UNIVERSITY OF PISA

MASTER'S DEGREE IN COMPUTER ENGINEERING

910II Industrial Application

AutoDoctor

Professors:

Pierfrancesco Foglia

Antonio Cosimo Prete

Students:

Tommaso Califano

Nicola Ramacciotti

Gabriele Suma

ACADEMIC YEAR 2024/2025

Abstract

In this study, we present a prototype system for rapid health assessment of vehicle occupants following traffic accidents, leveraging multimodal sensor data and AI-based inference. The system integrates cardiac, auditory, and visual inputs through an embedded Raspberry Pi platform, enabling a simplified Glasgow Coma Scale (GCS) estimation and vital sign monitoring to support emergency responders. Facial analysis has been consistently performed locally to ensure real-time responsiveness, while a hybrid processing strategy was evaluated for speech recognition, comparing local and remote inference. The impact of on-device speech inference on video sampling performance was specifically investigated, revealing trade-offs between model complexity and system responsiveness. Remote models demonstrated superior responsiveness and minimal interference with concurrent tasks, while local inference ensured operability in offline scenarios. The ultimate objective is to evolve the system into an autonomous, scalable tool for in-vehicle health monitoring, enhancing prehospital triage and contributing to next-generation automotive safety solutions.

Contents

Chapter 1

Introduction

In the critical moments following a vehicular accident, the timely and accurate assessment of passenger health can be the difference between life and death. Immediate knowledge of a passenger's physiological and neurological state enables emergency responders to prioritize interventions and allocate resources more effectively. Traditional emergency protocols often rely on manual assessment by first responders, which may be delayed or limited by environmental constraints. In response to this challenge, our project introduces a comprehensive in-vehicle health monitoring system designed to autonomously evaluate the condition of passengers in the aftermath of a crash.

Leveraging recent advances in sensor technology and artificial intelligence, the system integrates multiple modalities: cardiac sensors for vital sign monitoring, a camera and microphone for audio-visual assessment, and an audio output component to interact with passengers via vocal prompts. Among its core functionalities is the automatic initiation of a simplified Glasgow Coma Scale (GCS) assessment through voice interaction, enabling a preliminary estimation of neurological function. These signals are locally processed using embedded systems optimized for edge computing, with the possibility of cloud-based inference to enhance model performance when connectivity allows.

By combining multimodal physiological and behavioral data, the system aims to generate a rapid, objective, and data-driven health status report immediately after a collision. This information is formatted and transmitted to emergency services, facilitating faster and more informed medical response. The approach addresses a critical gap in post-accident care by automating the initial triage process, thus maximizing the window of opportunity for life-saving interventions. Through the integration of robust sensing technologies and hybrid AI processing pipelines, the proposed system represents a step forward in the evolution of intelligent, safety-focused automotive environments.

However implementing this system involves several key challenges, including limited computational resources for real-time multimodal data processing, sensor inaccuracies due to motion artifacts, and the need for non-invasive, unobtrusive data acquisition during driving. Environmental variability—such as lighting and cabin acoustics—adds complexity to visual and audio analysis, while reliance on remote AI services raises concerns about latency and data privacy. Addressing these issues is crucial for developing a reliable, practical solution for in-vehicle health monitoring.

1.1 Multimodal Health Monitoring

Passenger health status is a multifaceted condition encompassing physiological signals, behavioral responses, and cognitive indicators. In the aftermath of a traumatic event, accurately identifying and classifying this state becomes critical for initiating appropriate medical intervention. Just as psychological research has long sought to define and categorize complex internal experiences, our study aims to structure and interpret measurable signs of distress or unconsciousness using a multimodal approach.

1.1.1 Glasgow Coma Scale

The Glasgow Coma Scale (GCS) [1] is a widely adopted clinical scale designed to assess the level of consciousness in individuals who have suffered acute brain injuries, such as those resulting from trauma, stroke, or other neurological events. Developed in 1974 by Graham Teasdale and Bryan Jennett, the GCS offers a simple, standardized method to evaluate and communicate a patient's neurological status in emergency and intensive care settings.

The scale is based on the evaluation of three independent responses:

- **Eye Opening Response (E)** – scored from 1 to 4:
  1. *No response*: Eyes do not open, even to pain.
  2. *To pain*: Eyes open only in response to painful stimuli.
  3. *To speech*: Eyes open in response to verbal command.
  4. *Spontaneous*: Eyes open on their own, without stimulation.
- **Verbal Response (V)** – scored from 1 to 5:
  1. *No response*: No verbal output.
  2. *Incomprehensible sounds*: Moaning but no words.
  3. *Inappropriate words*: Random or exclamatory speech, no conversational exchange.
  4. *Confused*: Speech is coherent but the person is disoriented.
  5. *Oriented*: The person responds coherently and appropriately (knows who they are, where they are, and the time).
- **Motor Response (M)** – scored from 1 to 6:
  1. *No response*: No movement, even with painful stimulation.
  2. *Abnormal extension (decerebrate posture)*: Arms and legs extended with internal rotation.
  3. *Abnormal flexion (decorticate posture)*: Arms flexed, legs extended in response to pain.
  4. *Withdraws from pain*: Pulls away from painful stimulus.

5. *Localizes pain*: Purposeful movement to remove painful stimuli.
6. *Obeys commands*: Follows simple commands, such as moving a limb.

The total GCS score is the sum of the scores from the three categories, ranging from 3 (deep coma or death) to 15 (fully alert and oriented). In clinical practice, the GCS provides a fast and repeatable way to:

- Monitor changes in a patient's neurological status over time.
- Inform decisions about treatment urgency and resource allocation.
- Communicate the patient's condition between healthcare providers.

In the context of this study, the GCS serves as the reference standard for assessing consciousness. The system developed aims to automate the elicitation and evaluation of GCS-like responses using a combination of sensors and interactive prompts, supporting emergency services with timely and objective assessments in the aftermath of vehicular accidents.

1.1.2 Vital Sign Monitoring

Although the Glasgow Coma Scale provides a robust measure of neurological function, it offers only a partial view of the general physiological condition of a passenger. To obtain a comprehensive picture, especially in the critical moments after a traumatic incident, it is essential to complement the evaluation of consciousness with continuous monitoring of vital signs, especially respiratory and cardiac activity. These parameters offer direct insight into the integrity of life-sustaining systems and often signal deterioration earlier than behavioral symptoms alone.

Cardiac activity is among the most immediate and informative indicator of a person's physiological state, especially in the aftermath of a traumatic event. Cardiac monitoring provides essential insight into the body's capacity to maintain perfusion and oxygenation of vital organs. Parameters such as heart rate, heart rate variability (HRV), and pulse waveform quality offer real-time feedback on cardiovascular function. Tachycardia or bradycardia may suggest internal bleeding, hypovolemia, or neurogenic shock, while HRV reflects autonomic nervous system activity and can indicate physiological stress or unconsciousness. Abnormal pulse characteristics may signal impaired cardiac output, further raising clinical concern. These metrics can be captured unobtrusively using photoplethysmography (PPG) sensors embedded in seats or steering wheels or piezoelectric elements in contact with the body.

By integrating respiratory and cardiac monitoring with the Glasgow Coma Scale, an established measure of consciousness based on verbal, motor, and eye-opening responses, the system can provide a much more comprehensive assessment of a passenger's condition. While the GCS focuses on observable behavior and responsiveness, vital sign monitoring continuously tracks autonomic functions, even in patients who are nonverbal or unresponsive. This multimodal approach not only enhances early detection of critical deterioration but also supports injury severity assessment and prearrival triage. Ultimately, the synergy between behavioral evaluation and physiological data equips emergency responders with structured, actionable information,

helping them to prioritize care more effectively and improve survival outcomes in time-sensitive scenarios.

1.1.3 Challenges in Multimodal Health Monitoring

While the integration of physiological and behavioral data offers substantial potential for post-accident assessment, implementing such a system in real-world automotive environments presents several technical, ethical, and practical challenges:

- *Sensor Accuracy and Placement*: Developing a multimodal system that ensures high precision without compromising passenger comfort is inherently complex. Sensors must capture reliable data even during sudden impacts or irregular body positioning, all while remaining unobtrusive and seamlessly integrated into the vehicle interior (e.g., seatbelts, seats, or dashboards).

- *Non-Invasiveness*: The system must function passively, without requiring any active participation or wearing of external devices by passengers. This constraint limits the range of sensors that can be used and imposes strict requirements on data quality and robustness under varying conditions.

- *Computational Limitations*: The need to process large volumes of physiological, audio, and video data in real time often exceeds the capabilities of standard embedded systems. Balancing model accuracy, responsiveness, and hardware constraints, particularly in emergency situations where delay is unacceptable, poses a major design challenge.

- *Environmental Interference*: Factors such as ambient noise, lighting variability, and motion artifacts introduced during a crash can degrade data quality. Ensuring the system maintains reliability in uncontrolled or degraded conditions is critical for deployment in real vehicles.

- *Hardware Robustness*: All sensing and processing equipment must be resilient to mechanical shocks, vibrations, and impact forces typical of vehicular accidents. Components must continue to function reliably even under structural stress, making durability a key design consideration.

- *Privacy and Data Security*: Health-related data, especially when involving biometric and behavioral indicators, is highly sensitive. The system must comply with strict privacy regulations (e.g., GDPR) and ensure secure data handling, encryption, and minimal remote transmission, especially when AI models are executed in the cloud.

