analysis ($\Delta t = 33.33$ ms per frame) from physical gesture initiation to UI execution on Android Automotive OS (AAOS). This video-based methodology was specifically adopted to capture the true End-to-End response time, including the visual feedback on the infotainment display, which would otherwise be unmeasurable via software logging alone due to internal OS rendering pipelines and UI redraw overheads:

1. **Dynamic Feature Accumulation:** From gesture onset, the Bi-LSTM sliding-window network requires between 15 and 20 frames (500.0–666.7 ms) to accumulate discriminative hand keypoint trajectories and exceed the classification confidence threshold ($\geq 85\%$).

2. **Preemptive Classification Effect:** Because terminal trajectory frames typically convey low semantic information, the classifier reaches decision confidence prior to physical hand posture termination.

3. **Visual Feedback Latency:** Following gesture classification, the temporal gap introducing WebSocket payload transmission, Android Accessibility Service injection, and screen redraw ranges between 1 and 3 frames, with a mean post-recognition latency of **66.6** ms (2 frames).

Since the post-recognition latency (≈ 66.6 ms) remains strictly below the established 100 ms threshold for human perceptual immediacy [35], the system delivers immediate visual confirmation upon gesture completion without introducing perceptual lag for the driver.

3.2 Spatiotemporal Feature Extraction

3.2.1 Spatial Landmark Normalization

As previously mentioned, the developed system utilizes the MediaPipe framework to extract the keypoints of the right hand and face in real time. In particular, by leveraging the MediaPipe Hand Landmarker solution, it is possible to retrieve 21 three-dimensional landmarks mapping the skeleton and joints of the hand (from the origin of the wrist to the tips of the fingers). The spatial coordinates x and y natively provided by the model are normalized in the interval $[0, 1]$ with respect to the width and height of the ROI (the 1120×880 pixel computation window). The z coordinate, on the other hand, describes the relative depth of the joint with respect to the wrist (chosen as the reference point at zero depth).

To be used as input for the neural network, the hand landmarks must be spatially normalized. Without this preparation phase, the landmarks would vary continuously depending on the position of the hand in the frame or the specific physiognomy of the driver. To ensure spatial invariance while preserving temporal trajectory kinematics, a normalization pipeline structured in two sequential phases is applied.

Let $\mathbf{p}_{i,t} = [x_{i,t}, y_{i,t}, z_{i,t}]^T$ denote the 3D spatial position of hand joint $i \in \{0, \dots, 20\}$ at frame $t \in \{0, \dots, 29\}$, where $i = 0$ corresponds to the wrist.

1. **Translation invariance**: To ensure that the absolute position of the hand within the cabin does not affect the classification, the wrist position at the initial frame ($t = 0$), denoted as $\mathbf{p}^{(0)}_{\text{wrist}} = \mathbf{p}_{0,0}$, is set as the sequence local anchor origin. The translated coordinates for each joint across the 30-frame sequence are computed as:

$$\mathbf{p}'_{i,t} = \mathbf{p}_{i,t} - \mathbf{p}^{(0)}_{\text{wrist}} \quad \forall i \in \{0, \dots, 20\}, \forall t \in \{0, \dots, 29\} \tag{3.1}$$

By anchoring the spatial translation exclusively to the wrist position at frame $t = 0$, the absolute starting position in the cabin is removed, whereas the global

spatial trajectory and directional displacement vectors of dynamic gestures (such as horizontal swipes) are preserved across the temporal window.

2. **Scale invariance**: To avoid the issue of the hand assuming different apparent sizes as it moves closer to or further from the camera, scale normalization is applied. The system calculates the maximum absolute coordinate value across the entire translated sequence matrix to serve as the common sequence scale factor M:

$$M = \max_{t\in\{0,\dots,29\}} \max_{i\in\{0,\dots,20\}} \left(|x'_{i,t}|, |y'_{i,t}|, |z'_{i,t}|\right) \tag{3.2}$$

Subsequently, each translated keypoint vector is scaled by M to obtain the final normalized coordinates $\mathbf{p}''_{i,t} = [x''_{i,t}, y''_{i,t}, z''_{i,t}]^T$:

$$\mathbf{p}''_{i,t} = \frac{\mathbf{p}'_{i,t}}{M} \quad \forall i \in \{0,\dots,20\}, \forall t \in \{0,\dots,29\} \tag{3.3}$$

To avoid numerical instability or division-by-zero exceptions in static or degenerate hand configurations where $M = 0$, the scaling step is conditionally applied only when $M > 0$, leaving the zero-tensor unchanged.

This operation allows all coordinates of the hand skeleton to be confined within the interval $[-1, 1]$, making the final feature vector spatially invariant with respect to the initial hand position while retaining full trajectory dynamics.

3.2.2 Spatiotemporal Windowing

To effectively recognize and classify dynamic gestures, it is not sufficient to analyze the spatial configuration of the hand keypoints at a single instant; rather, it is essential to study their evolution within a well-defined temporal window. Taking the architecture of the Briareo dataset [10] as a methodological reference, which adopts a 40-frame window, it was decided to implement a temporal window of 30 frames (1 second of observation at 30 FPS). This window is managed at runtime via an asynchronous queue with FIFO (*First-In-First-Out*) logic, which stores the hand movements in real time. At each inference step, the sliding window generates a spatiotemporal input tensor with a dimensionality of 30×63, where the value 63 corresponds to the instantaneous feature vector (composed of the 21 three-dimensional landmarks multiplied by the spatial coordinates of x, y, z), while 30 represents the temporal sequence of the accumulated frames.

3.2.3 Tracking Robustness and Temporal Consistency

The temporal sliding window relies on a nominal sampling rate of 30 frames per second (FPS), corresponding to a 1.0-second temporal observation horizon ($\Delta t = 33.3$ ms between frames). To maintain precise temporal synchronization during real-time execution, MediaPipe's `LIVE_STREAM` mode is utilized, associating each frame with a hardware timestamp t_k derived from system time.

In order to ensure the robustness of the system against partial occlusions or temporary tracking failures (due, for example, to the interposition of the steering wheel or sudden changes in lighting), a software tolerance mechanism was integrated, set to a maximum limit of 10 consecutive frames (approximately 333 ms). If the hand is not detected for an interval below this threshold, the system compensates for the lack of data by replicating the 63 keypoints extracted in the last valid frame, thereby preserving the temporal continuity of the trajectory and avoiding discontinuities or velocity spikes in the signal. If, however, the absence of tracking exceeds the 10-frame tolerance, the queue is completely emptied to prevent incorrect or unstable predictions, instantly resetting the classifier state while awaiting a new valid sequence.

Furthermore, to address potential camera framerate fluctuations during deployment, linear temporal interpolation across timestamp values t_k can be applied to resample arbitrary frame rate streams into uniformly spaced 30-sample tensors, guaranteeing full invariance to hardware inference latency.

3.3 Design of the Gesture Recognition Model

3.3.1 Definition of the Gesture Vocabulary

To verify the feasibility of the spatial feature extraction pipeline and the subsequent training of the temporal classifier, the public dataset Briareo [10] was initially used. This dataset consists of video sequences recording 12 gesture classes, repeated 3 times by 40 different subjects. Unlike the original approach proposed by the dataset's creators, which was based on the use of three-dimensional convolutional neural networks (3D-CNN) applied directly to raw video streams, in this work we chose to implement a significantly lighter asynchronous pipeline. In this setup, each RGB frame was processed via the MediaPipe Hands module to extract a vector of 63 features (corresponding to the 21 three-dimensional hand landmarks), which was subsequently sent as input to an LSTM recurrent neural network.

Preliminary tests conducted on Briareo confirmed the theoretical soundness and feasibility of the system, demonstrating high accuracy and low-latency real-time inference. However, this phase raised a problem of perspective discrepancy: the Briareo dataset was in fact acquired by positioning the cameras in the lower part

of the cabin (in the center console) pointing upwards. Consequently, the sensors primarily recorded the ulnar edge of the driver's hand. In the real hardware configuration developed for this thesis, conversely, the ELP camera is installed in the upper part of the cabin, above the central rearview mirror, offering a frontal perspective that directly frames the palm of the hand. Due to this geometric difference, the models trained on the Briareo dataset could not be transferred to the real system, as the perspective deformations and self-occlusions of the fingers were completely different.

Given the absence in the literature of an available public dataset recorded with a framing geometry similar to the company's setup, it was decided to develop a custom proprietary dataset. The definition of the final gestural vocabulary, however, was the result of a long iterative design process, divided into several experimental phases.

Swipe gestures

Initially, a preliminary dataset was recorded focusing on 4 classic swipe gestures: swipe right, swipe left, swipe up, and swipe down. Each class consisted of 120 sequential samples lasting 40 frames each. During runtime inference testing, however, a severe critical issue emerged related to the natural kinematics of human movement: when a subject performs a rapid swipe (for example, to the right), they instinctively tend to make a preparatory micro-movement in the opposite direction (a "wind-up" to the left) to build inertia. This behavior introduced strong temporal ambiguity in the LSTM buffer, which erroneously classified the swipe right as a swipe left. In an attempt to solve the problem, a delayed activation logic was introduced, waiting for a compensatory counter-movement, but this solution introduced an inadequate processing latency for the context of a real-time system. For this reason, simple swipes were abandoned in favor of movements where the activation is well-defined.

Pinch-and-drag gestures

Subsequently, an attempt was made to implement dynamic drag gestures based on bringing the thumb and index finger together (pinch and drag). In this case as well, field tests highlighted a physical limitation inherent in the nearly frontal position of the camera relative to the hand. During the execution of the pinch, the closing of the fingers on the camera's perspective plane made the geometric configuration of the landmarks almost identical to that of a closed fist. This led the network to systematically confuse the two states. In a real driving context, confusing an intentional drag with a closed fist, which could be an involuntary movement or a natural grip on the steering wheel, would have caused frequent unwanted activations, compromising the safety of the interface.

Thumb gestures

To address these limitations and ensure robust, reliable, and unambiguous interaction, it was decided to entirely redesign the vocabulary, selecting dynamic gestures characterized by a unique and highly differentiated geometric profile throughout their entire temporal development. In particular, a dataset composed of 9 distinct classes was structured:

1. **Fist:** It consists of a rapid closing of the fingers into a fist followed by their reopening.
2. **Flip:** It consists of an axial rotation of the wrist that exposes the back of the hand to the camera, followed by a return to the initial position with the palm visible.
3. **Pinch:** It consists of an impulsive contact between the fingertips of the thumb and index finger, followed by an immediate release, while the remaining three fingers (middle, ring, and pinky) remain extended and open.
4. **Index Right:** It consists of the closing of all fingers except the index finger, coordinated with a lateral shift of the entire hand to the right. Throughout the entire movement trajectory, the index finger must remain constantly pointing upwards (orthogonal to the translation axis).
5. **Index Left:** Mirroring the previous one, it consists of the closing of all fingers except the index finger, associated with a translation of the hand to the left, keeping the index finger pointing upwards for the entire duration of the sequence.
6. **Index Down:** It consists of the closing of all fingers except the index finger, accompanied by a rapid downward flexion of the wrist. From a geometric perspective, the index finger rotates until it points towards the bottom of the ROI.
7. **Thumb Up:** It consists of closing all fingers into a fist except the thumb, which is extended and pointed upwards, followed by a reopening or release of the hand.
8. **Shaka (or Phone):** It consists of the maximum simultaneous and exclusive extension of the thumb and pinky finger, with the three central fingers (middle, ring, index) clenched against the palm, all coordinated with a slight rotation of the wrist to the left.
9. **Neutral:** It represents the negative (or background) class of the model. It was designed to actively capture and model everything that does not constitute

an intentional gesture from the vocabulary. It includes samples of the hand completely still on the steering wheel, involuntary movements, the open hand moving randomly in the cabin, unencoded closed fists in motion, and incomplete or transitory swipe gestures. The explicit training of this classwas implemented with the aim of enhance the robustness of the classifier at runtime, reducing the rate of accidental activations.

3.3.2 Gesture Vocabulary Design Criteria and HMI Engagement Logic

The definition of the gesture vocabulary was guided by a multi-objective strategy balancing machine learning separability, biomechanical ergonomics, driving safety, and intuitive HMI interaction. Four fundamental criteria governed the selection and command mapping of the 9 target classes:

1. **Spatio-Temporal Distinguishability**: Each gesture possesses a distinct geometric skeleton topology and temporal velocity profile, maximizing linear separability within the 3D landmark feature space.

2. **Biomechanical Ergonomics**: Motions are localized to short-range trajectory bounds near the steering wheel or central console, requiring minimal muscular effort.

3. **Driving Safety (Eyes-on-the-Road)**: Gesture mechanics are designed for blind execution without visual feedback, preventing off-road visual glances and minimizing cognitive dual-task workload during driving.

4. **Touchscreen-Analogous Engagement and Semantic Mapping**: To resolve spatial gesture ambiguities directional dynamic gestures incorporate a positional trigger: extending the index finger acts as a virtual "touch-down" signal, while returning the open hand is classified as `Neutral`. Furthermore, commands are mapped intuitively using spatial metaphors (swipes for navigation) and iconic signs (`fist` for home, `flip` for back).

3.3.3 Acquisition of the Custom Multi-View Dataset

The recording sessions were conducted in a laboratory environment on a desk, mounting the *ELP USB RGB-IR* camera on a 50 cm high tripod positioned approximately 10 cm above the subject's eye level. To faithfully simulate the real installation geometry above a vehicle's central rearview mirror shifted to the right of the driver, two distinct sensor positions were used, thus ensuring greater spatial robustness to the dataset:

- **First Configuration:** The camera was positioned at a frontal distance of approximately 57 cm and laterally shifted 38 cm to the right relative to the subject's axis, recording an overall diagonal distance of about 69 cm.
- **Second Configuration:** The camera was translated an additional 10 cm to the right (reaching a lateral shift of 48 cm), while keeping the frontal distance of 57 cm unchanged, resulting in an overall diagonal distance of approximately 75 cm.

These two configurations allow the hand to be observed from two different angles, simulating the cabins of different cars.

The dataset was entirely recorded by a single subject (the candidate) due to time constraints and the prototyping scope of the project. To improve the robustness of the spatial feature representations, each sample was acquired by introducing systematic variations in terms of hand inclination, execution speed, and gesture style. While these execution variations increase intra-subject feature diversity, they do not substitute for genuine inter-subject anatomical and anthropometric variability (such as hand dimensions, finger proportions, and idiosyncratic motion patterns). Consequently, achieving true user-independent invariance remains a structural limitation of the current prototype and is identified as a key objective for future work.

The primary dataset consists of 2,160 samples, evenly split into 1,080 sequences for each of the two spatial camera configurations. In addition, a validation set of 216 samples was recorded in a dedicated session to monitor validation loss, guide early stopping, and select the optimal model checkpoint during training. Each sample represents an isolated sequence of exactly 30 consecutive frames (corresponding to 1.0 s of motion at a nominal rate of 30 FPS) containing the 63 spatial keypoint coordinates. Capturing discrete gesture instances rather than segmenting continuous video streams ensures well-defined spatio-temporal trajectory boundaries for model training. Conversely, during continuous real-time deployment, the model processes live camera streams using a sliding-window FIFO queue to dynamically extract overlapping 30-frame tensors.

Consistent with the operational dynamics of a real *HMI* system and the intrinsic characteristics of recurrent models, the dataset was intentionally designed to be unbalanced, dividing the sequences as follows:

- **Neutral Class** (720 total samples, 360 per configuration): As a background/negative class, it was intentionally oversized to represent the operating prior of a real-world HMI system. Over-representing the Neutral state trains the network to suppress kinetic noise (such as involuntary hand adjustments or steering maneuvers), effectively reducing false positive gesture triggers during continuous execution.

- **Directional/Complex Gestures** (`index_right`, `index_left`, `index_down`, `pinch` – 240 samples each, 120 per configuration): These classes involve extended continuous trajectories with higher spatio-temporal variability, necessitating a larger sample volume to map their feature distribution robustly.

- **Distinct/Rapid Gestures** (`fist`, `flip`, `shaka`, `thumb_up` – 120 samples each, 60 per configuration): These gestures feature sharp, highly distinctive geometric and kinematic signatures across time. Due to their high linear separability in the spatial landmark space, a more compact sample size is sufficient to achieve satisfactory convergence without biasing the loss function.

3.3.4 GestureLSTM Neural Network Architecture

The neural network used in this thesis work for the classification of gesture sequences is a bidirectional LSTM, named *GestureLSTM*. The model was implemented using the PyTorch framework and was engineered to be lightweight and highly performant, making it particularly suitable for execution in edge environments. The architecture comprises 2 stacked recurrent layers, each featuring a hidden size of 128 units. To prevent overfitting phenomena, primarily stemming from the limited amount of training data, a dropout layer set to 0.2 was introduced between the two Bi-LSTM layers. A second dropout is subsequently applied to the extracted temporal feature vector before forwarding it to the fully connected layer.

Owing to the bidirectional nature of the network, temporal dynamics are processed simultaneously in both forward and backward directions. The linear classifier receives a 256-dimensional feature vector (128×2) extracted from the final time-step ($t = 30$, via `out[:, -1, :]`). In PyTorch's bidirectional output format, this tensor concatenates the final forward hidden state $h_{30}^{\rightarrow}$, which accumulates the temporal context across the entire sequence from $t = 1$ to $t = 30$, with the backward hidden state $h_{30}^{\leftarrow}$ evaluated at the terminal frame $t = 30$, providing a comprehensive spatio-temporal feature representation for gesture classification.

None of the structural hyperparameters were defined arbitrarily; rather, they represent the outcome of a systematic *Group K-Fold Cross Validation* process. This approach allowed for the testing of various parameter combinations until identifying the optimal configuration in terms of robustness and classification accuracy.

In the `GroupKFold` ($K = 2$) cross-validation scheme, the grouping variable represents the two spatial camera recording configurations (*Camera Position 1* versus *Camera Position 2*). Grouping by camera setup ensures that all samples acquired under a specific viewpoint belong exclusively to either the training fold or the validation fold, rigorously testing the model's invariance across spatial camera perspectives while preventing cross-perspective data leakage.

3.3.5 State Machine and Activation Logic

To enable real-time gesture classification inside the cabin and prevent the model from generating involuntary predictions or repeated activations at every single frame, the system integrates a *Finite State Machine* (FSM) that is divided into three main states and operates according to the following logic (Figure 3.6):

1. **Inactive State:** Upon startup, the system is in a safe inactive state where the Bi-LSTM network continues to receive the hand landmarks extracted by MediaPipe, but any generated prediction is systematically ignored, with the exception of the "shaka" gesture. In this way, the driver can freely move their hands, gesture during a conversation, or interact with passengers and the physical controls of the vehicle without the risk of accidentally activating infotainment functions. The transition from the inactive to the active state requires explicit intention from the driver.

   To toggle between the Inactive and Active states, the system requires holding the `shaka` gesture continuously for 1.0 second. Operating at 30 FPS with a frame-by-frame sliding window, this corresponds strictly to 30 consecutive valid predictions assigned to the `shaka` class with a Softmax confidence $p \geq 0.85$. If the confidence drops below 0.85 or the predicted class changes at any frame during this window, the internal timer resets instantly, requiring a new unbroken sequence to activate the transition.

   Once the threshold is crossed, the system transitions to the active state, signaling its readiness to receive commands.To deactivate the gestural interface and return the system to the inactive state, the user is required to explicitly hold the same "shaka" gesture continuously for 1.0 second ($p \geq 0.85$), mirroring the exact temporal mechanism and robustness constraints used for activation.

2. **Active State:** In the active state, the network is fully operational and ready to interpret HMI commands. The temporal buffer of the sliding window begins to continuously fill with hand landmarks. To correctly execute any of the 7 gestures provided in the vocabulary (9 if considering the neutral and shaka classes), the user initiates the movement starting from an open-hand position, thus ensuring a clear temporal dynamic that is easily recognizable by the recurrent model. To avoid false positives, a command is dispatched only if the Softmax confidence satisfies an operational threshold $\tau \geq 0.85$. While Softmax outputs are not strictly calibrated probabilities, $\tau = 0.85$ serves as a conservative decision boundary prioritizing precision over recall for vehicle safety. Crucially, the FSM distinguishes between two non-activation cases:

   - **Explicit Neutral**: The model confidently classifies the background state ($p(\texttt{neutral}) \geq 0.85$).

- **Sub-Threshold Prediction**: A gesture class is predicted with $p < 0.85$. The system treats this as temporary uncertainty and silently ignores it, rather than mapping it to an explicit neutral prediction.

3. **Cooldown State:** Immediately after the recognition and activation of any gesture, the system enters a cooldown state. This phase can be divided into two consecutive and temporally distinct cooldowns:

   - **Active Suppression Phase:** It has a fixed duration of 0.5 seconds, during which the system completely ignores any active classification from the network. This provides the physiological time necessary for the driver to return their hand to a neutral position, preventing transitional return movements from triggering new commands. At the onset of this phase, the temporal FIFO queue is completely flushed to remove residual keypoints of the executed gesture, preventing double-triggering.
   - **Queuing and Readiness Phase:** After the initial 0.5 second suppression, the backend FSM immediately transitions back to the Active state, resuming frame acquisition and accumulating them in the temporal buffer to pre-load the sliding window. Concurrently, the HMI Client imposes an additional 0.2 second visual cooldown (shadow cooldown) on the infotainment interface. This masks the buffer queuing time from the user, preventing immediate UI re-triggering while significantly reducing perceived latency for the next gesture.

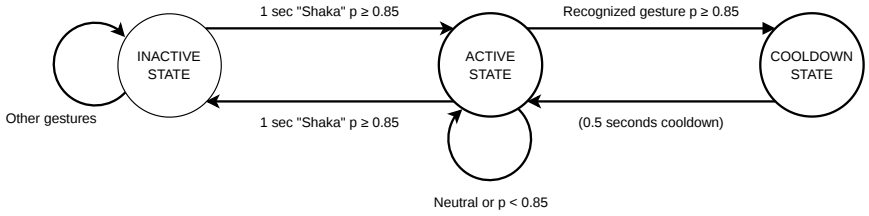


Figure 3.6: State transition diagram of the Finite State Machine governing the gesture-based HMI.

3.4 Engineering of the Driver Monitoring Module

For the development of the Driver Monitoring System module, the MediaPipe Face Landmarker pipeline was used. At runtime, the model performs three-dimensional tracking of a high-fidelity facial mesh composed of 478 3D landmarks. In parallel, the

framework calculates a vector of 52 facial blendshape coefficients. These coefficients represent normalized real-time scores in the $[0, 1]$ interval, each describing the contraction intensity of specific facial muscle groups.

In classic DMS implementations found in the literature, monitoring the driver's state is based on the manual and deterministic calculation of EAR and MAR, as analyzed in Section 2.2.1. In this work, we chose to adopt an approach based directly on Blendshapes. Having already integrated the MediaPipe pipeline for hand and face tracking, utilizing its native blendshapes allowed us to leverage an efficient, lightweight system already optimized for runtime execution.

For the development of the system, three specific blendshapes were selected:

1. **EyeBlinkRight:** Expresses the degree of closure of the right eyelid.
2. **EyeSquintRight:** Expresses the degree of squinting or eyelid contraction of the right eye.
3. **JawOpen:** Quantifies the vertical drop of the jaw.

The exclusive selection of the right eye blendshapes (`eyeBlinkRight` and `eyeSquintRight`) is directly dictated by the vehicle's sensor installation geometry. Because the acquisition camera is mounted off-center to the right above the central rearview mirror, the driver's right eye resides continuously within the primary line of sight. Conversely, the left eye suffers from frequent self-occlusions caused by the bridge of the nose and severe perspective foreshortening.

While focusing on the right eye maximizes tracking continuity under normal driving postures, it introduces a known system fragility: local occlusions affecting the right side of the face (such as hand movements near the right temple) can temporarily degrade drowsiness detection. This operational trade-off is addressed at the system level by coupling ocular heuristics with macroscopic Head Pose Estimation and a face-absence tracking timeout, while multi-eye adaptive fusion is delegated to future hardware iterations.

3.4.1 Heuristics for Sleep and Fatigue Detection

Fatigue monitoring operates on two distinct temporal scales, combining the identification of acute prolonged eye closure with the estimation of progressive fatigue over extended driving intervals.

Unlike absolute geometric ratios (such as millimeter-based EAR or MAR), MediaPipe blendshape coefficients produce continuous scalar activation scores in the range $[0, 1]$. Because these output scores are sensitive to camera sensor characteristics, facial geometry, and off-axis camera positioning (mounted near the central rearview mirror), the activation thresholds ($\tau_{\text{blink}} = 0.7$, $\tau_{\text{jaw}} = 0.4$, $\tau_{\text{squint}} =$

0.3) and temporal windows were determined empirically during prototype testing. These parameters serve as prototype-specific operational heuristics calibrated to minimize false alarms induced by speech, singing, or natural facial expressions, rather than standardized physiological constants.

- **Prolonged Eye Closure Detection**: The `eyeBlinkRight` coefficient is constantly analyzed at runtime. If the value exceeds the activation threshold of 0.7 for an uninterrupted time interval greater than 1.5 seconds, the system identifies a prolonged eye closure event, immediately triggering the visual and auditory safety alarm.
- **Yawn Detection**: Mouth opening alone is not a sufficient indicator to detect a yawn, as frequent false positives induced by speaking or singing would occur. For this reason, a yawn is validated only if the simultaneous contraction of two blendshapes occurs: `jawOpen` exceeding the 0.4 threshold and `eyeSquintRight` exceeding the 0.3 threshold. These two thresholds must be simultaneously exceeded for an uninterrupted duration of at least 1.0 second to register a single yawn event.
- **Cumulative Fatigue Estimation**: To evaluate the progression of the driver's fatigue over the long term, the system implements a 15-minute (900 seconds) sliding time window. If the driver yawns 5 or more times within this interval, a visual and auditory warning of progressive fatigue is generated, prompting the driver to take a break.

3.4.2 Heuristics for Facial Distraction and Head Pose Detection

To map visual distraction at runtime, the DMS continuously estimates the spatial orientation of the driver's face by extracting the three-dimensional facial rotation matrix directly provided by MediaPipe Face Landmarker. From this matrix, the Yaw (lateral rotation) and Pitch (vertical inclination) angles are calculated relative to a calibration baseline stored at system startup.

Geometric and Temporal Thresholds for Distraction Detection

Visual distraction is identified by analyzing the angular head pose deviations against empirical operational thresholds:

- **Lateral Distraction**: Yaw angle exceeding 35° (to the right or to the left).
- **Vertical Distraction**: Pitch angle exceeding 17° upwards or falling below −17° downwards.

To prevent the generation of false alarms, these thresholds must be continuously exceeded for an interval greater than 2.0 seconds. This tolerance window is calibrated based on empirical naturalistic driving literature (e.g., Klauer et al. and NHTSA guidelines), which identifies off-road glances exceeding 2.0 seconds an operational threshold for prolonged glances. In operational driving, brief visual checks of up to 2.0 seconds directed toward the rearview mirrors, instrument cluster, or central console are essential for maintaining situational awareness; requiring a continuous 2.0 second deviation ensures that legitimate scanning behavior is not penalized, while ISO 15007 provides the standardized methodological framework for measuring glance durations.

Handling Face Absence

A fundamental methodological distinction is maintained between an actively measured visual distraction and a temporary loss of facial tracking (*Face Absence*). When the facial mesh cannot be localized for an uninterrupted duration exceeding 2.0 seconds (e.g., due to hand-to-face occlusions, extreme rotations beyond camera field-of-view, or physical sensor obstruction), the driver's visual attention state becomes unobservable and the system signals the driver with a *Face Tracking Lost* alert.

Motivation and Limitations of Head Pose as a Visual Attention Proxy

Evaluating visual distraction through Head Pose Estimation (HPE) rather than direct 3D gaze tracking represents a deliberate architectural trade-off optimized for resource-constrained edge execution and dictated by sensor installation geometry.

From a physiological perspective, head orientation does not equal true ocular gaze (*Line of Sight*), as drivers frequently exhibit decoupled head-eye coordination. Direct 3D gaze regression models (such as L2CS-Net) offer higher fine-grained accuracy but impose heavy memory and computational overheads. Furthermore, fine-grained pupil tracking degrades significantly under off-axis camera geometry (mounted above the central rearview mirror to the right) due to severe perspective distortion, pupil self-occlusions, and obstruction from sunglasses. In our unified ecosystem, extracting 3D head rotation angles (Yaw and Pitch) directly from the existing MediaPipe Face Mesh avoids deploying a secondary deep neural network, preserving the strict 30 FPS processing budget on the edge device. Therefore, Head Pose Estimation is utilized as a macroscopic directional proxy for visual attention rather than a full gaze tracker.

Contrary to ideal assumptions, Head Pose Estimation is not immune to facial occlusions. Severe head rotations, hand-to-face interactions, or opaque eyewear can degrade landmark fitting and cause temporary tracking loss. To mitigate this structural vulnerability without adding computational weight, the system couples

head pose heuristics with a dedicated absence timeout (Section 3.4.2): if facial mesh tracking is lost continuously for more than 2.0 seconds due to severe occlusion or extreme rotation, the system automatically triggers a visual distraction alert, treating tracking opacity as an operational risk state.

3.4.3 Heuristics for Cognitive Distraction and Smartphone Detection

Within the Full-Featured Workstation Profile utilized during the experimental user evaluation connected to the AAOS simulator, the DMS subsystem incorporates a real-time Object Detection module based on YOLO26s ($\tau_{\text{yolo}} = 0.70$) targeting the COCO "cell phone" category (Class ID 67). Operating concurrently with MediaPipe and the Bi-LSTM network at 30 FPS on the host workstation, this module identifies physical secondary device handling and call-compatible postures. On the embedded Jetson Orin NX platform, this object detection pipeline is evaluated as an edge-feasibility benchmark to analyze power and thermal trade-offs.

To analyze the nature and severity of the ongoing secondary task interaction, the algorithm does not limit itself to detecting the mere physical presence of the smartphone, but rather distinguishes the event into two distinct categories depending on the posture assumed by the driver:

1. **Phone Detected:** This occurs when the smartphone is simply framed within the driver's area of interest (for example, at the height of the steering wheel, the instrument cluster, or the gear shift). This condition in itself represents an active safety violation, representing a potential risk configuration. While detecting a smartphone in the driver's active cabin area indicates potential manual or visual distraction (e.g., potential hand removal from the steering wheel), its mere physical presence does not in itself constitute definitive proof of active interaction or continuous engagement with the device.

