Secure Carpooling System using Vision Verification and Audio Processing

By

Yee Jia Hao

A REPORT

SUBMITTED TO

Universiti Tunku Abdul Rahman

in partial fulfillment of the requirements

for the degree of

BACHELOR OF COMPUTER SCIENCE (HONOURS)

Faculty of Information and Communication Technology

(Kampar Campus)

FEBRUARY 2026

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my supervisor, Ts Dr Aun Yichiet, for his invaluable guidance, constructive feedback, and continuous support throughout this project. His expertise and encouragement have been instrumental in shaping the success of this work.

I am also thankful to the Faculty of Information and Communication Technology at Universiti Tunku Abdul Rahman (UTAR) for providing the resources and a conducive learning environment that enabled me to complete this project successfully. In particular, I appreciate the knowledge and skills gained from various courses offered during my degree program, which laid the foundation for this work. Special mention goes to UCCD3223 (Mobile Applications Development), which provided essential insights and practical experience that helped me kick-start the development of this carpooling application.

Finally, I would like to extend my heartfelt thanks to my family and friends for their unwavering support, patience, and motivation throughout this journey. I am especially grateful to my relatives and friends who volunteered as system testers, participating in various simulated scenarios and providing invaluable feedback to refine the application.

COPYRIGHT STATEMENT

This Final Year Project proposal is submitted in partial fulfillment of the requirements for the degree of Bachelor of Computer Science (Honours) at Universiti Tunku Abdul Rahman (UTAR). This Final Year Project proposal represents the work of the author, except where due acknowledgment has been made in the text. No part of this Final Year Project proposal may be reproduced, stored, or transmitted in any form or by any means, whether electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the author or UTAR, in accordance with UTAR's Intellectual Property Policy.

ABSTRACT

University campuses are ideal for carpooling, yet adoption remains low due to safety, trust, and reliability concerns. This project introduces UniRide, a secure and scalable carpooling mobile application tailored for academic institutions like Universiti Tunku Abdul Rahman (UTAR). UniRide integrates multi-layered safety mechanisms for proactive protection, beginning with a pre-trip "Device Verification Guard" that prevents impersonation using ML Kit and a Zero-Trust Hybrid Cloud Architecture. This architecture utilises an encrypted Cloudflare Tunnel to a local Flask API, offloading ArcFace biometric processing to bypass cloud GPU costs while achieving sub-second verification latency. During trips, an edge-deployed YAMNet acoustic model operates directly on the smartphone to detect distress sounds locally. Anomalies immediately trigger a multi-modal validation pipeline using the Gemini 2.5 Flash/Lite API and IMU sensor data, confirming emergencies and dispatching automated alerts in an average end-to-end time of just 4.29 seconds. To ensure consistent service availability, UniRide features a semester-based scheduling system and a gamified Earn-On-Demand (EOD) economic model, allowing drivers to earn flexibly while keeping rides affordable for students. Developed with Flutter and Firebase, the modular application also includes QR-based authentication, geofencing, and dynamic routing with pass-by drop points to optimise match rates. Ultimately, by combining AI-driven safety and dynamic economic incentives, UniRide redefines campus carpooling as a trustworthy, sustainable mobility solution that actively supports SDG 11 (Sustainable Cities), SDG 13 (Climate Action), and SDG 16 (Peace and Justice).

Area of Study: Carpooling Safety Framework, Sustainable Mobility

Keywords: Multi-Modal Verification, Campus Transportation, Safety Mechanisms, Incentive Model, Mobile Application, Sustainable Development Goals, Solution for All Institutions

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ANS	Adaptive Noise Suppression
AOSP	Android Open-Source Project
AP	Average Precision
API	Application Programming Interface
APK	Android Package Kit
AST	Audio Spectrogram Transformer
AWS	Amazon Web Services
BaaS	Backend-as-a-service
CH	Contraction Hierarchies
CNN	Convolutional Neural Network
CRUD	Create, Read, Update, Delete
CUDA	Compute Unified Device Architecture
DNS	Domain Name System
EOD	Earn On Demand
ETA	Estimated Time of Arrival
FC	Fully Connected
FCM	Firebase Cloud Messaging
FP16	16-bit Floating-Point
FPR	False Positive Rate
FRR	False Reject Rate
GAR	Genuine Accept Rate

GMS	Google Mobile Service
GPS	Global Positioning System
HMS	Huawei Mobile Services
HTTPS	Hypertext Transfer Protocol Secure
ID	Identifier/Identification
IMU	Inertial Measurement Units
INT8	8-bit Integer
IPA	iOS App Store Package
IT	Information Technology
LLM	Large Language Model
MAC	Multiply-Accumulate
MCU	Microcontroller Unit
ML	Machine Learning
MVP	Minimum Viable Product
NLP	Natural Language Processing
OAuth	Open Authorisation
O-D	Origin-Destination
OOM	Out of Memory
OOM	Out of Memory
OS	Operating System
PCM	Pulse-Code Modulation
PR-AUC	Precision-Recall Area Under Curve
PTQ	Post-Training Quantisation

ReLU	Rectified Linear Unit
ROC	Receiver Operating Characteristic
ROC AUC	Receiver Operating Characteristic Area Under Curve
RTT	Round Trip Time
SDG	Sustainable Development Goals
SDK	Software Development Kit
SHAP	SHapley Additive exPlanations
SOV	Single Occupant Vehicle
SSL	Secure Sockets Layer
TFLite	TensorFlow Lite
TITP	Taxi Industry Transformation Programme
TLS	Transport Layer Security
TN	True Negative
TNG	Touch 'n Go
TP	True Positive
UI	User Interface
UTAR	Universiti Tunku Abdul Rahman
UX	User Experience
VGG	Visual Geometry Group (from VGG-11 / VGGish model)
VKT	Vehicle Kilometres Travelled
WSL	Windows Subsystem for Linux
XAI	Explainable AI

CHAPTER 1

Introduction

1.1 Problem Statement and Motivation

University campuses like UTAR represent a concentrated and largely untapped market for sustainable mobility solutions. As supported by Park et al. (2018), the high density of students with overlapping travel patterns creates an ideal environment for carpooling, especially at dense student accommodation and hotel locations. Yet, persistent safety concerns, unreliable availability, and lack of economic incentives hinder adoption.

This project is driven by the need to tackle existing mobility challenges by developing a secure, sustainable, and cost-effective carpooling mobile app tailored for the university community. The proposed system introduces several novel planning and design features to overcome the dual-sided market failure of carpool systems:

- **Trust and Safety**: Multi-factor identity verification using face to ensure driver authenticity, while QR-based trip verification and UTAR email registration add layers of gatekeeping and accountability. Real-time location tracking and optional distress detection further enhance rider safety, strengthening the SDG 16: Peace, Justice, and Strong Institutions.
- **Availability and Reliability**: A schedule-based booking system allows drivers to publish predefined routes, similar to train ticketing, with semester-long bidding options. This novel approach prioritises committed riders and improves trip planning reliability.
- **Economic Model**: The proposed system introduces a novel incentive framework called Earn on Demand (EOD). EOD allows drivers to set their own fare capped at UTAR's RM1 bus rate or even offer free rides, preserving the non-commercial spirit of carpooling while introducing flexible earning opportunities. Drivers can choose between purely voluntary participation or opt into a multi-tier reimbursement model based on trip frequency, demand, and service quality.

- **Routing Flexibility and Scalability:** The system incorporates pass-by drop points using campus grid mapping and algorithms like A* to optimise ride searching. This expands the definition of mutual destination increasing destination coverage and match rates. Designed with modular architecture and institutional integration, the system can be adopted by other universities or educational institutions facing similar transportation challenges.

By tackling both behavioural and structural barriers, this project redefines carpooling from a purely voluntary model to a flexible, campus-adjusted system. While it introduces minor compromises such as optional low-cost fares, it remains significantly more affordable than alternatives like GrabCar or bus. Through strong verification, dynamic routing, and demand-based incentives, the system ensures high availability and builds trust, supporting institutions' sustainability goals and broader low-carbon mobility efforts.

1.2 Project Objectives

The project main objective is to design, develop, and evaluate a secure, reliable, and efficient carpooling mobile application prototype tailored for UTAR university community, with scalability for future expansion and commercialisation.