- *Standardization of Medical Inference*: Translating raw sensor data into medically actionable information requires collaboration with clinical experts and adherence to health monitoring standards. Ensuring the system's outputs are interpretable and trustworthy for first responders is essential for real-world adoption.

By overcoming these challenges, the system aims to provide a reliable and ethical solution for real-time health monitoring, improving the speed and effectiveness of emergency response after accidents.

1.2 Scope of the Study

This study aims to design and prototype a comprehensive system capable of assessing a passenger's post-accident health status through multimodal sensing and intelligent processing. The overarching goal is to enable a fast, accurate, and privacy-conscious triage process that can assist emergency responders and improve survival outcomes. To achieve this, the research focuses on the following core objectives:

- *Design of a Complete Monitoring Framework:* The study proposes the creation of an integrated system that combines cardiac, audio, and visual signals to infer a passenger's physiological and neurological condition. This includes both hardware and software components working together in a cohesive and fault-tolerant architecture suitable for deployment in real vehicles.

- *Identification and Evaluation of AI Models:* A key focus is the selection and evaluation of artificial intelligence models capable of analyzing voice and visual data to autonomously perform assessments such as the Glasgow Coma Scale. These models must demonstrate high accuracy while being robust to noisy inputs and real-world conditions such as variable lighting, sound interference, and passenger positioning.

- *Adaptation for Embedded Devices:* The system is designed to run partially on low-power edge devices, with a particular focus on the Raspberry Pi 3 as a reference platform. This involves adapting inference pipelines and optimizing models to operate within constrained hardware environments. To ensure real-time responsiveness while maintaining performance, the system adopts a hybrid computation strategy, dynamically balancing workloads between the embedded device and a remote cloud backend. This architecture enables immediate local processing of critical data (e.g., vital signs), while deferring more intensive tasks (e.g., deep video analysis) to remote resources when necessary.

- *Development of a Dual-Component Application:* A dual-layer application architecture is implemented, comprising a local frontend and a cloud-connected backend. The frontend offers a unified interface that summarizes vital signs and assessment results in real time, providing a dashboard view suitable for first responders or in-vehicle monitoring. The backend handles data storage and system coordination.

- *Prototyping and Real-World Feasibility:* The study culminates in the development of a functional prototype that addresses several of the practical challenges discussed earlier. This prototype serves as a proof of concept, demonstrating the feasibility of deploying the system in a realistic automotive scenario and paving the way for future iterations and field testing.

By pursuing these objectives, the project establishes a foundation for a scalable and effective post-accident health monitoring solution, with the potential to significantly reduce emergency response times and improve patient outcomes.

Chapter 2

State of the Art

Recent advancements in automotive safety have extended far beyond traditional crashworthiness, increasingly focusing on post-crash monitoring and passenger condition assessment. Multiple initiatives from industry and research highlight a growing interest in multimodal in-cabin sensing systems, which combine vision, audio, and physiological data analysis to assess the real-time health status of occupants. This section reviews key developments in the field, drawing from regulatory frameworks, industrial implementations, and academic research.

The European New Car Assessment Program (Euro NCAP)[2] outlines a long-term vision in its document 'Vision 2030', focusing on the role of internal sensors capable of transmitting live physiological data, such as heart rate and breathing, from within the vehicle's cabin. This data could be critical for evaluating the consciousness level of occupants after a collision, potentially revolutionizing the response chain in post-crash scenarios by providing paramedics with actionable insights even before arrival.

From an industrial perspective, Philips has developed a vital signs camera specifically for automotive environments [3], designed to offer high accuracy even under challenging conditions such as motion artifacts or varying illumination. This camera is capable of tracking heart rate and respiration fluctuations in real time, thereby supporting early anomaly detection. Similarly, Continental Engineering Services has introduced Advanced Cabin Sensing Solutions [4], which encompass vital sign monitoring, thermal comfort analysis, and driver state tracking. These systems focus on non-contact, passive sensing to preserve comfort and avoid distraction.

A more comprehensive approach is demonstrated by the In-Cabin Monitoring System (IMS)[5] for Autonomous Vehicles, which integrates AI-enabled onboard devices (OBDs) with multiple cameras to support safety, security, and health monitoring. Notably, the system prioritizes user privacy through edge-based AI processing and the development of a specialized dataset reflecting multifaceted in-cabin scenarios. It also provides a comparative analysis of AI models used for occupant recognition, highlighting the need for tailored solutions in high-level autonomous mobility contexts.

Research efforts in health-related AI further demonstrate the feasibility and potential of multimodal monitoring. One study introduced a convolutional neural network (CNN) architecture for facial expression recognition (FER) to improve elderly safety monitoring [6]. This system incorporated temporal smoothing tech-

niques (e.g., moving average) to address the shortcomings of still image-based FER, achieving high performance on real-life datasets. The proposed Digital Healthcare (DHc) framework, implemented on a robotic platform, underlines the relevance of FER for health status estimation, particularly in care-related contexts.

More broadly, the relevance of in-vehicle healthcare monitoring systems (IVHS) has been emphasized in recent user-centered studies [7]. As daily commuting times increase, passengers express a growing interest in systems that can monitor their health unobtrusively during travel. Through co-design workshops with multidisciplinary participants, seven thematic areas were identified for IVHS development, ranging from real-time physiological data collection to communication and actuation. This led to the proposal of a conceptual framework that organizes these functionalities and provides future research directions, underscoring the importance of designing IVHS with user expectations and system feasibility in mind.

In summary, the current state of the art reveals a converging trend toward intelligent, multimodal, and privacy-conscious health monitoring systems in automotive environments. While regulatory bodies like Euro NCAP set the strategic direction, industrial stakeholders contribute robust sensor technologies, and academic research continues to refine the machine learning models and interaction paradigms necessary for effective real-world deployment. This convergence lays a solid foundation for the development of systems capable of evaluating passenger health in real time, ultimately enhancing automated emergency response and vehicle occupant safety.

Chapter 3

System Description

To support health monitoring objectives, an initial assessment was conducted to determine the most suitable set of sensors capable of fulfilling the system's functional requirements within the project scope. The ideal configuration included a heart rate sensor, a camera, a microphone, and a microcontroller specifically, a Raspberry Pi, to enable multimodal data acquisition. This configuration was selected to facilitate the estimation of a simplified Glasgow Coma Score (GCS) through a combination of visual, auditory, and physiological cues. The system architecture is illustrated in Figure 3.1.

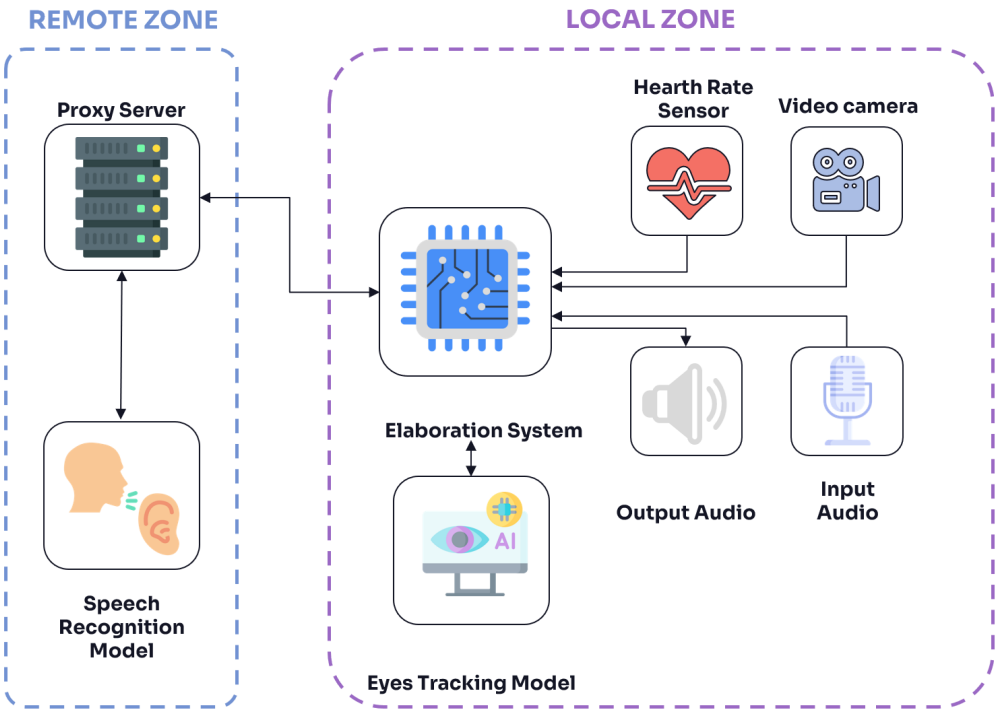


Figure 3.1: System architecture scheme.

- **Remote Zone**: Encompasses the Whisper AI model, which is hosted remotely due to its high computational requirements.

- **Local Zone**: Contains the embedded components installed within the vehicle, including the heart rate sensor, microphone, camera, audio output, and the Raspberry Pi microcontroller.

Throughout the development of the prototype, the system architecture was iteratively refined in response to hardware limitations and compatibility considerations.

The eye-tracking functionality was realized using AI models based on OpenCV [8] and Dlib [9], enabling facial recognition and image analysis directly within the vehicle. These models were selected for their computational efficiency, making local deployment feasible.