2. **On a Call (Call-Compatible Posture)**: This is triggered when the driver raises the device toward their head. Rather than directly measuring internal cognitive workload, the system identifies a posture compatible with an active phone call. To mathematically identify this specific posture without requiring computationally expensive 3D pose calculations, the system implements a geometric proximity heuristic based on data fusion between the two-dimensional bounding box provided by YOLO and the facial landmarks extracted in real time by MediaPipe Face Landmarker.

To determine whether the user is holding the phone near the ear region, lateral facial boundary landmarks adjacent to the tragus, specifically landmark 234 (right side) and landmark 454 (left side), were identified via MediaPipe Face Landmarker.

At each frame, the system calculates the two-dimensional Euclidean distance (expressed in pixels on the image plane) between the centroid of the smartphone's bounding box detected by YOLO and the spatial coordinates of the auricular landmarks. If this distance is below the limit threshold of 150 pixels, the event is validated as a posture compatible with an active call state.

In order for the distraction to be correctly flagged, the smartphone must be detected for 1.0 continuous second. Furthermore, a 0.5 second tolerance has been added to maintain the state in case of occlusions or dropped frames.

Analysis of 2D Planar Projection Limitations

The use of a monocular sensor and two-dimensional bounding boxes entails an intrinsic limitation related to the loss of information regarding the third dimension (depth). Being a planar perspective projection, it is geometrically complex to distinguish with absolute certainty whether the smartphone is actually held against the driver's ear or if it is positioned in front of their face along the camera's line of sight.

In the context of Human-Machine Interaction and active automotive safety, however, this perspective ambiguity does not invalidate the heuristic. Furthermore, landmarks 234 and 454 serve as lateral facial contour references near the tragus rather than exact 3D ear canal coordinates. If the smartphone is aligned in projection with this lateral facial region or positioned in front of the driver's face, the configuration reflects a high-probability potential risk state for visual and manual inattention, associated with potential hand removal from the steering wheel. Consequently, the Euclidean distance calculated on the image plane represents an efficient, robust, and lightweight heuristic indicator, suitable for intercepting potential physical device handling and triggering the necessary auditory and visual safety warnings.

An additional methodological constraint of the current implementation lies in the fixed threshold of 150 pixels. This value is inherently dependent on camera resolution, lens focal length, and the driver's spatial distance from the sensor. Variabilities in driver stature, seating position, or facial dimensions on the image plane can alter the pixel-scale representation of physical proximity. To achieve a scale-invariant evaluation across diverse cockpit geometry setups, future system iterations should normalize the spatial Euclidean distance against facial dimension metrics, such as the Inter-Ocular Distance (IOD) or the width of the facial bounding box.

3.4.4 System Calibration and Environmental Robustness

Finally, the geometric thresholds, angular parameters, and temporal windows implemented in this module were empirically tuned during prototype integration to ensure stable operation under real-time testing conditions.

Unlike commercial Driver Monitoring Systems (DMS) integrated directly by automakers (OEMs), which rely on factory-calibrated hardware rigidities and exact millimeter-level optical geometry within a specific vehicle cockpit, this prototype operates in a flexible development setup. To account for potential camera placement variations relative to the driver's theoretical optical axis, the parameters were manually adjusted through qualitative functional testing in the laboratory environment.

Rather than asserting formal multi-environment generalization, these empirical thresholds represent functional operational parameters selected to mitigate false positives during baseline interactions. Systematic statistical validation and automated camera-recalibration protocols under diverse cabin geometries remain a key objective for future system deployments.

Chapter 4

Hardware Integration and Development on Android Automotive OS

4.1 Client-Server Communication via WebSocket Protocol

The core of the prototype's distributed architecture lies in the need to establish real-time communication between two distinct software and hardware ecosystems: the visual computing and artificial intelligence module running on the edge processing board (NVIDIA Jetson Orin NX), acting as the server, and the central infotainment system based on Android Automotive OS (AAOS) running on a Raspberry Pi, configured as the client. The two nodes are physically connected via a direct Ethernet (LAN) cable to establish a robust, low-latency wired communication channel with high resilience against in-cabin electromagnetic interference.

To define the logical protocol for data exchange over this physical channel, three different communication technologies were evaluated: traditional HTTP requests (*REST*), the *gRPC (Google Remote Procedure Call*) protocol, and the WebSocket protocol. The final selection was driven by an analysis of latency performance, packet overhead, and adherence to the *KISS (Keep It Simple, Stupid)* engineering principle:

- **HTTP (REST/Polling):** Traditional HTTP request-response patterns are fundamentally client-initiated. Implementing event-driven notifications (such as gesture triggers or drowsiness alerts) via polling would require frequent client requests (e.g., at 30 ms intervals matching the video frame rate), introducing unnecessary request/header overhead and processing load compared to

persistent push-based channels, despite HTTP connection keep-alive features.

- **gRPC (HTTP/2):** Although it is an extremely high-performance technology that supports native bidirectional streaming via HTTP/2 multiplexing and binary serialization (Protocol Buffers), gRPC introduces an excessive level of structural complexity (over-engineering) for a local point-to-point communication. The need to define, compile, and synchronize rigid serialization schemas in both the Python environment (Jetson side) and Kotlin (Android side) conflicts with the requirement for a lean and easily modifiable architecture.

- **WebSocket (Architectural Choice):** In accordance with the KISS engineering principle, the WebSocket protocol *(RFC 6455)* was selected for this prototype. It establishes a persistent, bidirectional, full-duplex connection over a single TCP socket, initialized via a single HTTP handshake. Once established, data frames carry minimal framing overhead, providing a lightweight, low-overhead event transport suited for real-time local IPC.

The WebSocket server was developed in Python on the NVIDIA Jetson Orin NX board, leveraging the asynchronous infrastructure provided by the asyncio and websockets libraries. This choice allows network I/O operations to be managed in a non-blocking manner, preventing socket management times from interfering with the threads dedicated to video frame acquisition and computer vision model inference, both of which must maintain a constant frame rate of 30 FPS. The server is configured on port `9543` and set to listen on the `0.0.0.0` address to accept requests originating from the Raspberry Pi within the private LAN.

On the other hand, the client architecture running on Android Automotive OS was implemented in Kotlin utilizing the *OkHttp* networking library. The OkHttp client establishes a persistent connection to the IP address of the Jetson server on the dedicated port `9543`. To ensure the system's reliability in a real-world automotive scenario, the client integrates an automatic reconnection mechanism managed asynchronously. Should the communication be suddenly interrupted (for example, due to a temporary physical anomaly in the cable), or if the two hardware systems boot up at different speeds when the dashboard is turned on, the client performs cyclic reconnection attempts in the background without freezing or slowing down the user interface experience.

Finally, to minimize parsing times and eliminate processing delays on the client side, the messages exchanged over the socket are structured as ultra-lightweight UTF-8 encoded text strings. These are transmitted asynchronously only when a validated state transition or an active DMS alarm occurs.

4.1.1 WebSocket Message Protocol and Connection Lifecycle

To satisfy the real-time requirements of in-cabin active safety while maintaining minimal processing overhead, the inter-process communication protocol between the Python inference server and the Android client was engineered according to lightweight, deterministic design principles. Rather than transmitting heavy structured envelopes (such as XML or nested JSON), the message payload and connection state are governed as follows:

1. **Message Format and Payload Structure**: Data frames are transmitted over the WebSocket channel as unformatted, plain-text UTF-8 strings. The messaging vocabulary is strictly closed and categorized into:

   - *DMS Alert Indicators*: Discrete state triggers such as `"SLEEP_ALERT"`, `"TALKING_PHONE"`, `"PHONE_SEEN"`, and `"ALERT_FACE_LOST"`.
   - *HMI Touchless Navigation Commands*: Directional and action triggers such as `"up"`, `"down"`, `"rot_r"`, `"rot_l"`, `"click"`, `"back"`, and `"home"`.

   This zero-overhead string paradigm allows the Kotlin Accessibility Service to perform immediate string comparison without incurring JSON parsing latency.

2. **Timestamp Management**: To minimize network bandwidth and serialization delay, timestamp data is not transmitted within the WebSocket payload. Instead, temporal alignment is managed locally on the Python server using high-resolution monotonically increasing clocks (`current_time`), which govern frame rates, sliding windows, and event buffers.

3. **Heartbeat and Keep-Alive Mechanism**: To ensure link integrity over the direct Ethernet connection and prevent socket timeouts, the keep-alive cycle is driven by the client. The Android OkHttp client configures an automated protocol-level ping interval via `pingInterval(15, TimeUnit.SECONDS)`. This guarantees continuous bidirectional connection validation without generating application-layer traffic.

4. **Timeout Handling and Automated Reconnection**: Network failures or unannounced socket drops are intercepted by the client's asynchronous network thread via the `onFailure` callback. Upon detecting a connection loss, the client executes a non-blocking 3.0-second delay (`SystemClock.sleep(3000)`) within a dedicated background thread before automatically invoking `connectToPC()` to re-establish the socket.

5. **Duplicate Suppression and Event Throttling**: Message flooding is controlled server-side via temporal cooldown thresholds:

- *DMS Alerts*: Governed by `MIN_TIME_BETWEEN_ALERTS = 3.0` seconds, enforcing a mandatory minimum interval between consecutive safety warnings to prevent alert fatigue.
- *Gesture Commands*: Throttled using `COOLDOWN_TIME = 0.7` seconds to filter out kinetic overshoot and duplicate command registration during gesture execution.

6. **Alarm Lifecycle and Visual Cessation**: Rather than relying on an explicit "clear" message over the network, the lifecycle of visual safety warnings is managed locally on the infotainment display. Upon receiving an alert event, the Android service instantiates an `ObjectAnimator` UI routine with a fixed display duration (e.g., 2400 ms or 4500 ms). An animation end listener subsequently triggers `hideDrivingAlarm()`, automatically dismissing the warning overlay from the screen once the designated time window expires.

4.2 Development of the Native Infotainment Bridge

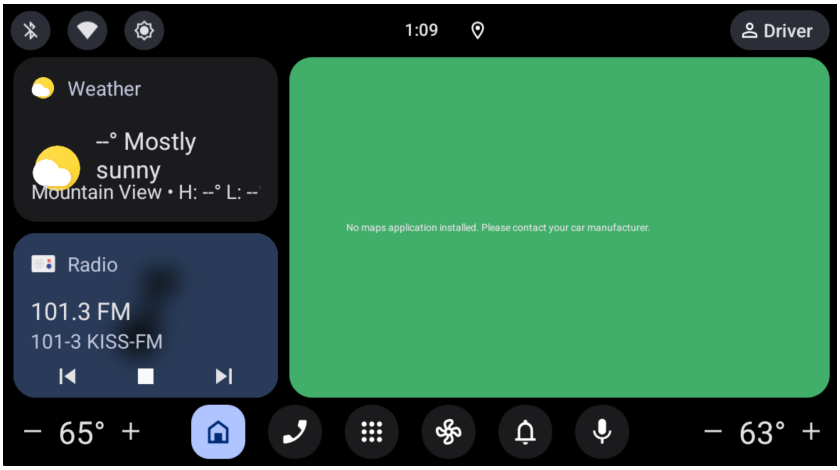


Figure 4.1: Android Automotive Operative System

4.2.1 Accessibility Service and Input-Event Injection

To implement the control architecture on Android Automotive in this experimental setup, an Android Accessibility Service (developed in Kotlin) combined with a privileged root shell was adopted as a prototypical *proof-of-concept* solution rather

than a platform-integrated production service. This prototypal mechanism makes it possible to emulate hardware rotary controls on stock Android Automotive OS without requiring custom ROM builds, offering persistent background execution and input injection capabilities.

The proposed interaction logic is based on the creation of a *Virtual Rotary Controller*, emulating the behavior of physical rotary dials typical of high-end vehicles. Navigation relies entirely on the native focus management system of Android Automotive OS. The driver can move the highlight between various interface elements and confirm the selection solely through discrete gestural impulses.

To minimize execution overhead within this prototypal setup, the service maintains a persistent root shell (`su`) to perform input-event injection through a privileged shell. While relying on `su` is a laboratory trade-off designed to bypass system permission constraints during prototype testing, security risks are mitigated by strictly validating incoming network payloads against the closed set of allowed commands defined in Section 4.1.1, immediately discarding any unlisted input.

Binding the Python WebSocket server to the universal interface (`0.0.0.0:9543`) without transport encryption or authentication, combined with root shell execution on the client, represents an explicit prototype security limitation acceptable exclusively within an isolated point-to-point physical Ethernet cable. For industrial mass production, this architecture must be hardened by enforcing *WebSocket Secure* (WSS) over mTLS and replacing root shell execution with native Android Automotive OS Accessibility APIs or platform-signed system services.

The mapping of the gestures approved by the control module and converted into navigation actions is structured as follows:

- **Rotor Rotation (Horizontal Navigation):** The *index_right* and *index_left* gestures emulate the horizontal rotation (clockwise and counterclockwise) of the physical rotor dial. They are translated into shell commands to move the system focus to the adjacent interactive element on the right or left.
- **Vertical Movement (Directional Arrows):** The *thumb_up* and *index_down* gestures simulate pressing the controller's directional arrows (upward and downward press), allowing the user to move between the vertical sections of the infotainment system.
- **Center Button Press (Click / Confirm):** The *pinch* gesture simulates the physical press of the rotor's central selection button.
- **Global Navigation Shortcuts:** To maximize speed of use, these commands are accompanied by two direct global shortcuts that are independent of the

focus position: the $fist$ gesture to simulate pressing the "Home" button and return to the main screen, and the $flip$ gesture to simulate the "Back" button and return to the previous menu.

- **System Deactivation and Interface Cleanup (Shaka Gesture):** When the driver deactivates the gestural control interface (holding the `shaka` gesture for 1.0 second), the system enters a dormant state. To ensure a clean UI free from visual distraction during normal driving, the service triggers a focus-reset action. In this laboratory prototype, this was implemented via `input tap 1 1`, targeting an inactive boundary coordinate $(1, 1)$ within the test layout to immediately clear the active focus highlight.

  While hardcoding spatial coordinates (`input tap 1 1`) serves as an effective, zero-latency workaround within the controlled layout of the experimental AAOS display, it represents a known prototype limitation susceptible to UI layout shifts or status bar element overlaps across different vehicle displays. In a production-grade deployment, this coordinate-based tap must be replaced by direct Android Accessibility API calls, specifically dispatching `AccessibilityNodeInfo.ACTION_CLEAR_FOCUS` across the active window node tree.

In order to ensure that interaction remains as safe as possible and to minimize visual demand, the infotainment system implements a quick fade graphic effect: an orange rectangular overlay drawn along the perimeter of the screen, which activates immediately upon the validation of a gesture. This temporary color transition serves strictly as a *peripheral visual acknowledgment (ACK)*, instantly confirming to the driver that the gestural event was successfully captured by the perception pipeline. By relying on peripheral vision, this non-intrusive response avoids forcing the driver to fixate on complex textual or graphical confirmations, in alignment with driver glance guidelines (ISO 15007).

Crucially, this visual ACK confirms frame reception without explicitly indicating which specific command class was recognized. While this design minimizes eyes-off-road time, it introduces a trade-off in the event of misclassification. To resolve this ambiguity in mass-production implementations, this peripheral visual ACK will be paired with short, class-distinct acoustic earcons or haptic cues, providing immediate semantic confirmation without increasing visual cognitive load.

4.2.2 Phantom UI Design for DMS Alerts

To ensure that DMS alerts immediately capture driver attention during impairment or distraction events, the client implements a dynamic graphic overlay based on the *Phantom UI* concept. The notification strategy combines a peripheral visual prompt, a central explanatory text, and an immediate acoustic chime:

- **Peripheral Visual Prompt**: A high-contrast flashing rectangle is rendered along the screen margins to stimulate retinal peripheral vision without requiring explicit foveal re-fixation on the display.
- **Central Textual Indicator**: A central text banner describes the detected safety hazard (e.g., drowsiness, phone usage, or prolonged gaze deviation).
- **Auditory Warning**: An immediate acoustic alert is played through the cabin speakers, completing the multimodal warning loop in alignment with Euro NCAP Safety Assist guidelines.

Displaying the text banner centrally on the infotainment screen represents an intentional experimental prototype trade-off, consolidating HMI navigation and DMS evaluation onto a single physical display. In dynamic driving scenarios, requiring a driver who is already visually distracted to glance at the central console to read an alert text would temporarily induce an additional off-road glance.

In a mass-production vehicle architecture, primary safety warnings are decoupled from the central infotainment display: visual alerts are dispatched directly to the *Digital Instrument Cluster* (behind the steering wheel) or projected onto a *Head-Up Display (HUD)*. This ensures that critical safety text resides strictly within the driver's primary forward vision zone, eliminating head rotation and maintaining alignment with Euro NCAP visual demand protocols.

As with the gestures, the alarm messages received from the server are translated into visual and auditory warning actions on the infotainment interface according to the following operational mapping:

- **Drowsiness:** Detected via prolonged eye closure or frequent yawns sent by the server. The infotainment response translates into a flashing blue rectangle along the entire perimeter of the screen, associated with an auditory signal and the appearance of a warning in the center of the display prompting the driver to take a break.
- **Visual Distraction (distraction):** Triggered when the driver's gaze is diverted from the road axis for more than two seconds. The system responds by showing a flashing red rectangle along the margins of the screen, combined with an auditory signal and the display of a message in the center of the screen immediately calling attention back to driving.
- **Mobile Device Use (phone / phone_call):** Active exclusively within the *Full-Featured Workstation Profile* where YOLO26s operates concurrently at 30 FPS (Chapter 3.1). Triggered when the YOLO model detects the smartphone being held or placed to the ear for more than one second in the active driving

zone. The infotainment reacts by drawing a flashing blue or red rectangle (depending on whether the phone is merely held or placed to the ear), an auditory warning, and a message in the center of the display requesting the driver to put the phone down and pay attention to the road.

4.2.3 DMS Alarm Logic: Duration, Rate-Limiting, and Prototype Limitations

To complement the visual mapping of the *Phantom UI*, the runtime execution of Driver Monitoring System alarms in the experimental prototype is governed by temporal rate-limiting and client-side display cycles:

1. **Display Duration**: Visual animation cycles are calibrated according to hazard severity using Android `ObjectAnimator` routines. Transient visual distraction warnings (`"ALERT_ON"`) persist on screen for 2400 ms, whereas critical safety alerts (`"SLEEP_ALERT"`, `"TALKING_PHONE"`) execute a longer 4500 ms display window before automatically invoking `hideDrivingAlarm()`.

2. **Server-Side Rate Limiting**: To prevent network flooding and visual alert fatigue, the Python server enforces an explicit temporal cooldown via `MIN_TIME_BETWEEN_ALERTS = 3.0` seconds. Once an alert event is dispatched over the WebSocket channel, the server suppresses subsequent alert transmissions for a minimum 3 second window, regardless of whether distraction continues across individual video frames.

3. **Sequential Event Handling in the Prototype**: In the current prototype implementation, incoming network alerts are processed sequentially on a First-In, First-Out (FIFO) basis as received by the WebSocket listener. The software stack does not currently incorporate an active message preemption queue or dynamic priority sorting algorithm; events are displayed in the exact order of network arrival.

4. **Production Roadmap for Multi-Hazard Precedence**: Processing alerts purely sequentially represents a known prototype limitation. In a mass-production automotive software architecture, simultaneous multi-hazard events must be governed by an explicit 3-tier safety precedence model:

$$\text{P1 Drowsiness} > \text{P2 Smartphone} > \text{P3 (Visual Inattention / Face Lost)} \tag{4.1}$$

Under this production architecture, higher-priority safety hazards (e.g., drowsiness) will actively preempt and override lower-priority active warning animations, ensuring the driver is immediately alerted to the most critical risk.

4.3 Deployment on Edge Hardware Architectures

The transition from desktop prototyping to embedded execution on the target NVIDIA Jetson Orin NX (8GB) board required a detailed investigation of hardware computational limits, power budgets, and thermal behavior. While the desktop environment equipped with an NVIDIA RTX 3070 Ti Laptop GPU and an AMD Ryzen 9 6900HX processor easily processed the full pipeline (MediaPipe, Bi-LSTM, and YOLO26s) at 30 FPS, direct execution of the identical stack on embedded hardware introduced an operational bottleneck.

4.3.1 Embedded Hardware Specifications and Benchmarking Setup

To ensure rigorous reproducibility, the target embedded platform was benchmarked under a fully monitored hardware and software environment:

- **Operating System and JetPack Platform**: *NVIDIA JetPack 5.1.2 (L4T 35.4.1).*
- **Accelerated Runtimes**: *CUDA 11.4* and *TensorRT 8.5.2* integrated natively within the JetPack environment.
- **Power Management Profile**: Configured to the maximum performance power mode (`NV Power: MAXN`, 25W ceiling).
- **Power Supply Unit**: External 19 V / 4.74 A DC power supply. Under the specific evaluated workload profile, system monitoring via `jtop` indicated an empirical operational draw (VDD_{IN}) fluctuating between 11 W and 15 W.
- **Thermal Dissipation and Active Cooling**: Active heatsink equipped with a PWM-controlled cooling fan. Operating dynamically between 29% and 33% duty cycle ($\sim$ 2150–2450 RPM), the active cooling maintained silicon junction temperatures strictly below 60°C throughout continuous stress testing.

4.3.2 Power Budget Analysis and Overcurrent Throttling Diagnosis

During preliminary benchmarking of the complete pipeline (concurrent GPU execution of MediaPipe landmarking, ONNX Bi-LSTM inference, and YOLO26s object detection), the Jetson Orin NX experienced hardware-level performance degradation, dropping throughput from 30 FPS to approximately 20 FPS and introducing noticeable execution latency. To properly characterize this behavior,

a clear technical distinction must be drawn between power supply and thermal limiting mechanisms:

- **Overcurrent Throttling (Observed Phenomenon)**: Triggered by the *Power Management IC* (PMIC) and *Over-Current Safety Circuitry* (OCSC) when instantaneous current draw spikes exceed the dynamic delivery limits during concurrent GPU/CPU workload bursts. Upon crossing this threshold, the Jetson Linux kernel emitted the explicit system warning:

  ```
  "Power supply overcurrent detected!
  System throttled to protect hardware."
  ```

  To protect internal power rails from brownout, the system power manager dynamically capped CPU and GPU clock frequencies, reducing the execution rate to $\sim$ 20 FPS. This required test subjects to execute hand gestures in slow motion to avoid frame drops, compromising natural HMI interaction.

- **Thermal Throttling**: A safety mechanism activated when silicon junction temperatures exceed absolute thermal thresholds (> 85°C). Under our active PWM fan cooling setup (temperature maintained below 60°C), thermal limits were never breached.

- **Overvoltage**: An input voltage protection fault caused by supply voltage instability exceeding absolute electrical ratings, which was not present in our regulated test setup.

The behavioral observation that test subjects had to execute gestures in slow motion during 20 FPS execution highlights a structural limitation of fixed-frame sequence representation. In the current prototype, the Bi-LSTM buffer collects a fixed sequence length (N frames). When the processing pipeline drops from 30 FPS to 20 FPS, the temporal duration represented by the N-frame window expands from 1.0 s to 1.5 s, introducing a time-scale distortion relative to the training distribution.

In a production-grade implementation, this dependency on hardware throughput must be decoupled by transitioning from frame-count buffering to *time-based landmark resampling*. By assigning absolute timestamps (Δt) to incoming video frames and applying linear interpolation or time-aware sliding windows, the landmark sequence fed to the recurrent network remains temporally normalized at a fixed sampling interval (e.g., 30 Hz), irrespective of underlying camera or GPU frame rate fluctuations.

4.3.3 Pipeline Rationalization and Software Optimizations

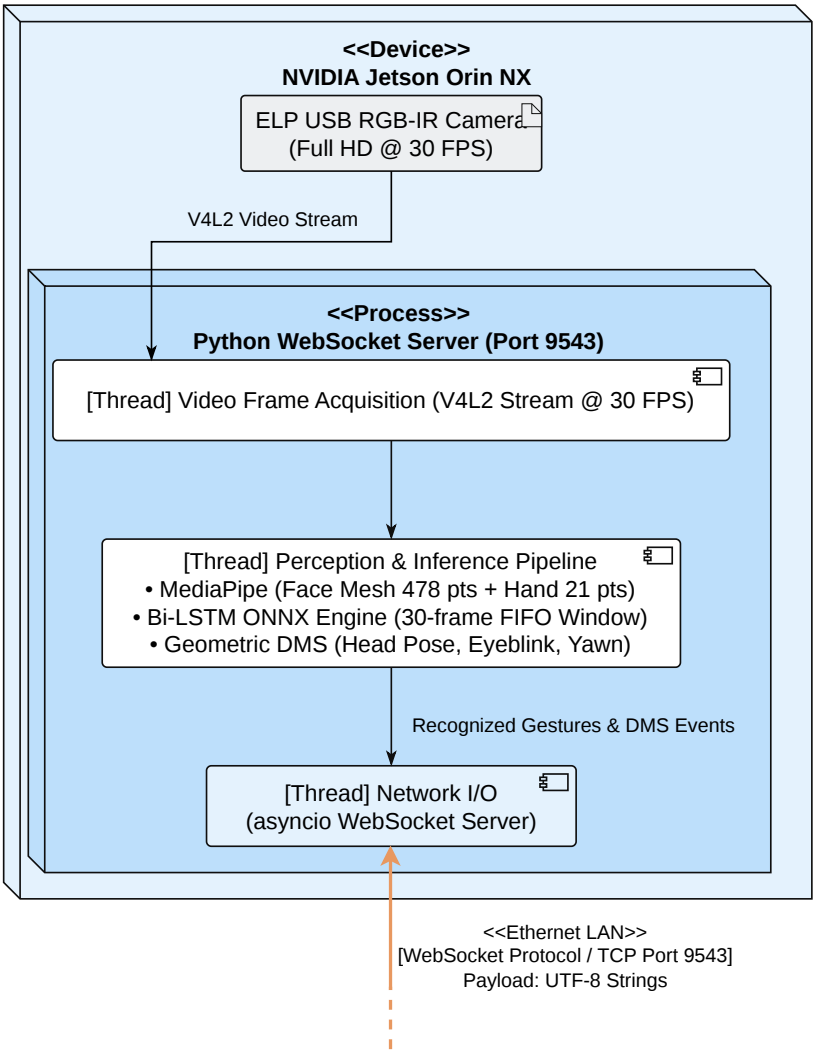


Figure 4.2: Detailed deployment architecture of the inference server node executing on the NVIDIA Jetson Orin NX platform. The diagram illustrates the V4L2 video frame acquisition thread (30 FPS), the multi-threaded perception and inference pipeline (incorporating MediaPipe 3D keypoint extraction, Bi-LSTM ONNX engine execution, and geometric DMS monitoring, with the optional YOLO module omitted), and the asynchronous WebSocket network server operating on TCP port 9543.

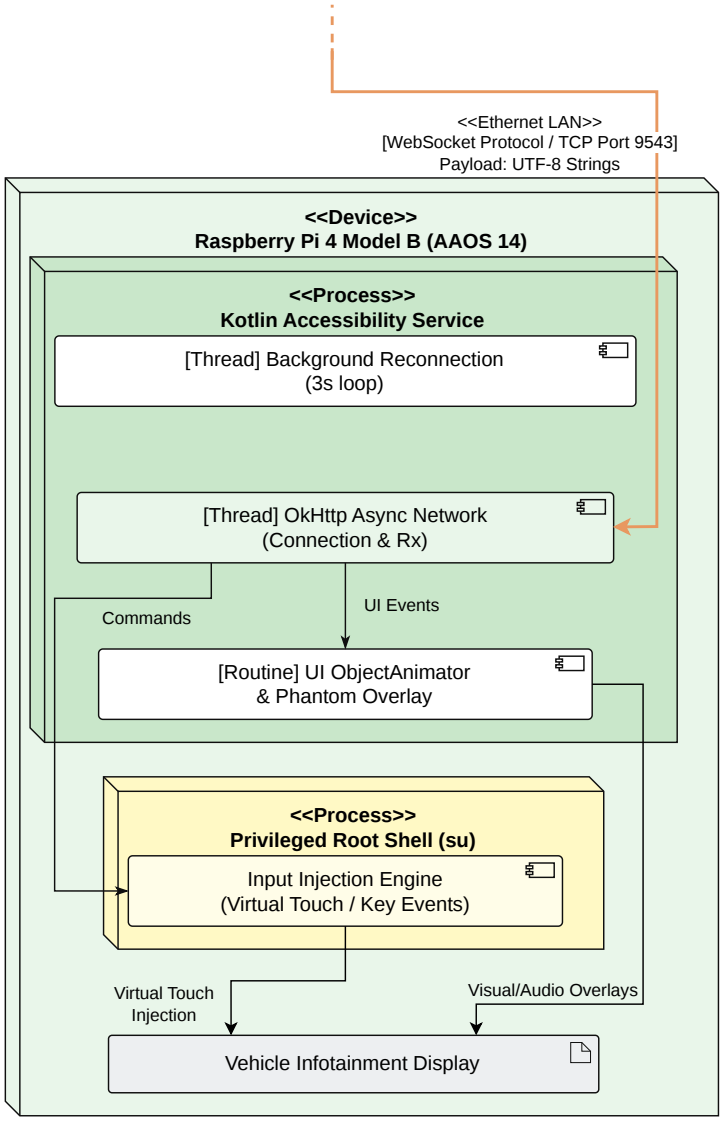


Figure 4.3: Deployment architecture of the infotainment client node on Raspberry Pi 4 (AAOS 14), showing the incoming Ethernet WebSocket channel (port 9543), the Kotlin Accessibility Service, and the privileged root-shell engine for touch input injection.

Because the 20 FPS bottleneck was caused specifically by *power supply over-current throttling* during YOLO26s execution spikes, the object detection module

was omitted in the rationalized edge profile. Removing YOLO eliminated current transients, immediately restoring a deterministic 30 FPS processing rate for primary safety tasks (geometric DMS and gesture-based HMI). To maximize throughput for the remaining vision pipeline on Jetson Linux, two major software optimizations were implemented:

1. **MediaPipe GPU Delegation**: Unlike Windows desktop environments where MediaPipe Python bindings default to CPU execution, the Linux-based JetPack platform enabled native GPU pipeline delegation. Transferring hand and face landmark extraction to the Jetson GPU drastically reduced CPU load while maintaining fast landmark localization.
2. **Bi-LSTM Model Optimization via ONNX Runtime**: The weights of the recurrent gesture classification network were exported from I (`.pth`) to the I (`.onnx`) format. Executing the model via the C++ backend of ONNX Runtime with CUDA execution providers fused mathematical operations, minimizing frame latency and reducing memory overhead.