This will be achieved through the following specific sub-objectives:

1. To design and develop a university-focused carpooling application supporting rider-driver registration, trip creation, bidding, scheduling, and campus-based ride matching.
2. To design and evaluate a pre-trip identity assurance mechanism using institutional email, face verification, liveness/device guard, and QR-based trip validation to reduce impersonation risk.
3. To design and evaluate a during-trip AI safety monitoring framework using edge-deployed YAMNet acoustic detection, IMU sensor validation, and Gemini-based contextual confirmation for automated distress alerting.
4. To integrate and evaluate EOD incentive, semester scheduling, and pass-by routing mechanisms to improve carpool availability, affordability, and match coverage.

5. To evaluate the system using AI model metrics, latency testing, spoofing tests, distress scenario tests, routing/matching analysis, and end-to-end safety workflow validation.

1.3 Project Scope and Direction

This project seeks to design and implement a carpooling app curated for widespread adoption across universities, offering convenient and sustainable transportation for students and lecturers, while integrating innovative safety features to foster a secure, comfortable, and trustworthy environment. The application will facilitate shared car rides between students and lecturers with similar destinations, focusing on safety and providing low-to-free fare rides. The project emphasizes the development of core functionalities for both drivers and riders, including trip submission, ride searching and bidding, Earn-on-Demand (EOD) reimbursement, and safety features such as a rating system.

Key safety features, such as multi-stage verification using vision models and QR codes, real-time location tracking, geofencing, and an emergency incident detection system powered by audio processing models, will be the main improvements over existing carpool systems. Additionally, this project will explore advanced ride-matching algorithms and a flexible EOD economic model. Initially, the system will focus on the UTAR Kampar Campus ecosystem, with potential for commercialisation and adoption by other educational institutions. The solution will leverage mobile phone hardware and online services to deliver an all-rounded carpool application with enhanced security and user engagement.

This project's scope will not include the development of a live payment gateway; the EOD model will calculate and record fare reimbursements, but actual financial transactions are considered out-of-scope; instead, it will utilise TNG's QR based transaction due to flexibility in transaction amount. The development of the backend server infrastructure from scratch is also excluded; the project will instead utilise backend-as-a-service (BaaS) platforms for core functions like authentication and database management. Finally, integration with external public transport systems or commercial ride-hailing services is not part of this project.

1.4 Application Contributions

This research introduces UniRide, a mobile carpooling ecosystem designed to resolve the first-and-last-mile connectivity issues within university campuses such as UTAR.

1. **Institutional Mobility Solution:** Provides a secure and affordable alternative to commercial ride-hailing and fixed-schedule campus buses, effectively bridging gaps in campus transit coverage.
2. **Economic Sustainability (EOD Model):** Introduces the Earn on Demand (EOD) incentive model, a novel fare-capping system that balances driver cost-recovery with student financial accessibility.
3. Environmental Impact: Facilitates the reduction of Single-Occupancy Vehicle (SOV) trips, directly contributing to lowered carbon emissions and reduced campus traffic congestion in alignment with SDG 13.
4. Scalable Architecture: Employs a registration-based, multi-tenant architecture that allows other Malaysian institutions to deploy dedicated, isolated instances of the system while maintaining centralised support.

1.5 Scientific and Technical Contributions

The primary technical contribution of this work is the formalisation of a Hierarchical Multimodal Safety Framework (HMSF) for closed-loop institutional environments.

1. **Proactive "Trigger-Analyse-Response" Logic:** Unlike traditional reactive safety features, this research contributes a modular pipeline that automates threat detection across the entire trip lifecycle.
2. **Edge-AI Acoustic Distress Modelling:** A significant contribution lies in the optimisation of the YAMNet deep learning model for mobile edge deployment. This allows for real-time, privacy-preserving acoustic classification (e.g. detecting screams or collisions) directly on the device without requiring cloud-based audio streaming.
3. **Multimodal Sensor Fusion for Validation:** The framework introduces a novel validation layer that synthesizes acoustic signals with kinetic data from the device's Inertial Measurement Unit (IMU). This cross-verification logic uses LLM-based reasoning (Gemini) to contextually differentiate between genuine emergencies and false positives, such as loud music or road noise.

4. **Integrated Biometric Identity Management**: The research demonstrates a method for achieving enterprise-grade face verification within a mobile environment by routing ArcFace embeddings through a Zero-Trust tunnel to a local API, ensuring sub-second verification latency.
5. **Service Zone Compliance via Geofencing:** Contributes a logic-based geofencing system that ensures service compliance within campus zones, fostering accountability and shared responsibility within the institutional community.

1.6 Background Information

Carpooling, a non-commercial service where participants share mutually aligned routes, offers a sustainable alternative to commercial e-hailing services like Grab and Uber. By reducing single-occupancy vehicles (SOVs), it directly alleviates traffic congestion and lowers carbon emissions [1, 2]. These environmental pressures are notably clear at Universiti Tunku Abdul Rahman (UTAR), where campus-related transportation significantly contributes to local fuel consumption and CO2 emissions, as shown in Figure 1.1. Recognising this impact, UTAR has introduced blueprints that nurture green transportation, including carpooling, and this project directly corresponds with those institutional goals and SDG 13: Climate Action.

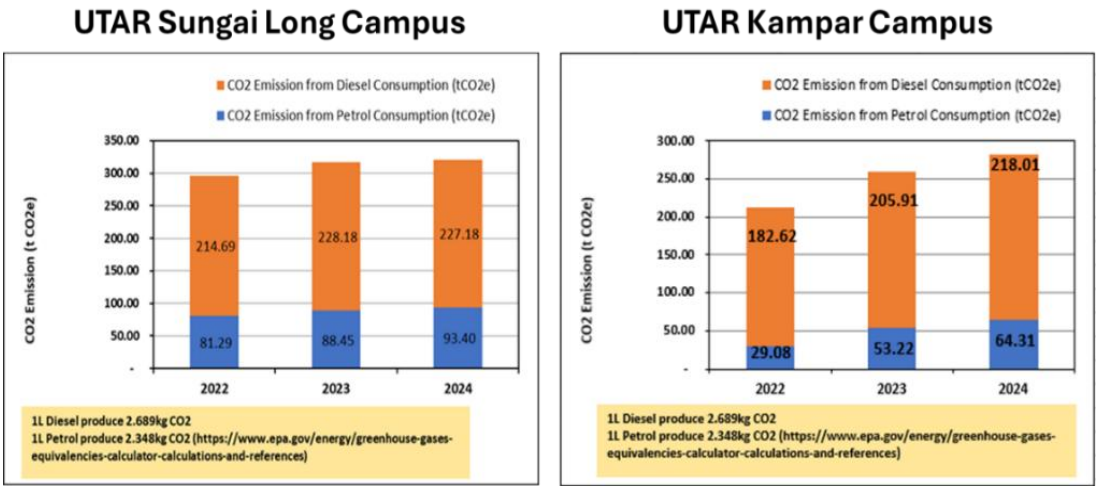


Figure 1.1 UTAR campuses CO2 emissions from 2022 to 2024

However, despite its clear benefits and policy support, carpooling has yet achieved extensive implementation, a challenge that has persisted for more than a

decade, with studies in academic institutions showing participation rates as low as 11-15% [3, 4]. This long-standing challenge stems from two fundamental barriers.

The first barrier is regarding the concern over safety and trust when sharing a vehicle with strangers, a fear worsened by risks of impersonation and the lack of formal regulatory oversight common in non-commercial systems [2, 4-6]. Icasiano et al. [7] stated that in Southeast Asia, there are several incidents in Malaysia where passengers and taxi drivers harassed ride-hailing drivers by vandalising their vehicles, reflecting a tense and sometimes unsafe environment [2, 5]. The second barrier is the inconsistent availability and reliability of the service. Unlike commercial platforms that use financial incentives to ensure a consistent supply of drivers, voluntary carpooling struggles to reach the critical mass of users needed to be a dependable transportation option for students with variable schedules.

To overcome these dual barriers, this proposal emphasizes on developing a secure, dependable, and scalable carpooling mobile app designed for the UTAR community and scalable. The system addresses the user trust deficit by integrating a robust multi-stage verification system and enhances in-ride security with proactive safety features. To solve for inconsistent availability, it introduces a novel hybrid incentive model and a semester-based scheduling system. By integrating these innovations, this project aims to build the foundational trust and reliability required to make carpooling a viable and widely adopted transportation solution for the institution's communities.