The heart rate sensing module was also implemented locally, leveraging its low resource consumption and ease of integration.

Conversely, the voice recognition component, powered by the Whisper model [10], was unsuitable for local execution due to its significant processing demands. To address this constraint, a remote zone was established. A proxy server was deployed to manage communication between the local and remote environments, allowing for remote inference of speech data while maintaining system responsiveness and modularity.

Chapter 4

Prototype

This chapter provides a detailed description of the hardware and software components that make up the developed prototype[1]. It aims to offer a comprehensive overview of the physical components used, their main technical specifications, and the software environment that ensures their correct operation and integration. The system is built on a low-power embedded platform and leverages processing tools designed for edge-device compatibility.

4.1 Hardware Description

This section presents the hardware setup used in the prototype. It describes the technical and functional characteristics of the main devices involved, including the *Raspberry Pi 3 Model B+*, the *Raspberry Pi Camera Module V2*, and the heart rate monitoring system composed of the *Polar T34 Transmitter* and *Receiver Module*. The analysis focuses on processing, communication, and interfacing capabilities, highlighting their relevance for embedded and biometric monitoring applications.

4.1.1 Raspberry Pi 3 model B+

The Raspberry Pi 3 Model B+ incorporates a Broadcom BCM2837B0 system on chip, a 64-bit quad-core ARM Cortex-A53 processor clocked at 1.4 GHz. RAM capacity stands at 1 GB of LPDDR2 SDRAM. This model maintains the same form factor as earlier variations of the Pi 2 and Pi 3, ensuring compatibility with existing accessories.

Connectivity & I/O

Equipped with dual-band Wi-Fi (2.4 GHz/5 GHz 802.11b/g/n/ac) and Bluetooth 4.2/BLE, the board supports both wireless and wired communication. Its GEM-like Gigabit Ethernet over USB 2.0 interface achieves data throughput of approximately 300Mb/s. Further connectivity options include:

- Four USB 2.0 ports

[1]The complete Python code developed for this study is publicly available at the following GitHub repository: `https://github.com/Dimerin/AutoDoctor`

- 40-pin GPIO header (compatible with standard HATs)
- Full-size HDMI (1.3a)
- DSI and CSI ribbon connectors
- 3.5mm analog audio/composite jack
- microSD slot for OS and storage

Performance and Power

The VideoCore IV GPU provides hardware acceleration for H.264 video up to 1080p30 and OpenGLES 1.1/2.0 graphics. Power is supplied via a 5 V/2.5 A micro-USB input or GPIO header; the board supports PoE with an optional HAT. It operates reliably between 0–50°C, with a mean time between failure (MTBF) of approximately 378,000 hours.

4.1.2 Raspberry Pi Camera Module V2

This camera features a Sony IMX219 sensor with 8 MP resolution, capturing still images up to 3280 × 2464 pixels. It supports video recording formats: 1080p at 30 fps, 720p at 60 fps, and 640 × 480 at 90 fps. The sensor uses a fixed-focus lens and includes software support via Raspberry Pi OS, including integration with the picamera library.

Form Factor & Interface

The compact camera board measures approximately 25 x 23 × 9 mm and weighs just over 3 g. Connection is via a CSI ribbon cable to the Pi, with mounting compatible with Pi Compute Module and Zero (using appropriate cables). Power, control, and image data are all transmitted through the CSI interface.

Performance Characteristics

The IMX219 sensor employs 1.12 μm pixels with OmniBSI technology, improving sensitivity and noise reduction. Built-in automatic controls include color balance, exposure, and luminance stabilization. The module captures high-definition images suitable for real-time video and computer-vision applications.

4.1.3 Polar T34 Heart Rate Transmitter

The Polar T34 is a chest-worn electrocardiogram (ECG) sensor that detects cardiac impulses using skin-contact electrodes. It transmits each heartbeat wirelessly using a 5 kHz analog signal burst. The transmitter is non-coded, meaning it does not use any unique pairing identifiers—making it compatible with all standard Polar receivers in range.

Physical and Electrical Characteristics

The device is integrated into a flexible, washable chest strap and includes a small, non removable electronics module. It is powered by a standard 3 V coin-cell battery (CR2032), with an expected life of up to 2500 hours of use. It is water-resistant up to 30 meters.

Transmission Behavior

The T34 emits a short radio burst each time it detects a heartbeat. Because the signal is not encrypted or coded, any compatible receiver within approximately 1–1.2 meters can detect the transmission. This facilitates easy interfacing but limits its use in environments where multiple sensors are in use simultaneously.

4.1.4 Polar Heart Rate Receiver

The receiver module detects the 5 kHz signal bursts transmitted by Polar chest straps (e.g., the T34) and outputs a logic-level digital pulse for each heartbeat. It simplifies heart rate monitoring by converting analog RF signals into a digital form directly usable by microcontrollers.

Electrical Interface and Link to the Raspberry Pi

The module is typically powered by 3.3 V to 5.5 V DC and features a 3-pin header interface:

- **VCC** – Supply voltage (3.3–5.5 V)
- **GND** – Ground
- **SIG** (Signal) – Digital output (HIGH on heartbeat)

Each heartbeat is represented as a high-level pulse approximately 15 ms wide. This output is compatible with standard digital I/O or interrupt lines on microcontrollers.

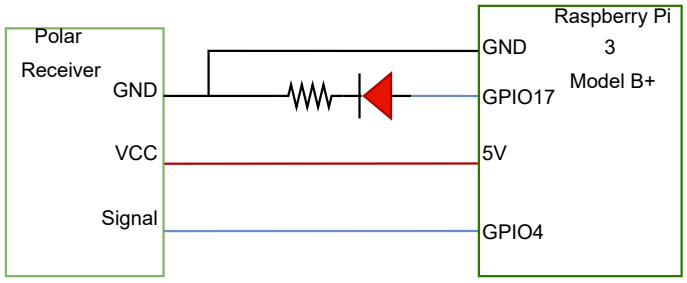


Figure 4.1: System Schematic

As depicted in Figure 4.1, the receiver SIG is connected to GPIO4 on the Raspberry Pi and powered by the 5V supply. The ground (GND) is also connected to

the Raspberry Pi. Additionally, there is an LED connected in series with a resistor to GPIO17 and the Raspberry Pi ground.

Detection Range and Behavior

The receiver has a typical effective range of up to 1.2 meters. Its built-in RF coil is tuned specifically for the Polar transmission frequency. It supports both coded and non-coded Polar transmitters, although interference may occur in the presence of multiple non-coded units.

4.1.5 MINI Microphone

The MI-305 Mini Microphone is an ultra-compact USB microphone dongle, designed for voice-centric applications such as podcasts, spoken-word recordings, audiobooks, and basic audio input on embedded systems. Its plug-and-play nature and USB 2.0 interface make it compatible with a wide range of devices, including PCs, laptops, and single-board computers (SBCs) such as the Raspberry Pi 3 B+. No additional drivers or software are required.

The microphone is powered directly via the USB interface and provides immediate functionality upon connection. For improved positioning or proximity to the sound source, a USB extension cable or swivel adapter may be employed. It can also operate in tandem with USB speakers for simultaneous recording and playback setups.

- **Model:** MI-305
- **Interface:** USB 2.0 (digital audio)
- **Power:** USB-powered, no external power supply required
- **Compatibility:** Plug-and-play with Windows, macOS, and Linux (including Raspberry Pi OS)
- **Sensitivity:** –67 dBV/pBar, –47 dBV/Pa ±4 dB
- **Frequency Response:** 100 Hz – 16 kHz
- **Noise Cancellation:** Built-in technology to suppress ambient background noise
- **Dimensions:** 22 mm × 19 mm × 7 mm

4.1.6 Asus Zenbook

This device was used as a benchmark platform to compare computational performance between edge and remote inference scenarios. Specifically, it served to evaluate the efficiency and latency of local execution on a high-performance laptop versus remote processing, in contrast with the resource-constrained Raspberry Pi setup.

- **Model:** ASUSTeK COMPUTER INC. ZenBook Pro 15 UX550GE_UX550GE

- **Processor:** Intel® Core™ i7-8750H × 12
- **RAM:** 16.0 GiB
- **Internal Graphics:** Intel® UHD Graphics 630 (CFL GT2)
- **Dedicated GPU:** NVIDIA GeForce GTX 1050 Ti
- **Disk Capacity:** 512.1 GB

4.2 Software Description

This section describes the software environment configured to support the prototype's functionalities. After installing the operating system on the Raspberry Pi platform, the necessary tools for data processing and speech recognition were set up. Details about the Python environment setup and performance optimizations implemented to ensure real-time capabilities are also provided.