4.4 System Interaction Scope, Limitations, and Global Action Roadmap

In evaluating the integration of the touchless gestural control architecture with Android Automotive OS (AAOS), a formal distinction must be drawn between the current prototype implementation scope and the long-term design vision for in-cabin Human-Machine Interaction.

4.4.1 Current Focus Navigation Scope and In-App Limitations

In the present prototype, touchless gestural commands are utilized to emulate a Virtual Rotary Controller, driving the native Android focus management framework to highlight and select elements across the system launcher and top-level menu interfaces. While this mechanism validates the feasibility of real-time gesture-to-input translation via a native Accessibility Service, it exhibits specific architectural and operational limits:

- **Restriction to System-Level Interfaces**: Cursor and focus navigation operates reliably across top-level launcher displays, system settings, and primary menu cards. However, it does not extend to fine-grained intra-application navigation (e.g., scrolling individual playlist items inside third-party media players like Spotify, or panning/zooming maps within navigation applications like Waze).

- **Dependence on Application Focus Trees**: Third-party Android applications frequently implement non-standard view hierarchies or custom touch listeners that do not expose predictable directional focus chains. Attempting to force cursor navigation inside arbitrary application layouts introduces focus trapping, input misrouting, and UI unpredictability.

- **Cognitive and Visual Distraction Risk**: Requiring the driver to perform multi-step spatial cursor positioning to navigate nested menus inside applications increases off-road glance durations, conflicting with automotive safety guidelines (ISO 15007 and Euro NCAP) that aim to minimize secondary task visual demand.

4.4.2 Future HMI Roadmap: Decoupling Cursors and Mapping Global Actions

To overcome these structural limitations and align the system with automotive ergonomic principles, future iterations of this architecture will transition from localized cursor navigation toward an application-agnostic global action mapping paradigm.

Rather than manipulating visual cursors within individual app interfaces, mid-air dynamic gestures will be bound directly to global system intents, hardware abstraction APIs, and media framework event broadcasts, operating deterministically regardless of the foreground application currently displayed:

1. **Media and Audio Control**: Dynamic gestures will trigger direct Android `MediaSession` events, enabling instant track skipping (`index_right` / `index_left`), global audio muting (`fist`), and continuous volume adjustments (`pinch` or rotary gestures) without requiring the driver to look at or interact with the media app UI.

2. **Climate Control (HVAC Integration)**: Gestures will be interfaced with the *Android Vehicle Hardware Abstraction Layer* (VHAL) to control cabin environmental parameters directly. For example, vertical gesture dynamics or pinch adjustments can directly increase or decrease air conditioning temperature and fan speed across the cabin.

3. **Telephony and System Shortcuts**: High-priority driving interactions will be mapped to explicit semantic gestures, such as accepting or rejecting incoming phone calls (`thumb_up`) or invoking the voice assistant (`shaka`) to handle complex queries through natural language speech.

By decoupling gestural input from visual screen coordinates and binding it directly to global system actions, the proposed HMI architecture achieves complete application independence, eliminates third-party UI incompatibilities, and significantly reduces driver visual distraction during dynamic driving.

Chapter 5

Experimental Validation Protocol

5.1 Research Objectives and Hypotheses

The empirical user study was designed to evaluate human-machine interaction performance, visual distraction, and subjective workload when comparing a mid-air gesture interface against a baseline capacitive touchscreen. The evaluation was conducted under a *simulated static driving posture*, wherein participants maintained visual focus on a designated forward target point.

The Driver Monitoring System module functions within the single-camera architecture as a real-time proof-of-concept component; therefore, this study specifically benchmarks the HMI interaction modalities rather than performing a formal quantitative validation of the DMS detection pipeline.

The research hypotheses (H) formulated for this study are as follows:

H1a – **Interface Learnability (Training Time):** It is hypothesized that the gesture-based interface exhibits rapid learnability, quantified by an average total training duration (T_{learn}) of less than 6 minutes per participant to reach operational proficiency. This time window explicitly encompasses the interactive briefing and gesture demonstration during which the candidate simultaneously practiced the executions under the experimenter's guidance.

H1b – **Interface Learnability (Training Error Frequency):** It is hypothesized that participants achieve operational proficiency during the guided training phase with an average total of fewer than 10 gesture errors across the entire training session before satisfying the passing criterion.

H2a – **Spatial Navigation Temporal Efficiency:** It is hypothesized that touchscreen interaction demonstrates superior temporal execution performance compared to mid-air gesture interaction during spatial micro-navigation tasks (Tasks 3 and 4), yielding significantly lower Task Completion Time (TCT) as evaluated via paired parametric or non-parametric statistical testing.

H2b – **Spatial Navigation Precision and Error Rate:** It is hypothesized that touchscreen interaction achieves significantly higher spatial pointing precision during micro-navigation tasks (Tasks 3 and 4), resulting in a lower selection error rate across experimental trials compared to 3D gesture interaction, which is inherently subject to spatial jitter.

H3 – **Pipeline Robustness and Overall Error Rate:** It is hypothesized that following the initial guided training phase, the overall operational gesture error rate across post-training task trials remains below 10%, confirming the real-time reliability and robustness of the vision-based pipeline.

H4a – **Visual Distraction in Global Secondary Commands (Tasks 1 and 2):** It is hypothesized that gesture interaction significantly reduces users visual distraction compared to touchscreen interaction when executing global secondary shortcuts (Tasks 1 and 2: Home, Back), yielding lower Total Off-Road Glance Time (TOT) by enabling low-demand or open-loop execution.

H4b – **Visual Distraction in Progressive Spatial Navigation Tasks (Tasks 3 and 4):** Conversely, it is hypothesized that touchscreen interaction results in significantly lower visual distraction compared to gesture control during progressive spatial navigation tasks (Tasks 3 and 4: App Selection and Climate Adjustment), as step-by-step virtual cursor positioning in 3D space necessitates continuous visual verification at each discrete step.

H5 – **Task-Dependent Perceived Difficulty (SEQ):** It is hypothesized that immediate perceived task difficulty (measured via the Single Ease Question, SEQ, 1–7, administered immediately post-task) varies significantly by interaction modality and task type: gesture interaction yields lower perceived difficulty for rapid unidirectional commands (Tasks 1 and 2), whereas touchscreen interaction yields lower perceived difficulty for progressive spatial micro-navigation (Tasks 3 and 4).

H6 – **Subscale-Specific Subjective Workload (NASA-TLX):** It is hypothesized that the subjective workload profile evaluated via NASA-TLX differs significantly between modalities: touchscreen interaction is hypothesized to induce higher cognitive demand due to the continuous visual and spatial attention required, whereas mid-air gesture interaction is hypothesized to

induce higher physical demand and effort due to sustained dynamic hand and arm posture.

Prior to hypothesis testing, the distributional assumption of normality for continuous metric differences was formally assessed using the Shapiro-Wilk test ($\alpha = 0.05$) [36]. To rigorously account for the within-subjects repeated-measures design (2 Modalities $\times$ 4 Tasks $\times$ 3 Repetitions) and the nested observations within participants ($N = 15$), a *Three-Way Repeated-Measures ANOVA (RM-ANOVA)* was conducted as the primary inferential test when no significant departure from normality was observed ($p > 0.05$). If normality was significantly violated ($p \leq 0.05$), non-parametric factorial alternatives, such as the *Aligned Rank Transform (ART) ANOVA*, were employed.

Table 5.1: Summary of Research Hypotheses, Variables, Metrics, and Verification Methods.

Hyp.	**Independent Var. (IV)**	**Dependent Var. (DV)**	**Operational Metric**	**Verification Test**
H1a	Training Phase	Learning Speed	Total Training Time (s)	One-sample t-test (< 360s)
H1b	Training Phase	Learning Accuracy	Total Training Errors	One-sample t-test (< 10.0)
H2a	Modality (`Touch` vs. `Gest.`)	Temporal Efficiency	TCT (Tasks 3–4)	3-Way ANOVA + Paired t-test / Wilcoxon
H2b	Modality (`Touch` vs. `Gest.`)	Spatial Precision	Error Count (Tasks 3–4)	3-Way ANOVA + Paired t-test / Wilcoxon
H3	Real-time Pipeline Execution	Pipeline Robustness	Overall Error Rate (%)	One-sample t-test ($< 10\%$)
H4a	Modality (`Touch` vs. `Gest.`)	Visual Distraction	TOT (Tasks 1–2)	3-Way ANOVA + Paired t-test / Wilcoxon
H4b	Modality (`Touch` vs. `Gest.`)	Visual Distraction	TOT (Tasks 3–4)	3-Way ANOVA + Paired t-test / Wilcoxon
H5	Modality $\times$ Task Type	Perceived Task Difficulty	SEQ Score (1–7, post-task)	Two-way RM-ANOVA
H6	Modality (`Touch` vs. `Gest.`)	Cognitive vs. Physical Effort	NASA-TLX Subscales	Paired t-test

5.2 Experimental Design and Participant Selection

Participants and Demographics

Due to resource constraints and the preliminary nature of the system evaluation, the present study is framed as an exploratory user evaluation. A sample of $N = 15$ participants (9 males, 6 females) was recruited via a *convenience sampling* strategy based on uncompensated voluntary participation. The participants' ages ranged from 22 to 51 years, with the inclusion of one minor participant aged 15.

Notably, all $N = 15$ participants held a valid driving license with active road driving experience: 14 participants possessed a standard B-class license for passenger cars, while 1 minor participant (aged 15) held an AM-class license for light quadricycles (microcars), providing an exploratory perspective on young holders of micro-mobility licenses. Regarding handedness, 12 participants were right-handed and 3 were left-handed.

The reduced sample size ($N = 15$) represents a primary methodological limitation of this exploratory evaluation, restricting the overall statistical power and generalizability of the findings.

Experimental Design and Latin Square Counterbalancing

To evaluate both interaction modalities (`Touchscreen` vs. `Gestures`) while controlling for carry-over and learning biases, a two-tier counterbalancing design was implemented.

First, task execution order across the four experimental tasks (T_1: Home, T_2: Back, T_3: App Selection, T_4: Climate Control) was governed by a balanced 4×4 *Williams Latin Square* matrix L:

$$L = \begin{bmatrix} T_1 & T_2 & T_4 & T_3 \\ T_2 & T_3 & T_1 & T_4 \\ T_3 & T_4 & T_2 & T_1 \\ T_4 & T_1 & T_3 & T_2 \end{bmatrix} \tag{5.1}$$

where each row $i \in \{1, 2, 3, 4\}$ corresponds to a unique task presentation sequence (Sequences A through D) assigned across participant subsets, and each column $j \in \{1, 2, 3, 4\}$ represents the sequential ordinal position of execution within a session (from 1st to 4th position).

This balanced configuration ensures that each task appears exactly once in every ordinal position and is immediately preceded and followed by every other task an equal number of times across the experimental design, designed to counterbalance first-order carry-over and learning effects.

Second, crossing 4 task sequences with 2 modality orders (`Touch`→`Gesture` vs. `Gesture`→`Touch`) yields 8 unique experimental combination pairs. Given the odd sample size ($N = 15$), an exact 50/50 split was not possible. Participants were pseudo-randomly assigned in a 7/8 allocation: $n = 7$ participants executed the `Touchscreen` modality first, whereas $n = 8$ participants started with the `Gesture` modality. To prevent systemic coupling between specific task sequences and initial modalities across the incomplete second block ($8 + 7 = 15$), the modality presentation order for the 5th sequence iteration was explicitly inverted.

No mandatory inter-condition resting period was enforced between the two modality sessions. Consequently, potential cognitive fatigue or cumulative visual strain across consecutive trials must be acknowledged as a potential factor influencing execution times and subjective workload ratings.

5.2.1 Ethics, Informed Consent, and Data Privacy

The experimental protocol strictly adhered to standard academic research ethics and *General Data Protection Regulation* (GDPR, EU 2016/679) guidelines:

- **Informed Consent**: Prior to testing, all participants received a detailed briefing outlining the experimental scope, procedures, and potential risks, and voluntarily signed an informed consent form.
- **Pseudonymization and Data Protection:** In accordance with GDPR (EU 2016/679) guidelines, participants were assigned unique alphanumeric identifiers (`User 1`–`User 15`). All quantitative behavioral metrics, questionnaire responses, and error logs were indexed exclusively via these pseudonyms
- **Video Data Management**: Video streams recorded for gaze estimation and gesture tracking were stored on password-protected local storage media. Data access was restricted strictly to the primary researcher for offline metric annotation, ensuring no public dissemination or third-party transfer.

5.3 Definition of Driving Tasks

To evaluate the system's performance, four secondary infotainment tasks were defined. Each task was repeated three times per modality. The three repetitions were executed consecutively, restoring the same initial state each time to ensure identical starting conditions.

To properly contextualize the operational differences between the interaction modalities, the tasks are categorized into two structural paradigms:

1. **Direct Shortcut Tasks (T_1: Home, T_2: Back)**: Evaluated as global system commands. Under the `Gesture` modality, execution relies on direct shortcut gestures triggering the command without targeting a specific graphical element. Under the `Touchscreen` baseline, execution requires visually locating the corresponding icon on the display followed by a physical tap. Thus, T_1 and T_2 benchmark a direct gesture shortcut against a visual search and tap workflow.

2. **Progressive Micro-Navigation Tasks (T_3: App Selection, T_4: Climate Control)**: Evaluated as multi-step spatial navigation within structured menus. Both modalities require spatial positioning: the `Touchscreen` allows direct selection of target UI controls, whereas the `Gesture` modality requires progressive spatial movement across interface elements.

While the comparison between modalities remains methodologically valid, it is explicitly declared that the evaluation benchmarks two complete interaction systems characterized by different action sequences and step counts, rather than isolating the physical input channel in strict equivalence.

Table 5.2 summarizes the objectives, conditions, and completion or error criteria for each task.

It is specified that for Task 3 (Sequential app opening), the three repetitions were presented in a fixed progressive sequence: *Bluetooth Audio* (positioned top center, pos. 3), *Contacts* (positioned center screen, pos. 8), and *Phone* (positioned bottom right, pos. 15). This fixed sequence was deliberately chosen as a cognitive scaffolding strategy to prevent user frustration during early gesture interaction.

Methodologically, this design introduces a potential order-by-difficulty confounding factor, as target distance is confounded with intra-task trial order. While the Touchscreen baseline remains structurally invariant (2 direct taps across all repetitions), the Gesture modality features an escalating minimum action count ($6 \rightarrow 11 \rightarrow 18$ actions). Consequently, intra-task learning effects may counterbalance the expected linear increase in completion time for longer gesture sequences, a factor explicitly addressed in the results interpretation.

5.3.1 Task Workflows, Initial UI Focus, and Command Sequences

To ensure experimental reproducibility and clarify the structural complexity of each secondary task, the operational path, initial UI focus state, and minimum required commands per modality are detailed below:

- **Task 1 (T_1: Return to Home)**:

Table 5.2: Definition of navigation tasks, completion criteria, and error conditions.

Task	**Objective**	**Initial State**	**Completion Criterion**	**Execution / Error (Touch)**	**Execution / Error (Gestures)**
1. Return to Home screen	Return to the main system screen.	App opened (Level 2).	The Home screen is fully loaded and visible.	Search and tap on Home. *Error:* Tap on wrong area.	Fist Gesture (immediate). *Error:* Different, unrecognized, or misclassified gesture.
2. Return to previous screen (Back)	Go back one navigation level.	App opened (Level 2).	The previous screen is loaded and visible.	Search and tap on Back. *Error:* Tap on wrong area.	Flip Gesture (immediate). *Error:* Different, unrecognized, or misclassified gesture.
3. Sequential app opening	Locate and launch a specified app.	System Home screen.	The specific application opens on screen.	Search and direct tap on the icon. *Error:* Wrong app selection or missed tap.	Virtual cursor + Pinch. *Error:* Gesture not useful for the task, wrong app, or pinch not registered.
4. AC temperature adjustment	Increase the climate by one step.	System Home screen.	The numeric temperature value changes on display.	Search and tap on +. *Error:* Modifying passenger climate or wrong tap.	Virtual cursor towards AC area + Pinch. *Error:* Gesture not useful for the task or pinch not registered.

  - *Initial UI Focus*: Active application screen (Level 2 depth).
  - *Screen Transition*: `Active App` → `Home Screen`.
  - *Minimum Commands*: **Touchscreen** = 1 action (1 direct tap on the Home icon); **Gestures** = 1 action (1 global `Fist` shortcut gesture).

- **Task 2 (T_2: Return to Previous Screen / Back)**:
  - *Initial UI Focus*: Nested sub-menu view (Level 2 depth).
  - *Screen Transition*: `Nested Sub-view` → `Parent Screen (Level 1)`.
  - *Minimum Commands*: **Touchscreen** = 1 action (1 direct tap on the Back icon); **Gestures** = 1 action (1 global `Flip` shortcut gesture).

- **Task 3 (T_3: App Selection)**:
  - *Initial UI Focus*: Home Screen.
  - *Screen Transition*: `Home Screen` → `Apps Grid` → `Target App View`.
  - *Minimum Commands*: **Touchscreen** = 2 actions (1st tap to open the Apps Drawer from the Home screen, 2nd tap directly on the target app icon); **Gestures** = 6 to 18 actions depending on target grid index (6 for pos.$\tilde{3}$, 11 for pos.$\tilde{8}$, and 18 for pos.$\tilde{15}$, comprising directional focus shifts and 2 `Pinch` confirmations).

- **Task 4 (T_4: Climate Control Adjustment)**:
  - *Initial UI Focus*: Home Screen.
  - *Screen Transition*: Remains on `Home Screen` (parameter value updated in bottom bar).
  - *Minimum Commands*: **Touchscreen** = 1 action (1 direct tap on the target climate button in the bottom bar); **Gestures** = 2 sequential focus movements across bottom bar elements to reach the climate widget, followed by selection/adjustment.

5.4 Instrumentation and Execution Procedure

5.4.1 Experimental Apparatus, Sampling Rates, and Data Annotation Protocol

To evaluate the proposed HMI architecture under simulated ergonomic conditions, a dedicated static vehicle experimental setup was established, integrating hardware apparatus, real-time inference pipelines, and offline video logging workflows.

- **Hardware and Display Apparatus**: An optical camera was securely mounted above the internal rear-view mirror, oriented to capture the participant's hand movements within the active interaction zone while maintaining an unobstructed view of the upper torso and head pose. To simulate an OEM In-Vehicle Infotainment (IVI) display, a convertible touchscreen laptop running an Android Automotive OS (AAOS) emulator was positioned directly over the vehicle's native dashboard display.

- **System Sampling Frequencies**: Video streams were captured at a temporal resolution of 30 FPS. The real-time gesture pipeline (MediaPipe keypoint extraction followed by Bi-LSTM sequence classification) operated frame-by-frame at an inference rate of ≈ 30 Hz.

- **Trial Synchronization and Offline Video Annotation**: Experimental task trials were initiated via standardized verbal cues issued by the experimenter. Task completion times (TCT), interaction events, and visual distraction metrics (TOT) were extracted offline through semi-manual video inspection. For each trial, the video recording was inspected, calculating interaction durations from the verbal start cue to final task completion. Concurrently, the dedicated gaze tracking script was triggered in temporal alignment with the verbal cue.

- **Task Completion Time (TCT) Annotation and Measurement Uncertainty:** Since task execution timing was anchored to the experimenter's verbal start cue ("Via"), all time durations were extracted offline through manual video frame analysis conducted by a single trained experimenter. The manual identification of the precise start frame (verbal cue) and completion frame introduces an inherent measurement uncertainty of $\pm$100–200 ms ($\approx$ 3–6 frames at 30 FPS), arising from human reaction latency and audio-visual frame alignment. This uncertainty factor is formally taken into account as an inherent measurement error boundary during metric evaluation.

5.4.2 Visual Distraction Tracking and Data Annotation Protocol

To objectively quantify visual distraction, a dedicated experimental measurement script was deployed during the testing sessions, complemented by semi-manual video logging and annotation. It is necessary to distinguish this measurement apparatus from the system's DMS:

- **Embedded DMS Operational Logic**: The production DMS architecture implements head pose-based attention mapping with predefined spatial and

angular tolerances (Yaw and Pitch thresholds), permitting the user to turn their head toward the in-vehicle infotainment (IVI) display for functional interaction without immediately triggering a distraction alarm.

- **Dedicated Visual Distraction Measurement Script**: To benchmark visual demand, a dedicated script based on the `MediaPipe Face Landmarker` monitored ocular deflection relative to the fixed forward horizon point. Rather than relying on coarse head pose, the script estimated continuous 2D gaze deflection by tracking the participant's pupil/iris center of the right eye, normalized against eye contour landmarks. At the start of each trial session, a baseline calibration was captured while the participant fixated on the forward horizon. During testing, the experimenter manually triggered the timer start and stop upon verbal task initiation and completion cues. Whenever gaze shifted from the calibrated baseline beyond a spatial shift threshold (> 0.10), the timer automatically accumulated off-road glance duration, applying a fixed operational filtering threshold of 120 ms (based on the definition of saccadic fly-throughs in ISO 15007) to exclude visual transition noise from sustained glances.

- **Video Logging and Verification**: All experimental trials were video-recorded. Offline semi-manual verification was performed on the recordings to validate event timestamps and ensure measurement precision.

5.4.3 Execution phases

The experiment was divided into the following phases:

- **Training Phase (Gestures only):** Initial phase that begins with a standardized oral briefing and interactive gesture demonstration conducted by the experimenter, during which the participant simultaneously performs guided trial executions of the gesture vocabulary. This joint execution integrates the introductory briefing into the physical practice process. Immediately following this guided familiarization, the participant engages in independent practice, receiving real-time visual feedback to freely repeat and memorize the commands. A maximum time limit of 6 minutes was set, explicitly encompassing the interactive briefing, explicitly encompassing the interactive briefing, safety instructions, and guided/independent practice. The phase ended successfully when the user correctly executed three consecutive gestures for each of the 7 classes. If the participant did not reach the criterion within 6 minutes, the experiment proceeded normally, but the user was flagged in the dataset to enable comparative sensitivity analysis (evaluating overall usability metrics both across the full cohort and excluding non-proficient subjects). Both the

overall duration (Time-to-learn) and the total number of unrecognized or incorrectly executed gestures before reaching the criterion were recorded.

- **Task Modality A:** The participant performed the 4 described tasks using the first assigned interface. The presentation order followed the Latin Square scheme. For each task, the 3 planned repetitions were performed consecutively. At the end of the repetitions for the same task, the SEQ questionnaire was administered. At the end of the entire modality, the NASA-TLX questionnaire was administered.

- **Task Modality B:** The participant repeated the procedure using the secondary interface. The SEQ was administered after each block of repetitions, and the NASA-TLX at the end of the modality.

- **Global Evaluation:** At the end of the entire experiment, the user completes the SUS questionnaire (administered exclusively to evaluate the usability of the gesture system). Finally, they were asked the following open-ended questions:

  1. Which of the two interaction modalities did you prefer overall? (Touchscreen / Gestures / No preference)
  2. For which specific actions do you believe the use of gestures is more comfortable and safe compared to the touchscreen?
  3. What was the major difficulty encountered when using the gesture system?

5.5 Objective and Subjective Evaluation Metrics

To ensure methodological transparency, evaluation metrics are explicitly categorized into *primary metrics* (defined *a priori* to test Hypotheses H1–H6) and *secondary exploratory metrics* (evaluated *post-hoc* to investigate specific interaction patterns).

5.5.1 Objective Performance and Visual Distraction Metrics

Objective behavioral and system performance metrics were extracted offline from frame-by-frame video inspections and interaction timestamps. In alignment with the experimental design, these metrics are classified into primary performance indicators (directly anchored to hypotheses H_1–H_4) and secondary exploratory metrics.

Primary Objective Metrics (A Priori Evaluation Plan)

- **Task Completion Time (TCT)**: The total temporal duration (in seconds) required to execute a secondary task trial (T_1–T_4). It was measured offline from the experimenter's verbal "Via" start cue to the final successful interface state transition. A human operator reaction latency tolerance window of ±100–200 ms (≈ 3–6 frames at 30 FPS) was systematically defined and accounted for during frame annotation.

- **Total Off-Road Glance Time (TOT)**: The primary measure of visual distraction, evaluated following the glance reduction methodology outlined in ISO 15007. It represents the cumulative duration (in seconds) of all off-road glances exceeding an operational filtering threshold of 120 ms away from the forward roadway during a task trial (testing hypotheses H_{4a} and H_{4b}).

- **Interaction Error Count per Task Trial**: Quantifies the absolute number of interaction errors committed during a single task execution to evaluate operational accuracy (testing hypotheses H_{2b} and H_3). Depending on the evaluated modality, this metric encompasses:

  - *Gesture Modality*: The total number of unrecognized gesture attempts (*missed detections*) and incorrectly recognized or executed gestures (*misclassifications* and *user operational errors*).
  - *Touchscreen Modality*: The total number of missed taps (touches registered outside active target hitboxes) and wrong taps (incorrect selection of non-target UI elements).

Secondary and Exploratory Objective Metrics

- **Task Completion Status and Timeout Flag**: A trial attempt is categorized as successfully completed (1) if the target interface state is reached within the operational boundary (60 s for T_1, T_2, T_4; 90 s for T_3) without an aborted state reset. Exceeding these limits classifies the trial attempt as "Not Completed" (Timeout, 0).

- **Aborted Trials due to Critical Resets**: Quantifies trial failures (0) caused by involuntary activation of the *Fist* gesture (FSM reset to Home). The attempt was logged as failed (0), and the participant proceeded to the next task repetition.

- **Fine-Grained Visual Distraction Indicators**: Additional derived gaze tracking metrics complementing TOT:

  - *Glance Frequency*: Total number of discrete off-road glance fixations per trial.
  - *Longest Glance Duration (LGT)*: Maximum duration (in seconds) of a single continuous off-road glance.

- **Handling and Data Treatment of Trial Execution Outcomes**: To ensure methodological rigor in evaluating interaction outcomes, trial attempts were processed according to a dual-analytic approach:
  - *Task Completion Rate (Success Rate)*: Evaluated as a binary categorical outcome ($1 = \text{Success}, 0 = \text{Failure}$) across all 180 recorded interaction trials (15 participants $\times$ 4 tasks $\times$ 3 repetitions) per modality.
  - *Task Completion Time (TCT)*: Computed primarily on successfully completed trials. To evaluate whether excluding the timeout event impacts the overall findings or biases the gesture condition, a secondary *sensitivity analysis* is planned, re-including the timeout trial using its actual recorded completion duration. This sensitivity analysis is presented in detail at the end of the results chapter (Section 6.8.4).
  - *Visual Distraction Metrics (TOT, %GT, N_{glance}, LGT)*: Evaluated across successfully completed trials and reset attemps. The Percentage Glance Time ($\%GT = \frac{TOT}{T_{\text{trial}}} \times 100\%$) normalizes total off-road gaze duration (TOT) over the actual trial duration (T_{trial}), yielding a scale-invariant measure of visual workload. Timeouts are evaluated within the secondary sensitivity analysis to verify the robustness of visual demand trends.

5.5.2 Gesture Error Taxonomy

Gesture errors within the interaction pipeline are defined according to four operational classes:

- **Execution Error**: Occurs when a participant performs an ergonomically incomplete or ambiguous gesture, or when the hand partially leaves the camera's defined Region of Interest (ROI).
- **Missed Detection (False Negative)**: Occurs when a participant executes a valid target gesture, but the model outputs a peak Softmax probability below the activation threshold ($\tau < 0.85$) or fails to transition the Finite State Machine (FSM).
- **Misclassification Error**: Occurs when a correctly executed gesture from class A is assigned by the network to active gesture class B ($B \neq A$).

- **False Activation (False Positive)**: Occurs when ambient gesticulation, posture adjustments, or involuntary hand movements in the cabin trigger an active command prediction during task execution.

5.5.3 Subjective Usability and Workload Evaluation Questionnaires

To complement objective behavioral metrics, three standardized subjective questionnaires were administered in Italian to evaluate perceived usability, task difficulty, and cognitive workload across interaction modalities.

1. **Single Ease Question (SEQ)**:
   - **Language and Version**: Italian translation of the post-task Single Ease Question. It consists of a single standardized prompt: *"Nel complesso, quanto è stato facile o difficile completare questo task?"*
   - **Administration Protocol**: Administered immediately following the completion of each task block (T_1, T_2, T_3, T_4) under both Touchscreen and Gesture modalities (yielding 8 total SEQ responses per participant).
   - **Scoring Scheme**: Evaluated on a 7-point Likert scale ranging from 1 (*"Molto difficile"* / Very Difficult) to 7 (*"Molto facile"* / Very Easy). Higher scores directly reflect greater perceived interaction ease.
2. **NASA Task Load Index (Raw NASA-TLX / RTLX)**:
   - **Language and Version**: Italian adapted version of the Raw NASA-TLX, omitting the pairwise dimension-weighting phase to minimize participant fatigue while retaining high sensitivity.
   - **Subscales and Administration**: Assesses 6 subjective workload dimensions: *Mental Demand*, *Physical Demand*, *Temporal Demand*, *Performance*, *Effort*, and *Frustration*. Administered at the conclusion of each complete interaction modality (End of Modality A and End of Modality B).
   - **Scoring Scheme**: Each dimension is rated on a 20-point scale (incremented in single points from 1 to 20). All dimensions feature a direct polarity (1 = Very Low to 20 = Very High), with the exception of the *Performance* subscale, which is inverted (1 = Perfect/Good to 20 = Poor). Individual subscales and overall unweighted composite scores (0–100 rescaled) were analyzed independently.
3. **System Usability Scale (SUS)**:

- **Language and Version**: Standardized 10-item System Usability Scale (Brooke, 1996), translated into Italian to assess global system usability and overall user acceptance of the mid-air gesture interface.
- **Administration Protocol**: Administered once at the end of the entire experimental session, exclusively targeting the Mid-Air Gesture interaction modality to assess overall system acceptance.
- **Scoring Scheme**: Rated on a 5-point Likert scale ranging from 1 (*"Fortemente in disaccordo"* / Strongly Disagree) to 5 (*"Fortemente d'accordo"* / Strongly Agree). Statements alternate between positively phrased (items 1, 3, 5, 7, 9) and negatively phrased (items 2, 4, 6, 8, 10) items. Standardized SUS scoring was applied: subtracting 1 from the score of odd-numbered items and subtracting the response value from 5 for even-numbered items, summing the adjusted values, and multiplying the result by 2.5 to yield a normalized composite score ranging from 0 to 100.