1.7 Report Organisation

This report is organised into seven distinct chapters, detailing the progressive research, design, implementation, and evaluation of the UniRide system:

Chapter 1: Introduction

Introduces the fundamental problem statement, motivation, and primary objectives of the research. It outlines the project's scope, its contributions to sustainable mobility (SDG 11, 13, and 16), and the underlying background of campus transportation challenges.

Chapter 2: Literature Review

Presents a comprehensive analysis of existing carpooling systems, safety mechanisms (e.g. DriverAuth, SafeShareRide, JustSave), and their practical limitations. The chapter justifies the architectural decisions made for UniRide, including the selection of YAMNet for edge audio recognition and the necessity of hybrid incentive models.

Chapter 3: System Methodology and Technology

Details the Agile (Kanban) development methodology and outlines the system specifications. It introduces the high-level hybrid system architecture, detailing the integration of the Flutter client, Firebase managed services, Cloudflare Tunnels, and the localised Flask API.

Chapter 4: System Design

Outlines the comprehensive blueprint of the carpool safety framework across the Pre-Trip, During-Trip, and Post-Trip phases. It details the Earn-On-Demand (EOD) economic model, semester-based scheduling, and the mathematical and structural design of the YAMNet deep-learning architecture for distress detection.

Chapter 5: System Implementation

Describes the technical realisation of the project. It covers the setup of the development environment, the establishment of the Zero-Trust Hybrid Cloud Architecture, the integration of biometric and distress monitoring pipelines, and the resolutions to critical implementation challenges.

Chapter 6: System Evaluation and Discussion

Evaluates the system's operational and AI performance. It provides a detailed analysis of the Face Biometric Verification robustness, the YAMNet classification accuracy (via Confusion Matrices and ROC AUC curves), and the end-to-end integration latency metrics of the hybrid cloud architecture.

Chapter 7: Conclusion and Recommendations

Summarises the project's overall achievements, highlights how it successfully addressed the initial problem statements, and provides actionable recommendations for future enhancements, system scalability, and commercialisation.

CHAPTER 2

Literature Review

2.1 Previous Works on Carpool System

2.1.1 Safety and Trust

Building a trusted user base is essential for community-driven carpooling platforms, therefore must account for both participants and driver security across all stages of a trip: before, during, and after. While commercial platforms like Grab [10] have introduced notable safety mechanisms across these phases shown in Figure 2.1, additional strategies from academic and developer communities offer complementary approaches. The following subsections study a range of safety measures proposed by researchers and implemented in real-world applications. These include pre-trip identity verification, during-trip monitoring, and post-ride accountability. Each approach is discussed in terms of its strengths and limitations, with resolve to build a well-rounded safety-focused carpooling system.

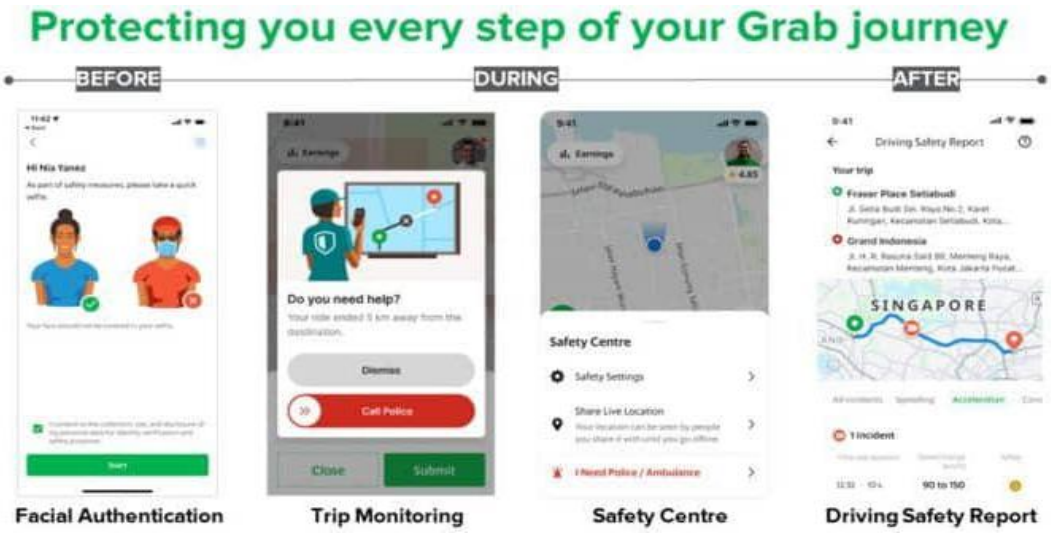


Figure 2.1 Grab journey safety lifecycle

Pre-Trip

Campus-based carpool systems often rely on institutional email domains (e.g. @1utar.my and @utar.my) to verify user identity, creating a closed and trusted environment without requiring complex mechanisms [12, 13]. While this method helps exclude external participants, it remains vulnerable to impersonation; email credentials

and profile pictures can be spoofed, and physical impersonation at pickup is still a risk when visual cues are the only verification [2, 6, 8]. These limitations highlight the need for stronger identity safeguards that go beyond static credentials. Table 2.1 provides a comparative overview of trust and verification mechanisms across campus systems, DriverAuth, commercial platforms like GrabCar, and enhanced models.

Table 2.1 Existing Pre-Trip safety verification listing

Ride-Sharing	Role	Verification Method	Purpose
Campus based Carpool Solutions [12], UTAR's Past FYP DinoPool [13]	**Driver, Rider**	Institutional Email (Proactive)	**Prevent** external communities from joining the system **but prone to impersonation and email spoofing.**
GrabCar [14, 15, 10]	**Driver**	Slow manual screening with license and background check (Proactive)	**Effectively prevent** unqualified or risky drivers from providing the service.
	Rider	Selfie for verification (Passive)	Store selfies for post incident **accountability**; not used for prevention.
		Itinerary Sharing (Proactive)	Allow rider to share their ride detail, real time location and driver information to anyone.
DriverAuth [16]	**Driver**	Repeatedly verify the driver after accepting a new ride request, using: • Swipe gesture • Text-independent voice • Face (Proactive)	**Effectively Prevent** driver impersonation but severely compromise driver's UX and delay the driver from initiating the ride.
SafeShareRide [17]	**Driver**	Driver verification when registering vehicle. (Proactive)	**Effectively prevent** unqualified or risky drivers from entering the platform
	Rider	Itinerary Sharing (Proactive)	Allow rider to share their ride detail, real time location and

			driver information to trusted contacts.
Key Takeaways	**Driver, Rider**	**For a Well-Rounded Safety and UX:** • Institutional Email, • Fast facial verification against existing databases, • Repeated QR code scan to verify rider-driver.	**Prevent** impersonation and ensure trust within a closed community like university without compromising any roles UX.

In contrast, commercial platforms like GrabCar apply different trust mechanisms based on user roles. For drivers, Grab enforces a slow, manual onboarding process involving license scans and background checks, serving as a strong proactive filter against unsafe individuals. For riders, however, Grab prioritises a seamless user experience by implementing quicker facial verification during app use by taking selfie. This verification is not used to restrict access but to store facial data for post-ride accountability in case of safety incidents occurred [14, 15]. This creates a split trust model where drivers are actively tightly screened while riders are passively verified for accountability purposes. This passive approach means any user who uses the app is accepted first (blacklist concept), and the system does not verify every passenger individually; a ride may include multiple unregistered participants, making it unsuitable for a high-trust campus environment.

To address these vulnerabilities, Gupta et al. [16] propose more proactive solutions through their DriverAuth authentication scheme, which focuses specifically on verifying driver identity. To ensure driver integrity, DriverAuth integrates facial, voice, and gesture-based verification in a multi-modal process triggered at each ride confirmation. As shown in its architecture in Figure 2.2, this moves verification from a one-time event at registration to a continuous, dynamic process. However, DriverAuth relies heavily on a stable connection to a central server for biometric processing, which may pose limitations in areas with poor mobile connectivity or where data availability is limited. Additionally, the continuous nature of verification can potentially impact user experience (UX) and practicality issues.