4.2.1 Operating Systems

The Raspberry Pi used for deployment was equipped with a 64 GB SanDisk microSD card (UHS-I, up to *20* MB/s). The official *Raspberry Pi Imager* was used to flash the following operating system:

- **OS Version:** Raspberry Pi OS with desktop
- **Release Date:** November 19, 2024
- **System Architecture:** 64-bit
- **Kernel Version:** 6.6
- **Debian Base:** Version 12 (Bookworm)
- **Image Size:** 1,179 MB

The Asus ZenBook, employed for remote inference, was configured with the following operating system:

- **Firmware Version:** UX550GE.308
- **Operating System:** Ubuntu 24.04.2 LTS
- **System Architecture:** 64-bit
- **Desktop Environment:** GNOME 46
- **Display Server:** Wayland
- **Kernel Version:** Linux 6.11.0-26-generic

4.2.2 Python Environment

Raspberry Pi OS comes with Python 3.11.2 pre-installed. The deployment environment can be set up by installing all required dependencies with the following command, based on the requirements file shown in Listing 4.1:

```
pip install -r requirements.txt
```

```
1     customtkinter==5.2.2
2     dlib==19.24.8
3     numpy==1.24.4
4     opencv_python==4.11.0.86
5     picamera2==0.3.27
6     Pillow==9.4.0
7     Prototype==0.2
8     psutil==5.9.4
9     python_vlc==3.0.21203
10    Requests==2.32.4
11    rpi_lgpio==0.6
12    scipy==1.15.3
13    sounddevice==0.5.2
14
```

Listing 4.1: Requirements file for Pi 3

4.2.3 Voice Transcription with Whisper

To transcribe speech audio, the open-source project `whisper.cpp` was employed. This is a C/C++ implementation of OpenAI's Whisper model, providing a fully offline, CPU-optimized speech-to-text solution tailored for low-resource devices. The project also includes a lightweight HTTP server (`whisper-server`) that enables local API-based transcription.

Setup and Installation

The server was built using `CMake`, and models in the Geometric Graph Memory Layout (GGML) format were downloaded for inference. The installation and setup procedure is as follows:

```
git clone https://github.com/ggml-org/whisper.cpp.git
cd whisper.cpp
sh ./models/download-ggml-model.sh base.en
cmake -B build -DWHISPER_BUILD_SERVER=ON
cmake --build build --config Release
```

The server can then be launched from the `whisper.cpp` directory using:

```
build/bin/whisper-server -m models/<model> --host <ip-addr>
```

where `<ip-addr>` specifies the IP address the server will listen on and `<model>` indicates the model used for inference (e.g., `ggml-base.en.bin`).

Models Evaluated

The evaluation presented in this study is limited to performance-related metrics — specifically execution time and memory usage — since the transcription accuracy of individual Whisper models has already been thoroughly validated in existing literature. The following Whisper models were assessed across the tested platforms:

- **Tiny (quantized)** — evaluated on both the Raspberry Pi 3 and the ASUS laptop.
- **Base (quantized)** — evaluated on both the Raspberry Pi 3 and the ASUS laptop.
- **Large (non-quantized)** — evaluated exclusively on the ASUS laptop due to hardware constraints.

Quantized models are optimized for inference speed and memory consumption, with minimal loss of transcription accuracy. For instance, the `base.en` model size is reduced from 142 MiB to approximately 57 MiB by applying 5-bit quantization.

Although accuracy was not re-evaluated in this work, previously published Word [11] Error Rate (WER) benchmarks can offer a useful reference. The table below reports the zero-shot performance (greedy decoding) of the selected Whisper models on the standard LibriSpeech [12] `test-clean` and `test-other` subsets.

`test-clean` is a benchmark dataset consisting of clean, well-articulated read speech recorded under ideal conditions. In contrast, `test-other` includes more challenging audio samples, such as speech with regional accents, less careful pronunciation, or lower recording quality. Together, they provide a more comprehensive view of automatic speech recognition (ASR) robustness.

Model	Parameters	WER test-clean (%)	WER test-other (%)
Tiny	39M	7.6	16.9
Base	74M	5.0	12.4
Large	1550M	2.7	5.6

Table 4.1: WER on LibriSpeech `test-clean` and `test-other`, using greedy decoding.

Raspberry Pi Integration and Protocol Execution

To integrate Whisper's transcription capabilities into the Raspberry Pi–based health monitoring prototype, a dedicated Python module was developed. This module interacts with the local `whisper-server` instance via HTTP, enabling low-latency, offline voice transcription.

The implemented class, `VoiceAgent`, encapsulates the end-to-end speech interaction pipeline. It performs the following core tasks:

- **Prompt Playback:** Pre-recorded prompts are played using the `python-vlc` bindings. Questions are sequentially presented to the user to elicit verbal responses.

- **Audio Capture:** Audio is recorded using the `sounddevice` library at a sampling rate of 16 kHz. Each utterance is saved as a temporary `.wav` file in the local filesystem.
- **Audio Submission:** The recorded file is sent via HTTP POST to the `whisper-server` endpoint. The server, running locally on the Raspberry Pi, returns the transcription in JSON format.
- **Response Processing:** The raw transcription is normalized and parsed to extract semantically relevant responses (e.g., identifying affirmative or negative replies).
- **Threaded Execution:** Each question-response cycle is managed in a separate thread, ensuring responsiveness and facilitating precise timing analysis. Thread-safe queues are used to collect transcription results and execution times for each interaction.
- **Time Monitoring:** The module captures user reaction times and Whisper processing times independently for each prompt, allowing for quantitative evaluation of latency across sessions.

All temporary audio files are automatically deleted after processing to minimize memory usage on resource-constrained devices. The system architecture supports rapid iteration and modular deployment, ensuring compatibility with various Whisper models (e.g., `tiny.en.q5`, `base.en.q5`) and minimal latency for real-time applications on embedded platforms.

4.2.4 Heart Rate Monitoring

The heart rate monitoring functionality is based on a chest-worn Polar T34 ECG transmitter and a compatible receiver module connected to the Raspberry Pi.

This setup enables continuous and non-invasive monitoring of cardiac activity with high temporal resolution. Since the system operates without the need for external processing units or cloud-based services, it ensures data privacy and low-latency response. Additionally, the simplicity of the hardware interface allows for seamless integration with embedded platforms such as the Raspberry Pi.

Raspberry Pi Integration

The heart rate monitoring system on the Raspberry Pi is implemented by interfacing a Polar-compatible receiver module directly to one of the GPIO pins (GPIO4). The software component is encapsulated in a custom `HeartRateSensor` class, which is responsible for configuring the GPIO interface and computing the heart rate in beats per minute (BPM) based on the timing between received digital pulses.

To ensure responsiveness and minimize CPU usage, the system leverages an interrupt-driven mechanism rather than a polling-based approach. Specifically, the Raspberry Pi is configured to trigger an interrupt service routine (ISR) on each rising

edge detected on the signal pin. This method ensures that each heartbeat is captured with minimal latency and without the need for continuous active monitoring of the pin state.

Within the ISR, the time interval between successive heartbeats is computed using high-precision system timers, and the corresponding BPM value is derived from this interval. To handle concurrency and data consistency, a semaphore is used to protect access to the latest heart rate sample, which can be retrieved asynchronously by other components of the application.

This interrupt-based design offers significant advantages over polling: it reduces CPU overhead, enables accurate timing even under variable system load, and guarantees immediate reaction to heart rate events. Furthermore, it allows the main application thread—responsible for GUI updates and camera processing—to operate independently, thus maintaining smooth and responsive real-time monitoring.

The relevant methods responsible for configuring the GPIO pin and processing the heart rate signal are shown in Listing 4.2. The `setup()` method initializes the GPIO pin in input mode with a pull-down resistor to avoid false triggers when idle. It also registers a rising edge interrupt on the selected pin, invoking the `heart_rate_ISR()` method when a new heartbeat pulse is detected.

To reduce noise and avoid false multiple triggers from the same signal edge, a debouncing interval is configured using the `bouncetime` parameter. In this case, a value of 260 milliseconds is set, meaning that after an interrupt is triggered, further interrupts on the same pin are ignored for that duration.

The chosen debounce interval of 260 ms represents a compromise between filtering out false triggers caused by signal noise or contact bounce, and maintaining the ability to detect heartbeats at physiologically plausible maximum rates. Since a typical maximum heart rate does not exceed approximately 220 beats per minute (bpm), corresponding to a minimum interval of about 270 ms between pulses, setting the debounce slightly below this threshold helps avoid spurious detections without missing valid beats.

The ISR function `heart_rate_ISR()` calculates the time elapsed since the previous pulse and converts it into a BPM value.

```
1    def setup(self):
2         GPIO.setmode(GPIO.BCM)
3         GPIO.setwarnings(False)
4
5         GPIO.setup(
6             self._gpio_pin_hr,
7             GPIO.IN,
8             pull_up_down=GPIO.PUD_DOWN
9         )
10
11        GPIO.add_event_detect(
12            self._gpio_pin_hr,
13            GPIO.RISING,
14            callback=self.heart_rate_ISR,
15            bouncetime=260
16        )
17
```

```
18      def heart_rate_ISR(self, channel):
19          if channel != self._gpio_pin_hr:
20              return
21          current_time = Time.time()
22          if self._last_pulse_time is not None:
23              interval = current_time - self._last_pulse_time
24              if interval > 0:
25                  bpm = 60 / interval
26                  with self._semaphore_hr:
27                      self._last_heart_rate_sample = bpm
28
29          self._last_pulse_time = current_time
30
```

Listing 4.2: Board pin setup for heart rate monitoring.

4.2.5 Eye Tracking Monitoring

The eye tracker implemented in this study utilizes the `dlib` library, an open-source toolkit for facial recognition and image analysis, known for its efficiency and accuracy. `dlib` employs machine learning algorithms, notably a *regression tree ensemble* model for facial landmark detection, to precisely localize key facial features, including the eyes.

The tracking process begins with face detection using a classifier based on *Histogram of Oriented Gradients* (HOG) combined with a Support Vector Machine (SVM). Once the face is detected, the regression model predicts 68 facial landmarks, among which are those outlining the shape and contour of the eyes and surrounding regions.