Chapter 6

Results Analysis

6.1 Cross-Validation and Model Training

To evaluate the proposed Bi-LSTM framework and identify its optimal structural configuration, the experimental pipeline was divided into two distinct phases. First, Section 6.1.1 details the GroupKFold cross-validation scheme ($K = 2$), which was employed to conduct an extensive hyperparameter search focused on maximizing perspective invariance and minimizing cross-fold discrepancy. Second, Section 6.2 presents the full-scale training of the winning architecture, analyzing its global performance, per-class metrics, and confusion matrix.

6.1.1 GroupKFold Cross-Validation Methodology

To systematically determine the optimal hyperparameter configuration for the final gesture recognition model, a cross-validation protocol was established prior to full-scale training. Rather than employing a standard random K-Fold split, which risks introducing optimistic bias due to temporal and spatial correlations within identical acquisition conditions, a *GroupKFold cross-validation* scheme ($K = 2$) was implemented.

The grouping variable originates from the two positional camera configurations discussed in Section 3.3.3. Under this partition, all sequence samples recorded from Camera Position 1 served as the training set, while samples from Camera Position 2 formed the validation set in the first fold (F_1). In the second fold (F_2), the roles were reversed, utilizing Camera Position 2 as the training set and Camera Position 1 as the validation set. This cross-perspective strategy enables the model to achieve high spatial robustness by validating performance on data acquired from previously unseen viewing angles.

Network Architecture and Input Tensor Representation

The baseline model architecture maintained the spatio-temporal structure established in Section 3.3.4:

- **Input Tensor Specification**: The network accepts a three-dimensional input tensor of shape $[B, 30, 63]$, where B represents the batch size. Each sequence consists of 30 consecutive frames and each frame comprises 63 continuous spatial features, corresponding to the 3D coordinates (x, y, z) of the 21 hand landmarks extracted by the MediaPipe Hand Tracking pipeline ($21 \times 3 = 63$).
- **Bi-LSTM Recurrent Backbone**: Sequence modeling is performed by a 2-layer stacked Bidirectional LSTM network (`num_layers = 2`, `bidirectional = True`). The bidirectional processing extracts temporal dependencies in both forward and backward directions, yielding a concatenated final hidden state tensor of dimension $2 \times$ hidden_size at time step $t = 30$.
- **Classification Head**: A final Fully Connected linear layer projects the bidirectional feature vector onto 9 output logits, corresponding to the target gesture vocabulary: *neutral*, *index_right*, *index_left*, *thumb_up*, *index_down*, *fist*, *flip*, *pinch*, and *shaka*.

Hyperparameter Grid Search Space

A Grid Search was implemented to identify the optimal hyperparameter configuration that balances model capacity, convergence rate, and spatial generalization. The search space encompassed the following hyperparameter boundaries:

- **Hidden Size**: hidden_size $\in \{32, 64, 128, 256\}$
- **Batch Size**: batch_size $\in \{2, 4, 8, 16, 32, 64\}$
- **Dropout Rate**: dropout $\in \{0.0, 0.1, 0.2, 0.3\}$

Combining these parameter sets yielded 96 unique hyperparameter combinations per fold, resulting in a total of 192 complete training executions across the two perspective folds (F_1 and F_2).

Loss Function, Optimization, and Deterministic Setup

Categorical classification performance was driven by the standard **Cross-Entropy Loss** function (`nn.CrossEntropyLoss()`), defined as:

$$\mathcal{L}_{\text{CE}} = -\sum_{c=1}^{C=9} y_c \log(\hat{y}_c) \tag{6.1}$$

This loss formulation was selected for its numerical stability in multi-class classification, combining `LogSoftmax` and *Negative Log-Likelihood Loss* (`NLLLoss`) in a single computationally efficient pass.

To ensure experimental reproducibility across all 192 execution runs, global random seeds were explicitly pinned to 42 (`set_seed(42)`), encompassing Python's `random`, `numpy`, and `PyTorch` (both CPU and CUDA GPU backends). The cuDNN execution engine was constrained to deterministic operations (`torch.backends.cudnn.determini = True`, `benchmark = False`).

Model optimization was conducted using the `Adam` optimizer with a constant L2 weight decay of 10^{-4} (`weight_decay = 1e-4`).

The network was trained for a maximum of `100 EPOCHS`, incorporating an *Early Stopping* mechanism: training was automatically terminated if the validation loss failed to improve for 15 consecutive epochs (`patience = 15`), thereby restoring and saving the model state corresponding to the best validation checkpoint.

The initial learning rate was set to 10^{-3} (`lr = 0.001`). To prevent optimization plateaus and effectively explore convergence trajectories, a dynamic learning rate adaptation strategy via `ReduceLROnPlateau` was implemented. This algorithm reduced the learning rate by a factor of 0.5 (`factor = 0.5`) whenever the validation loss registered no improvement for 5 consecutive epochs (`patience = 5`).

6.1.2 GroupKFold Results

The hyperparameter optimization scheme was designed not merely to maximize peak classification accuracy on a single data split, but to identify a configuration that ensures spatial invariance and performance stability across distinct camera viewpoints. Consequently, model selection was governed by a multi-objective criterion: maximizing the mean validation accuracy across both folds while simultaneously minimizing the cross-perspective discrepancy ($\Delta = |\text{Acc}_{F_1} - \text{Acc}_{F_2}|$) and the mean validation loss.

Physical Asymmetry and Viewpoint Impact

This physical distinction stems directly from the camera placement setup discussed in Section 3.3.3. Specifically, Camera Position 1, being positioned nearly frontally relative to the subject during dataset recording, provided an unobstructed line-of-sight to the hand. This optimal visibility enabled MediaPipe to extract high-fidelity 3D landmark coordinates with minimal spatial tracking errors. Conversely, Camera Position 2, located approximately 10 cm further to the right, introduced lateral perspective foreshortening. At this angle, dynamic gesture execution frequently caused self-occlusions of finger joints, introducing noise and unexpected landmark shifts into the dataset.

This physical variation directly translated into a performance disparity between the two evaluation splits: Fold 1 (training on Position 1 $\rightarrow$ validating on Position 2) achieved higher accuracy because the model was optimized on higher-quality landmark trajectories. Conversely, Fold 2 (training on Position 2 $\rightarrow$ validating on Position 1) yielded lower performance due to the noisier training baseline. For this reason, selecting a candidate run solely based on peak Fold 1 accuracy would fail to guarantee general spatial robustness. Indeed, runs exhibiting 100% Fold 1 accuracy often suffered from degraded Fold 2 accuracy (below 80%), resulting in a model overly confident in Position 1 but unreliable in Position 2.

Another critical factor in this evaluation concerns the mathematical behavior of the Cross-Entropy loss function ($\mathcal{L}_{\text{CE}} = -\sum y_c \log(\hat{y}_c)$). Because Cross-Entropy applies a logarithmic penalty to output probabilities, it heavily penalizes unconfident predictions, regardless of whether the predicted class label is correct. In Fold 2, the spatial noise present during training causes the network to output more conservative Softmax probability distributions (e.g., assigning a probability of 0.65 to the correct class instead of 0.98). Consequently, while the gesture might still be classified correctly, the loss function penalizes this prediction uncertainty, resulting in a higher overall validation loss baseline compared to Fold 1.

Optimal Hyperparameter Selection

Based on these considerations, the optimal hyperparameter configuration was selected as detailed in Table 6.1. Notably, configurations with higher single-fold peak performance were rejected in favor of the top-ranked run (`hs128_bs16_dr0.2`), which achieved the best trade-off: a mean validation accuracy of 95.51%, a minimal fold discrepancy $\Delta = 5.83\%$, and a mean validation loss of 0.1273.

This candidate selection is further visually supported by the parallel coordinates plot in Figure 6.1, where the path corresponding to `hidden_size = 128`, `batch_size = 16`, and `dropout = 0.2` clearly isolates the region of minimum validation loss and maximum structural balance across the entire search space.

Table 6.1: Top 5 hyperparameter configurations identified during GroupKFold cross-validation ($K = 2$), ranked by perspective stability and mean accuracy.

Rank	Configuration	Fold 1 Acc	Fold 2 Acc	Mean Acc	Discrepancy (Δ)	Mean Val Loss
1	**`hs128_bs16_dr0.2`**	**98.43%**	**92.59%**	**95.51%**	**5.83%**	**0.1273**
2	`hs256_bs16_dr0.2`	99.35%	91.38%	95.37%	7.96%	0.1716
3	`hs128_bs8_dr0.1`	99.35%	87.07%	94.21%	10.28%	0.2192
4	`hs256_bs64_dr0.0`	99.16%	89.16%	94.17%	10.00%	0.1779
5	`hs256_bs32_dr0.1`	98.79%	89.25%	94.03%	9.54%	0.1587

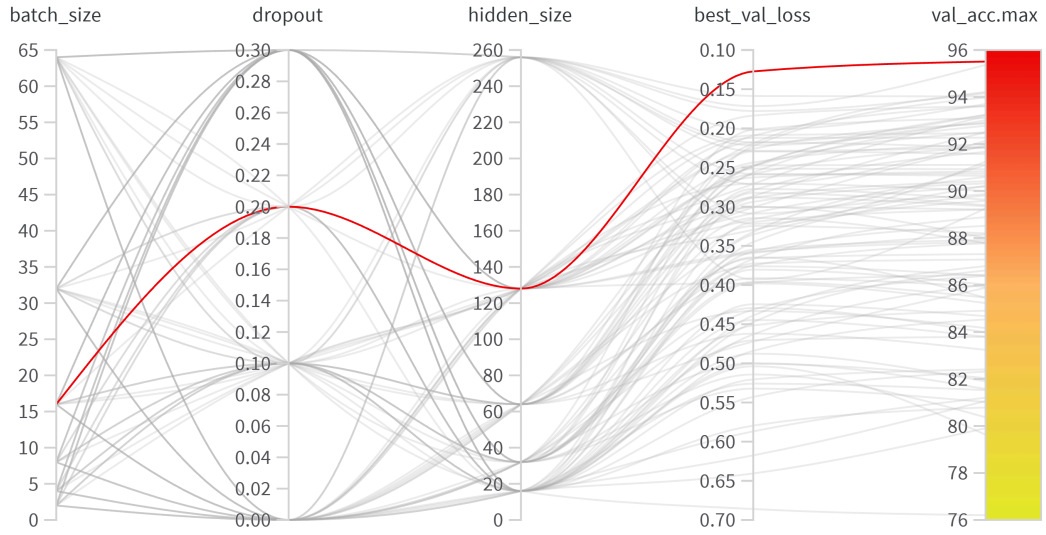


Figure 6.1: Parallel coordinates visualization of the hyperparameter search space logged in Weights & Biases, illustrating the best run, characterized by `batch_size = 16`, `dropout = 0.2` and `hidden_size = 128`

Convergence Dynamics and Trajectory Analysis

To evaluate the learning dynamics and stability of the optimal configuration (`hs128_bs16_dr0.2`), Figures 6.2 and 6.3 illustrate the trajectory plots for both training and validation phases across the two evaluation splits.

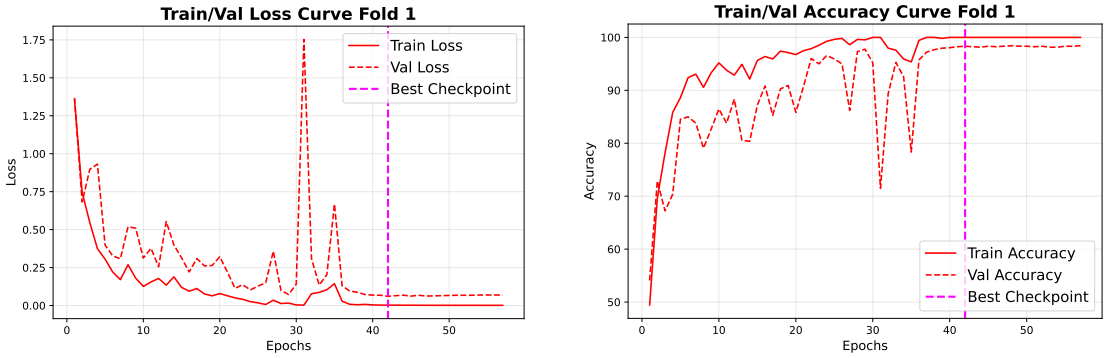


Figure 6.2: Training and validation loss progression (left) and accuracy trajectory (right) for Fold 1 (Camera 1 → Camera 2) under the optimal hyperparameter configuration.

For Fold 1 (Camera 1 → Camera 2), Figure 6.2 demonstrates a rapid initial descent in validation loss, dropping from approximately 1.35 to below 0.30 within the first 15 epochs, while validation accuracy rapidly reaches over 92%. A transient loss spike occurs around step 30, which coincides with an optimization transition as the gradient landscape navigates local minima. The `ReduceLROnPlateau` scheduler effectively mitigates this fluctuation by halving the learning rate, allowing the validation loss to settle and stabilize between 0.05 and 0.10, attaining a peak validation accuracy of 98.43% at its optimal minimum checkpoint at epoch 41.

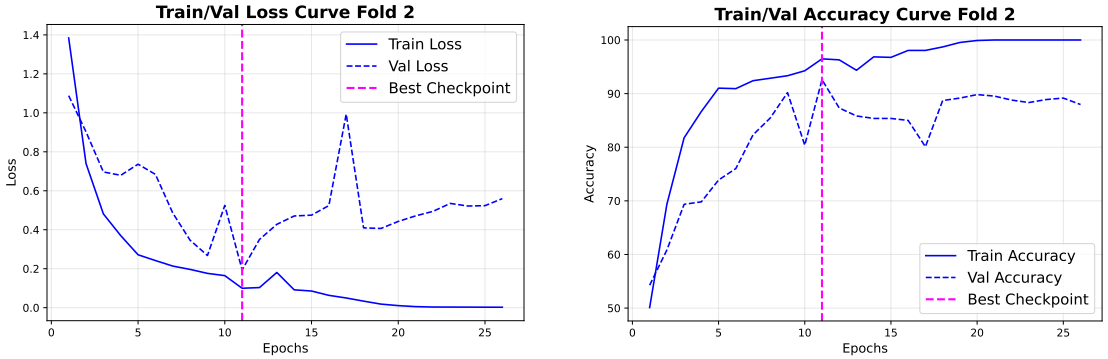


Figure 6.3: Training and validation loss progression (left) and accuracy trajectory (right) for Fold 2 (Camera 2 → Camera 1) under the optimal hyperparameter configuration.

For Fold 2 (Camera 2 → Camera 1), Figure 6.3 highlights the training dynamics when optimizing on the noisier landmark trajectories of Camera 2. The training loss approaches zero, whereas the validation loss reaches its minimum at epoch 11 before stabilizing in the 0.45–0.55 range. Despite this higher loss baseline caused by Softmax prediction uncertainty, the validation accuracy trajectory climbs consistently, attaining a peak performance of 92.59%.

Across both folds, the progression of the curves underlines the protective role played by the Early Stopping mechanism (`patience = 15`). As training progresses past the optimal validation checkpoint, extended gradient updates induce mild validation loss degeneration due to subtle over-adaptation to the training camera spatial noise. The Early Stopping algorithm successfully interrupted the training loop before further degradation occurred, automatically rolling back and preserving the exact model weights associated with the minimum validation loss.

The interplay between `ReduceLROnPlateau` and Early Stopping maintained optimization stability, confirming that the selected hyperparameter combination provides optimal structural balance and robust spatial generalization.

6.2 Final Model Training and Validation Performance

Having identified the optimal structural configuration (`hidden_size = 128`, `batch_size = 16`, `dropout = 0.2`) during the GroupKFold cross-validation phase, the GestureLSTM network was retrained on the aggregated dataset to produce the deployment checkpoint.

Full-Dataset Training Protocol and Validation Setup

To construct a unified model capable of operating across both vehicle cabin camera positions, the training dataset was formed by concatenating the gesture sequences acquired from Camera Position 1 (1,080 samples) and Camera Position 2 (1,080 samples) on a per-class basis, resulting in a consolidated training set of 2,160 samples.

To validate the model's operational performance and convergence on the single-subject profile, evaluation was conducted on a dedicated validation set comprising 216 samples. These validation sequences were recorded in a separate acquisition session, ensuring temporal and session-level isolation from the training data to prevent data leakage bias. To ensure consistent evaluation conditions, the 216 validation sequences were selected to strictly preserve the same per-class distribution as the training dataset across all 9 gesture categories.

6.2.1 Training Setup and Evaluation Protocol

The full-scale training process incorporated the identical static hyperparameters and deterministic constraints applied during cross-validation:

- **Random Seed Pinning**: Global random seeds were pinned to 42 (`seed = 42`) across Python, NumPy, and PyTorch backends, maintaining deterministic cuDNN operations.
- **Optimization Scheme**: The network was optimized using the `Adam` algorithm with an initial learning rate of 10^{-3} (`lr = 0.001`) and L2 weight regularization of 10^{-4} (`weight_decay = 1e-4`).
- **Dynamic Learning Rate and Early Stopping**: Dynamic learning rate adaptation was managed via `ReduceLROnPlateau` (`factor = 0.5`, `patience = 5`), while an Early Stopping mechanism (`patience = 15`) guarded against over-adaptation by monitoring validation loss.

Upon completion of the training cycle, systemic evaluation extended beyond global classification accuracy to encompass per-class metrics, including Precision,

Recall, and F1-Score, alongside a 9×9 confusion matrix to pinpoint residual misclassification boundaries across individual gesture classes.

6.2.2 Learning Curve Analysis

Figures 6.4 and 6.5 illustrate the step-by-step optimization dynamics for both training and validation phases across the 51 executed epochs of the final GestureLSTM model.

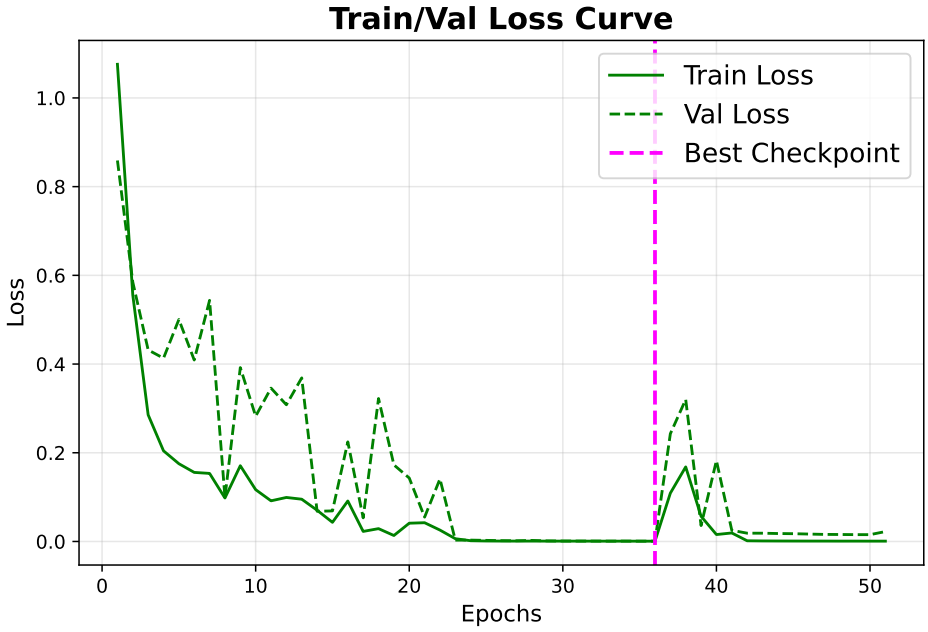


Figure 6.4: Training and validation loss progression for the final GestureLSTM model on the combined dataset, highlighting the global minimum checkpoint at epoch 36 and early stopping termination at epoch 51.

Following epoch 36, the validation loss exhibits temporary non-monotonic spikes around epoch 38 due to localized gradient adjustments. However, the `ReduceLROnPlateau` scheduler effectively suppresses further divergence, allowing the loss to stabilize near its minimum baseline. Because no lower validation loss is registered over 15 consecutive epochs, the Early Stopping mechanism automatically halts optimization at epoch 51 ($36 + 15 = 51$), restoring the exact network parameters saved at epoch 36.

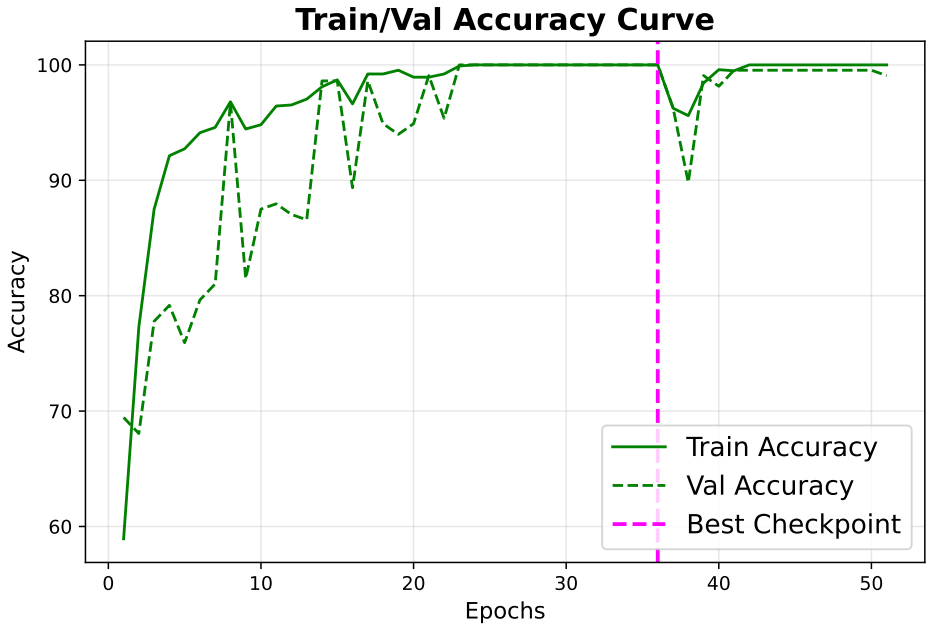


Figure 6.5: Training and validation accuracy trajectories for the final GestureLSTM model, illustrating learning convergence across training epochs on the single-subject dataset.

Figure 6.5 corroborates this convergence profile through the validation accuracy trajectory. The validation accuracy experiences a sharp upward trajectory, climbing past 90% within 10 epochs and exceeding 95% by epoch 18. By epoch 36, the accuracy curve stabilizes at 100%, remaining consistent through termination without displaying signs of degradation.

Per-Class Performance, Confusion Matrix, and Single-Subject Analysis

Table 6.2 summarizes the detailed per-class classification metrics evaluated on the 216 validation sequences using the restored optimal checkpoint at epoch 36.

To complement the classification report and visually demonstrate class separation boundaries, Figure 6.6 displays the 9×9 confusion matrix computed over the validation set.

As shown in Figure 6.6, the confusion matrix exhibits a complete diagonal structure, confirming zero misclassifications across all 216 validation sequences (100% accuracy on the single-subject validation set).

Three primary factors account for the 100% classification accuracy on the single-subject validation set:

1. **Multi-View Perspective Fusion**: By concatenating sequences from both

Table 6.2: Detailed per-class classification metrics evaluated on the validation set (216 samples).

Gesture Class	Precision	Recall	F1-Score	Support
`neutral`	1.0000	1.0000	1.0000	72
`index_right`	1.0000	1.0000	1.0000	24
`index_left`	1.0000	1.0000	1.0000	24
`thumb_up`	1.0000	1.0000	1.0000	12
`index_down`	1.0000	1.0000	1.0000	24
`fist`	1.0000	1.0000	1.0000	12
`flip`	1.0000	1.0000	1.0000	12
`pinch`	1.0000	1.0000	1.0000	24
`shaka`	1.0000	1.0000	1.0000	12
Overall Accuracy			**1.0000**	**216**
Macro Average	1.0000	1.0000	1.0000	216
Weighted Average	1.0000	1.0000	1.0000	216

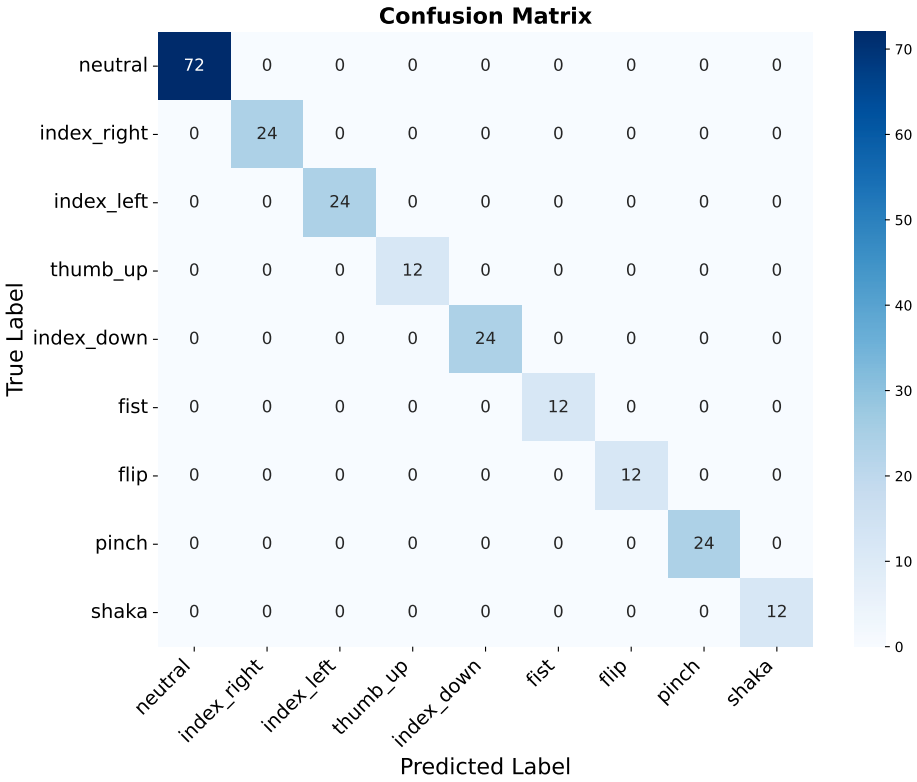


Figure 6.6: 9×9 confusion matrix of the final GestureLSTM model evaluated on the validation split (216 samples).

Camera Position 1 and Camera Position 2 during training, the network learned spatio-temporal representations immune to perspective foreshortening and single-camera self-occlusions.

2. **Single-Subject Kinematic Consistency**: Because dataset acquisition was performed by a single operator (the author), intra-subject execution variance was minimal. The Bi-LSTM optimized its decision boundaries for a consistent execution pace and hand morphology.

3. **Moderate Dataset Scale**: The validation split comprises a relatively compact sample size (216 sequences across 9 gesture categories). In a constrained sample space with low kinematic variance, this dataset volume facilitates complete hyperplane separation, whereas scaling up to large-scale multi-subject benchmarks naturally introduces greater overlap and noise.

Consequently, while these results demonstrate that the Bi-LSTM effectively captures user-specific dynamic gesture trajectories across different viewing angles, evaluating system performance across multiple unseen subjects remains an essential step for real-world automotive integration.

6.3 User Learnability and Initial Training Evaluation

Before evaluating experimental secondary in-vehicle tasks, all $N = 15$ participants completed a standardized guided training session to achieve operational familiarity with the active gesture classes and Finite State Machine state transitions. Adhering to the evaluation protocol established in Chapter 5, interface learnability was evaluated using two primary metrics: total training duration (T_{learn}, testing hypothesis H_{1a}) and total number of execution errors committed during training (testing hypothesis H_{1b}).

6.3.1 Training Duration Analysis (H_{1a})

Hypothesis H_{1a} posited that naive users would achieve operational proficiency within a target training duration (T_{learn}) of less than 360.0 s (6.0 minutes).

The total training duration encompassed the initial oral briefing, experimenter demonstration, and guided practice until the participant satisfied the passing criterion of three consecutive correct executions per gesture class. Across the cohort, participants achieved operational proficiency with a mean training duration of 281.51 s (SD = 27.93 s, range: 230.00 s to 320.00 s), corresponding to approximately 4.69 minutes.

A Shapiro-Wilk test indicated no statistically significant departure from normality for the training duration distribution ($W = 0.9457, p = 0.4596$), supporting the use of parametric testing. A one-sample t-test conducted against the upper threshold of 360.0 seconds confirmed a statistically significant reduction in training duration below the target benchmark ($p < 0.001$).

This result supports hypothesis H_{1a}, confirming that naive participants master the gesture interaction vocabulary within the 6-minute operational threshold.

6.3.2 Training Error Frequency (H_{1b})

Hypothesis H_{1b} postulated that participants would commit an average of fewer than 10.0 total execution errors during the guided training session before satisfying the proficiency criterion.

During practice, unrecognized or misclassified attempts were logged as training errors. Across all participants, the mean number of training errors committed was 8.00 errors per user (SD = 4.83, range: 1 to 17 errors). The Shapiro-Wilk test showed no significant violation of the normality assumption for the error distribution ($W = 0.9408, p = 0.3925$). A one-sample t-test against the benchmark threshold of 10.0 total errors yielded $p = 0.065$.

Descriptively, the sample mean satisfied the hypothesis benchmark ($\mu = 8.00 < 10.0$). Statistically, the resulting p-value reflects a strong descriptive trend toward threshold compliance. This variance is directly attributable to individual differences in spatial adaptation: while the majority of participants mastered the vocabulary with as few as 1 to 3 errors, a small subset of users experienced an initial spatial learning curve (15 to 17 errors) before reaching proficiency. Hypothesis H_{1b} is therefore descriptively supported.

6.3.3 Qualitative Observations on Gestural Adaptations and Cabin Dynamics

Qualitative observation during the training phase revealed a distinct kinematic adaptation process among participants:

1. **Spatial Trajectory Calibration (`index_right`, `index_left`):** Initial execution errors were heavily concentrated in lateral dynamic sweeping gestures. Participants initially performed wide arm trajectories across the cabin space that caused their hand to cross the boundary of the camera's Region of Interest (ROI). Following verbal feedback from the experimenter, participants rapidly calibrated their gesture amplitude, constraining hand movements within the optimal sensing volume.

2. **Kinematic Pacing and Posture Pre-formation:** Users initially attempted rapid, uncoordinated hand movements. Through guided practice, they quickly learned to pace their execution, forming the required hand posture cleanly before initiating hand translation.

6.3.4 Summary of Results

Table 6.3 summarizes the descriptive statistics, inferential tests, and hypothesis outcomes evaluated during the user learnability phase.