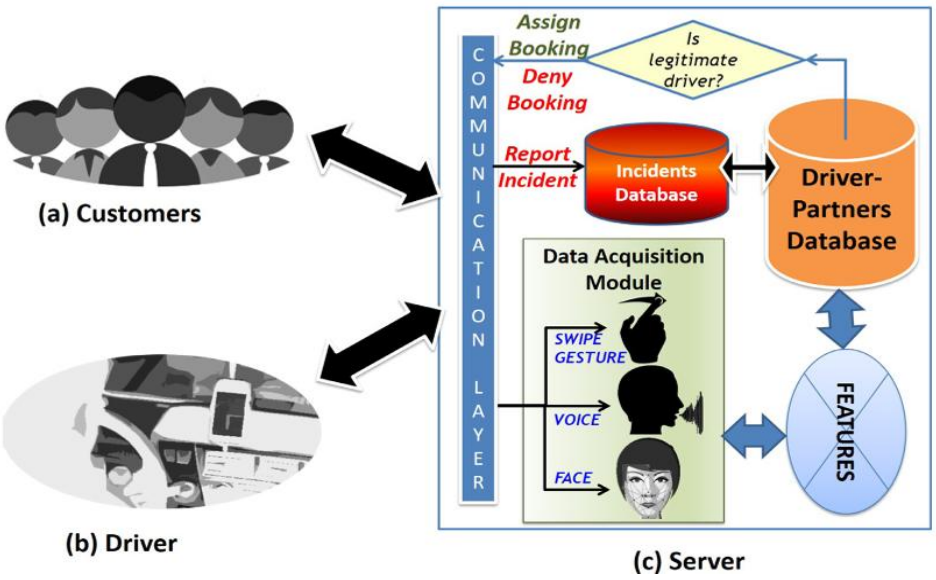


Figure 2.2 New ride assignment flow (DriverAuth [16])

An enhanced approach is therefore feasible by combining fast facial verification for both drivers and riders with institutional email validation, cross-referenced against existing university databases. This dual-layer model enables proactive gatekeeping while preserving post-ride accountability. Unlike continuous biometric checks, a QR-based verification system can be introduced to streamline the process, offering real-time identity validation with minimal user effort. This approach delivers a tightly controlled and enforceable trust framework, surpassing the limitations of open commercial platforms and existing university carpool solutions, without compromising UX for any users.

During-Trip

Chaudhry et al. [5] emphasised the importance of real-time GPS location sharing and an easily accessible emergency panic button to enhance passenger safety during a ride. While these features provide reactive protection, they depend heavily on user interaction. If a passenger is unable to activate the SOS button, assistance may be delayed or unavailable, especially when the emergency feature in GrabCar is not directly displayed but nested within the “Safety Centre” [10]. Likewise, location sharing requires manual monitoring by friends or family, which may not be consistently reliable. These limitations underscore the need for more proactive and automated safeguards during the trip.

Grab addresses this gap through its Trip Monitoring Technology [10], which automatically detects potential safety violations by analysing ride status, GPS data, traffic conditions, telematics (Like speeding, breaking in shock and hard cornering information from phone sensor), and map context. This system can identify unplanned stops, route abnormalities, or accidents and alert a central mediator without requiring user input. However, such a system may be too costly for campus-based carpool platforms. A more practical alternative would be geofencing, it can be used to ensure drivers and passengers remain within a designated service area, triggering alerts if driver surpass the boundary. Additionally arm with an audio-based distress detection system that responds to keywords or signs of conflict, such as repeated shouting or phrases like "Help" could offer a more responsive and automated safety layer beyond the limitations of a static panic button [10].

MYSafeZone, an emergency response application utilising AI context, was developed during the UMPSA Hackathon where it's earned the 1st Runner Up award. This recognition validates the "Trigger-Analyse-Response" framework as a viable proof of concept for the UniRide safety layer [11]. As illustrated in Figure 2.3, the architecture operates in three distinct phases:

- **Pre-Trigger Loop:** The system monitors specific Keyword Triggers (via Microphone) or Motion Triggers (via IMU magnitude thresholds) in a low-power background state.
- **Post-Trigger Data Collection & Analysis:** Upon activation, the system captures an 8-second window of multi-modal data. This context is processed by the Gemini API, which generates a high-confidence binary verdict (True/False) to identify incidents like robberies, falls, or vehicular crashes. This dual-sensor approach significantly minimises False Positives.
- **Emergency Response:** Once validated, the system maintains live GPS updates, secures evidence in cloud storage, and notifies nearby authorities to facilitate rapid intervention.

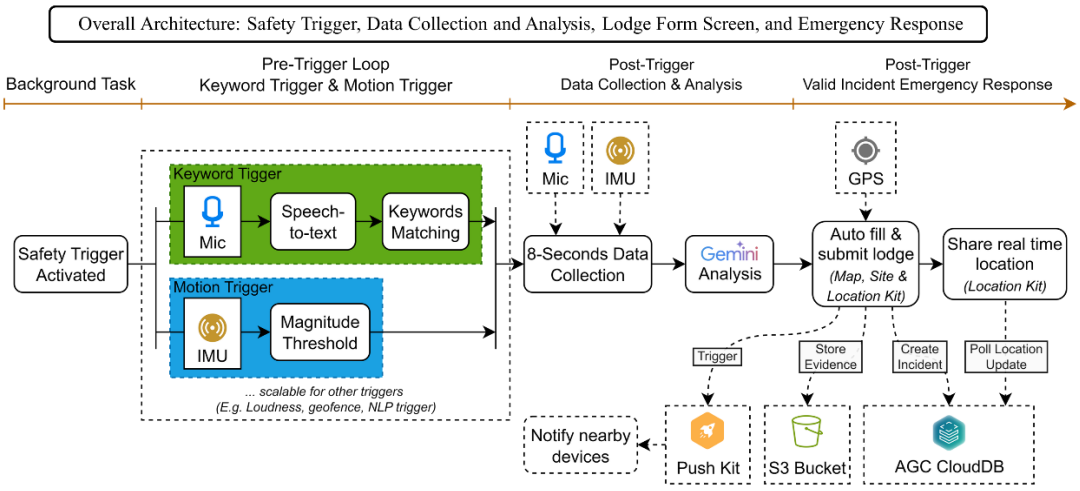


Figure 2.3 Architecture of MYSafeZone safety trigger and emergency response

Despite its success as a multi-modal emergency trigger, the original MYSafeZone architecture was limited by its reliance on speech-to-text keyword and phrase detection. This linguistic-based approach was fundamentally unable to interpret critical non-verbal acoustic signals like high-pitched screams or the impact sounds of a collision which are often inaudible or unintelligible to traditional transcription engines. While phonetic matching was considered, the extensive manual input required for phonetic dictionaries proved impractical for capturing the diverse variations of human distress vocalisations.

However, due to the modular and scalable nature of the MYSafeZone architecture, the system was designed to support diverse triggering inputs without requiring a fundamental redesign. This architectural flexibility allows for the seamless replacement of the Keyword Trigger module with a high-fidelity, AI-driven acoustic classifier at the Pre Trigger phase. By swapping the linguistic-based input for an acoustic-based pattern recognition system, the UniRide module can recognise nuanced, non-verbal distress signals while maintaining the downstream logics.

Post-Trip

Li et al. [4] and Acheampong [6] suggest that implementing a rating system for both drivers and riders is essential for building and maintaining long-term trust and safety in ride-sharing services. These systems help identify and filter out users who constantly display misconduct or problematic over time, thereby preserving the integrity of the platform and reinforcing user confidence in its safety standards. GrabCar [10], for instance, improves its rating system with a Driver Safety Report based on telematic

data, offering insights into driving behaviours such as speeding, hard braking, and phone usage. Therefore, drivers who involve in hazardous behaviours on the road will be penalised while those exhibit safe driving habits may receive incentives, encouraging continued positive behaviour.