These landmarks are used to isolate the eye region for further analysis and also to provide reference points for tracking localized movement. The position of the pupil is estimated through segmentation techniques and shape-based analysis, relying on contrast and intensity variations within the eye landmark region. Additionally, frame-to-frame displacements of these landmarks are analyzed to assess motion within the facial region, enabling detection of dynamic activity beyond pupil movement alone.

The `dlib`-based approach enables a non-invasive and markerless tracking system, robust under varying lighting conditions, and suitable for real-time deployment on low-power embedded devices such as the Raspberry Pi 3 B+. By combining reliable face detection with precise landmark tracking, the system supports both eye state inference and general facial motion analysis, while remaining computationally efficient for embedded health monitoring applications.

Raspberry Pi Integration

To achieve real-time motion tracking on resource-constrained hardware, the implementation was specifically tailored for the Raspberry Pi 3 B+ platform. The system integrates the `Picamera2` module for efficient video acquisition and utilizes the `dlib`

and `OpenCV` (`cv2`) libraries for facial landmark detection and motion estimation, respectively.

A dedicated class-based architecture orchestrates the tracking pipeline. The `EyeTracker` class encapsulates the logic for facial landmark extraction, eye state estimation, and movement detection. Landmark prediction is performed using the pre-trained 68-point facial shape model from `dlib`, which identifies key features such as the eyes and their contours. From these landmarks, geometric descriptors—such as the Eye Aspect Ratio (EAR)—are computed to infer eye openness. This ratio allows the system to classify the eye as *open*, *slightly closed*, or *closed* based on empirically tuned thresholds.

Motion detection is based on the Lucas-Kanade pyramidal optical flow algorithm applied to facial landmark regions across consecutive grayscale frames. The displacement vectors derived from the tracked landmarks are temporally smoothed using a short buffer of past frames, and their average magnitude is compared against a calibrated threshold to determine whether the observed facial region is either *moving* or *stationary*. This enables detection of both ocular and general upper facial movements, making the approach suitable for broader behavioral monitoring.

To maintain responsiveness in the presence of concurrent tasks (e.g., heart rate sampling and GUI updates), the video acquisition and tracking operations are executed within a dedicated background thread. This threaded design decouples image processing from the main application loop, ensuring real-time performance and minimizing latency on the embedded platform.

Frame capture and preprocessing are managed by a dedicated `CameraHandler` class, which configures and controls the `Picamera2` interface, delivering video frames in the appropriate format for downstream analysis. The asynchronous thread continuously retrieves frames from `CameraHandler`, performs grayscale conversion, and invokes the `EyeTracker` for landmark-based analysis. The computed motion status and eye state are updated in real time and exposed to the broader health monitoring protocol for further integration.

An example of the functioning eye tracking module is shown in Figure 4.2.

Figure 4.2: Eye tracking example.

4.3 Protocol Description

This section outlines the protocol adopted for data acquisition and computation of the Glasgow Coma Scale (GCS) score. The protocol integrates multimodal information—specifically eye state, motor activity, heart rate, and verbal responsiveness—to assess a subject's level of consciousness.

The procedure is initiated by the user via the **"Estimate GCS"** button in the graphical user interface (GUI), illustrated in Figure 4.3, after selecting the desired inference mode (**local** or **remote**).

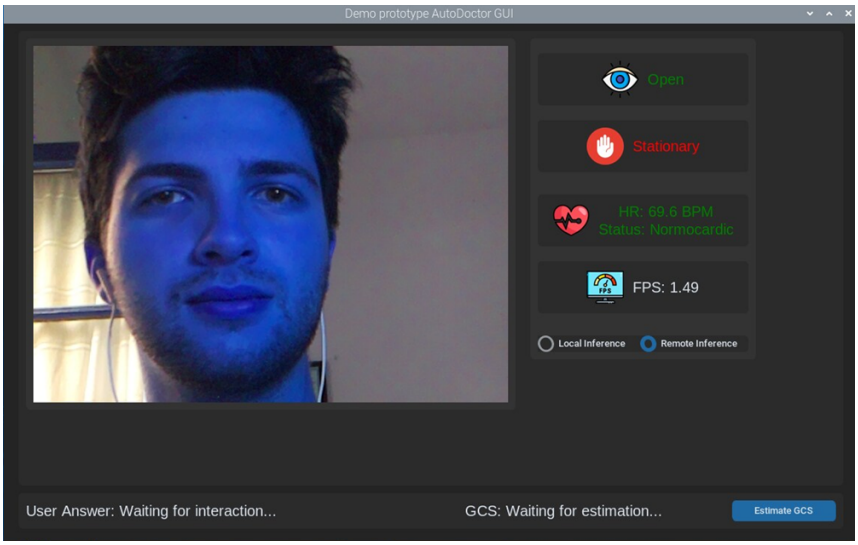


Figure 4.3: Screenshot of the graphical user interface.

4.3.1 Data Sampling

Upon activation, the system begins sampling the following data streams:

- **Physiological signals:**
  - *Eye state:* classified as `Open`, `Slightly Closed`, or `Closed`
  - *Eye movement:* detected as either `Moving` or `Stationary`
  - *Heart rate:* measured in beats per minute (BPM)
- **Verbal responsiveness:** determined through speech recognition (using Whisper) on the subject's answers to the following three binary (`yes/no`) questions:
  1. "Are you okay?"
  2. "Can you open your eyes?"
  3. "Can you move?"
- **System performance metrics:**
  - Video frame rate (FPS)

  - CPU, memory, and swap usage
  - Inference latency for Whisper transcription

During the protocol execution, a red LED remains continuously lit to indicate the ongoing acquisition phase.

4.3.2 Protocol Termination and GCS Estimation

At the conclusion of the verbal query phase, the system performs the following actions:

1. **Terminates all data sampling** procedures.
2. **Estimates the GCS** based on a multi-factorial decision model, summarized below:
   (a) The most frequent *eye state* is assigned a score:
      - `Open`: 5 points
      - `Slightly Closed`: 3 points
      - `Closed`: 1 point
      - Unknown: 0 points
   (b) The most frequent *movement state* is mapped to a score:
      - `Moving`: 5 points
      - `Stationary`: 2 point
      - Unknown: 0 points
   (c) The number of affirmative answers (`yes`) from the subject determines the verbal score, ranging from 1 to 5.
   (d) The **total GCS** is obtained by summing the individual component scores:

$$\text{GCS} = \text{Eye Score} + \text{Movement Score} + \text{Verbal Response Score}$$

3. **Stores all collected data and the selected inference mode** for further analysis or audit.

For the heart rate analysis, the standard cardiology classification is used to determine the patient's status based on the heart rate value:

- **60–100 BPM**: Normocardic
- **40–59 BPM**: Bradycardic
- $\geq$ **100 BPM**: Tachycardic
- $<$ **40 BPM**: Severe Bradycardic

Monitoring heart rate in conjunction with the Glasgow Coma Scale provides a more comprehensive assessment of the patient's physiological state, especially in emergency scenarios such as road traffic accidents. While the GCS is a validated and widely adopted tool for evaluating neurological function and consciousness level, it does not offer direct insight into systemic physiological responses to trauma, such as hypovolemia, internal bleeding, or shock.

Heart rate, by contrast, serves as a sensitive indicator of autonomic nervous system activity and cardiovascular stability. For instance, tachycardia may reflect compensatory mechanisms in response to hemorrhage or hypoxia, while severe bradycardia may signal elevated intracranial pressure or impending cardiac arrest. Integrating heart rate classification—ranging from normocardic to severe bradycardic—enables early detection of critical conditions that may not be immediately apparent through neurological scoring alone.

In prehospital and resource-constrained environments, such as inside a damaged vehicle, rapid and automatic acquisition of both neurological and cardiological indicators is crucial. The inclusion of heart rate monitoring thus enhances triage decisions, supports early warning systems, and enables timely medical interventions that may improve survival outcomes.

Chapter 5

Performance Evaluation

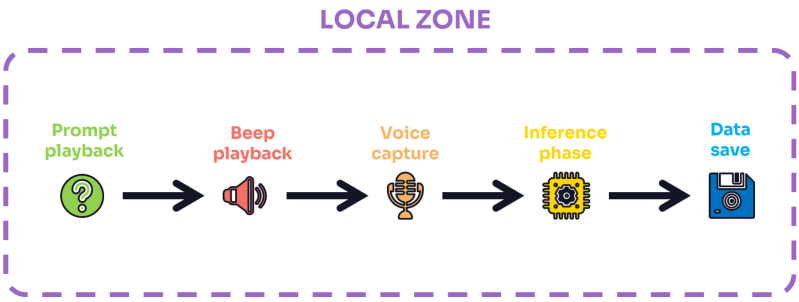
During the development of the application, the limited hardware resources of the Raspberry Pi 3 Model B+ posed significant constraints. To address these limitations, the system architecture was designed to perform the majority of computations locally, within the vehicle,thereby ensuring functionality even in the absence of external network connectivity such as 4G, 5G, or Wi-Fi.

To evaluate the feasibility of this approach, a series of performance tests were conducted under different deployment configurations. Initially, the application was executed entirely on the Raspberry Pi, utilizing all of its available resources. Subsequently, the most computationally intensive component, the Whisper speech recognition engine, was offloaded to a remote server to alleviate the local processing load. The two possible configurations are shown in Figure 5.1.