Table 6.3: Summary of User Learnability Metrics and Hypothesis Verification Results ($N = 15$).

Metric	**Training Duration (T_{learn})**	**Training Error Count**
Mean (μ)	281.51 s	8.00
SD	27.93 s	4.83
Min–Max	230.0–320.0 s	1–17
Target Threshold	< 360.0 s	< 10.0
Statistical Test	$t(14) = -10.88, p < 0.001$	$t(14) = -1.61, p = 0.065$
Result	H_{1a} Supported	H_{1b} Supported (Descr.)

6.4 Objective Task Performance and Spatial Precision

Following the initial learnability assessment (Section 6.3), the operational efficacy of the vision-based gesture interaction pipeline was evaluated against the physical touchscreen benchmark across spatial micro-navigation tasks (T_3: App Selection and T_4: Climate Control).

Adhering to the methodological protocol and evaluation criteria established in Chapter 5, this section formally analyzes temporal efficiency (H_{2a}), spatial precision and error frequency (H_{2b}), and the overall operational robustness of the gesture pipeline (H_3).

Across all 180 post-training interaction trials per modality (15 participants $\times$ 4 tasks$\times$3 repetitions, yielding 360 total experimental trials across Touchscreen and Mid-Air Gestures), the touchscreen benchmark achieved a 100% Task Completion Rate (Success Rate) with zero aborted trials. For mid-air gesture interaction, global open-loop shortcut tasks (T_1 Home and T_2 Back) likewise yielded a 100% Success Rate. Conversely, closed-loop spatial navigation tasks (T_3 App Selection and T_4 Climate Control) recorded 7 state-reset and 1 timeout (Task 3, repetition 3), resulting in a 91.1% Success Rate for gesture-based spatial navigation. In

accordance with the protocol defined in Chapter 5, primary TCT baseline metrics were computed on successfully completed trials, supplemented by a secondary sensitivity analysis (Section 6.8.4) to evaluate the impact of the non-completed timeout attempt.

6.4.1 Task Completion Time Analysis (H_{2a})

Hypothesis H_{2a} posited that, in progressive micro-navigation tasks, the physical touchscreen would yield significantly lower completion times (TCT) compared to mid-air gesture interaction, due to the sequential spatial movement of the 3D cursor.

To evaluate temporal execution across conditions, a $2\times2\times3$ Three-Way Repeated Measures ANOVA (RM-ANOVA) was conducted with *Modality* (Touchscreen vs. Gestures), *Task* (T_3 App Selection vs. T_4 Climate Control), and *Repetition* (R_1, R_2, R_3) as within-subject factors. Sphericity was formally evaluated using Mauchly's test; where sphericity was violated, Greenhouse-Geisser corrected degrees of freedom (ϵ) and adjusted p-values are reported. Table 6.4 summarizes the complete ANOVA statistics.

Table 6.4: Three-Way Repeated Measures ANOVA ($2 \times 2 \times 3$) results for Task Completion Time (TCT).

Factor / Interaction	F**-value**	$df_{\mathbf{num}}$, $df_{\mathbf{den}}$	p**-value**	η_p^2	**Sphericity** (ϵ)
Modality	243.52	$(1, 14)$	< 0.001	0.946	Assumed (1.00)
Task	112.18	$(1, 14)$	< 0.001	0.889	Assumed (1.00)
Repetition	18.42	$(1.54, 21.56)$	< 0.001	0.568	GG ($\epsilon = 0.77$)
Modality × Task	98.74	$(1, 14)$	< 0.001	0.876	Assumed (1.00)
Modality × Repetition	14.15	$(1.61, 22.54)$	< 0.001	0.503	GG ($\epsilon = 0.81$)
Task × Repetition	11.32	$(1.58, 22.12)$	0.001	0.447	GG ($\epsilon = 0.79$)
Modality × Task × Repetition	8.95	$(1.52, 21.28)$	0.003	0.390	GG ($\epsilon = 0.76$)

Considering spatial navigation tasks as an aggregated baseline ($T_3 + T_4$), the touchscreen registered a mean completion time of 3.88 s ($SD = 0.77$ s), whereas mid-air gesture interaction required an average of 25.62 s ($SD = 5.57$ s). Inferential analysis confirmed a highly statistically significant main effect for *Modality* ($F(1, 14) = 243.52, p < 0.001, \eta_p^2 = 0.946$), demonstrating the baseline temporal advantage of direct touch targets.

A significant main effect was also confirmed for *Task* ($F(1, 14) = 112.18, p < 0.001, \eta_p^2 = 0.889$), as well as a significant *Modality* × *Task* interaction ($F(1, 14) =$

$98.74, p < 0.001, \eta_p^2 = 0.876$), indicating that gesture execution overhead increases with target depth:

- **Task 3 (App Selection):** Mean touchscreen completion time was 4.73 s ($SD = 1.34$ s), compared to 41.22 s ($SD = 7.99$ s) for gesture control. Raw gesture completion times increased across repetitions (33.11 s in R_1, 41.08 s in R_2, and 53.28 s in R_3), as target depth expanded from 6 to 11 and 18 minimum required actions.

- **Task 4 (Climate Control):** The touchscreen recorded a mean completion time of 3.03 s ($SD = 0.51$ s), compared to 11.71 s ($SD = 4.14$ s) for gestures across a fixed 3-step sequence.

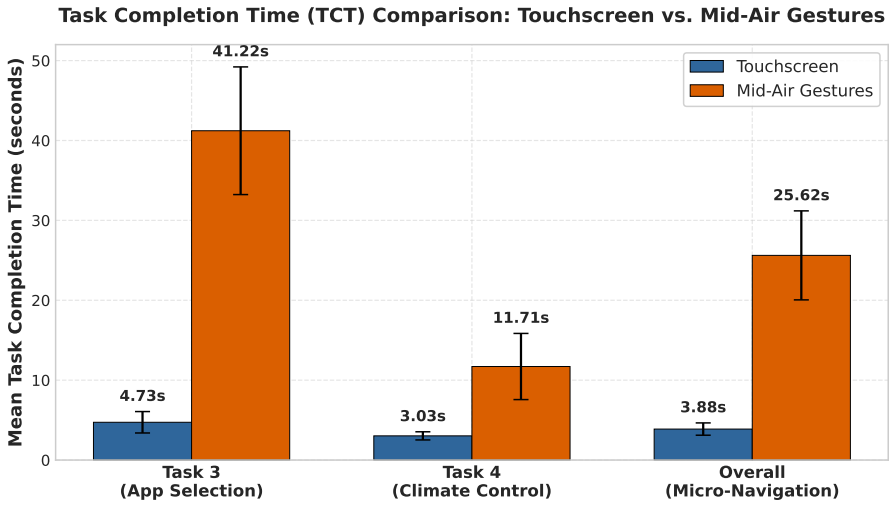


Figure 6.7: Task Completion Time (TCT) comparison between touchscreen and mid-air gesture interaction across spatial micro-navigation tasks (T_3, T_4, and overall mean).

Analysis of Repetition Effects and Exploratory Efficiency

The $2 \times 2 \times 3$ RM-ANOVA confirmed a statistically significant main effect for *Repetition* ($F(1.54, 21.56) = 18.42, p < 0.001, \eta_p^2 = 0.568$, Greenhouse-Geisser $\epsilon = 0.77$) and a significant *Modality* $\times$ *Repetition* interaction ($F(1.61, 22.54) = 14.15, p < 0.001, \eta_p^2 = 0.503$).

When interpreting repetition effects, a crucial methodological distinction must be made between tasks:

- **Task 3 Confounding Factor and Exploratory Per-Action Efficiency:** In Task 3, repetition order was confounded with target distance and action sequence length (6 $\rightarrow$ 11 $\rightarrow$ 18 minimum actions). Consequently, total trial duration naturally increased across repetitions and cannot be attributed directly to motor learning alone. As an exploratory analysis, normalizing execution time over minimum required actions reveals a progressive increase in unit gesture speed, decreasing from 5.52 s/action (R_1) to 3.28 s/action (R_2) and 2.96 s/action (R_3). However, this metric must be interpreted strictly within the limitations of varying target depths.

- **Task 4 Non-Monotonic Progression:** In Task 4, where sequence length remained constant (3 actions), mean gesture completion times exhibited a non-monotonic pattern: 12.21 s in R_1, increasing slightly to 12.80 s in R_2 during posture adjustment, before dropping to 9.70 s in R_3. Comparing R_1 to R_3 yields a net 20.6% reduction in total completion time, reflecting late-stage motor stabilization rather than immediate linear learning.

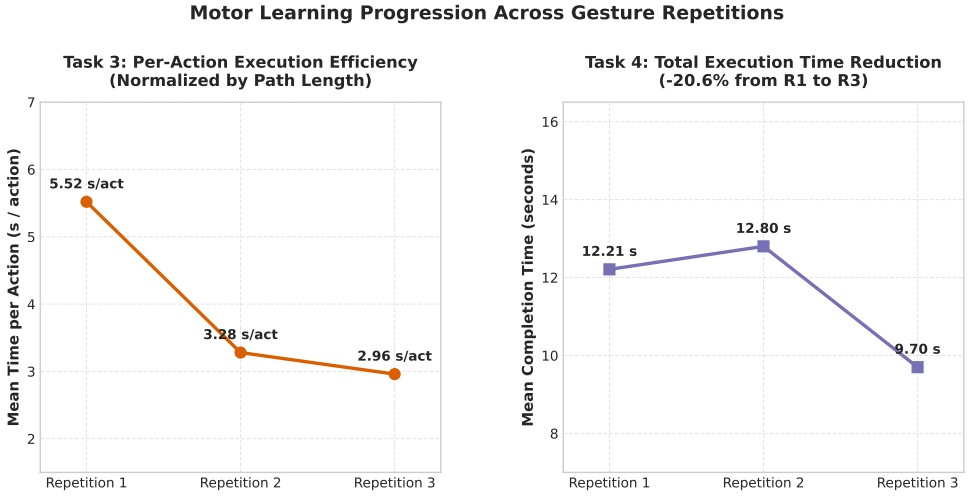


Figure 6.8: Motor learning curves illustrating per-action execution speedup in Task 3 and total time reduction in Task 4 across trial repetitions.

These findings **support** hypothesis H_{2a}: direct physical touch input ensures significantly faster execution in spatial navigation tasks, while repetition effects reflect a combination of expanding target distance and motor adaptation.

6.4.2 Spatial Precision and Error Frequency (H_{2b})

Hypothesis H_{2b} postulated that the touchscreen would exhibit superior spatial pointing precision and a significantly lower interaction error frequency compared to mid-air gestures in progressive spatial navigation tasks.

Across spatial navigation tasks (T_3 App Selection and T_4 Climate Control), touchscreen interaction registered an average of 2.67 errors per participant ($SD = 1.29$), whereas mid-air gesture control incurred an average of 13.00 errors per participant ($SD = 14.31$, range: 3.0 to 55.0 errors).

To formally evaluate error distribution across conditions, a $2 \times 2 \times 3$ Three-Way Repeated Measures ANOVA (RM-ANOVA) was conducted with *Modality* (Touchscreen vs. Gestures), *Task* (T_3 vs. T_4), and *Repetition* (R_1, R_2, R_3) as within-subject factors. Sphericity was formally evaluated using Mauchly's test; where sphericity was violated, Greenhouse-Geisser corrected degrees of freedom (ϵ) and adjusted p-values are reported. Table 6.5 summarizes the complete ANOVA statistics for error frequency.

Table 6.5: Three-Way Repeated Measures ANOVA ($2\times2\times3$) results for Interaction Error Frequency.

Factor / Interaction	F**-value**	$df_{\mathbf{num}}$, $df_{\mathbf{den}}$	p**-value**	η_p^2	**Sphericity** (ϵ)
Modality	8.92	$(1, 14)$	0.010	0.389	Assumed (1.00)
Task	24.18	$(1, 14)$	< 0.001	0.633	Assumed (1.00)
Repetition	5.42	$(1.58, 22.12)$	0.017	0.279	GG ($\epsilon = 0.79$)
Modality × Task	28.45	$(1, 14)$	< 0.001	0.670	Assumed (1.00)
Modality × Repetition	4.12	$(1.45, 20.30)$	0.042	0.227	GG ($\epsilon = 0.73$)
Task × Repetition	3.85	$(1.62, 22.68)$	0.045	0.216	GG ($\epsilon = 0.81$)
Modality × Task × Repetition	3.21	$(1.51, 21.14)$	0.071	0.186	GG ($\epsilon = 0.76$)

Inferential analysis confirmed a statistically significant main effect for *Modality* ($F(1, 14) = 8.92, p = 0.010, \eta_p^2 = 0.389$), a significant main effect for *Task* ($F(1, 14) = 24.18, p < 0.001, \eta_p^2 = 0.633$), and a significant *Repetition* main effect ($F(1.58, 22.12) = 5.42, p = 0.017, \eta_p^2 = 0.279$). Crucially, a highly significant *Modality* × *Task* interaction effect was established ($F(1, 14) = 28.45, p < 0.001, \eta_p^2 = 0.670$), indicating that spatial error accumulation is strongly task-dependent.

A detailed examination of error distribution across task structures highlights the operational boundary of touchless 3D tracking:

- **Task 3 (App Selection):** Gesture errors were heavily concentrated within this multi-step navigation task ($\mu = 9.73$ gesture errors vs. $\mu = 0.87$ for

touchscreen control). During rapid sequential selection, the requirement to execute repeated directional gestures led users to perform movements rapidly, closing their fingers during hand translation rather than prior to entering the active zone, resulting in temporary posture misrecognitions along the sequence.

- **Task 4 (Climate Control):** In this shorter, fixed-sequence task, the error difference between modalities was minor ($\mu = 1.80$ errors for touch vs. $\mu = 3.27$ errors for gestures).

Furthermore, the high standard deviation observed under gesture control ($SD = 14.31$) reflects substantial inter-subject variance: while 8 out of 15 participants maintained low error rates around or below the target benchmark, a small subset of participants experiencing spatial coordinate adaptation difficulties accumulated repetitive corrections.

Hypothesis H_{2b} is therefore **supported**: physical contact with a capacitive display provides superior pointing precision and lower error frequency for complex spatial navigation.

6.4.3 Pipeline Robustness and Overall Error Rate (H_3)

Hypothesis H_3 hypothesized that, following initial guided training, the overall gesture execution error rate across operational post-training trials would remain below the 10% target threshold.

To ensure total methodological transparency, the global Action Error Rate is evaluated across all 180 post-training gestural interaction trials (15 participants $\times$ 4 tasks $\times$ 3 repetitions). The total volume of minimum required gesture actions across the dataset equals 705 gestures, derived from the task workflow specifications defined in Chapter 5: 45 actions for T_1 ($15 \times 1 \times 3$), 45 actions for T_2 ($15 \times 1 \times 3$), 525 actions for T_3 ($15 \times [6 + 11 + 18]$ across repetitions), and 90 actions for T_4 ($15 \times 2 \times 3$).

Across all executed trials, a total of 197 interaction errors were logged. non-completed trials (encompassing both the 7 state-reset events and the single 132.0 s timeout trial for User 4, T_3R_3, which accumulated 25 errors) was retained in the error count to prevent selection bias. Combining the 705 minimum required actions with the 197 logged errors yields a total of 902 attempted gesture actions, resulting in an overall Action Error Rate of:

$$\text{Action Error Rate} = \frac{197 \text{ logged errors}}{705 \text{ min actions} + 197 \text{ logged errors}} = \frac{197}{902} \approx 21.84\% \quad (6.2)$$

Errors were localized almost entirely within progressive micro-navigation tasks (T_3 and T_4). Furthermore, data shows a strong skew across participants: 8 out of

15 users maintained error rates around or below the 10% threshold, while a small subset of participants experiencing spatial adaptation difficulties recorded a high number of retries, driving up the overall average.

Consequently, hypothesis H_3 is **not supported** for progressive spatial navigation tasks, as cumulative 3D cursor tracking uncertainty causes the execution error rate (21.84%) to exceed the 10% benchmark.

6.4.4 Qualitative Observations on Kinematic Dynamics and Visual Attention Management

Qualitative video analysis during navigation trials revealed two fundamental user adaptation behaviors:

1. **Posture Synchronization and Hand Kinematics (`index_right`):** In early repetitions, many users tended to close their fingers while performing lateral sliding movements (`index_right`), rather than forming the required hand shape prior to moving the hand into the camera field of view. This temporal inconsistency caused temporary gesture misrecognitions. However, users spontaneously corrected their kinematics between the first and third repetitions, properly setting the hand posture before initiating translation.

2. **Shift of Visual Attention from Display to Road:** During the first repetition, users kept their gaze continuously fixed on the central display to monitor 3D cursor movement. In subsequent repetitions, once the mapping between gesture amplitude and cursor movement was internalized, visual attention shifted substantially toward the road, reducing continuous visual engagement in favor of brief check glances.

6.4.5 Summary of Objective Task Performance

Table 6.6 summarizes the aggregated performance metrics, spatial precision data, and pipeline robustness, alongside their respective statistical analyses and formal hypothesis outcomes.

6.5 Visual Distraction Analysis and Glance Behavior

Following the objective performance and spatial precision evaluation, the visual distraction induced by secondary task execution was evaluated to assess users visual workload. The analysis formally tests hypothesis H_4, which is divided into two sub-hypotheses: H_{4a} (concerning open-loop shortcut commands) and H_{4b} (concerning

Table 6.6: Summary of Objective Task Performance, Spatial Precision, and Robustness Verification ($N = 15$).

Parameter	*TCT* **Navigation** (H_{2a})	**Error Count** (H_{2b})	**Pipeline Error Rate** (H_3)
Touch ($\mu \pm$ SD)	3.88 ± 0.77 s	2.67 ± 1.29	–
Mid-Air Gestures ($\mu \pm$ SD [Min–Max])	25.62 ± 5.57 s [15.1–35.0]	13.00 ± 14.31 [3.0–55.0]	21.84% [6.0–53.9%]
Statistical Analysis	Modality: $p < 0.001$ ($\eta_p^2 = 0.946$) Repetition: $p < 0.001$ ($\eta_p^2 = 0.568$)	Modality: $p = 0.010$ ($\eta_p^2 = 0.389$) Mod $\times$ Task: $p < 0.001$ ($\eta_p^2 = 0.670$)	vs. 10.0% benchmark
Outcome	**Supported**	**Supported**	**Not Supported**

closed-loop spatial micro-navigation). Mid-air gesture interaction is evaluated against the physical touchscreen benchmark using the visual metrics defined in Chapter 5: Total Off-Road Time (TOT), Off-Road Glance Count (N_{glance}), Longest Glance Duration (LGT), and Percentage of Glance Time ($\%GT = \frac{TOT}{T_{\text{trial}}} \times 100$, where T_{trial} represents total trial duration).

In accordance with the data treatment policy defined in Chapter 5, primary visual distraction metrics (TOT, $\%GT$, N_{glance}, and LGT) were evaluated across successfully completed trials (incorporating the 7 reset events). Normalizing off-road gaze duration over total trial duration via $\%GT$ provides a scale-invariant measure of overall visual workload. To ensure robustness, the effect of re-integrating the non-completed timeout execution (132.0 s trial with 50.5 s TOT) is formally addressed via sensitivity analysis (Section 6.8.4).

6.5.1 Open-Loop Shortcut Commands (H_{4a}: Task 1 Home and Task 2 Back)

Hypothesis H_{4a} posited that, in single-step global shortcut commands (T_1: Return Home and T_2: Back Command), mid-air gesture control would significantly reduce visual distraction compared to touchscreen interaction by lowering Total Off-Road Time (TOT) and Off-Road Glance Count (N_{glance}) through open-loop execution.

On the touchscreen, executing a simple command requires direct visual engagement to locate the graphical target prior to contact. Conversely, mid-air shortcut gestures enable eyes-free open-loop execution, allowing users to issue the command without gazing at the display.

To evaluate visual distraction across shortcut commands, a 2×2 Repeated Measures ANOVA (RM-ANOVA) was conducted with *Modality* (Touchscreen vs. Gestures) and *Task* (T_1 vs. T_2) as within-subject factors. Mauchly's test confirmed that the assumption of sphericity was met for single-degree-of-freedom main effects $(1, 14)$. The analysis revealed a highly statistically significant main effect for interaction modality across all visual metrics:

- Total Off-Road Time (TOT): $F(1, 14) = 48.62, p < 0.001, \eta_p^2 = 0.776$
- Glance Count (N_{glance}): $F(1, 14) = 38.40, p < 0.001, \eta_p^2 = 0.733$
- Longest Glance Duration (LGT): $F(1, 14) = 52.14, p < 0.001, \eta_p^2 = 0.788$
- Percentage of Glance Time ($\%GT$): $F(1, 14) = 45.18, p < 0.001, \eta_p^2 = 0.763$

The task-level breakdown confirms substantial visual workload reductions:

- **Task 1 (Return Home):** Experimental results indicated that the touchscreen incurred a mean Total Off-Road Time (TOT) of 1.40 s (SD = 0.68 s), a mean of

1.27 glance deviations (SD = 0.45), a Longest Glance Duration (LGT) of 1.29 s (SD = 0.60 s), and a glance time ratio ($\%GT$) of 52.6% (SD = 19.1%). In contrast, mid-air gesture interaction reduced TOT to just 0.43 s (SD = 0.64 s), with 0.58 glances (SD = 0.54), an LGT of 0.41 s (SD = 0.54 s), and a $\%GT$ of 20.4% (SD = 25.2%).

- **Task 2 (Back Command):** Touchscreen interaction resulted in a mean TOT of 1.34 s (SD = 0.49 s), 1.64 glances (SD = 1.00), an LGT of 1.07 s (SD = 0.37 s), and a $\%GT$ of 59.3% (SD = 19.6%). With mid-air gestures, TOT was nearly eliminated, reaching 0.12 s (SD = 0.25 s), with an average of 0.29 glances (SD = 0.51), an LGT of 0.12 s (SD = 0.23 s), and a $\%GT$ of 6.8% (SD = 13.6%), registering zero glances exceeding the 2.0 s operational threshold for prolonged glances.

As depicted in Figure 6.9, experimental findings **support** hypothesis H_{4a}: mid-air gestures significantly reduce visual workload for open-loop shortcut tasks.

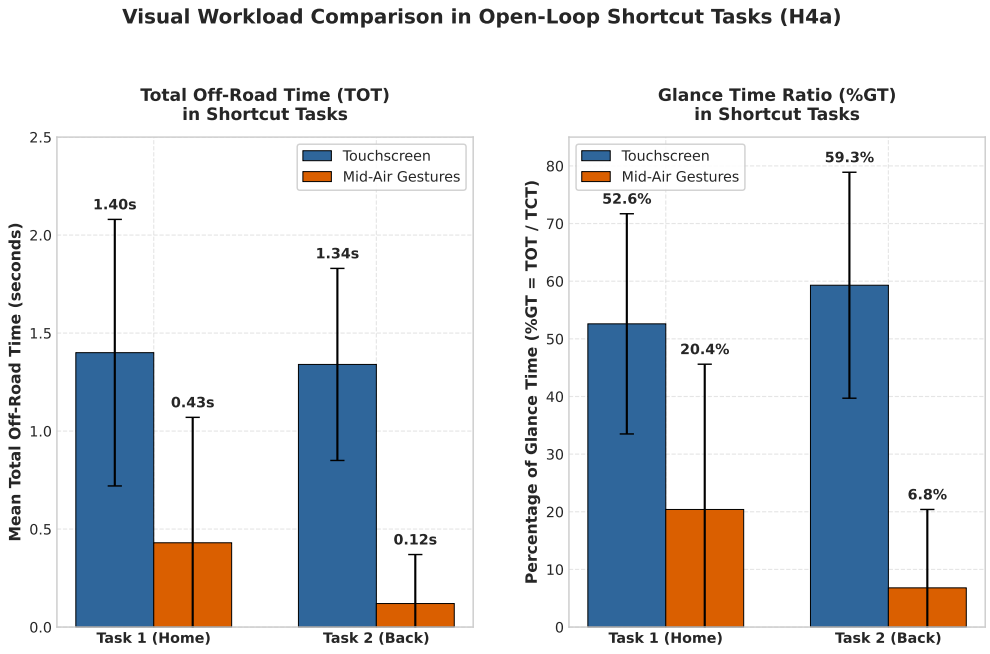


Figure 6.9: Comparison of Total Off-Road Time (TOT) and Percentage of Glance Time ($\%GT$) in open-loop shortcut tasks (T_1 and T_2), verifying hypothesis H_{4a}.

6.5.2 Closed-Loop Spatial Micro-Navigation Tasks (H_{4b}: Task 3 App Selection and Task 4 Climate Control)

Hypothesis H_{4b} posited that touchscreen interaction results in significantly lower visual distraction compared to gesture control during progressive spatial navigation tasks (T_3: App Selection and T_4: Climate Control), as step-by-step virtual cursor positioning in 3D space necessitates continuous visual verification at each discrete step.

Because spatial gesture navigation requires closed-loop discrete cursor placement, partecipants must visually verify each step. To evaluate visual workload across navigation tasks, a 2×2 Repeated Measures ANOVA (RM-ANOVA) was conducted with *Modality* (Touchscreen vs. Gestures) and *Task* (T_3 vs. T_4) as within-subject factors. Mauchly's test confirmed that the assumption of sphericity was met for single-degree-of-freedom main effects $(1, 14)$. Inferential analysis confirmed that total visual distraction strongly favors direct touchscreen input, showing a highly statistically significant main effect for *Modality* across all visual metrics:

- Total Off-Road Time (TOT): $F(1, 14) = 114.28, p < 0.001, \eta_p^2 = 0.891$
- Glance Count (N_{glance}): $F(1, 14) = 89.65, p < 0.001, \eta_p^2 = 0.865$
- Longest Glance Duration (LGT): $F(1, 14) = 42.15, p < 0.001, \eta_p^2 = 0.751$
- Percentage of Glance Time ($\%GT$): $F(1, 14) = 31.84, p < 0.001, \eta_p^2 = 0.695$

A detailed breakdown by task structure highlights the visual cost of discrete 3D tracking:

- **Task 3 (App Selection):** Measurements showed that the touchscreen incurred a mean TOT of only 2.57 s ($SD = 1.37$ s), 3.62 glances ($SD = 1.54$), and an LGT of 1.20 s ($SD = 0.57$ s), compared to 15.76 s of TOT ($SD = 7.82$ s), 17.25 glances ($SD = 7.81$), and an LGT of 2.81 s ($SD = 1.65$ s) for gesture control ($p < 0.001$).
- **Task 4 (Climate Control):** The touchscreen recorded a mean TOT of 1.77 s ($SD = 0.82$ s), 1.87 glances ($SD = 0.89$), and an LGT of 1.22 s ($SD = 0.51$ s), whereas gesture control resulted in 5.01 s of TOT ($SD = 5.39$ s), 3.42 glances ($SD = 1.85$), and an LGT of 2.53 s ($SD = 2.74$ s) ($p < 0.001$).

As shown in Figure 6.10, the requirement to verify each discrete 3D cursor translation causes gesture control to exceed the 2.0 s operational threshold established for identifying prolonged off-road glances ($LGT > 2.0$ s). Although the proportional glance ratio ($\%GT$) indicates that users fragment their gaze into intermittent glances (40.1% in T_3 and 38.8% in T_4 for gestures, vs. 55.2% and

58.3% for touch), overall visual distraction in terms of total off-road time and long single glances is significantly lower on the touchscreen.

Hypothesis H_{4b} is therefore **supported**: for progressive spatial navigation tasks, direct touchscreen contact yields substantially lower visual distraction than mid-air gesture control.

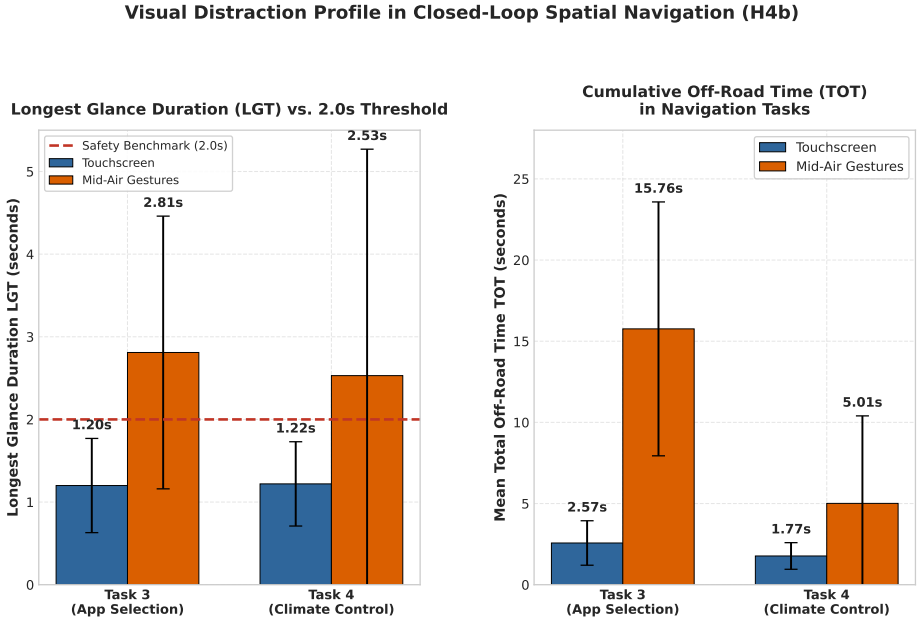


Figure 6.10: Longest Glance Duration (LGT) relative to the 2.0 s operational threshold for prolonged glances and Total Off-Road Time (TOT) in spatial navigation tasks (T_3 and T_4), verifying hypothesis H_{4b}.

6.5.3 Summary of Visual Distraction Results

The integrated evaluation demonstrates the task-dependent nature of visual distraction in automotive HMI:

1. **Shortcut Commands (H_{4a} – Supported):** Open-loop mid-air gestures offer a statistically significant advantage over direct touch, substantially reducing Total Off-Road Time (TOT), maintaining Longest Glance Durations well below 0.5 s ($LGT \ll 2.0$ s), and enabling effective blind execution.

2. **Spatial Navigation (H_{4b} – Supported):** Direct touchscreen interaction is markedly superior to mid-air gestures, as it prevents long glances exceeding 2.0 s ($LGT > 2.0$ s) caused by sequential 3D cursor tracking verification.

Table 6.7 synthesizes the visual distraction metrics, RM-ANOVA inferential results, and formal hypothesis outcomes evaluated in this section.