While rating systems and safety reports are effective, a campus-based carpool platform may benefit from incorporating gamification elements to promote engagement and trust. For example, consistently well-rated participants could earn "Verified" or "Trusted" badges; like trust indicators used in e-commerce and social media platforms. These badges serve not only as lightweight trust signals for riders but also as motivational rewards, reinforcing positive behaviours through visible recognition without requiring complex backend infrastructure. This approach not only enhances perceived safety but also fosters a sense of accomplishment of drivers and riders. Furthermore, drivers are also given access to evaluate rider behaviours, supporting the next cycle of trip safety and reinforcing trust within each community that uses the carpool system.

2.1.2 Availability and Reliability

Service availability in non-commercial carpooling systems depends entirely on voluntary driver participation, a model that struggles to reach the critical mass of users needed to generate consistent availability and momentum for a reliable service. These systems typically rely on weak incentives such as volunteerism, environmental concern, or basic cost-sharing for resources like fuel, which may not appeal to all participants [16]. The key advantage of this method is its focus on fostering a community-driven, non-commercial model. It avoids the complex regulatory burdens faced by commercial services and is relatively simple to implement. As shown in Table 2.2, factors like cost savings are significant motivators for driver and ride, yet purely voluntary systems fail to provide strong enough financial rewards to ensure driver consistency.

Table 2.2 Motivation and demotivation factors using carpool services

	Support (%)		
	Driver	**Passenger**	**Both Role**
Motivation Factors			
A guaranteed ride home	38.6	56.7	56.8

Cost of gasoline	29.8	35.3	45.2
Cheaper than driving alone	29.8	35.0	46.6
Discouraging Factors			
I have irregular schedule	61.4	59.0	67.6
I do not like to depend on stranger	71.9	64.5	60.9

In contrast, commercial platforms maintain high availability by offering financial benefits and using dynamic pricing models to actively balance driver supply with passenger demand; economically favouring the driver compared to rider. The main weakness of the non-commercial model like carpool is that minimal incentives are insufficient to motivate a consistent demand of drivers. This leads to variable and undependable service, which is unfitting for students and staff with fixed and important academic schedules. Inconsistent availability becomes a major obstacle that prevents the system from scaling. When driver supply collapses under high rider demand, adoption rates remain persistently low, often around 13% in institutional environments, as reported by Li et al. [4].

An older study in 2013 by Graziotin [9] proposed a psychologically attractive incentive: reducing parking fees for drivers who participate in carpooling. This approach directly appeals to cost-conscious individuals and could be effective, but it requires institutional or governmental sponsorship to function. There is still potential for implementation. Building on gamification strategies, drivers who consistently behave well, participate frequently, and transport the most riders could be granted privileges such as priority bidding or discounts on campus parking. This makes earning incentives more engaging and competitive, which could increase carpool availability; especially in institutions where parking spaces are scarce and may requiring bidding while also fulfilling the convenient parking.

To narrow this gap, a hybrid incentive strategy can be introduced. A novel Earn on Demand (EOD) framework based on tiered reimbursement allows drivers to set a low, university-capped fare (e.g. RM1 bus fare) or offer free rides. This provides tangible motivation while preserving affordability and the original spirit of carpooling. During periods of high demand, drivers may earn up to four times the base fare per trip for a five-seater vehicle, effectively offsetting fuel costs while reducing the carbon footprint of four individual passengers. This model is supported by the integration of

E-Wallet services such as Touch ‘n Go, which enable flexible micro-transfers starting from as low as RM0.01, ensuring seamless and scalable fare handling fulfilling the flexible carpool fare needs.

To improve reliability, the system can incorporate a novel semester-based scheduling and bidding feature. Drivers submit their timetables and preferred departure times to and from campus, while riders commit to recurring trips by bidding for seats over the course of the semester. This transforms the service from ad-hoc arrangements into a dependable, schedule-based transportation option tailored to the university community. To ensure greater scheduling integrity, drivers should confirm their trips as the scheduled date approaches. This prevents last-minute cancellations and ensures accountability. If driver cancels the carpool without a valid reason, their credibility score may be reduced, similar to how safety violations are handled. This mechanism helps protect institution community from sudden trip disruptions and supports stable academic planning by giving them sufficient time to replan the transportation.

2.1.3 Inflexible Routing and Trip Searching

Traditional carpooling systems are often constrained by rigid origin-destination (O-D) matching, where participants must have nearly identical predefined start and end points as the driver, as noted by Liu et al. [17]. This approach is particularly effective because it combines computational simplicity with minimal route deviation, which is critical for user-friendly, voluntary carpooling systems.

However, the fundamental weakness of rigid O-D matching lies in its strict requirement for exact destination alignment. This constraint severely limits the pool of potential matches, resulting in low match rates and underutilisation of the system. As illustrated in Figure 2.4, a trip from point O to destination D may include several viable drop-off points that are omitted by the system. These convenient “pass-by” locations are often overlooked, even when they fall logically along the driver’s route.

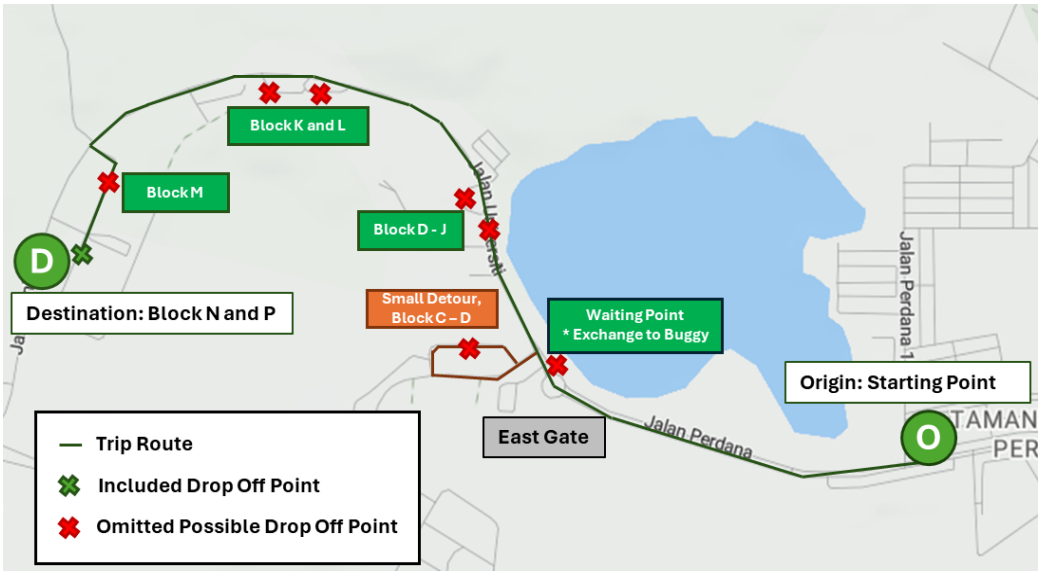


Figure 2.4 Missing opportunity for past by drop-off zone

A literature overview by Si et al. [2] highlights that this limitation is especially pronounced in geographically dense environments such as university campuses. For example, UTAR features a circular layout and dual-gate entrances, which present numerous routing opportunities. In such contexts, flexible routing could unlock significantly more possible matches and maximise the potential of the carpooling system.

To overcome the inefficiencies of rigid O-D matching, recent research has explored algorithmic innovations aimed at improving ride-matching flexibility and system scalability. Meshkani and Farooq [18] proposed GMOMatch, an algorithm leveraging graph structures to enable many-to-one ride matching aimed to optimise shared trips. When tested on Toronto's downtown network, GMOMatch significantly reduces vehicle kilometres travelled (VKT) and traffic travel time, although it requires substantial computational resources and careful parameter tuning.

In parallel, Shahi et al. [19] conducted a comparative study of pathfinding algorithms and found that Contraction Hierarchies (CH), despite their high preprocessing demands, are highly effective at reducing trip times and adapting to dynamic traffic conditions. These findings suggest that integrating clustering techniques, graph-based matching algorithms, and efficient navigation systems can enable reliable, real-time carpooling platforms. Such systems are particularly useful for enabling flexible pickup and drop-off arrangements, including fetching additional participants along the route before reaching the destination.