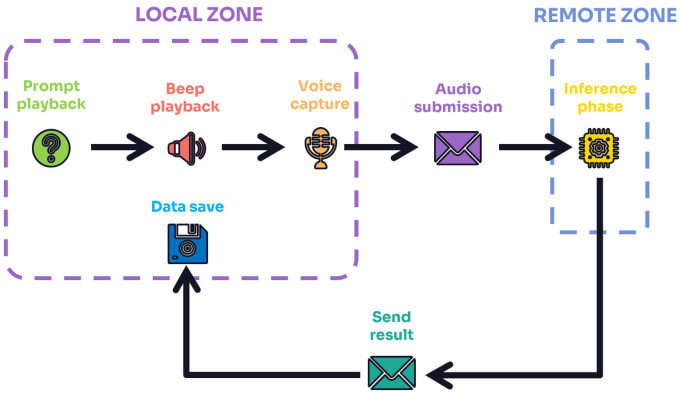
The system was further tested with various Whisper model variants to assess their viability for local deployment. Specifically, the tiny, base, and large models were evaluated. The tiny and base models were tested both locally and remotely, while the large model, due to its substantial computational demands, was only tested on the remote server. For each configuration (local/remote and model variant), five repetitions of the experiment were performed to ensure consistency and reliability of the results.

Performance metrics collected from these test scenarios are illustrated in the following graphs. These results provide insight into the trade-offs between local and distributed processing and help identify the most suitable Whisper model for on-device deployment given the hardware constraints.

Each graph illustrates the impact of using the tiny, base, and large models, both in local and remote setups (with the large model tested only remotely). The data highlights the varying computational demands of each model and the performance benefits of offloading heavy processing to a remote server.



(a) Local inference of whisper model.



(b) Remote Computation of whisper model.

Figure 5.1: Audio recording and Whisper inference on the two different setups.

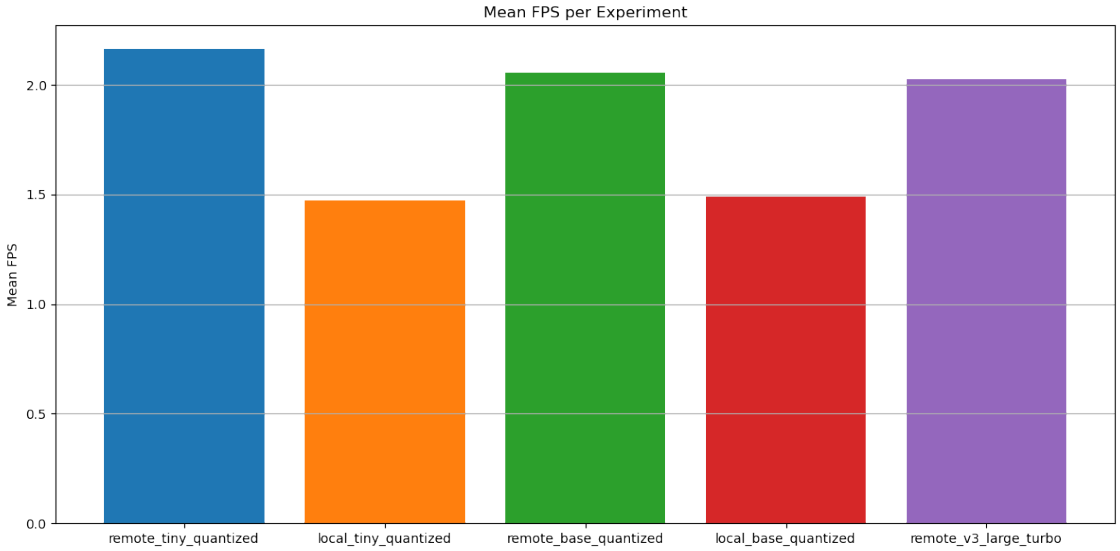


Figure 5.2: Camera FPS under different Whisper model configurations and deployment strategies.

Figure 5.2 illustrates the average frame rate (FPS) sustained by the camera during live sampling, under different Whisper inference configurations. Rather than measuring model inference speed directly, this graph illustrates how each model configuration impacts the system's ability to maintain real-time video acquisition. A clear trend emerges: heavier models executed locally significantly reduce the camera sampling rate, with FPS dropping sharply due to the CPU being overwhelmed by concurrent video processing and inference tasks. Interestingly, despite their difference in size and complexity, both the tiny and base quantized models exert a similar negative impact when run locally, indicating a limitation in system resource access by the models due to process scheduling priorities—where any sustained inference workload on the Raspberry Pi is sufficient to disrupt video acquisition, regardless of the model's relative computational weight.

Conversely, remote execution minimizes interference, enabling the Raspberry Pi to maintain higher FPS, close to the optimal hardware limit. This behavior is consistent across all remote configurations, with performance differences that are minimal but measurable. Among them, the tiny model executed remotely yields the highest FPS, as expected due to its low computational load and fast inference time. The base model follows closely, while the large model—despite being executed remotely—results in slightly lower FPS, though still within an acceptable range. These minor differences mirror the time each thread spends waiting for the server to return inference results, reflecting how even remote model complexity can subtly affect real-time performance on resource-constrained clients.

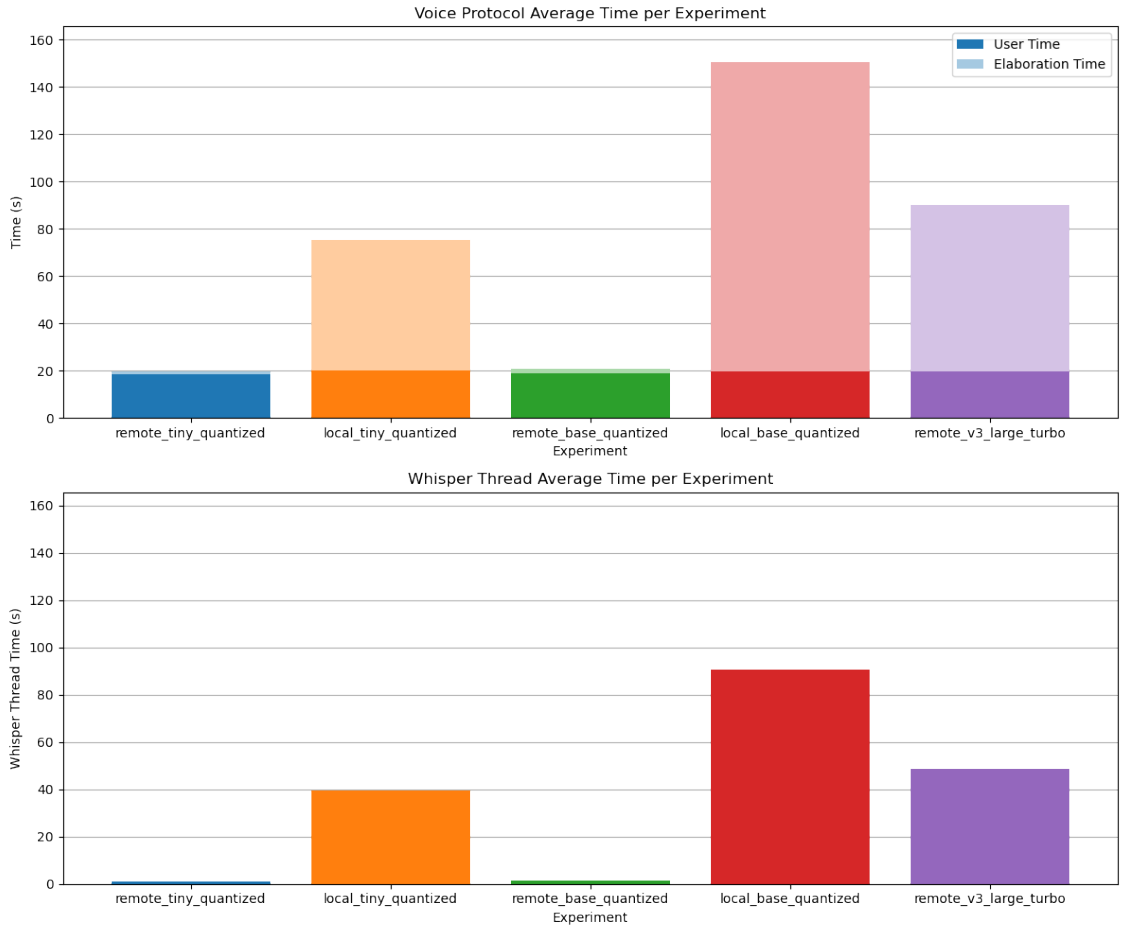


Figure 5.3: Average voice protocol time under different Whisper model configurations and deployment strategies.

Figure 5.3 presents a breakdown of user time and elaboration time during the transcription process. The user time represents the fixed interaction time dictated by the user protocol, i.e., the duration spent speaking and advancing through the predefined questions. As expected, this value remains nearly constant across all configurations, confirming that user behavior is unaffected by the underlying inference strategy.

On the other hand, the elaboration time corresponds to the cumulative processing time required to transcribe all utterances, taking into account the partial concurrency of the process—while the user is responding to a question, the system is already transcribing the previous response. This pipelined execution reduces overall latency, yet still reveals substantial differences depending on model placement and size.