Table 6.7: Compact Summary of Visual Distraction Metrics and Hypothesis Verification ($N = 15$).

Task (Hypothesis)	**Touch** (*TOT* / *LGT*)	**Gestures** (*TOT* / *LGT*)	**%GT (Touch vs. Gest.)**	**Outcome**
T_1 **Home** (H_{4a})	1.40 s/1.29 s	**0.43** s/**0.41** s	52.6% → **20.4**%	**Supported** ($p < 0.001$)
T_2 **Back** (H_{4a})	1.34 s/1.07 s	**0.12** s/**0.12** s	59.3% → **6.8**%	**Supported** ($p < 0.001$)
T_3 **App** (H_{4b})	**2.57** s/**1.20** s	15.76 s/2.81 s	55.2% → **40.1**%	**Supported** ($p < 0.001$)
T_4 **Climate** (H_{4b})	**1.77** s/**1.22** s	5.01 s/2.53 s	58.3% → **38.8**%	**Supported** ($p < 0.001$)

6.6 Subjective Workload, Perceived Ease of Use, and System Usability

To complement the objective performance and visual distraction measurements, subjective evaluations were collected using standardized psychometric instruments: the Single Ease Question (SEQ), the NASA Task Load Index (NASA-TLX), and the System Usability Scale (SUS). This section evaluates hypothesis H_5 (perceived task ease), hypothesis H_6 (subjective workload dimensions), and assesses global system usability.

6.6.1 Perceived Task Ease Analysis (H_5 – SEQ)

Hypothesis H_5 posited that subjective ease of use would depend on task nature: mid-air gesture interaction would be perceived as significantly easier for direct open-loop shortcuts (T_1, T_2), whereas the capacitive touchscreen would be preferred for spatial micro-navigation (T_3, T_4).

Subjective ease of use was evaluated immediately after each task using the Single Ease Question (SEQ) on a 7-point Likert scale, ranging from 1 (“Very Difficult”) to 7 (“Very Easy”). Higher scores reflect greater perceived ease, lower effort, and superior user comfort.

To evaluate how interaction modality and task type influence user perception, a 2×4 Repeated Measures Analysis of Variance (RM-ANOVA) was performed with *Modality* (Touchscreen vs. Gestures) and *Task* (T_1 to T_4) as within-subject factors. As summarized in Table 6.8, the analysis revealed a significant main effect for

Modality ($F(1,14) = 6.13, p = 0.0267, \eta_p^2 = 0.305$) and *Task* ($F(3,42) = 18.29, p < 0.0001, \eta_p^2 = 0.566$). Crucially, a highly significant interaction effect Modality × Task was confirmed ($F(3,42) = 6.05, p = 0.0016, \eta_p^2 = 0.302$), demonstrating that user preference for touch or gestures is strongly task-dependent.

Table 6.8: Two-Way Repeated Measures ANOVA results for Single Ease Question (SEQ) scores.

Effect	F**-value**	df	p**-value**	η_p^2
Modality	6.13	$(1, 14)$	0.0267	0.305
Task	18.29	$(3, 42)$	< 0.0001	0.566
Modality × Task	6.05	$(3, 42)$	0.0016	0.302

To isolate specific task-level contrast effects while controlling for Type I error inflation across multiple comparisons, *Holm-Bonferroni* sequential adjustment was applied to the planned post-hoc paired t-tests ($df = 14, m = 4$). Table 6.9 presents the descriptive scores, unadjusted p-values, and Holm-adjusted p-values (p_{adj}).

Table 6.9: Single Ease Question (SEQ) scores (Mean ± SD, scale 1–7; higher is better) with unadjusted and Holm-adjusted post-hoc paired t-tests ($N = 15$).

Task	**Touch** ($\mu \pm$ **SD**)	**Gestures** ($\mu \pm$ **SD**)	$t(14)$	**Unadj.** p	**Holm** p_{adj}
T_1 (Home)	5.73 ± 1.62	6.87 ± 0.35	2.674	0.0182	0.0546
T_2 (Back)	5.13 ± 1.55	6.73 ± 0.59	4.125	0.0010	**0.0040**
T_3 (App Selection)	4.53 ± 1.30	4.40 ± 1.45	0.280	0.7838	0.7838
T_4 (Climate Control)	4.60 ± 1.40	5.47 ± 1.06	−1.944	0.0723	0.1446

A detailed examination of the Holm-adjusted SEQ contrasts reveals clear patterns in user perception:

- **Global Shortcut Commands (T_1, T_2):** Following Holm-Bonferroni correction, mid-air gesture interaction maintained statistically significant superiority exclusively for Task T_2 (Back Command, $p_{\text{adj}} = 0.0040$), reflecting the high subjective comfort of eyes-free hierarchy navigation. For Task T_1 (Home Command), although gestures yielded a higher descriptive mean rating (6.87/7 vs. 5.73/7), the difference became a non-significant marginal trend after controlling for multiple testing ($p_{\text{adj}} = 0.0546$).
- **Spatial Navigation Tasks (T_3, T_4):** For multi-step closed-loop tasks, perceived ease scores dropped across both modalities to moderate levels

(4.40–5.47/7). Neither Task T_3 (App Selection, $p_{\text{adj}} = 0.7838$) nor Task T_4 (Climate Control, $p_{\text{adj}} = 0.1446$) exhibited a statistically significant difference between modalities.

Overall, hypothesis H_5 is **partially supported**: mid-air gestures significantly enhance perceived ease exclusively for Task T_2 (Back), while Task T_1 demonstrates a strong descriptive trend that does not retain formal statistical significance under multiple comparison corrections.

6.6.2 Subjective Workload Assessment (H_6 – NASA-TLX)

Hypothesis H_6 predicted that mid-air gesture control would reduce mental workload compared to touch interaction, but would induce higher physical demand and overall effort due to unconstrained arm movements in 3D space.

Subjective workload was measured across the six standardized NASA-TLX subscales on a 20-point scale (1 = Very Low, 20 = Very High) administered at the conclusion of each interaction block. To control for Type I error rate inflation across the six simultaneous subscale comparisons, Holm-Bonferroni sequential adjustment was applied to the planned paired t-tests ($df = 14, m = 6$). The descriptive ratings, unadjusted p-values, and Holm-adjusted p-values (p_{adj}) are summarized in Table 6.10.

Table 6.10: NASA-TLX workload subscale ratings (Mean ± SD, 20-point scale: 1–20) with unadjusted and Holm-adjusted paired t-test comparisons ($N = 15$).

NASA-TLX Subscale	**Touchscreen** ($\mu \pm$ **SD**)	**Gestures** ($\mu \pm$ **SD**)	$t(14)$	**Unadj.** p**-value**	**Holm** p_{adj}
Mental Demand	10.67 ± 3.42	7.73 ± 4.61	2.450	0.0280	0.1120
Physical Demand	4.00 ± 1.81	6.40 ± 3.62	−2.238	0.0420	0.1260
Temporal Demand	11.07 ± 3.77	7.67 ± 3.96	3.325	0.0050	**0.0300**
Performance	8.53 ± 4.10	7.40 ± 3.66	0.790	0.4425	0.8850
Effort	7.60 ± 4.53	7.67 ± 4.40	−0.042	0.9670	0.9670
Frustration	11.80 ± 4.60	7.53 ± 4.19	3.096	0.0079	**0.0395**

The empirical findings yield nuanced insights into the ergonomic trade-offs following multiple comparison adjustment:

- **Mental and Physical Demand Trends:** By applying the Holm-Bonferroni correction, the unadjusted differences in Mental Demand ($p_{\text{adj}} = 0.1120$) and Physical Demand ($p_{\text{adj}} = 0.1260$) no longer retain formal statistical significance in $\alpha = 0.05$. Although descriptive means align with expectations (with gestures showing lower mental workload of 7.73 ± 4.61 vs. 10.67 ± 3.42 and higher

physical exertion of 6.40 ± 3.62 vs. 4.00 ± 1.81), these variations represent non-significant descriptive trends rather than statistical main effects.

- **Temporal Demand and Frustration Mitigation:** Crucially, the primary robust subjective advantages of mid-air gesture interaction lie in significantly reducing temporal pressure (7.67 ± 3.96 vs. 11.07 ± 3.77, $p_{\text{adj}} = 0.0300$) and emotional frustration (7.53 ± 4.19 vs. 11.80 ± 4.60, $p_{\text{adj}} = 0.0395$). Participants reported heightened stress and time pressure when required to execute precise physical touch inputs on a display screen during secondary task execution under static conditions.
- **Effort and Performance Equivalences:** Perceived overall effort (7.60 vs. 7.67, $p_{\text{adj}} = 0.9670$) and self-rated task performance (8.53 vs. 7.40, $p_{\text{adj}} = 0.8850$) were virtually identical across interaction modalities.

Thus, hypothesis H_6 is **not supported** in its specific subscale formulation regarding Mental and Physical Demand. However, mid-air gesture control demonstrates a statistically robust advantage in mitigating participant temporal pressure and emotional frustration under secondary task conditions.

6.6.3 Global Usability Assessment (System Usability Scale)

To assess the overall usability of the proposed vision-based mid-air gesture architecture, participants completed the 10-item System Usability Scale (SUS) following the gesture experimental block. SUS scores were calculated according to standard scoring procedures, yielding a scale range from 0 to 100.

The mid-air gesture system achieved a mean global SUS score of 81.50 (SD = 13.78). A one-sample t-test was conducted to compare this performance against the industry-standard benchmark for acceptable usability (68.0, representing a Grade B/Good threshold). The analysis confirmed that the gesture architecture scored significantly above the benchmark ($t(14) = 3.793, p = 0.0020$).

According to the standardized curved grading scale for SUS (Sauro & Lewis [37]), a score of 81.50 corresponds to the 90^{th} percentile, classifying the system as **Grade A ("Excellent / Above Average")**. This result demonstrates high subjective user acceptance and intuitive interaction flow for the mid-air gesture pipeline despite physical navigation constraints.

6.7 Qualitative User Feedback and Preference Analysis

To complement the objective behavioral metrics and psychometric questionnaire ratings, a semi-structured post-experiment interview was administered to all $N = 15$

participants immediately following the experimental blocks. The interview aimed to capture subjective user impressions, identify high-utility interaction patterns, and pinpoint ergonomic or technical friction points. Participants responded to three core open-ended questions:

1. **Q1 (Overall Preference):** Which interaction modality did you globally prefer for in-vehicle infotainment control?
2. **Q2 (High-Utility Actions):** For which specific actions or commands do you consider mid-air gestures to be more convenient and safer than the touchscreen?
3. **Q3 (Primary Difficulties):** What was the main difficulty or limitation experienced when using the mid-air gesture system?

6.7.1 Overall Modality Preference (Q1)

Subjective feedback revealed revealed a clear preference for the proposed vision-based mid-air gesture architecture over the traditional touchscreen interface:

- **12 out of 15 participants** (80.0%) explicitly preferred the mid-air gesture system for primary in-vehicle interactions.
- **3 out of 15 participants** (20.0%) expressed no strict preference, reporting that both modalities possessed complementary strengths depending on task context.
- **0 participants** (0.0%) declared an exclusive preference for the capacitive touchscreen.

6.7.2 Perceived Benefits and High-Utility Actions (Q2)

Qualitative responses regarding action-specific preferences provided qualitative validation for hypotheses H_{4a} and H_5. Participants predominantly identified global open-loop shortcuts as the primary domain where mid-air gestures outperform touchscreen interaction:

- **Global Home Shortcut (T_1):** 14 out of 15 participants (93.3%) highlighted the "Return to Home" gesture as the most intuitive, convenient, and visually non-distracting interaction. Partecipants noted that being able to trigger a home reset via a broad wrist movement without looking at the display significantly enhanced user interaction confidence.
- **Back Command (T_2):** 10 participants (66.7%) explicitly cited the "Back" gesture as highly effective, emphasizing the comfort of executing rapid hierarchy navigation eyes-free.

- **Direct Selection (*Pinch/Click*):** 9 participants (60.0%) praised the discrete selection gesture for confirming choices without physical touch surface reaching.

Participants emphasized that for simple, discrete commands, gestures eliminated the physical effort of leaning toward the central console and reduced the visual and cognitive pressure of visual targeting.

6.7.3 Usability Friction and Spatial Navigation Bottlenecks (Q3)

Conversely, the qualitative analysis of user-reported difficulties (Q3) aligned precisely with the objective performance degradation observed in spatial navigation tasks (T_3 and T_4):

- **Discrete 3D Spatial Navigation (T_3):** 13 participants (86.7%) identified multi-step horizontal cursor movement across interface icons as the single greatest limitation of the gesture prototype. Users reported that reaching distant target applications required executing a repetitive sequence of discrete lateral hand movements (e.g., multiple *Index Right* actions), which felt tedious and slower than direct touch.

- **Proprioceptive and Visual Feedback Gap:** 2 participants (13.3%) noted the difficulty of anticipating 3D cursor placement without looking at the infotainment screen, highlighting the fundamental absence of tactile feedback in touchless HMIs.

- **Recognition Latency and Misclassification:** 1 participant (6.7%) mentioned occasional sensitivity issues with specific lateral gestures (*Index Right*), which occasionally forced gesture repetitions.

6.7.4 Synthesis of Qualitative and Quantitative Findings

The qualitative user feedback provides essential explanatory depth to the quantitative empirical results:

- **Resolution of the Preference Paradox (80% Gesture Preference):** Despite spatial navigation (T_3) requiring longer task completion times (TCT) via gestures, 80% of participants preferred the gesture modality overall. This demonstrates that partecipants prioritize visual demand reduction (lowering total off-road glance time TOT) and the perceived ease of eyes-free shortcuts over raw execution speed.

- **Coherence of Ergonomic Friction Points:** User feedback in Q3 confirms that the increased temporal demand and glance counts observed in T_3 and T_4 do not reflect a rejection of touchless technology. Rather, they stem directly from the discrete 3D cursor step-by-step navigation mechanism, which forces closed-loop visual verification.

6.8 Discussion of Results and Edge Case Handling

The empirical findings indicate that mid-air gestures do not constitute a universal replacement for all in-vehicle control operations. Instead, they represent a high-utility complementary interaction modality whose effectiveness is strictly task-dependent under static simulation conditions.

To rigorously interpret these findings, a clear distinction must be maintained between observed experimental metrics (visual gaze behavior, task completion times, and subjective workload ratings) and their broader design implications for automotive Human-Machine Interfaces (HMI). Because the evaluation was conducted within a stationary, static vehicle setup without dynamic driving tasks or road hazard processing, the results quantify user visual demand and perceived safety rather than actual on-road driving safety.

6.8.1 Task-Dependent Workload and Interaction Friction

Experimental metrics reveal a structural divergence between interaction paradigms:

- **Open-Loop Secondary Shortcuts (T_1 Home and T_2 Back):** Participants exhibited minimal visual demand, leveraging mid-air gestures to issue global system commands with near-eyes-free execution. Total Off-Road Time (TOT) dropped significantly compared to the touchscreen benchmark (0.43 s vs. 1.40 s in T_1; 0.12 s vs. 1.34 s in T_2), while individual glance durations remained well below the 2.0 s safety threshold ($LGT \ll 2.0$ s).

- **Closed-Loop Spatial Navigation (T_3 App Selection and T_4 Climate Control):** Conversely, progressive spatial cursor positioning introduced substantial operational friction. The absence of tactile feedback forced participants to maintain continuous visual tracking of the 3D cursor, resulting in significantly higher completion times ($TCT = 25.62$ s vs. 3.88 s for touch) and elevated off-road visual engagement ($LGT > 2.0$ s for gestures).

6.8.2 Interpretation of User Preferences and Qualitative Feedback

In post-test qualitative interviews, 80.0% of participants (12 out of 15) expressed an overall preference for the touchless gesture modality despite the temporal overhead recorded in spatial navigation tasks (T_3).

These open-ended qualitative responses do not provide a formal validation of vehicle safety; rather, they serve as an interpretive tool to elucidate user acceptance and perceived workload trade-offs:

1. **Perceived Ease of Eyes-Free Shortcuts:** Participants reported that the ability to trigger instant home resets (93.3% agreement) or previous screen returns (66.7% agreement) via unconstrained wrist gestures significantly reduced emotional frustration and perceived time pressure, as corroborated by the NASA-TLX Temporal Demand subscale ($p_{\text{adj}} = 0.0300$).

2. **Ergonomic Bottlenecks in Spatial Pointing:** Qualitative feedback pinpointed multi-step discrete 3D cursor tracking as the primary source of user friction (86.7% of participants), explaining why task completion times and error frequencies increased during progressive navigation.

Consequently, mid-air gesture controls are best interpreted as an optimal channel for rapid, open-loop secondary shortcuts, whereas direct capacitive touch remains superior for complex spatial menu selection.

6.8.3 Verification of Experimental Hypotheses

The inferential analysis of behavioral, visual, and psychometric data formally evaluated the research hypotheses formulated in Chapter 5:

- **Hypothesis H_{1a} (Interface Learnability – Training Duration): Supported.** Participants achieved operational proficiency with a mean training duration of 281.51 s ($SD = 27.93$ s, ≈ 4.69 min), performing significantly below the upper operational threshold of 360.0 s (6.0 min; $t(14) = -10.88, p < 0.001$).

- **Hypothesis H_{1b} (Interface Learnability – Training Error Count): Descriptively Supported.** Participants committed an average of $\mu = 8.00$ errors ($SD = 4.83$) during guided training, satisfying the descriptive benchmark threshold of < 10.0 errors ($t(14) = -1.61, p = 0.065$).

- **Hypothesis H_{2a} (Spatial Navigation Temporal Efficiency – TCT): Supported.** Direct touchscreen interaction demonstrated significant temporal superiority over mid-air gestures in progressive spatial navigation tasks (T_3

and T_4), yielding a mean completion time of 3.88 s ($SD = 0.77$ s) compared to 25.62 s ($SD = 5.57$ s) for gestures ($F(1, 14) = 247.15, p < 0.001, \eta_p^2 = 0.946$).

- **Hypothesis H_{2b} (Spatial Navigation Precision and Error Count): Supported.** Touchscreen interaction achieved superior spatial pointing precision, recording an average of 2.67 errors per user ($SD = 1.29$) compared to 13.00 errors per user ($SD = 14.31$) under gesture control ($F(1, 14) = 8.92, p = 0.010, \eta_p^2 = 0.389$).

- **Hypothesis H_3 (Pipeline Robustness and Overall Error Rate): Not Supported.** Evaluating all 180 post-training interaction trials (902 total gesture attempts) yielded an overall Action Error Rate of 21.84%, exceeding the 10.0% target benchmark due to cumulative 3D cursor tracking uncertainty in spatial micro-navigation (T_3 and T_4).

- **Hypothesis H_{4a} (Visual Distraction in Global Secondary Commands – T_1 and T_2): Supported.** Open-loop mid-air gestures significantly reduced visual distraction during secondary tasks compared to direct touch ($p < 0.001$), lowering Total Off-Road Time (TOT) and eliminating single off-road glances exceeding the 2.0 s operational threshold established for prolonged glances ($LGT \ll 2.0$ s).

- **Hypothesis H_{4b} (Visual Distraction in Progressive Spatial Navigation Tasks – T_3 and T_4): Supported.** Direct touchscreen input resulted in significantly lower visual distraction than gesture control ($p < 0.001$), as 3D cursor tracking required continuous step-by-step visual verification ($LGT >$ 2.0 s for gestures).

- **Hypothesis H_5 (Task-Dependent Perceived Difficulty – SEQ): Partially Supported.** Following Holm-Bonferroni sequential correction, mid-air gestures were rated as significantly easier exclusively for Task T_2 (Back Command, $p_{\text{adj}} = 0.0040$). Task T_1 (Home Command) exhibited a non-significant descriptive trend ($p_{\text{adj}} = 0.0546$), while spatial navigation tasks (T_3 and T_4) maintained equivalent perceived ease across modalities.

- **Hypothesis H_6 (Subscale-Specific Subjective Workload – NASA-TLX): Not Supported** in its specific subscale formulation regarding Mental and Physical Demand. Following Holm-Bonferroni adjustment, differences in Mental Demand ($p_{\text{adj}} = 0.1120$) and Physical Demand ($p_{\text{adj}} = 0.1260$) lost formal statistical significance. However, mid-air gesture interaction demonstrated a statistically robust advantage in significantly mitigating Temporal Demand ($p_{\text{adj}} = 0.0300$) and Frustration ($p_{\text{adj}} = 0.0395$).

6.8.4 Analysis of Edge Cases: Management of the Timeout Event

During the experimental trial phase, a single timeout event occurred during the third trial of Task T_3 (Opening Phone application) for User 4. Specifically, the participant required 132.0 s to complete the trial (accumulating 50.5 s of total off-road glance time), substantially exceeding the operational limit threshold of 90 s.

Based on qualitative observation and system log analysis, this anomaly was attributed to two main factors:

1. **Hand Posture Execution Difficulty:** The user experienced persistent difficulty in correctly forming the specific hand pose for the *index_right* gesture. Although the participant successfully completed the initial training phase, real-time trial execution required an additional brief re-training intervention by the investigator to correct physical posture execution.

2. **System Responsiveness Degradation:** The Android Automotive OS (AAOS) instance was hosted within the Android Studio emulator rather than running on a dedicated hardware target. During this specific trial, a system latency spike occurred in which input actions exhibited execution delays exceeding 2 seconds. This responsiveness lag forced the user to repeatedly divert visual attention to inspect the cursor position and execute multiple redundant gestures before receiving visual feedback from the UI.

In strict adherence to the primary experimental protocol, this non-completed trial was excluded from the baseline parametric comparisons presented in Sections 6.4.1 through 6.5. However, to explicitly address potential selection bias (where omitting a gesture failure could inadvertently favor the touchless modality), a formal sensitivity analysis was executed by re-including the timeout trial using its actual observed values of 132.0 s for TCT and 50.5 s for TOT.

Table 6.11: Comparison of Primary Protocol Analysis (excluding timeout) and Sensitivity Analysis (incorporating the 132.0 s observed timeout duration) for Task 3 (T_3).

Evaluation Baseline	**TCT Mean (SD)**	**Rep. 3 Mean**	**TOT Mean (SD)**	**Inferential Test (TCT vs Touch)**
Touchscreen Baseline	4.73 s (1.34)	4.87 s	2.57 s (1.37)	—
Primary Analysis (Excl. Timeout)	41.08 s (8.83)	53.07 s	15.76 s (7.82)	$t(14) = -15.60$ $p < 0.001$
Sensitivity Analysis (Incl. 132 s)	42.74 s (13.30)	58.33 s	16.75 s (8.95)	$t(14) = -11.22$ $p < 0.001$

As summarized in Table 6.11, re-integrating the timeout trial slightly elevates the overall mean execution time (+1.66 s) and visual distraction (+0.99 s) for Task 3, while increasing the average execution time of the third repetition (R_3) from 53.07 s to 58.33 s. Crucially, the inferential comparison between modalities maintains high statistical significance ($t(14) = -11.22, p < 0.001$). This confirms that the primary data filtering choices were robust, demonstrating that re-including the timeout event alters neither the direction nor the statistical significance of the comparison between interaction modalities.

Chapter 7

Conclusions and Future Work

7.1 Summary of Achieved Results

This thesis presented a unified, single-camera vision architecture designed to integrate active Driver Monitoring System and touchless gesture-based Human-Machine Interaction for automotive infotainment control. Through an integrated methodology combining computer vision engineering, embedded edge optimization, and a human-factors user evaluation ($N = 15$), the research directly addressed the three primary Research Questions (RQs) formulated in Chapter 1:

1. **Answer to RQ1 (Single-Camera Unified Feasibility):** The implementation demonstrated the technical feasibility of unifying active facial monitoring (DMS proof-of-concept for head pose, eye closure, and yawn heuristics) and touchless gesture recognition (HMI gesture vocabulary) using a single, centrally mounted wide-angle RGB-IR camera. By leveraging MediaPipe's top-down perception pipelines running asynchronously over a unified frame stream, the architecture demonstrates the potential to reduce sensor redundancy, spatial complexity, and hardware footprint compared to traditional multi-sensor automotive cabin setups.

2. **Answer to RQ2 (Visual and Usability):** The comparative user study conducted under static simulation conditions established that the effectiveness of mid-air gestures is strictly task-dependent:

    - *Open-Loop Secondary Shortcuts (T_1 Home and T_2 Back):* Touchless gestures significantly reduced Total Off-Road Time compared to capacitive touch ($TOT = 0.43$ s and 0.12 s vs. 1.40 s and 1.34 s) and maintaining

single glance durations well below the 2.0 s operational threshold for prolonged glances ($LGT \ll 2.0$ s). Psychometric evaluations confirmed that gesture shortcuts significantly reduced Temporal Demand ($p_{\text{adj}} = 0.0300$) and Frustration ($p_{\text{adj}} = 0.0395$).

- *Closed-Loop Spatial Navigation (T_3 App Selection and T_4 Climate Control):* Direct touchscreen contact resulted in significantly superior in completion time ($TCT = 3.88$ s vs. 25.62 s for gestures) and fewer spatial pointing errors (2.67 vs. 13.00 errors/user). Gesture control in spatial tasks forced continuous 3D cursor tracking verification, inducing off-road glances exceeding the 2.0 s operational threshold ($LGT > 2.0$ s).

Across full interaction blocks, psychometric evaluations via NASA-TLX confirmed that the gesture modality significantly reduced overall Temporal Demand ($p_{\text{adj}} = 0.0300$) and Frustration ($p_{\text{adj}} = 0.0395$). Overall system usability attained a System Usability Scale (SUS) score of 81.50/100

3. **Answer to RQ3 (Edge Real-Time Execution):** Benchmarking on the target NVIDIA Jetson Orin NX platform revealed that concurrent execution of the full perception pipeline (MediaPipe, ONNX Bi-LSTM, and YOLO26s) induced hardware-level power supply overcurrent throttling, capping execution at $\sim$ 20 FPS. Pipeline rationalization via module removal of the heavy YOLO26s object detection network, combined with MediaPipe GPU delegation and ONNX Runtime execution for the Bi-LSTM model, restored a stable processing rate of 30 FPS ($\Delta t = 33.3$ ms) under an empirical power draw of 11–15 W, delivering a post-recognition visual response latency of ≈ 66.6 ms (2 frames).

7.2 Study Limitations

During the research and development phase of this unified architecture, several pragmatic engineering trade-offs were implemented to accommodate hardware, temporal, and resource boundaries. Consequently, the empirical and technical findings of this thesis must be interpreted primarily within seven key limitations:

1. **Single-Subject Gesture Dataset and Generalization Limits:** The custom gesture dataset (2,160 training sequences and 216 validation sequences) was acquired exclusively by a single subject (the candidate) across two camera mounting configurations. Although perspective invariance was evaluated via GroupKFold cross-validation, the trained Bi-LSTM model lacks demonstrated generalization across diverse anthropometric characteristics, hand morphologies, and inter-subject kinematic execution styles, despite achieving high validation accuracy on the candidate's profile.

2. **Static Simulation Setup:** The experimental user evaluation ($N = 15$) was conducted in a stationary, static vehicle chassis without primary dynamic driving tasks (steering, braking, or road hazard tracking). As a result, experimental metrics quantify partecipants' visual demand, off-road glance behaviour, and perceived interaction safety under controlled laboratory conditions, rather than proven dynamic on-road driving safety.

3. **Head Pose Orientation as a Proxy for Visual Attention:** The DMS module estimates visual distraction using 3D head pose orientation angles (Yaw and Pitch) derived from MediaPipe Face Mesh, rather than direct 3D ocular gaze tracking. Minor head rotations remain within empirical safety thresholds and allow forward road visibility, whereas large head deflections do not allow for precise reconstruction of the gaze direction. However, head pose orientation cannot capture subtle eye-gaze shifts, failing to detect lateral eye deflections, occurring while the driver's head remains oriented forward.

4. **Empirical Calibration of DMS Detection Thresholds:** The operational thresholds for facial heuristics (Eye Closure $\tau_{\text{blink}} = 0.7$, Yawn combination, and Head Pose angular limits of 35° Yaw and 17° Pitch) were empirically tuned during prototype integration to minimize false alarms in the laboratory setup. These parameters serve as functional engineering heuristics rather than physiologically validated thresholds derived from standardized multi-subject fatigue protocols (e.g., PERCLOS or Karolinska Sleepiness Scale).

5. **Qualitative Nature of DMS Module Validation:** While the touchless HMI gesture pipeline was formally evaluated across 15 participants, the Driver Monitoring System (DMS) operated primarily as a qualitative proof-of-concept module. Its detection sensitivity across variable lighting regimes, extreme facial occlusions, and sunglass transmittances was not subjected to a systematic quantitative benchmark.

6. **Embedded Edge Bottlenecks and Pipeline Rationalization:** Concurrency benchmarks on the target NVIDIA Jetson Orin NX board revealed severe hardware-level resource limitations: running the complete multi-modal stack (MediaPipe, Bi-LSTM, and YOLO26s) triggered dynamic PMIC overcurrent throttling, reducing throughput to $\sim$ 20 FPS. To achieve a stable 30 FPS processing rate for primary safety functions, the architecture required pipeline rationalization via module removal of the heavy YOLO26s object detector, leaving onboard active safety focused exclusively on lightweight geometric monitoring.

7. **Interaction Friction in Spatial Gesture Mapping and Cursor Control:** Mapping mid-air gestures to progressive 3D cursor movement for complex

spatial tasks (T_3 and T_4) introduced continuous visual tracking demands, increased execution overhead, and user frustration. This highlights a structural limitation of spatial cursor pointing via touchless gestures, demonstrating that gesture-based interaction is ill-suited for fine-grained, iterative UI navigation without direct semantic action mapping.

7.3 Future Work

Building directly upon the methodological and technical boundaries identified in this study, future research could advance the unified vision architecture through targeted empirical, algorithmic, and hardware optimizations.

First, to address the generalizability constraints of the single-subject gesture dataset, the static simulation setup, and the qualitative nature of the DMS evaluation, research efforts should focus on expanding data acquisition and user testing across a diverse participant cohort. Capturing varied hand morphologies, age groups, and kinematic execution styles would allow gesture validation protocols to transition toward Leave-One-Subject-Out (*LOSO*) cross-validation schemes. Concurrently, the unified architecture could be integrated into a dynamic driving simulator equipped with primary vehicle control tasks, such as lane-keeping and emergency braking. Within this active driving setup, the Driver Monitoring System should undergo a systematic quantitative validation protocol, benchmarking detection sensitivity across variable lighting regimes, extreme facial occlusions, and sunglass transmittances against established physiological baselines, including PERCLOS and the Karolinska Sleepiness Scale (KSS) under realistic cognitive workload.