Building on these insights, a more effective system should replace rigid O-D matching with a flexible routing model that incorporates convenient pass-by drop points and emphasizes a unified pickup location. By applying route optimisation algorithms such as A* or Dijkstra to a campus grid map, the system can redefine a "match" not solely by identical destination endpoints, but by overall route compatibility. This includes intermediate red crossed drop-off points ignored in the Figure 2.4 above. The inclusion of rider tracking can further increase availability by allowing drivers to conveniently pick up riders at alternate points along their route. This enables dynamic identification of riders whose journeys align with a driver's path, requiring only minimal deviation to complete the trip. As a result, matching efficiency improves, destination coverage expands, and the overall usability of the carpooling platform is significantly enhanced.

To ensure that drivers remain motivated even when slight detours are needed, the EOD framework can be integrated to offer tiered incentives. This mechanism compensates for additional effort while preserving affordability and reinforcing participation. By aligning route flexibility with incentive responsiveness, the system becomes more resilient and scalable, eventually enhancing service availability and making carpooling as a better alternative.

2.1.4 Existing Model and Designs

To address critical adoption issues in carpooling, particularly safety and service reliability, researchers have proposed targeted technological architectures. Real-world applications such as the Grab App also offer practical insights into how ridesharing platforms, and carpool systems in particular, are designed and operated.

DriverAuth by Gupta et al. [16] is a proactive, server-based system designed to prevent impersonation by using multi-modal biometrics (face, voice, swipe) to authenticate a driver's identity before each ride acceptance as shown in Figure 2.5. On the left, the diagram illustrates the client-side interface used by the driver, along with its local processing logic prior to server transmission. The right side represents the server-side architecture, which handles biometric verification and decision-making after receiving data from the network.

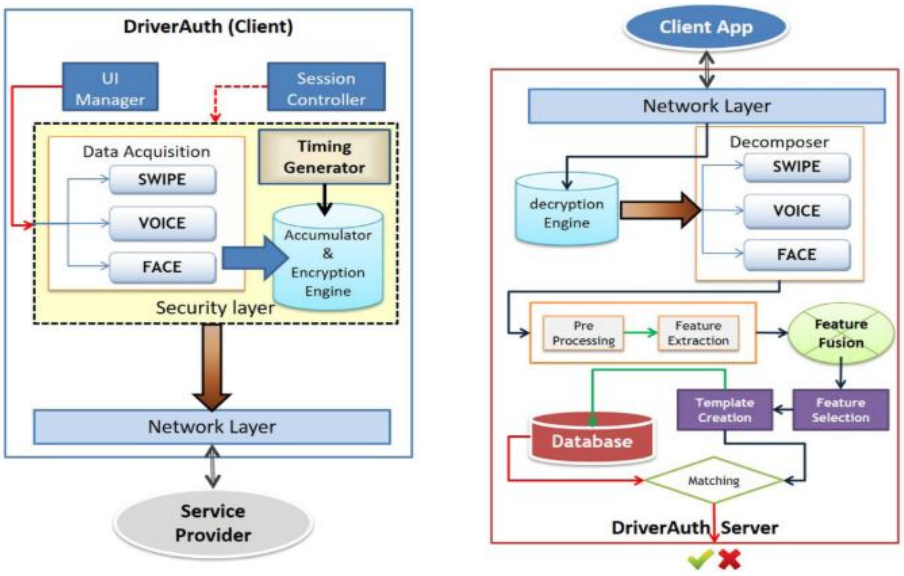


Figure 2.5 DriverAuth architecture

According to the architecture, once a driver accepts a ride, the client-side application triggers a blocking-call process to sequentially gather biometric data, including swipe gestures, voice samples, and facial images, ensuring all required inputs are captured before continuing. These data points are then processed through the acquisition module and passed to the accumulator and encryption engine, where they are encrypted, timestamped, and securely transmitted to the server. Upon receipt, the server decrypts the information and separates it into individual biometric modalities. Each modality undergoes signal preprocessing and feature extraction, after which the extracted features are combined using a multimodal fusion technique and filtered to retain the most relevant attributes for authentication. A driver profile template is generated from these selected features and stored in the database. For identity verification, the same process is applied to incoming data to create a test template, which is then compared against existing records in the database. Additionally, the biometric query is cross-referenced with a flagged incident repository to identify any prior violations or suspicious activity.

This mechanism ensures driver authenticity prior to service initiation, thereby enhancing passenger safety and mitigating potential risks. However, as highlighted in Section 2.1.1, DriverAuth exhibits inefficiencies in data processing because it requires the transmission of encrypted raw biometric inputs over the internet for every ride, leading to noticeable delays and diminished user experience. To overcome this limitation, a QR code-based verification method is proposed as an alternative, enabling

riders to confirm the driver's identity through a randomly generated QR code displayed on the driver's device, streamlining the authentication process. Despite these shortcomings, the DriverAuth framework provides a strong architectural foundation. By shifting preprocessing and feature extraction to the client side and transmitting only encrypted feature data to the server, bandwidth dependency can be minimised. This approach not only reduces data consumption for drivers but also alleviates server load, resulting in a more efficient and cost-effective carpooling system.

Next, SafeShareRide by Zhang et al. [17] is a reactive edge-based system that operates on a smartphone to detect in-trip incidents. Since in-trip systems rely on response rather than prevention, proactive measures must be strictly implemented during the pre-trip phase. The system uses two triggers: audio (via microphone) and driving behaviour (via On-Board Diagnostics, phone sensors, and GPS), which activate video capture that is then sent to the server, as illustrated in Figure 2.6.

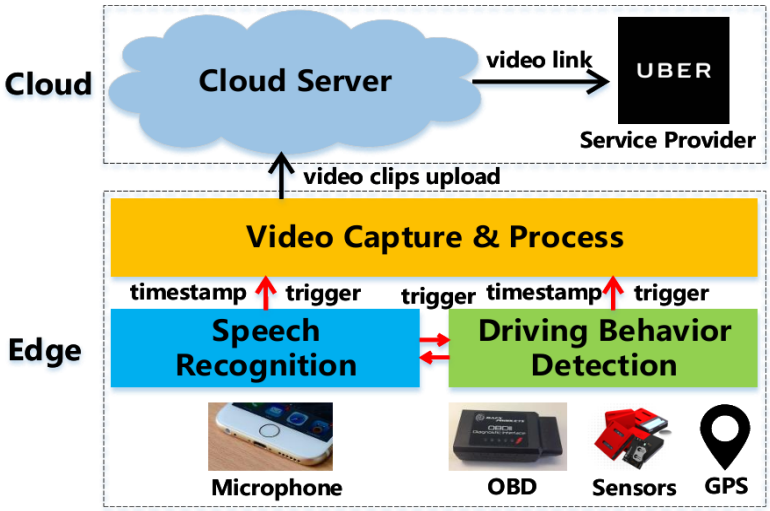


Figure 2.6 Edge based safety mechanism triggers

In the proposed system, speech recognition is used to detect abnormal sounds such as high-pitched noises and screaming, as well as emergency keywords like "Help". This is achieved through speech and keyword recognition models, including Hidden Markov Models for speech decoding and CMU Sphinx for keyword spotting. A key advantage of this architecture is its trigger-based video capture, which conserves user bandwidth and battery while remaining resilient to the unstable connectivity often encountered in moving vehicles.

DriverAuth's primary limitation lies in its reliance on a centralised server for authentication and its exclusive focus on pre-trip proactive safety. SafeShareRide

addresses this gap by introducing in-trip detection and geofencing as reactive safety measures, thereby complementing the overall safety framework. For post-trip, a review system is essential to prevent misconduct by drivers from recurring in future services. Additionally, the scope of these safety measures can be broadened by requiring drivers to authenticate themselves using the same concept as DriverAuth, but with an improved architecture. During-trip safety mechanisms should also be extended to all carpool participants to reduce false negatives, especially since not all devices have uniform microphone and sensor capabilities.

Grab Model

JustSave in Malaysia (known as GrabShare in other countries) is a carpool service provided by GrabCar [20] starting from 2023 May that allows riders to share rides with up to two passengers heading in a similar direction. Users benefit from fare reductions of up to 20% compared to Grab Standard, while drivers earn more per origin-destination (O-D) trip by serving multiple groups simultaneously, saving time and fuel. If any group cancels the ride, a penalty fee is applied to the cancelling party, while the fare for the remaining group remains unaffected. However, this service is only available in high-demand areas in Malaysia, such as KLCC and the Klang Valley, due to its reliance on dense urban zones with a high volume of riders traveling to popular destinations.