Remote execution significantly lowers elaboration time on the Raspberry Pi by offloading the computational load to the server, allowing the device to act mainly as a passive conduit. In contrast, local execution results in noticeably higher elaboration times, especially for the base model, reflecting the increased burden on the CPU. This disparity becomes even more pronounced when comparing local and remote

versions of the same model, demonstrating the cost of on-device inference under constrained resources. The large-v3 model, which was not tested locally due to feasibility limits, still shows substantial elaboration time when executed remotely, owing to its sheer complexity and size—even when processed on more powerful hardware.

These observations are further supported by the analysis of the average per-thread elaboration time, i.e., the time taken to process each individual transcription task in isolation. Due to the concurrency built into the system, this value is not exactly one-third of the total elaboration time (despite there being three questions), but remains indicative of the processing load associated with each configuration. The gap between local and remote execution remains evident even when parallelism is not accounted for, reinforcing the benefits of distributed processing in resource-constrained environments.

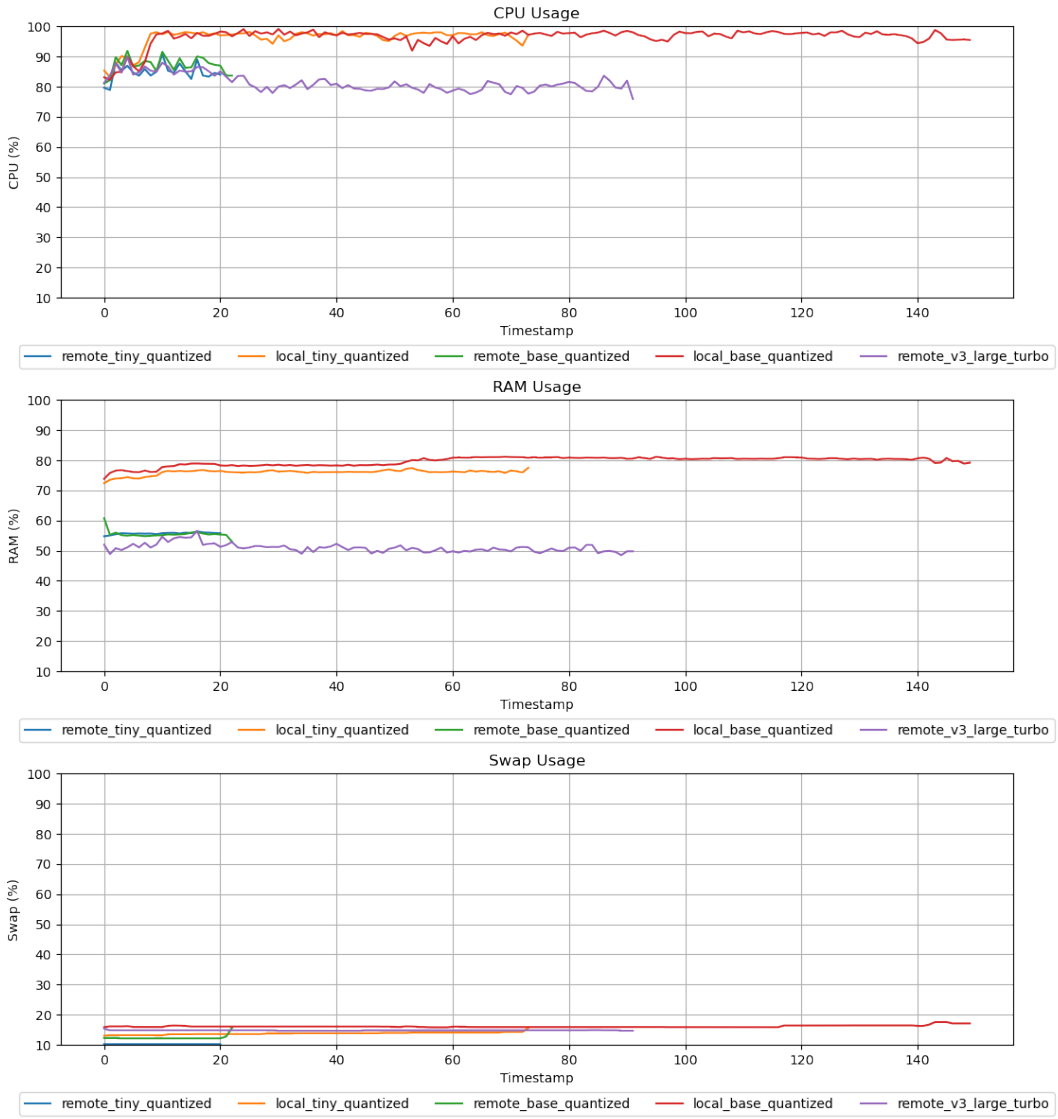


Figure 5.4: System resource utilization (CPU, RAM, and swap) under different Whisper model configurations and deployment strategies.

Figure 5.4 illustrates CPU, RAM, and swap usage across the various configurations tested. Contrary to initial expectations, swap usage remains consistently low in all scenarios, ranging between 10% and 20%. This indicates that none of the configurations exhaust physical memory to the extent that disk-based virtual memory becomes a significant factor, reflecting good memory handling even under stress.

RAM usage, on the other hand, shows a stark difference between local and remote execution. Local inference of both the tiny and base models leads to substantial RAM occupation, nearing full capacity on the Raspberry Pi. Interestingly, despite the difference in size and complexity, both models result in very similar levels of memory and CPU usage, suggesting that the system's inference allocates similar processing buffers or overhead structures regardless of the model scale. In contrast, remote execution significantly reduces RAM usage, with all remote models main-

taining low and stable memory footprints. In particular, the large v3-turbo model shows RAM usage that is comparable to the other remote configurations, albeit slightly lower.

Deeper in the consumption of CPU resources, the analysis focused on the average usage of the three main processes:

- **Whisper**: corresponds to the speech-to-text model.
- **dlib + HR**: includes the OpenCV pipeline used for face detection,which rely on real-time video frame processing, and the heart rate sampling module.
- **labwc**: is the lightweight Wayland window compositor responsible for managing the graphical user interface (GUI), including rendering and user interaction components. It was selected as a reference point to monitor the resource consumption specifically attributable to the graphical interface.

By isolating these processes, the data reveals how each component contributes to overall CPU load, allowing for a clearer comparison between configurations with and without GUI.

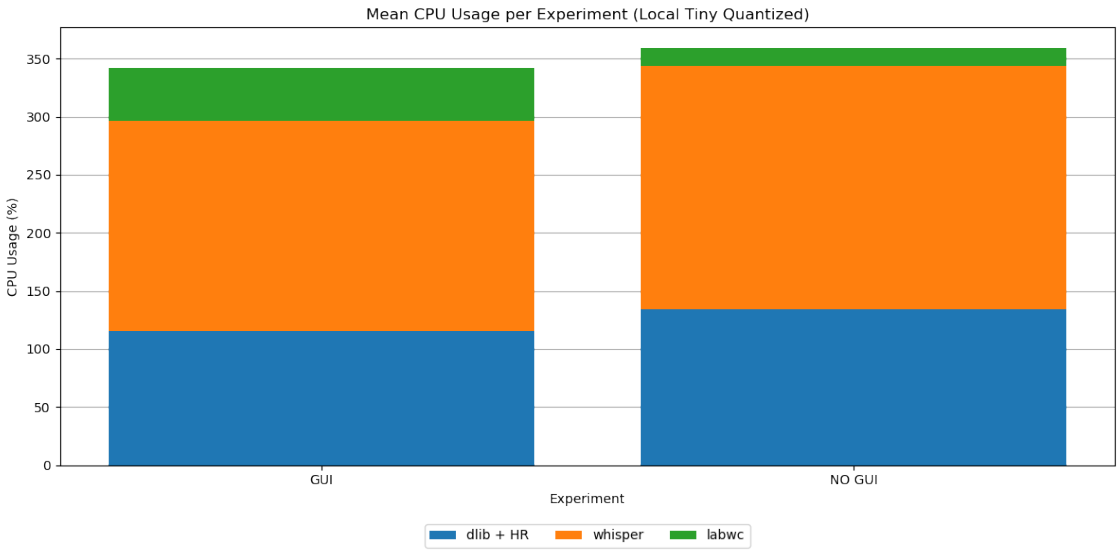


Figure 5.5: CPU usage of the main process involved in the GCS estimation protocol.

Starting with Figure 5.5, we observe some differences in total and per-process CPU consumption between the two configurations. While the overall CPU usage is slightly higher in the no-GUI scenario, this increase is strategically distributed. Specifically:

- **Whisper** consistently consumes the largest share of resources in both setups, but it receives more CPU time when the GUI is disabled, indicating improved prioritization of inference-related tasks.
- **dlib + HR** also sees a modest increase in CPU usage when the GUI is removed, benefiting from reduced contention.

- **labwc** drops significantly in CPU consumption from a noticeable portion under the GUI to a minimal share without it highlighting its role in resource overhead.