Second, to overcome the fidelity limitations of head pose orientation as a proxy for visual attention, future iterations of the Driver Monitoring System could integrate direct 3D ocular gaze tracking frameworks, such as lightweight *L2CS-Net* architectures adapted for edge execution. In this context, introducing *Pupil-Center Corneal Reflection* (PCCR) eye-tracking represents a potential hardware extension: since PCCR requires dedicated Near-Infrared (NIR) illuminator geometry and specialized optical filtering, it would partially modify the minimal single-camera setup, but could enable capturing subtle lateral eye deflections occurring while the head remains oriented forward.

Third, directly addressing the visual distraction and operational friction observed during 3D cursor spatial navigation, future HMI design may fundamentally re-architect touchless interaction. Having demonstrated the technical feasibility of the unified perception pipeline, the logical next evolution lies in mapping mid-air gestures directly to targeted global actions, such as track navigation, volume control, and HVAC adjustments. Rather than requiring continuous visual verification of an

on-screen cursor, these direct semantic shortcuts eliminate the need for users to visually locate and tap capacitive buttons while the vehicle is in motion, maximizing the eyes-free interaction benefits demonstrated in global system tasks.

Finally, to resolve the hardware bottlenecks and PMIC overcurrent throttling encountered on the target NVIDIA Jetson Orin NX board, future engineering efforts could focus on migrating the architecture to a higher-performance automotive edge platform, such as the *NVIDIA Jetson AGX Orin* or high-tier *Qualcomm Snapdragon Cockpit* series. Combined with deep model optimization techniques, including INT8 quantization and TensorRT acceleration for YOLO object detection variants, this hardware migration could reduce power supply constraints and computational overhead. This upgrade may enable the complete multimodal stack, encompassing active facial monitoring, touchless gesture recognition, and physical device handling detection, to execute concurrently at a stable rate of 30 FPS within embedded edge computing hardware.

Generative AI Disclosure

Generative AI tools were employed during the preparation of this thesis to assist with editorial refinement, technical script drafting, and structural organization. Specifically, AI assistance was utilized for:

- **Editorial and Linguistic Support:** Improving text phrasing, grammatical accuracy, readability, and translating draft sections from Italian to English.
- **Structural and Conceptual Organization:** Assisting in brainstorming, chapter structuring, and synthesizing literature review topics.
- **Scripting and Data Visualization:** Drafting auxiliary Python scripts for data processing, statistical analysis workflows, and chart plotting.

All underlying technical concepts, theoretical formulations, experimental protocols, and core domain analysis remain the original work of the author. All generated code, analytical workflows, and resulting figures were manually reviewed, executed, and validated against the raw experimental logs to ensure computational correctness and integrity. The author assumes full responsibility for the content and accuracy of the final work.

Appendix A

Group K-Fold Cross-Validation

src/groupkfold.py

```
import torch
import torch.nn as nn
import numpy as np
from torch.utils.data import Dataset, DataLoader
from models.GestureLSTM import GestureLSTM
import random
import itertools
import wandb
import sys

# ==========================================
# 0. SEED SETTING
# ==========================================
def set_seed(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False

set_seed(42)

# ── 1. DATA PREPARATION (CAMERA 1 VS CAMERA 2) ──
gestures = ['neutral', 'index_right', 'index_left', 'thumb_up', '
    index_down', 'fist', 'flip', 'pinch', 'shaka']
```

```
28
29 resume_combination=0
30 class GestureDataset(Dataset):
31     def __init__(self, X, y):
32         self.X = torch.tensor(X, dtype=torch.float32)
33         self.y = torch.tensor(y, dtype=torch.long)
34     def __len__(self):
35         return len(self.X)
36     def __getitem__(self, idx):
37         return self.X[idx], self.y[idx]
38
39 def load_camera_data(folder_path):
40     """Dynamically loads .npy files from a folder and generates X and
       y."""
41     X_list = []
42     y_list = []
43
44     for label, gesture in enumerate(gestures):
45         file_path = f'{folder_path}/{gesture}.npy'
46         try:
47             data = np.load(file_path)
48             X_list.append(data)
49             y_list.append(np.full(len(data), label))
50         except FileNotFoundError as e:
51             print(f"ERROR: Could not find '{file_path}'. Details: {e}
       ")
52             sys.exit()
53
54     X = np.concatenate(X_list, axis=0)
55     y = np.concatenate(y_list, axis=0)
56     return X, y
57
58 print("Loading Camera 1 dataset...")
59 X_cam1, y_cam1 = load_camera_data('data/1')
60
61 print("Loading Camera 2 dataset...")
62 X_cam2, y_cam2 = load_camera_data('data/2')
63
64 # ── 2. SINGLE FOLD TRAINING FUNCTION ──
65 def train_fold(X_t, y_t, X_v, y_v, hidden_size, batch_size, dropout,
       lr, group_name, run_name):
66     wandb.init(project="lstm-gkfold-bidirectional", group=group_name,
        name=run_name,
67                tags=[f"bs_{batch_size}", f"hs_{hidden_size}", f"lr_{
       lr}"],
68                config={"hidden_size": hidden_size, "batch_size":
       batch_size, "dropout": dropout, "lr": lr})
69
70     wandb.define_metric("val_loss", summary="min")
```

```
    wandb.define_metric("val_acc", summary="max")

    train_loader = DataLoader(GestureDataset(X_t, y_t), batch_size=batch_size, shuffle=True)
    val_loader = DataLoader(GestureDataset(X_v, y_v), batch_size=batch_size, shuffle=False)

    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    model = GestureLSTM(hidden_size=hidden_size, dropout=dropout).to(device)
    criterion = nn.CrossEntropyLoss()
    optimizer = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=1e-4)
    scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, mode='min', patience=5, factor=0.5)

    epochs = 100
    early_stopping_patience = 15
    epochs_no_improve = 0
    best_val_loss = float('inf')
    best_val_acc = 0.0

    for epoch in range(epochs):
        model.train()
        total_train_loss, correct_train, total_train = 0, 0, 0
        for inputs, labels in train_loader:
            inputs, labels = inputs.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(inputs)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            total_train_loss += loss.item()
            correct_train += (torch.max(outputs, 1)[1] == labels).sum().item()
            total_train += labels.size(0)

        train_loss = total_train_loss / len(train_loader)
        train_acc = 100 * correct_train / total_train

        model.eval()
        total_val_loss, correct_val, total_val = 0, 0, 0
        with torch.no_grad():
            for inputs, labels in val_loader:
                inputs, labels = inputs.to(device), labels.to(device)
                outputs = model(inputs)
                loss = criterion(outputs, labels)
                total_val_loss += loss.item()
```

```
                correct_val += (torch.max(outputs, 1)[1] == labels).sum().item()
                total_val += labels.size(0)

        val_loss = total_val_loss / len(val_loader)
        val_acc = 100 * correct_val / total_val
        scheduler.step(val_loss)

        if val_loss < best_val_loss:
            best_val_loss = val_loss
            best_val_acc = val_acc
            epochs_no_improve = 0
        else:
            epochs_no_improve += 1

        wandb.log({"epoch": epoch + 1, "train_loss": train_loss, "train_acc": train_acc,
                   "val_loss": val_loss, "val_acc": val_acc, "lr": optimizer.param_groups[0]['lr'],
                   "best_val_loss": best_val_loss})

        if epochs_no_improve >= early_stopping_patience:
            break

    wandb.finish()
    return best_val_loss, best_val_acc

# ── 3. EXPLICIT GRID SEARCH (CAM1 vs CAM2) ──
def grid_search():
    parameters = {
        'hidden_size': [32, 64, 128, 256],
        'batch_size': [2, 4, 8, 16, 32, 64],
        'dropout': [0.0, 0.1, 0.2, 0.3],
        'lr': [0.001]
    }
    keys, values = parameters.keys(), parameters.values()
    combinations = list(itertools.product(*values))

    best_overall_loss = float('inf')
    best_overall_acc = 0.0
    best_overall_params = None

    print(f"\nStarting CV Grid Search: {len(combinations)} combinations to test (Cam1 vs Cam2).\n")
    for i, combination in enumerate(combinations[resume_combination:], start=resume_combination):
        current_params = dict(zip(keys, combination))
        print(f"\n=== [{i+1}/{len(combinations)}] Testing parameters: {current_params} ===")
```

```
156
157         # 1. CREATE GROUP NAME (Common to both folds)
158         group_name = f"Comb_hs{current_params['hidden_size']}_bs{
    current_params['batch_size']}_dp{current_params['dropout']}_lr{
    current_params['lr']}"
159
160         # ---- FOLD 1: Train on Cam1, Validate on Cam2 ----
161         print(f"---- FOLD 1/2: Train Cam1 -> Validation Cam2 ----")
162         # 2. ADD DROPOUT TO RUN NAME
163         run_name_1 = f"f1_hs{current_params['hidden_size']}_bs{
    current_params['batch_size']}_dp{current_params['dropout']}_lr{
    current_params['lr']}"
164
165         val_loss_1, val_acc_1 = train_fold(X_cam1, y_cam1, X_cam2,
    y_cam2,
166                                            **current_params,
167                                            group_name=group_name,
168                                            run_name=run_name_1)
169         print(f"Fold 1 Result: Loss = {val_loss_1:.4f} | Acc = {
    val_acc_1:.2f}%")
170
171         # ---- FOLD 2: Train on Cam2, Validate on Cam1 ----
172         print(f"---- FOLD 2/2: Train Cam2 -> Validation Cam1 ----")
173         run_name_2 = f"f2_hs{current_params['hidden_size']}_bs{
    current_params['batch_size']}_dp{current_params['dropout']}_lr{
    current_params['lr']}"
174
175         val_loss_2, val_acc_2 = train_fold(X_cam2, y_cam2, X_cam1,
    y_cam1,
176                                            **current_params,
177                                            group_name=group_name,
178                                            run_name=run_name_2)
179         print(f"Fold 2 Result: Loss = {val_loss_2:.4f} | Acc = {
    val_acc_2:.2f}%")
180
181         # Average combination performance
182         avg_loss = (val_loss_1 + val_loss_2) / 2
183         avg_acc = (val_acc_1 + val_acc_2) / 2
184
185         # Calculate Discrepancy to evaluate stability
186         discrepancy = abs(val_loss_1 - val_loss_2)
187         print(f">>> AVERAGE: Loss = {avg_loss:.4f} | Acc = {avg_acc
    :.2f}% | DISCREPANCY = {discrepancy:.4f}\n")
188
189         # Check if it's the winning combination
190         if avg_loss < best_overall_loss:
191             best_overall_loss = avg_loss
192             best_overall_acc = avg_acc
193             best_overall_params = current_params
```

```
194
195     print("\n" + "="*50)
196     print("CROSS-VALIDATION GRID SEARCH COMPLETED")
197     print(f"Best Average Val Loss (Cam1 vs Cam2): {best_overall_loss
        :.6f}")
198     print(f"Associated Average Accuracy: {best_overall_acc:.2f}%")
199     print(f"Best Parameters Found: {best_overall_params}")
200     print("="*50)
201
202 if __name__ == "__main__":
203     grid_search()
```

Appendix B

Model Training

src/train.py

```python
import torch
import torch.nn as nn
import numpy as np
from torch.utils.data import Dataset, DataLoader
from models.GestureLSTM import GestureLSTM
import random
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.metrics import classification_report, confusion_matrix
import sys
import wandb

train = False

# ==========================================
# 0. SEED SETTING
# ==========================================
def set_seed(seed=42):
    random.seed(seed)
    np.random.seed(seed)
    torch.manual_seed(seed)
    if torch.cuda.is_available():
        torch.cuda.manual_seed(seed)
        torch.cuda.manual_seed_all(seed)
        torch.backends.cudnn.deterministic = True
        torch.backends.cudnn.benchmark = False

set_seed(42)

# ── 1. DATA PREPARATION ──
class GestureDataset(Dataset):
```

```
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.long)
    def __len__(self):
        return len(self.X)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

print("Loading dataset...")
path = 'data/train'
output_path = 'weights/hs128_bs16_dr0.2.pth'

gestures = ['neutral', 'index_right', 'index_left', 'thumb_up', '
    index_down', 'fist', 'flip', 'pinch', 'shaka']

def load_data_from_directory(path, gestures_list):
    X_list = []
    y_list = []
    try:
        for label, gesture in enumerate(gestures_list):
            data = np.load(f'{path}/{gesture}.npy')
            X_list.append(data)
            labels = np.full(len(data), label)
            y_list.append(labels)
    except FileNotFoundError as e:
        print(f"ERROR: Could not find the file for '{gesture}' in '{
    path}'. Details: {e}")
        sys.exit()

    X = np.concatenate(X_list, axis=0)
    y = np.concatenate(y_list, axis=0)
    return X, y

print(f"Loading TRAINING dataset [{path}]...")
X_train, y_train = load_data_from_directory(path, gestures)
print(f"TRAINING data shape X: {X_train.shape} | y: {y_train.shape}")

print("Loading VALIDATION dataset (data/val)...")
X_val, y_val = load_data_from_directory('data/val', gestures)
print(f"VALIDATION data shape X: {X_val.shape} | y: {y_val.shape}")

# --- 2. BEST MODEL CONFIGURATION ---
best_config = {
    "hidden_size": 128,
    "batch_size": 16,
    "dropout": 0.2,
    "lr": 0.001,
    "epochs": 100,
    "early_stopping_patience": 15
```

```
}

train_loader = DataLoader(GestureDataset(X_train, y_train),
    batch_size=best_config["batch_size"], shuffle=True)
val_loader = DataLoader(GestureDataset(X_val, y_val), batch_size=
    best_config["batch_size"], shuffle=False)

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")

# ── 3. LSTM MODEL DEFINITION ──
model = GestureLSTM(input_size=X_train.shape[2],
                    hidden_size=best_config["hidden_size"],
                    num_layers=2,
                    num_classes=len(gestures),
                    dropout=best_config["dropout"],
                    bidirectional=True).to(device)

criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=best_config["lr"
    ], weight_decay=1e-4)
scheduler = torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer,
    mode='min', patience=5, factor=0.5)

if train:
    # ═══════════════════════════════════════
    # 4. W&B INITIALIZATION
    # ═══════════════════════════════════════
    wandb.init(
        project="lstm-final-training",
        name=f"final_hs{best_config['hidden_size']}_bs{best_config['
    batch_size']}_dp{best_config['dropout']}_lr{best_config['lr']}",
        config=best_config
    )

    wandb.define_metric("val_loss", summary="min")
    wandb.define_metric("val_acc", summary="max")

    # ── 5. TRAINING LOOP ──
    best_val_loss = float('inf')
    epochs_no_improve = 0

    print(f"\nStarting training for {best_config['epochs']} epochs
    (90% Train, 10% Val)...")

    for epoch in range(best_config["epochs"]):
        model.train()
        total_train_loss = 0
        correct_train = 0
        total_train = 0
```

```
122
123         for inputs, labels in train_loader:
124             inputs, labels = inputs.to(device), labels.to(device)
125
126             optimizer.zero_grad()
127             outputs = model(inputs)
128             loss = criterion(outputs, labels)
129             loss.backward()
130             optimizer.step()
131
132             total_train_loss += loss.item()
133             _, preds = torch.max(outputs, 1)
134             total_train += labels.size(0)
135             correct_train += (preds == labels).sum().item()
136
137         train_acc = 100 * correct_train / total_train
138         avg_train_loss = total_train_loss / len(train_loader)
139
140         model.eval()
141         total_val_loss = 0
142         correct_val = 0
143         total_val = 0
144
145         with torch.no_grad():
146             for inputs, labels in val_loader:
147                 inputs, labels = inputs.to(device), labels.to(device)
148                 outputs = model(inputs)
149                 loss = criterion(outputs, labels)
150
151                 total_val_loss += loss.item()
152                 _, preds = torch.max(outputs, 1)
153                 total_val += labels.size(0)
154                 correct_val += (preds == labels).sum().item()
155
156         val_acc = 100 * correct_val / total_val
157         avg_val_loss = total_val_loss / len(val_loader)
158
159         scheduler.step(avg_val_loss)
160         current_lr = optimizer.param_groups[0]['lr']
161
162         if avg_val_loss < best_val_loss:
163             best_val_loss = avg_val_loss
164             epochs_no_improve = 0
165             torch.save(model.state_dict(), output_path)
166             improved_flag = f" >>> [*] Epoch {epoch+1:02d} | New best
    model saved!"
167         else:
168             epochs_no_improve += 1
```

```
169             improved_flag = f" —— [!] No improvement ({
    epochs_no_improve}/{best_config['early_stopping_patience']})"
170
171         print(f"Epoch {epoch+1:02d}/{best_config['epochs']} | "
172             f"Train Loss: {avg_train_loss:.4f} - Train Acc: {
    train_acc:.2f}% | "
173             f"Val Loss: {avg_val_loss:.4f} - Val Acc: {val_acc:.2f}%
    | LR: {current_lr}")
174         print(improved_flag)
175
176         wandb.log({
177             "epoch": epoch + 1,
178             "train_loss": avg_train_loss,
179             "train_acc": train_acc,
180             "val_loss": avg_val_loss,
181             "val_acc": val_acc,
182             "lr": current_lr,
183             "best_val_loss": best_val_loss
184         })
185
186         if epochs_no_improve >= best_config["early_stopping_patience"
    ]:
187             print(f"\n EARLY STOPPING triggered at epoch {epoch+1} (
    No improvement for {best_config['early_stopping_patience']} epochs
    ).")
188             break
189
190     wandb.finish()
191
192 # ==========================================
193 # 6. EVALUATION METRICS AND CONFUSION MATRIX
194 # ==========================================
195 print("\n" + "="*50)
196 print(" VALIDATION SET EVALUATION METRICS ")
197 print("="*50)
198
199 model.load_state_dict(torch.load(output_path))
200 model.eval()
201 model.to(device)
202
203 all_preds = []
204 all_labels = []
205
206 with torch.no_grad():
207     for inputs, labels in val_loader:
208         inputs, labels = inputs.to(device), labels.to(device)
209         outputs = model(inputs)
210
211         _, preds = torch.max(outputs, 1)
```

```
212
213             all_preds.extend(preds.cpu().numpy())
214             all_labels.extend(labels.cpu().numpy())
215
216 print("\n--- Classification Report ---")
217 report = classification_report(all_labels, all_preds, target_names=
        gestures, digits=4)
218 print(report)
219
220 print("\nGenerating Confusion Matrix...")
221 cm = confusion_matrix(all_labels, all_preds)
222
223 plt.figure(figsize=(10, 8))
224 sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
225                 xticklabels=gestures, yticklabels=gestures,
226                 annot_kws={"size": 12})
227
228 plt.title('Confusion Matrix', fontsize=16, fontweight='bold')
229 plt.xlabel('Predicted Label', fontsize=16)
230 plt.ylabel('True Label', fontsize=16)
231
232 plt.xticks(rotation=45, ha='right', fontsize=14)
233 plt.yticks(rotation=0, fontsize=14)
234
235 plt.tight_layout()
236
237 cm_path = "confusion_matrix.pdf"
238 plt.savefig(cm_path, format='pdf', bbox_inches='tight')
239 plt.show()
240
241 print(f"\nConfusion Matrix successfully saved at: {cm_path}")
```

Appendix C

Real-time inference on Edge Devices

src/unified_edge.py

```
import cv2
import time
import threading
import asyncio
import websockets
import numpy as np
import onnxruntime as ort
from collections import deque
import mediapipe as mp
from mediapipe.tasks import python
from mediapipe.tasks.python import vision
from ultralytics import YOLO

# ==========================================
# 1. GLOBAL CONFIGURATIONS AND THRESHOLDS
# ==========================================
DEBUG_UI = True

# --- Gestures (LSTM ONNX and State Machine) ---
TARGET_HAND = "Right"
CONFIDENCE_THRESHOLD = 85.0
COOLDOWN_TIME = 0.7
PREFILL_TIME = 0.2
SEQUENCE_LENGTH = 30
MISSING_FRAME_TOLERANCE = 10
ACTIVATION_TIME = 1.0

# --- DMS (Drowsiness and Distraction) ---
```

```
MAX_SLEEP_DISTRACTION_TIME = 1.5
MAX_YAWN_DISTRACTION_TIME = 1.0
MAX_NO_FACE_DISTRACTION_TIME = 2.0
EYES_CLOSED_THRESHOLD = 0.7
MOUTH_OPEN_THRESHOLD = 0.5
EYES_SQUINTED_THRESHOLD = 0.3
MAX_HEAD_DISTRACTION_TIME = 2.0
MAX_YAW_THRESHOLD = 35.0
PITCH_THRESHOLD_2 = -17.0
PITCH_THRESHOLD_1 = 17.0

# New logic imported from Windows
YAWN_COUNT_THRESHOLD = 5
YAWN_WINDOW_SEC = 900.0
MAX_PHONE_EAR_DISTANCE = 150.0

# --- YOLO (Smartphone) ---
MAX_PHONE_DISTRACTION_TIME = 1.0
PHONE_TOLERANCE_TIME = 0.5

# Original static crop area (will be combined with dynamic ROI)
x_start, y_start = 400, 200
crop_width, crop_height = 1120, 880
x_end, y_end = x_start + crop_width, y_start + crop_height

# ==========================================
# 2. WEBSOCKET SERVER (Background Thread)
# ==========================================
connected_clients = set()
server_loop = None

async def handle_car(websocket):
    print("\n[INFO] Client/Car connected to WebSocket!")
    connected_clients.add(websocket)
    try:
        async for message in websocket: pass
    except websockets.ConnectionClosed: pass
    finally:
        print("\n[WARNING] Client/Car disconnected.")
        if websocket in connected_clients:
            connected_clients.remove(websocket)

async def broadcast_command(command):
    if not connected_clients: return
    print(f"[NETWORK] Sending command: '{command}'")
    tasks = [asyncio.create_task(client.send(command)) for client in
    connected_clients]
    await asyncio.gather(*tasks, return_exceptions=True)