Despite its benefits, JustSave has several critical limitations. First, it does not maximise seat capacity, restricting each trip to only two participants regardless of vehicle size. Second, the absence of advance booking or a pre-trip bidding mechanism can result in only one participant being matched, resulting in a standard Grab trip. This undermines the cost-saving intent, leads to longer waiting times, and causing inconsistent fare reductions.

This model could be improved and adapted for university areas with dense student travel to common destinations. Such settings offer a promising opportunity to extend carpool services like JustSave outside urban centres and into academic communities. The predictable campus destinations increase rider demand. Furthermore, maximising seat utilisation and establishing designated grouping points, where participants share similar O-D routes are essential to the efficiency of carpooling. This approach minimises detours and pickup delays, preserving driver convenience while

significantly improving upon the limitations of the current JustSave carpool model. Moreover, the maximum 20% fare cut on top of standard grab fare could not make the carpool solution cheaper, e.g. RM2.00 reduced to RM1.60, which is a minimal discount.

The analysis of existing carpooling systems: DriverAuth, SafeShareRide, and Grab's JustSave highlights key limitations in safety, efficiency, and economic viability across different ride phases. DriverAuth offers strong pre-trip authentication through multimodal biometrics but suffers from server dependency and processing delays. SafeShareRide complements this by introducing reactive in-trip detection using smartphone-based edge processing yet lacks broader participant verification. JustSave, while commercially deployed, restricts capacity to two passengers, lacks advance booking, and offers only marginal fare reductions, making it less effective in practice.

These gaps suggest a need for a unified, context-aware framework tailored to university environments, where predictable travel patterns, dense student populations, and static destinations offer ideal conditions for scalable carpooling. By integrating proactive authentication, in-trip monitoring, post-trip accountability, and dynamic seat utilisation with designated grouping points, a more resilient and student-friendly system can be developed. Combining insights from research and real-world applications enables the creation of an economically viable carpool model that enhances safety, reduces cost, and supports sustainable mobility in academic communities.

2.2 Limitation and Proposed Solution Comparison

The analysis of existing carpooling systems, research, technological architectures, and commercial applications, reveals that while carpooling is a promising solution as an alternative sustainable mobility, its widespread adoption is constantly low due to interconnected challenges in safety, reliability, and efficiency.

The literatures shows that current campus-based systems often fail due to weak verification methods and a lack of compelling incentives, leading to low user trust and inconsistent service availability. Specialised academic solutions like DriverAuth and SafeShareRide offer robust but fragmented approaches to safety, addressing either pre-trip or in-trip security without a unified, user-friendly design. While commercial models such as Grab's Standard and JustSave, demonstrating market viability, are limited by restricted capacity, marginal economic benefits, and trust model that cannot

enforced in closed campus community. These gaps highlight the need for an all-rounded solution, context-aware framework tailored to the unique environment of a university campus, where predictable travel patterns and a trusted user base offer ideal conditions for a scalable carpooling model. The following Table 2.3 summarises the primary limitations identified across the literature and contrasts them with the integrated solutions proposed by this project.

Table 2.3 Comparative summary of existing limitations and proposed solutions

Area of Concern	Limitations in Existing Solution	Proposed Solution
Safety & Trust (Pre-Trip)	• Weak verification (e.g. institutional email) is prone to impersonation and spoofing [6, 2, 8]. • Commercial models use passive/split trust that is unsuitable for a closed community [14, 15]. • Specialised academic solutions (e.g. DriverAuth) are robust but can be cumbersome and have high server dependency [16].	• Unified, proactive gatekeeping using institutional email, fast facial verification against university databases, and a lightweight QR-code system for mutual driver-rider verification before each trip.
Safety & Trust (During-Trip)	• Safety features are often reactive (e.g. manual SOS buttons) and rely on user interaction, which may not be possible during an emergency [17, 5]. • Advanced automated trip monitoring is effective but often too costly for non-commercial platforms [10]. • Edge-based detection systems like SafeShareRide are efficient but reactive, detecting attacks in progress rather than preventing them [17]. • While MYSafeZone utilised keyword and phrase detection, this approach was limited to transcribable speech and could not identify non-verbal acoustic signals, such as screams or the high-frequency impact of a car crash. Furthermore, while phonetic matching was	• A multi-layered approach combining geofencing for operational control with automated, audio-based distress detection for proactive, hands-free alerts. • Readapting MYSafeZone safety trigger, analyse and response system using AI context to determine if distress situation had occurred, while incorporating Edge-based trigger system.

	considered, it proved impractical due to the extensive manual input required for phonetic dictionaries and its inability to reliably capture the diverse variations of human distress vocalisations.	
Safety & Trust (Post-Trip)	• Standard rating systems are subjective, can be biased or manipulated, and are a lagging indicator of user misconduct [16].	• A gamified reputation system with objective metrics (e.g. punctuality, low cancellation rates) and visible trust badges ("Verified," "Trusted") to create a more nuanced and reliable trust framework.
Availability & Reliability	• Voluntary participation and weak incentives lead to inconsistent driver supply and unreliability [3, 1]. • Systems fail to reach "critical mass", making them unsuitable for users with fixed schedules and leading to low adoption rates [4, 1].	• A hybrid "Earn on Demand" (EOD) incentive model with capped fares and tiered reimbursement to motivate drivers.
Routing & Efficiency	• Rigid Origin-Destination (O-D) matching severely limits the potential for matches, leading to low system efficiency [2]. • Commercial carpool models (e.g. JustSave) have artificially restricted passenger capacity and marginal fare reductions, undermining their effectiveness [20].	• Integration with the EOD framework to financially incentivise minor detours, maximising both efficiency and driver participation.

2.3 YAMNet for Edge Audio Recognition Task

While the machine learning literature offers theoretically larger audio classification models such as VGGish [21] or highly accurate Audio Spectrogram Transformers (AST) [22], YAMNet was selected as the foundation for the UniRide distress detection module due to its specialised and localised edge-computing efficiency and its suitability for transfer learning.

2.3.1 Comparative Architectural Efficiency

Unlike VGGish, which is built upon the parameter-dense VGG-11 architecture and relies on standard convolutional layers requiring millions of weights; YAMNet utilises a highly optimised MobileNetV1 backend [23]. The defining advantage of YAMNet over other edge networks lies in its use of Depthwise Separable Convolutions. This architectural choice bifurcates standard network convolutions into two hyper-efficient steps: a depthwise spatial convolution, followed by a 1x1 pointwise convolution.

This reduction in mathematical Multiply-Accumulate (MAC) operations shrinks the model footprint to approximately 3.7 MB. As illustrated in Table 2.4, YAMNet provides the optimal balance between the extreme constraints of TinyML models (e.g. MicroSpeech) and the high computational demands of transformer-based architectures. This ensures the UniRide safety monitor can operate in the background without degrading the performance of mobile device during usage.

Table 2.4: Comparative analysis of audio classification architectures

Model	Backbone	Parameter Count / Footprint	Edge Suitability	Semantic Depth (Distress Detection)
YAMNet	MobileNetV1	**~3.7M / 3.7MB**	**High** (Native TFLite & Depthwise Conv)	**High** (pre-trained on 521 AudioSet classes)
VGGish	VGG-11	~62M / 250MB+	Low (Requires Cloud/Server offloading)	**High** (General purpose)
AST	Vision Transformer	~86M / 340MB	Very Low (High CPU/NPU demand)	**Very High** (State-of-the-art accuracy)
MicroSpeech	TinyML CNN	**< 0.1M / < 1MB**	**Very High** (Designed for MCUs)	Low (Limited to Keyword Spotting only)

2.3.2 Suitability for Distress Detection via Transfer Learning

The selection of YAMNet is further justified by its robust training lineage. Trained by Google on AudioSet [24], a corpus of over 2 million human-labelled

YouTube clips; YAMNet has already learned the acoustic signatures of 521 diverse classes. Crucially, this includes event classes essential for carpooling safety, such as Screaming, Crying, Shouting, and Environmental Noise.

In the UniRide context, YAMNet serves as a high-fidelity feature extractor. By utilising a transfer-learning approach, the system bypasses YAMNet's final classification layer and instead extracts a 1024-dimensional continuous embedding [25]. Because YAMNet is already pre-conditioned to recognise the nuances of human distress, the custom binary classifier built for UniRide can achieve high sensitivity with significantly less training data than would be required if training a model from scratch.