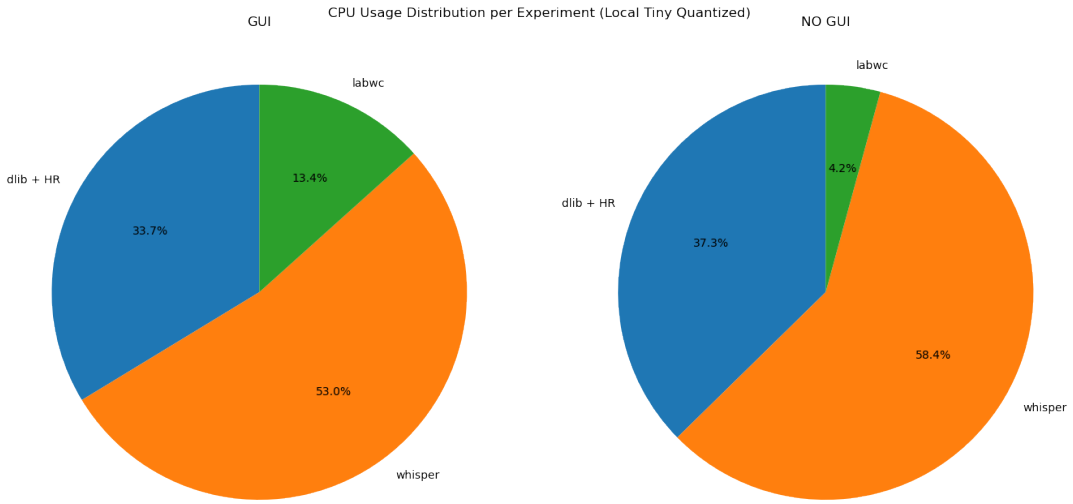


Figure 5.6: Pie charts representation of the distribution of cpu usage on local tiny quantized model.

These distinctions become even more apparent when analyzing the CPU usage pie charts (Figure 5.6). In the GUI-enabled setup, labwc alone accounts for over 13% of CPU usage. This is a non-negligible load, especially on constrained hardware like the Raspberry Pi. When the GUI is disabled, labwc's share plummets to just 4.2%, freeing valuable CPU time.

The redistribution of CPU cycles primarily benefits whisper, whose share increases from 53% to 58.4%. This suggests that the model scales to utilize available processing capacity more effectively in headless mode. Similarly, dlib + HR increases from 33.7% to 37.3%, further emphasizing that these compute-heavy modules thrive when GUI-related overhead is removed.

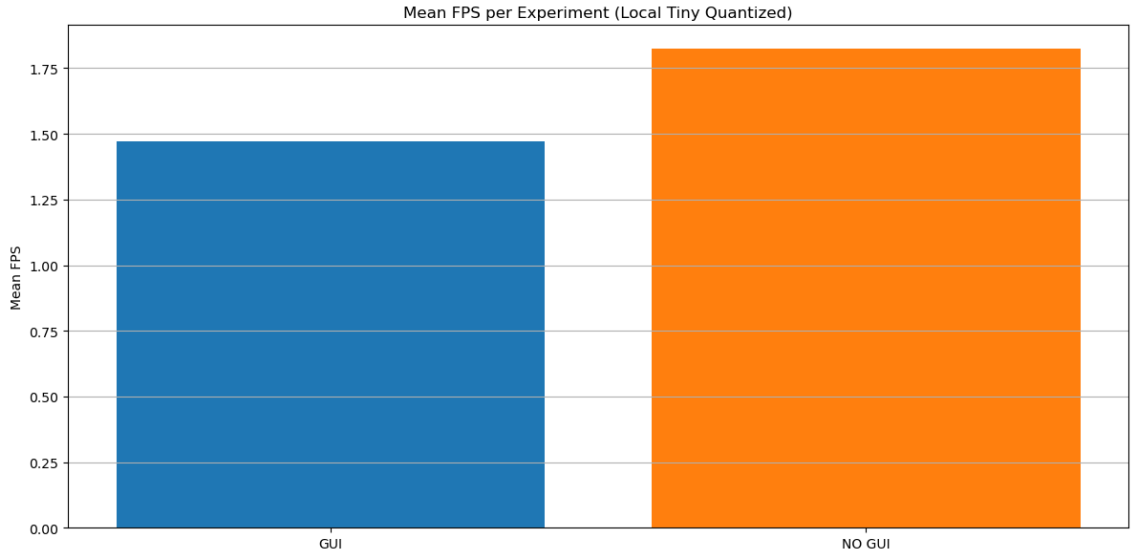


Figure 5.7: Mean FPS with and without GUI on local inference with tiny model quantized.

The computational benefits observed in the mean FPS plot reflect the redistribution of system resources previously seen once the GUI is removed. In this context, the freed resources are slightly more allocated to Whisper rather than Dlib. Interestingly, this still results in a minor increase in Dlib's performance, as indicated by a small rise in FPS. While this represents a positive improvement, its impact on overall processing efficiency remains relatively minor.

Based on the experimental findings, it is evident that offloading Whisper inference to a remote server markedly enhances overall system performance and responsiveness, particularly when utilizing larger and more computationally demanding models. Conversely, in fully offline scenarios where network connectivity is unavailable or unreliable, the tiny quantized model represents the most viable option, balancing inference speed and resource consumption effectively on the Raspberry Pi hardware. Although the base model can be deployed locally, it imposes a significant strain on system resources, potentially compromising real-time responsiveness during prolonged operation. The large model, due to its substantial computational requirements, remains impractical for on-device execution on the Raspberry Pi 3 Model B+, reinforcing the necessity of distributed processing architectures for advanced speech recognition tasks in resource-constrained embedded environments.

Chapter 6

Conclusion

By integrating multimodal physiological and behavioral data through a modular embedded system, this project addresses the critical need for rapid and automated health assessment of vehicle occupants following accidents. The prototype combines sensors for cardiac, visual and auditory inputs, enabling a comprehensive estimation of a simplified Glasgow Coma Scale (GCS) and vital sign monitoring to support first responders in emergency scenarios.

Benchmarking different AI inference strategies highlighted key trade-offs between local and remote processing. While local inference on the Raspberry Pi offers autonomy in offline conditions, offloading intensive tasks such as speech recognition to remote AI models significantly improves responsiveness and reduces embedded hardware load. This hybrid approach balances computational efficiency with real-time performance, reinforcing the feasibility of deploying such systems in resource-constrained automotive environments.

The system's architecture exemplifies adaptability and scalability. The use of a Raspberry Pi as an edge hub facilitates sensor fusion and initial data processing, while the integration with remote AI services allows for enhanced analysis without overwhelming embedded resources. This flexible design supports future expansion, including the addition of new sensor modalities and refined AI models tailored for embedded deployment.

Despite its promising results, the project underscores challenges inherent to real-world applications, such as variability in sensor data quality due to environmental factors and the limitations of current embedded hardware for complex AI workloads. Addressing these issues through model optimization and hardware advancements will be essential for broader adoption.

Looking ahead, this multimodal embedded system lays the foundation for advanced prehospital health monitoring solutions in automotive safety. Further developments could explore deeper multimodal data fusion, on-device AI acceleration, and extensive field validation to enhance reliability and impact in emergency response.

In conclusion, this work demonstrates a meaningful step toward integrating AI-driven health assessment within vehicle safety systems, offering scalable and effective tools to improve rapid triage and occupant care after accidents.

Bibliography

[1] Wikipedia contributors, *Glasgow Coma Scale*, *Wikipedia, The Free Encyclopedia*, https://en.wikipedia.org/wiki/Glasgow_Coma_Scale, accessed May 30, 2025.

[2] Euro NCAP, *A Safer Future for Mobility – Euro NCAP Vision 2030*, https://www.euroncap.com, 2020.

[3] Philips, *Vital Signs Camera for Automotive*, Philips Official Website, 2023.

[4] Continental Engineering Services, *Advanced Cabin Sensing Solutions*, CES Solutions Brochure, 2022.

[5] A. Mishra, S. Lee, D. Kim, and S. Kim, *In-Cabin Monitoring System for Autonomous Vehicles*, *Sensors*, vol. 22, no. 12, article 4360, 2022. DOI: 10.3390/s22124360.

[6] F. Wang, H. Chen, L. Kong, and W. Sheng, *Real-time Facial Expression Recognition on Robot for Healthcare*, in *Proceedings of the 2018 IEEE International Conference on Intelligence and Safety for Robotics (ISR)*, pp. 402–406, 2018. DOI: 10.1109/IISR.2018.8535710.

[7] J. Bai, Y. Zhang, X. Sun, S. Zhou, R. Lan, and X. Jiang, *Investigate the In-Vehicle Healthcare System Design Opportunities: Findings from a Co-design Study*, in *HCI in Mobility, Transport, and Automotive Systems*, H. Krömker (Ed.), Springer, Cham, pp. 123–133, 2022. DOI: 10.1007/978-3-031-04987-3_8.

[8] OpenCV, *Open Source Computer Vision Library*, Official site opencv.org, 2023. Available at: `https://opencv.org`

[9] Dlib, *Dlib C++ Library*, Official Site dlib.net, 2023. Available at: `http://dlib.net`

[10] ggml-org, *whisper.cpp: Port of OpenAI's Whisper model in C/C++*, GitHub repository, https://github.com/ggml-org/whisper.cpp, accessed June 3, 2025.

[11] A. Radford, J. W. Kim, T. Xu, G. Brockman, C. McLeavey, and I. Sutskever, *Robust Speech Recognition via Large-Scale Weak Supervision*, arXiv preprint, 2022. Available at: arxiv.org/abs/2212.04356.

[12] V. Panayotov, G. Chen, D. Povey and S. Khudanpur, "Librispeech: An ASR corpus based on public domain audio books," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, South Brisbane, QLD, Australia, 2015, pp. 5206–5210, doi: 10.1109/ICASSP.2015.7178964.