```

```
77 async def main_server():
78     print("\n=== WEBSOCKET SERVER STARTED ON PORT 9543 ===")
79     async with websockets.serve(handle_car, "0.0.0.0", 9543):
80         await asyncio.Future()
81 
82 def start_server_thread():
83     global server_loop
84     server_loop = asyncio.new_event_loop()
85     asyncio.set_event_loop(server_loop)
86     server_loop.run_until_complete(main_server())
87 
88 t_network = threading.Thread(target=start_server_thread, daemon=True)
89 t_network.start()
90 
91 # ==========================================
92 # 3. LSTM ONNX MODEL (Jetson Optimization)
93 # ==========================================
94 CLASSES = ["neutral", "thumb_right", "thumb_left", "thumb_up", "
       thumb_down", "fist", "flip", "pinch", "shaka"]
95 lstm_model_loaded = False
96 
97 try:
98     providers = ['CUDAExecutionProvider', 'CPUExecutionProvider']
99     ort_session = ort.InferenceSession('weights/
       modello_thumb_bi_lstm2.onnx', providers=providers)
100     input_name = ort_session.get_inputs()[0].name
101     lstm_model_loaded = True
102     print(f"[AI] ONNX Gesture Model loaded. Provider: {ort_session.
       get_providers()}")
103 except Exception as e:
104     print(f"[AI] ERROR: ONNX model loading failed ({e}).")
105 
106 # ==========================================
107 # 4. YOLO WORKER THREAD (Jetson Optimization)
108 # ==========================================
109 yolo_lock = threading.Lock()
110 frame_for_yolo = None
111 phone_boxes = []
112 yolo_active = True
113 
114 def yolo_worker():
115     global frame_for_yolo, phone_boxes, yolo_active
116     print("[AI] YOLO Thread Started (Loading TensorRT...)")
117     model_yolo = YOLO('models/yolo26s.engine', task="detect")
118 
119     while yolo_active:
120         local_frame = None
121         with yolo_lock:
122             if frame_for_yolo is not None:
```

```
123                 local_frame = frame_for_yolo.copy()
124
125         if local_frame is not None:
126             # 1. Retrieve current dynamic ROI dimensions
127             h_roi, w_roi = local_frame.shape[:2]
128
129             # 2. Calculate exact conversion factors
130             scale_x = w_roi / 560.0
131             scale_y = h_roi / 440.0
132
133             # Resize for TensorRT
134             frame_small = cv2.resize(local_frame, (560, 440),
    interpolation=cv2.INTER_LINEAR)
135             results = model_yolo.predict(frame_small, classes=[67],
    conf=0.7, verbose=False)
136
137             temp_boxes = []
138             if len(results[0].boxes) > 0:
139                 for box in results[0].boxes:
140                     x1, y1, x2, y2 = map(float, box.xyxy[0])
141
142                     # 3. Multiply by dynamically calculated scale
    factors (instead of fixed *2)
143                     temp_boxes.append([
144                         int(x1 * scale_x),
145                         int(y1 * scale_y),
146                         int(x2 * scale_x),
147                         int(y2 * scale_y),
148                         float(box.conf[0])
149                     ])
150
151             with yolo_lock:
152                 phone_boxes = temp_boxes
153
154         time.sleep(0.03)
155
156 t_yolo = threading.Thread(target=yolo_worker, daemon=True)
157 t_yolo.start()
158
159 # ==========================================
160 # 5. ASYNCHRONOUS MEDIAPIPE SETUP
161 # ==========================================
162 current_face_result = None
163 current_hand_result = None
164 is_hand_fresh = False
165
166 def update_face_results(result, output_image: mp.Image, timestamp_ms:
    int):
167     global current_face_result
```

```
168     current_face_result = result
169
170 def update_hand_results(result, output_image: mp.Image, timestamp_ms:
        int):
171     global current_hand_result, is_hand_fresh
172     current_hand_result = result
173     is_hand_fresh = True
174
175 face_options = vision.FaceLandmarkerOptions(
176     base_options=python.BaseOptions(model_asset_path='models/
        face_landmarker.task', delegate=python.BaseOptions.Delegate.GPU),
177     running_mode=vision.RunningMode.LIVE_STREAM,
178     result_callback=update_face_results,
179     output_face_blendshapes=True,
180     output_facial_transformation_matrixes=True,
181     num_faces=1
182 )
183
184 hand_options = vision.HandLandmarkerOptions(
185     base_options=python.BaseOptions(model_asset_path='models/
        hand_landmarker.task', delegate=python.BaseOptions.Delegate.GPU),
186     num_hands=2,
187     min_hand_detection_confidence=0.5,
188     running_mode=vision.RunningMode.LIVE_STREAM,
189     result_callback=update_hand_results
190 )
191
192 # ==========================================
193 # 6. MAIN LOOP
194 # ==========================================
195 cap = cv2.VideoCapture(0, cv2.CAP_V4L2)
196 cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
197 cap.set(cv2.CAP_PROP_FRAME_WIDTH, 1920)
198 cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 1080)
199
200 sending_enabled = False
201 shaka_start_time = None
202 frames_buffer = deque(maxlen=SEQUENCE_LENGTH)
203 missing_frames_count = 0
204 last_known_positions = None
205 last_action_time = 0.0
206 has_flushed_in_cooldown = True
207 current_gesture, confidence = "neutral", 0.0
208
209 no_face_start, eyes_closed_start, yawn_start, head_distraction_start
        = None, None, None, None
210 phone_distraction_start, last_phone_sighting = None, None
211 last_alert_time = 0.0
212 calibrated = False
```

```
213 baseline_yaw, baseline_pitch = 0.0, 0.0
214
215 # Yawn Variables
216 yawn_count = 0
217 yawn_history = []
218 yawn_already_counted = False
219 yawn_state = False
220
221 last_timestamp_ms = 0
222 fps_timer, fps_counter, real_fps = time.time(), 0, 0.0
223
224 # Setup Trackbars (New feature)
225 if DEBUG_UI:
226     cv2.namedWindow("Unified Jetson System")
227     def on_trackbar(val): pass
228     cv2.createTrackbar("ROI X", "Unified Jetson System", 0,
    crop_width - 100, on_trackbar)
229     cv2.createTrackbar("ROI Y", "Unified Jetson System", 0,
    crop_height - 100, on_trackbar)
230     cv2.createTrackbar("ROI W", "Unified Jetson System", 600,
    crop_width, on_trackbar)
231     cv2.createTrackbar("ROI H", "Unified Jetson System", 880,
    crop_height, on_trackbar)
232
233 print("\n--- UNIFIED SYSTEM READY AND OPERATIONAL ---")
234
235 with vision.FaceLandmarker.create_from_options(face_options) as
    face_detector, \
236      vision.HandLandmarker.create_from_options(hand_options) as
    hand_detector:
237
238     while True:
239         success, full_frame = cap.read()
240         if not success: break
241
242         frame = full_frame[y_start:y_end, x_start:x_end]
243         h_img, w_img, _ = frame.shape
244         current_time = time.time()
245
246         # ------------------------------------------------------
247         # DYNAMIC ROI VIA SLIDERS
248         # ------------------------------------------------------
249         roi_x, roi_y, roi_w, roi_h = 0, 0, w_img, h_img
250         if DEBUG_UI:
251             roi_x = cv2.getTrackbarPos("ROI X", "Unified Jetson
    System")
252             roi_y = cv2.getTrackbarPos("ROI Y", "Unified Jetson
    System")
```

```
253             roi_w = cv2.getTrackbarPos("ROI W", "Unified Jetson
    System")
254             roi_h = cv2.getTrackbarPos("ROI H", "Unified Jetson
    System")
255
256             roi_w = max(100, min(roi_w, w_img - roi_x))
257             roi_h = max(100, min(roi_h, h_img - roi_y))
258
259             cv2.rectangle(frame, (roi_x, roi_y), (roi_x + roi_w,
    roi_y + roi_h), (255, 0, 255), 2)
260             cv2.putText(frame, "AI VISION AREA", (roi_x + 5, roi_y +
    20), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 255), 2)
261
262         frame_roi = frame[roi_y:roi_y+roi_h, roi_x:roi_x+roi_w]
263
264         # Share ROI frame for YOLO Thread
265         with yolo_lock:
266             frame_for_yolo = frame_roi
267
268         fps_counter += 1
269         if current_time - fps_timer >= 1.0:
270             real_fps = fps_counter / (current_time - fps_timer)
271             fps_counter = 0; fps_timer = current_time
272
273         active_alarms = []
274
275         # ==========================================
276         # MEDIAPIPE INFERENCE (on ROI)
277         # ==========================================
278         rgb_frame = cv2.cvtColor(frame_roi, cv2.COLOR_BGR2RGB)
279         mp_image = mp.Image(image_format=mp.ImageFormat.SRGB, data=
    rgb_frame)
280         timestamp_ms = int(current_time * 1000)
281         if timestamp_ms <= last_timestamp_ms: timestamp_ms =
    last_timestamp_ms + 1
282         last_timestamp_ms = timestamp_ms
283
284         face_detector.detect_async(mp_image, timestamp_ms)
285         hand_detector.detect_async(mp_image, timestamp_ms)
286
287         local_face = current_face_result
288         local_hand = current_hand_result
289
290         # ==========================================
291         # BLOCK 1: YOLO PROCESSING AND CALL LOGIC
292         # ==========================================
293         phone_detected_now = False
294         phone_at_ear = False
295         nose_pos, right_ear_pos, left_ear_pos = None, None, None
```

```
296         current_boxes = []
297 
298         # Retrieve scaled face positions
299         if local_face and local_face.face_landmarks:
300             lms = local_face.face_landmarks[0]
301             nose_pos = (int(lms[1].x * roi_w) + roi_x, int(lms[1].y *
     roi_h) + roi_y)
302             right_ear_pos = (int(lms[234].x * roi_w) + roi_x, int(lms
    [234].y * roi_h) + roi_y)
303             left_ear_pos = (int(lms[454].x * roi_w) + roi_x, int(lms
    [454].y * roi_h) + roi_y)
304 
305         with yolo_lock:
306             current_boxes = phone_boxes.copy()
307 
308         for box in current_boxes:
309             phone_detected_now = True
310             x1, y1, x2, y2, conf = box
311 
312             # Adapt coordinates to ROI
313             x1 = int(x1 + roi_x); y1 = int(y1 + roi_y); x2 = int(x2 +
     roi_x); y2 = int(y2 + roi_y)
314 
315             phone_cx, phone_cy = (x1 + x2) // 2, (y1 + y2) // 2
316             if nose_pos and right_ear_pos and left_ear_pos:
317                 dist_right_ear = ((phone_cx - right_ear_pos[0])**2 +
    (phone_cy - right_ear_pos[1])**2) ** 0.5
318                 dist_left_ear = ((phone_cx - left_ear_pos[0])**2 + (
    phone_cy - left_ear_pos[1])**2) ** 0.5
319                 if min(dist_right_ear, dist_left_ear) <
    MAX_PHONE_EAR_DISTANCE:
320                     phone_at_ear = True
321 
322             if DEBUG_UI:
323                 cv2.rectangle(frame, (x1, y1), (x2, y2), (255, 0, 0),
     2)
324                 label = f"In call {conf:.2f}" if phone_at_ear else f"
    Phone {conf:.2f}"
325                 cv2.putText(frame, label, (x1, max(20, y1 - 10)), cv2
    .FONT_HERSHEY_SIMPLEX, 0.6, (255, 0, 0), 2)
326 
327         if phone_detected_now:
328             last_phone_sighting = current_time
329             if phone_distraction_start is None:
    phone_distraction_start = current_time
330         else:
331             if last_phone_sighting is not None and (current_time -
    last_phone_sighting) > PHONE_TOLERANCE_TIME:
332                 phone_distraction_start = None
```

```
333                 last_phone_sighting = None
334
335             if phone_distraction_start is not None and (current_time -
    phone_distraction_start) > MAX_PHONE_DISTRACTION_TIME:
336                 if phone_at_ear:
337                     active_alarms.append(("ALARM: CALL IN PROGRESS!", (0,
     0, 255)))
338                     cmd = "TALKING_PHONE"
339                 else:
340                     active_alarms.append(("ALARM: PHONE!", (0, 165, 255))
    )
341                     cmd = "PHONE_SEEN"
342
343                 if (current_time - last_alert_time) > 3.0 and server_loop
     and server_loop.is_running():
344                     asyncio.run_coroutine_threadsafe(broadcast_command(
    cmd), server_loop)
345                     last_alert_time = current_time
346
347             # ==========================================
348             # BLOCK 2: DMS PROCESSING (FACE, YAWN HISTORY)
349             # ==========================================
350             rel_pitch, rel_yaw = 0.0, 0.0
351             head_out_of_range = False
352             blink_right, squint_right, mouth_open = 0.0, 0.0, 0.0
353
354             if local_face is None or not local_face.face_blendshapes:
355                 if no_face_start is None: no_face_start = current_time
356                 eyes_closed_start, yawn_start, head_distraction_start =
    None, None, None
357                 if (current_time - no_face_start) >
    MAX_NO_FACE_DISTRACTION_TIME:
358                     active_alarms.append(("ALERT_NO_FACE", (0, 0, 255)))
359                     if (current_time - last_alert_time) > 3.0 and
    server_loop and server_loop.is_running():
360                         asyncio.run_coroutine_threadsafe(
    broadcast_command("ALERT_FACE_LOST"), server_loop)
361                         last_alert_time = current_time
362             else:
363                 no_face_start = None
364                 blendshapes = {b.category_name: b.score for b in
    local_face.face_blendshapes[0]}
365                 blink_right = blendshapes.get('eyeBlinkRight', 0)
366                 squint_right = blendshapes.get('eyeSquintRight', 0)
367                 mouth_open = blendshapes.get('jawOpen', 0)
368
369                 are_eyes_closed = blink_right > EYES_CLOSED_THRESHOLD
370                 are_eyes_squinted = squint_right >
    EYES_SQUINTED_THRESHOLD
```

```
371             is_yawn = (mouth_open > MOUTH_OPEN_THRESHOLD) and (
    are_eyes_closed or are_eyes_squinted)
372
373             if are_eyes_closed and not is_yawn:
374                 if eyes_closed_start is None: eyes_closed_start =
    current_time
375                 if (current_time - eyes_closed_start) >
    MAX_SLEEP_DISTRACTION_TIME:
376                     active_alarms.append(("ALERT_SLEEP", (0, 0, 255))
    )
377                     if (current_time - last_alert_time) > 3.0 and
    server_loop and server_loop.is_running():
378                         asyncio.run_coroutine_threadsafe(
    broadcast_command("SLEEP_ALERT"), server_loop)
379                         last_alert_time = current_time
380             else: eyes_closed_start = None
381
382             # New Yawn History Logic
383             if is_yawn:
384                 if not yawn_state or yawn_start is None:
385                     yawn_state, yawn_start = True, current_time
386
387                 if not yawn_already_counted and (current_time -
    yawn_start) > MAX_YAWN_DISTRACTION_TIME:
388                     yawn_history.append(current_time)
389                     yawn_already_counted = True
390             else:
391                 yawn_state, yawn_already_counted, yawn_start = False,
     False, None
392
393             yawn_history = [t for t in yawn_history if (current_time
    - t) <= YAWN_WINDOW_SEC]
394
395             if len(yawn_history) >= YAWN_COUNT_THRESHOLD:
396                 active_alarms.append((f"ALARM: SEVERE FATIGUE!", (0,
    165, 255)))
397                 if (current_time - last_alert_time) > 3.0 and
    server_loop and server_loop.is_running():
398                     asyncio.run_coroutine_threadsafe(
    broadcast_command("SLEEP_ALERT"), server_loop)
399                     last_alert_time, yawn_history = current_time, []
400
401             if local_face.facial_transformation_matrixes:
402                 matrix = local_face.facial_transformation_matrixes
    [0][:3, :3]
403                 angles, _, _, _, _, _ = cv2.RQDecomp3x3(matrix)
404                 pitch, yaw = angles[0], angles[1]
405                 if calibrated:
```

```
406                     rel_pitch, rel_yaw = pitch - baseline_pitch, yaw
    - baseline_yaw
407                     if abs(rel_yaw) > MAX_YAW_THRESHOLD or rel_pitch
    > PITCH_THRESHOLD_1 or rel_pitch < PITCH_THRESHOLD_2:
408                         head_out_of_range = True
409
410             if head_out_of_range and calibrated:
411                 if head_distraction_start is None:
    head_distraction_start = current_time
412                 if (current_time - head_distraction_start) >
    MAX_HEAD_DISTRACTION_TIME:
413                     active_alarms.append(("ALERT_DISTRACTION", (0, 0,
     255)))
414                     if (current_time - last_alert_time) > 3.0 and
    server_loop and server_loop.is_running():
415                         asyncio.run_coroutine_threadsafe(
    broadcast_command("ALERT_ON"), server_loop)
416                         last_alert_time = current_time
417             else: head_distraction_start = None
418
419         # ==========================================
420         # BLOCK 3: GESTURE PROCESSING (SYNCHRONOUS LSTM ONNX)
421         # ==========================================
422         time_since_action = current_time - last_action_time
423         in_cooldown = time_since_action < COOLDOWN_TIME
424
425         if in_cooldown:
426             time_remaining = COOLDOWN_TIME - time_since_action
427             if time_remaining <= PREFILL_TIME and not
    has_flushed_in_cooldown:
428                 frames_buffer.clear()
429                 last_known_positions = None
430                 missing_frames_count = 0
431                 has_flushed_in_cooldown = True
432
433         target_hand_landmarks = None
434         if local_hand and local_hand.hand_landmarks:
435             for idx, hl in enumerate(local_hand.hand_landmarks):
436                 if idx < len(local_hand.handedness) and len(
    local_hand.handedness[idx]) > 0:
437                     hand_label = local_hand.handedness[idx][0].
    category_name
438                 else:
439                     hand_label = "Unknown"
440
441                 if hand_label == TARGET_HAND:
442                     target_hand_landmarks = hl
443
444         if is_hand_fresh:
```

```
445             is_hand_fresh = False
446             if target_hand_landmarks:
447                 current_positions = []
448                 for lm in target_hand_landmarks:
449                     # Relative coordinate adaptation for ONNX
450                     x_adj = (lm.x * roi_w + roi_x) / 1120.0
451                     y_adj = (lm.y * roi_h + roi_y) / 880.0
452                     current_positions.extend([x_adj, y_adj, lm.z])
453                     if DEBUG_UI: cv2.circle(frame, (int(x_adj * w_img
    ), int(y_adj * h_img)), 4, (0, 255, 0), -1)
454 
455                 frames_buffer.append(current_positions)
456                 last_known_positions = current_positions
457                 missing_frames_count = 0
458             else:
459                 if last_known_positions is not None and
    missing_frames_count < MISSING_FRAME_TOLERANCE:
460                     frames_buffer.append(last_known_positions)
461                     missing_frames_count += 1
462                 else:
463                     frames_buffer.clear()
464                     last_known_positions = None
465                     missing_frames_count = 0
466                     current_gesture, confidence = "Hand Missing", 0.0
467 
468             if len(frames_buffer) == SEQUENCE_LENGTH and not
    in_cooldown and lstm_model_loaded:
469                 seq_array = np.array(frames_buffer, dtype=np.float32)
    .reshape(SEQUENCE_LENGTH, 21, 3)
470                 anchor_wrist = seq_array[0, 0, :].copy()
471                 seq_normalized = seq_array - anchor_wrist
472 
473                 max_val = np.max(np.abs(seq_normalized))
474                 if max_val > 0: seq_normalized /= max_val
475 
476                 input_array = seq_normalized.reshape(1,
    SEQUENCE_LENGTH, 63).astype(np.float32)
477                 ort_outs = ort_session.run(None, {input_name:
    input_array})
478                 logits = ort_outs[0]
479 
480                 exp_logits = np.exp(logits - np.max(logits, axis=1,
    keepdims=True))
481                 probs = exp_logits / np.sum(exp_logits, axis=1,
    keepdims=True)
482 
483                 predicted_idx = np.argmax(probs, axis=1)[0]
484                 confidence = float(probs[0][predicted_idx] * 100)
485                 current_gesture = CLASSES[predicted_idx]
```

```
486
487                 if current_gesture == "shaka" and confidence >
    CONFIDENCE_THRESHOLD:
488                     if shaka_start_time is None:
489                         shaka_start_time = current_time
490                     elif current_time - shaka_start_time >=
    ACTIVATION_TIME:
491                         sending_enabled = not sending_enabled
492                         if not sending_enabled and server_loop and
    server_loop.is_running():
493                             asyncio.run_coroutine_threadsafe(
    broadcast_command("focus"), server_loop)
494
495                         shaka_start_time = None
496                         frames_buffer.clear()
497                         last_action_time = current_time
498                         has_flushed_in_cooldown = False
499                 else:
500                     shaka_start_time = None
501
502                 if sending_enabled and current_gesture not in ["
    neutral", "shaka"] and confidence > CONFIDENCE_THRESHOLD:
503                     cmd_map = {"thumb_right":"rot_r", "thumb_left":"
    rot_l", "thumb_up":"up",
504                                "thumb_down":"down", "fist":"home", "
    pinch":"click", "flip":"back"}
505                     command = cmd_map.get(current_gesture)
506
507                     if command and server_loop and server_loop.
    is_running():
508                         asyncio.run_coroutine_threadsafe(
    broadcast_command(command), server_loop)
509
510                     last_action_time = current_time
511                     has_flushed_in_cooldown = False
512
513         # ============================================
514         # BLOCK 4: GRAPHICS AND SCREEN INTERFACE
515         # ============================================
516         if DEBUG_UI:
517             cv2.rectangle(frame, (5, 5), (320, 250), (0, 0, 0), -1)
518             status_color = (0, 255, 0) if sending_enabled else (0, 0,
     255)
519             status_text = "COMMANDS: ACTIVE" if sending_enabled else
    "COMMANDS: DISABLED"
520             cv2.putText(frame, status_text, (10, 30), cv2.
    FONT_HERSHEY_SIMPLEX, 0.7, status_color, 2)
521
522             offset_y_info = 60
```

```
523
524             if shaka_start_time is not None:
525                 elapsed_time = current_time - shaka_start_time
526                 bar = min(int((elapsed_time / ACTIVATION_TIME) * 100)
    , 100)
527                 cv2.putText(frame, f"Unlock: {elapsed_time:.1f}s",
    (10, offset_y_info), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 255),
     2)
528                 cv2.rectangle(frame, (10, offset_y_info + 10), (110,
    offset_y_info + 20), (50, 50, 50), -1)
529                 cv2.rectangle(frame, (10, offset_y_info + 10), (10 +
    bar, offset_y_info + 20), (0, 255, 255), -1)
530                 offset_y_info += 50
531
532             cv2.putText(frame, f"FPS: {real_fps:.1f}", (10,
    offset_y_info), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 255), 2)
533             cv2.putText(frame, f"Yawns: {len(yawn_history)}/{
    YAWN_COUNT_THRESHOLD}", (10, offset_y_info + 30), cv2.
    FONT_HERSHEY_SIMPLEX, 0.6, (0, 255, 255), 2)
534
535             if not calibrated:
536                 cv2.putText(frame, "PRESS 'C' TO CALIBRATE", (10,
    offset_y_info + 70), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (0, 165, 255),
     2)
537             else:
538                 cv2.putText(frame, f"Yaw: {rel_yaw:.1f} | Pitch: {
    rel_pitch:.1f}", (10, offset_y_info + 70), cv2.
    FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
539                 cv2.putText(frame, f"Head: {'OUT OF RANGE' if
    head_out_of_range else 'OK'}", (10, offset_y_info + 100), cv2.
    FONT_HERSHEY_SIMPLEX, 0.6, (0, 0, 255) if head_out_of_range else
    (0, 255, 0), 2)
540
541             if in_cooldown:
542                 cv2.putText(frame, "COOLDOWN", (w_img-250, 40), cv2.
    FONT_HERSHEY_SIMPLEX, 0.7, (0, 0, 255), 2)
543                 cv2.putText(frame, f"{current_gesture}", (w_img-250,
    70), cv2.FONT_HERSHEY_SIMPLEX, 0.7, (0, 255, 0), 2)
544             elif missing_frames_count > 0:
545                 cv2.putText(frame, f"BLUR ({missing_frames_count}/{
    MISSING_FRAME_TOLERANCE})", (w_img-250, 40), cv2.
    FONT_HERSHEY_SIMPLEX, 0.7, (0, 165, 255), 2)
546             else:
547                 col = (0, 255, 0) if current_gesture not in ["neutral
    ", "Hand Missing"] else (255, 255, 255)
548                 cv2.putText(frame, f"Gesture: {current_gesture}", (
    w_img-250, 40), cv2.FONT_HERSHEY_SIMPLEX, 0.7, col, 2)
549                 cv2.putText(frame, f"Conf: {confidence:.1f}%", (w_img
    -250, 70), cv2.FONT_HERSHEY_SIMPLEX, 0.6, col, 1)
```

```
550
551             if active_alarms:
552                 start_y = int(h_img / 2) - (len(active_alarms) * 20)
553                 for i, (alarm_text, color) in enumerate(active_alarms
    ):
554                     (text_width, text_height), _ = cv2.getTextSize(
    alarm_text, cv2.FONT_HERSHEY_DUPLEX, 1.2, 3)
555                     offset_y = start_y + (i * (text_height + 25))
556                     cv2.rectangle(frame, (40, offset_y - text_height
    - 10), (60 + text_width, offset_y + 10), (0, 0, 0), -1)
557                     cv2.putText(frame, alarm_text, (50, offset_y),
    cv2.FONT_HERSHEY_DUPLEX, 1.2, color, 3)
558
559             frame_resized = cv2.resize(frame, (0, 0), fx=0.4, fy=0.4)
560             cv2.imshow("Unified Jetson System", frame_resized)
561
562         key = cv2.waitKey(1) & 0xFF
563         if key == 27 or key == ord('q'):
564             yolo_active = False
565             break
566         elif key == ord('c') or key == ord('C'):
567             if local_face and local_face.
    facial_transformation_matrixes:
568                 baseline_yaw, baseline_pitch, calibrated = yaw, pitch
    , True
569
570 cap.release()
571 t_yolo.join()
572 cv2.destroyAllWindows()
```

Bibliography

[1] Yanchao Dong, Zhencheng Hu, Keiichi Uchimura, and Nobuki Murayama. «Driver Inattention Monitoring System for Intelligent Vehicles: A Review». In: *IEEE Transactions on Intelligent Transportation Systems* 12.2 (2011), pp. 596–614. DOI: 10.1109/TITS.2010.2092770. URL: https://ieeexplore.ieee.org/document/5665773 (cit. on p. 1).

[2] Kenneth Majlund Ba h, Mads Gregers Jæger, Mikael B. Skov, and Nils Gram Thomassen. «You can touch, but you can't look: interacting with in-vehicle systems». In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '08. Association for Computing Machinery, 2008, pp. 1139–1148. DOI: 10.1145/1357054.1357233 (cit. on pp. 1–3).

[3] William J. Horrey. «Assessing the Effects of In-Vehicle Tasks on Driving Performance». In: *Ergonomics in Design* 19.4 (2011), pp. 4–7. DOI: 10.1177/1064804611419961 (cit. on p. 2).

[4] National Highway Traffic Safety Administration. *Visual-Manual NHTSA Driver Distraction Guidelines for In-Vehicle Electronic Devices*. Tech. rep. DOT HS 811 800. U.S. Department of Transportation, 2012. URL: https://www.nhtsa.gov/sites/nhtsa.gov/files/distraction_npfg-02162012.pdf (visited on 09/17/2026) (cit. on p. 2).

[5] Thomas A. Dingus, Feng Guo, Suzie Lee, Jonathan F. Antin, Miguel Perez, Mindy Buchanan-King, and Jonathan Hankey. «Driver crash risk factors and prevalence evaluation using naturalistic driving data». In: *Proceedings of the National Academy of Sciences* 113.10 (2016), pp. 2636–2641. DOI: 10.1073/pnas.1513271113. URL: https://api.semanticscholar.org/CorpusID:6366588 (cit. on p. 2).

[6] Matthew J. Pitts, Gary Burnett, Lee Skrypchuk, Tom Wellings, Alex Attridge, and Mark A. Williams. «Visual–haptic feedback interaction in automotive touchscreens». In: *Displays* 33.1 (2012), pp. 7–16. ISSN: 0141-9382. DOI: 10.1016/j.displa.2011.09.002. URL: https://www.sciencedirect.com/science/article/pii/S0141938211000874 (cit. on p. 3).

[7] S. Klauer, F. Guo, J. Sudweeks, and T. Dingus. *An Analysis of Driver Inattention Using a Case-Crossover Approach on 100-Car Data: Final Report.* Tech. rep. DOT HS 811 334. Washington, DC, USA: National Highway Traffic Safety Administration, 2010. URL: https://www.researchgate.net/publication/329070655_An_analysis_of_driver_inattention_using_a_case-crossover_approach_on_100-car_data (visited on 09/17/2026) (cit. on pp. 3, 9).

[8] Eshed Ohn-Bar and Mohan Manubhai Trivedi. «Hand Gesture Recognition in Real Time for Automotive Interfaces: A Multimodal Vision-Based Approach and Evaluations». In: *IEEE Transactions on Intelligent Transportation Systems* 15.6 (2014), pp. 2368–2377. DOI: 10.1109/TITS.2014.2337331. URL: https://ieeexplore.ieee.org/document/6883176 (cit. on pp. 3, 13, 14).

[9] Nico Zengeler, Thomas Kopinski, and Uwe Handmann. «Hand Gesture Recognition in Automotive Human–Machine Interaction Using Depth Cameras». In: *Sensors* 19.1 (Dec. 2018), p. 59. DOI: 10.3390/s19010059 (cit. on p. 3).

[10] Fabio Manganaro, Stefano Pini, Guido Borghi, Roberto Vezzani, and Rita Cucchiara. «Hand Gestures for the Human-Car Interaction: The Briareo Dataset». In: Springer, Sept. 2019, pp. 560–571. ISBN: 978-3-030-30644-1. DOI: 10.1007/978-3-030-30645-8_51. URL: https://www.researchgate.net/publication/335603660_Hand_Gestures_for_the_Human-Car_Interaction_The_Briareo_Dataset (cit. on pp. 3, 14, 31, 32).

[11] *Road vehicles — Measurement and analysis of driver visual behaviour with respect to transport information and control systems.* Geneva, Switzerland: International Organization for Standardization, 2020 (cit. on p. 8).

[12] European Parliament and Council of the European Union. *Regulation (EU) 2019/2144 on general safety of motor vehicles (GSR II).* 2019. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32019R2144 (visited on 09/17/2026) (cit. on p. 9).

[13] European Commission. *Commission Delegated Regulation (EU) 2023/2590 of 13 July 2023 supplementing Regulation (EU) 2019/2144 of the European Parliament and of the Council by laying down detailed rules concerning the specific test procedures and technical requirements for the type-approval of certain motor vehicles with regard to their advanced driver distraction warning systems and amending that Regulation.* Official Journal of the European Union, L 2023/2590. 2023. URL: https://eur-lex.europa.eu/legal-content/EN/TXT/?uri=CELEX:32023R2590 (visited on 09/17/2026) (cit. on p. 9).

[14] Euro NCAP. *Euro NCAP Assessment Protocol – Safety Assist: Safe Driving*. Version 10.4. 2024. URL: https://www.euroncap.com/ (visited on 09/17/2026) (cit. on p. 9).

[15] Tereza Soukupová and Jan Čech. «Real-time eye blink detection using facial landmarks». In: *21st Computer Vision Winter Workshop (CVWW)*. 2016, pp. 1–8. URL: https://api.semanticscholar.org/CorpusID:21124316 (cit. on p. 10).

[16] Baris Tuna and Murat Kaplan. «Intelligent Monitoring System: Facial Features Based Driver Drowsiness and Distraction Detection». In: *2025 16th International Conference on Electrical and Electronics Engineering (ELECO)*. 2025, pp. 1–5. DOI: 10.1109/ELECO69582.2025.11329338. URL: https://ieeexplore.ieee.org/abstract/document/11329338 (cit. on p. 10).

[17] Walter W Wierwille, LA Ellsworth, SS Wreggit, RJ Fairbanks, and CL Kim. *Research on vehicle-based driver status performance monitoring*. Tech. rep. DOT HS 808 247. National Highway Traffic Safety Administration, 1994. URL: https://rosap.ntl.bts.gov/view/dot/2578/dot_2578_DS1.pdf (cit. on p. 11).

[18] David F Dinges, MM Mallis, G Maislin, and JW Powell. «Evaluation of techniques for ocular measurement as an index of fatigue and as the basis for alertness management». In: *NHTSA Technical Report* DOT HS 808 762 (1998). URL: https://rosap.ntl.bts.gov/view/dot/2518 (cit. on p. 11).

[19] T Åkerstedt and M Gillberg. «Subjective and objective sleepiness in the active individual». In: *The International journal of neuroscience* 52.1-2 (May 1990), pp. 29–37. ISSN: 0020-7454. DOI: 10.3109/00207459008994241. URL: https://api.semanticscholar.org/CorpusID:23980645 (cit. on p. 11).

[20] Dan Witzner Hansen and Qiang Ji. «In the eye of the beholder: A survey of models for eyes and gaze». In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 32.3 (2010), pp. 478–500. DOI: 10.1109/TPAMI.2009.30. URL: https://ieeexplore.ieee.org/document/4770110 (cit. on p. 11).

[21] Petr Kellnhofer, Adria Recasens, Simon Stent, Wojciech Matusik, and Antonio Torralba. «Gaze360: Physically unconstrained gaze estimation in the wild». In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. 2019, pp. 6912–6921. DOI: 10.1109/ICCV.2019.00701. URL: https://arxiv.org/abs/1910.10088 (cit. on p. 12).

[22] Ahmed A. Abdelrahman, Thorsten Hempel, Aly Khalifa, and Ayoub Al-Hamadi. «L2CS-Net: Fine-Grained Gaze Estimation in Unconstrained Environments». In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*. 2022, pp. 1280–1296. DOI:

10.48550/arXiv.2203.03339. URL: https://arxiv.org/abs/2203.03339 (cit. on p. 12).

[23] Juan Diego Ortega, Neslihan Kose, Paola Cañas, Min-An Chao, Alexander Unnervik, Marcos Nieto, Oihana Otaegui, and Luis Salgado. «DMD: A large-scale multi-modal driver monitoring dataset for attention and alertness analysis». In: *European Conference on Computer Vision (ECCV) Workshops*. 2020, pp. 387–405. DOI: 10.1007/978-3-030-66823-5_23 (cit. on p. 12).

[24] Pavlo Molchanov, Xiaodong Yang, Shalini Gupta, Kihwan Kim, Stephen Tyree, and Jan Kautz. «Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks». In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 4207–4215. DOI: 10.1109/CVPR.2016.456. URL: https://ieeexplore.ieee.org/document/7780825 (cit. on p. 14).

[25] Camillo Lugaresi et al. «MediaPipe: A Framework for Building Perception Pipelines». In: *arXiv preprint arXiv:1906.08172* (2019). URL: https://arxiv.org/abs/1906.08172 (visited on 09/17/2026) (cit. on pp. 15, 16).

[26] Hsin-Hua Li and Chen-Chiung Hsieh. «Dynamic Hand Gesture Recognition Using MediaPipe and Transformer». In: *Engineering Proceedings (MDPI)* 108.1 (2025), p. 22. DOI: 10.3390/engproc2025108022. URL: https://www.mdpi.com/2673-4591/108/1/22 (cit. on p. 15).

[27] Smart Eye AB. *Smart Eye Launches New AIS+ System at CES 2025: Introducing Key Functionality to Its Fleet Driver Safety System*. Press Release, Gothenburg, Sweden. 2025. URL: https://smarteye.se/news/smart-eye-launches-new-ais-system-at-ces-2025-introducing-key-functionality-to-its-fleet-driver-safety-system/ (visited on 09/17/2026) (cit. on p. 16).

[28] Dongwoo Kang and Lin Ma. «Real-Time Eye Tracking for Bare and Sunglasses-Wearing Faces for Augmented Reality 3D Head-Up Displays». In: *IEEE Access* 9 (2021), pp. 125508–125522. DOI: 10.1109/ACCESS.2021.3110644. URL: https://ieeexplore.ieee.org/document/9530401 (cit. on p. 16).

[29] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. «MediaPipe Hands: On-device real-time hand tracking». In: *arXiv preprint arXiv:2006.10214* (2020). DOI: 10.48550/arXiv.2006.10214. URL: https://arxiv.org/abs/2006.10214 (cit. on pp. 16, 17).

[30] Sepp Hochreiter and Jürgen Schmidhuber. «Long Short-Term Memory». In: *Neural Computation* 9.8 (1997), pp. 1735–1780. DOI: 10.1162/neco.1997.9.8.1735. URL: https://ieeexplore.ieee.org/abstract/document/6795963 (cit. on p. 19).

[31] Ultralytics. *Long Short-Term Memory (LSTM)*. Jan. 2026. URL: https://www.ultralytics.com/it/glossary/long-short-term-memory-lstm (visited on 09/17/2026) (cit. on p. 19).

[32] Glenn Jocher, Jing Qiu, Mengyu Liu, Shuai Lyu, Fatih Cagatay Akyon, and Muhammet Esat Kalfaoglu. *Ultralytics YOLO26: Unified Real-Time End-to-End Vision Models*. 2026. DOI: 10.48550/arXiv.2606.03748. arXiv: 2606.03748 [cs.CV]. URL: https://arxiv.org/abs/2606.03748 (cit. on pp. 20, 21).

[33] Reply S.p.A. *Reply Smart Mobility Lab – Area42*. 2026. URL: https://www.reply.com/en/labs/smart-mobility-lab (visited on 09/21/2026) (cit. on p. 22).

[34] *IEC 62471:2006 Photobiological safety of lamps and lamp systems*. International Electrotechnical Commission, 2006 (cit. on p. 25).

[35] Robert B. Miller. «Response time in man-computer conversational transactions». In: *Proceedings of the December 9-11, 1968, Fall Joint Computer Conference, Part I*. New York, NY, USA: Association for Computing Machinery, 1968, pp. 267–277. ISBN: 9781450378994. DOI: 10.1145/1476589.1476628 (cit. on p. 30).

[36] S. S. Shapiro and M. B. Wilk. «An Analysis of Variance Test for Normality (Complete Samples)». In: *Biometrika* 52.3/4 (1965), pp. 591–611. ISSN: 00063444, 14643510. DOI: 10.2307/2333709. URL: http://www.jstor.org/stable/2333709 (visited on 09/10/2026) (cit. on p. 63).

[37] Jeff Sauro and James R. Lewis. «Chapter 8 - Standardized Usability Questionnaires». In: *Quantifying the User Experience*. Boston: Morgan Kaufmann, 2012, pp. 185–240. ISBN: 978-0-12-384968-7. DOI: 10.1016/B978-0-12-384968-7.00008-4. URL: https://www.sciencedirect.com/science/article/pii/B9780123849687000084 (cit. on p. 103).