2.3.3 Suitability and Performance for the Car Environment

While YAMNet's pre-training on AudioSet [24] provides a strong foundation, its application within the UniRide framework requires specific adaptation to the highly complex acoustic environment of a vehicle cabin. A car interior is not a controlled audio environment; it is characterised by overlapping layers of background noise, including human conversations, engine rumble, wind shear, and tire-road friction, other motorists sound, which vary significantly based on vehicle speed and road conditions.

Because UniRide utilises YAMNet primarily as a pre-trained feature extractor rather than retraining the entire network from scratch [25], the challenge lies in effectively differentiating genuine distress signals from benign, high-energy in-cabin sounds. If deployed without contextual constraints, the model is susceptible to a high rate of false positives triggered by:

- **Road Noise and Environmental Sounds:** Sudden acoustic spikes from passing sirens, heavy rain, or driving over potholes can mimic the sudden energy burst of an impact or physical struggle.
- **Media Playback:** Loud music, particularly tracks containing high-pitched vocals, heavy bass drops, or cinematic sound effects (e.g. sirens or crashes within a song), can be misinterpreted by the model as real-world distress.
- **Benign Human Interactions:** Enthusiastic conversations, loud laughter, or excited shouting among carpool participants share similar frequency ranges and acoustic intensity with genuine distress vocalisations (like screaming or crying).

To mitigate these false positives inherent to pre-trained acoustic models, the UniRide framework does not rely solely on the YAMNet audio classification. Instead, YAMNet acts as the initial, highly sensitive trigger within a multimodal validation pipeline. When YAMNet detects an acoustic anomaly, it prompts the secondary layer the Gemini API (utilising the 2.5 Flash Lite model for minimal-cost, low-latency and high-rate usage) to contextually analyse the audio snippet and YAMNet data alongside kinetic data from the smartphone's IMU. This design ensures that a loud laugh or heavy bass track might trigger the edge model but will be correctly filtered as a false positive during the multimodal validation phase, ensuring alarms are only raised for genuine emergencies.

Furthermore, utilizing YAMNet as an intelligent edge gatekeeper significantly reduces API invocations compared to relying on a simple decibel-threshold sound detection method. A basic volume-based trigger would indiscriminately invoke the API for every loud ambient noise such as a slammed car door or hitting a pothole leading to excessive API calling, high bandwidth consumption, and rapid battery drain. By filtering out structurally benign noises locally, YAMNet ensures the Gemini API is invoked only when the acoustic patterns resemble human distress or hazardous events, thereby optimising the system's operational efficiency and cost.

2.3.4 Edge-Deployment Advantages

For a safety-critical application where response latency is critical, relying on massive models would necessitate offloading audio to a remote cloud server. This introduces critical vulnerabilities including network latency delays, safety loss in internet dead zones, and user privacy violations as passenger audio need to move across internet.

By executing inferences directly on the user’s smartphone via TensorFlow Lite, UniRide ensures that audio never leaves the phone. Furthermore, the efficiency of the MobileNet backbone ensures negligible battery drain and prevents thermal throttling, ultimately proving YAMNet is the optimal planning choice for edge mobile deployment.

CHAPTER 3

System Methodology and Technology

This carpool mobile application development will follow the Kanban methodology; an agile framework based on Lean principles. The project will prioritise the creation of a Minimum Viable Product (MVP) to validate the core carpooling solution with a minimal set of essential features. The initial MVP will include only the core and essential features:

1. A generic carpooling mobile application with user account creation and email domain gate guarding,
2. A basic profile system and the ability for drivers to post a trip and for riders to search and request to join.

Following the successful validation of the MVP, the project will progress by incrementally developing and integrating the planned and additional modules. For continuous workflow, these features will be developed in a prioritised order:

1. Multi-stage user verification and safety system,
2. Simple scheduling system,
3. EOD model, routing algorithm,
4. Distress detection system and model training
5. Semester-based scheduling system,
6. Gamification features (Leaderboard, events, trusted badge system, etc.)
7. More features or improvements as the development progresses.

Machine Learning Methodology

The development of the edge-based safety framework relies on an industrial grade machine learning pipeline for the YAMNet distress detection model. The methodology involves a structured sequence of data acquisition, audio preprocessing and augmentation, model transfer-learning, and comparative evaluation. The system is designed to input environmental and human vocalisation audio data to accurately classify distress events locally on the mobile device, while offloading backend cost.

3.1 Design Specification

3.1.1 Android and User Requirements

This Android Carpool mobile app development is intended for general use and aiming for wide compatibility; therefore, it is designed to function on any device meeting the specified Android specification listed in Table 3.1.

Table 3.1 Targeted Android mobile specification

Description	Specifications
Android	> Android 7.0 Nougat (API 24)
RAM	≥ 4 GB
Storage needed	< 100MB
Sensors	Camera, Microphone, GPS, Inertial Measurement Unit (IMU: Gyroscope, accelerometer and magnetometer)

3.1.2 Software and Services

The app development will use Flutter and its Dart language to utilise its wide support for native Android device hardware features, such as the camera, GPS, microphone, and IMUs which are essential for the core safety systems in the carpool system. This choice also ensures the application is highly portable and scalable for other operating systems, allowing for efficient future expansion to iOS or web app platforms from a single codebase.

The following Table 3.2 represents the tools used for development,

Table 3.2 Software tools and services for development

Tools	Description
Visual Studio Code	IDE for coding, debugging and testing Flutter application.
Flutter SDK	Core toolkit for Dart language, framework libraries, and CLI tools for building apps.
Android Studio and SDK	Required for compiling, running, and debugging Flutter apps on emulators or physical devices.
Git and GitHub Desktop	Version control tracking.

Firebase	Backend-as-a-Service (BaaS) that accelerate the development by eliminating the server management. It also provides managed services that can be utilised for this project and will be further discussed below.
Flask	To host the ArcFace Face Verification model directly, rather than relying on Firebase Functions, to avoid cold start latency and associated costs. This approach also improves embedding generation speed, resulting in faster and more reliable verification.
Cloudflare	Cloudflare is used to forward requests from the main domain to a subdomain, which connects to the Flask service running locally. This setup makes the local Flask app accessible globally under the exact domain name.
Gemini 2.5 Flash/Lite API	Employed as the secondary, multi-modal validation layer within the AI distress monitoring pipeline. It processes contextual data (such as YAMNet acoustic anomalies and IMU sensor data) to validate genuine emergencies, minimise false positives.

Firebase is cloud-based application development platform that provides varieties of software tools and services that can fulfil the user centric safety and high-availability carpool system by utilising the service provided,

Firebase Backend Services:

1. **Authentication**
    - To manage secure user registration and login. This service will be configured to enforce registration only through a verified institutional email domain with Inbox access (e.g. *@utar.my* and *@1utar.my*), requiring email link verification to create a foundational layer of trust within the closed-community environment.
2. **Cloud Firestore**
    - Application Data: Serves as the primary real-time NoSQL database for the system. It stores all dynamic application data, including user profiles (with ratings and trust badges), trip details and history, semester-based schedules, ride bids and booking information, and live driver GPS coordinates, and secure links to emergency audio/video recordings
    - Institutional Directory: A secure dummy collection to represent the institutional student and staff directory. This will contain essential data for identity matching and include secure links to the verification photos stored in Cloud Storage.

3. **Cloud Storage**
    - To securely store and manage the initial biometric templates required for the facial recognition and authentication workflows.
    - To store user-generated, including profile pictures for display within the application.
    - To store encrypted video and audio recordings triggered by the audio distress detection system, making them available for security and incident review purposes.
4. **Cloud Functions (Alternative to self-hosted "Flask + Cloudflare")**
    - Used to execute all server-side logic within a serverless environment. It handles critical operations such as running the ride-matching algorithm, calculating Earn-On-Demand (EOD) fares based on trip data, processing new ride bids, and triggering push notifications for emergency responses through Cloud Messaging.
5. **Cloud Messaging**
    - Responsible for sending real-time push notifications to participants for important events, including ride-matching alerts, trip confirmations, cancellations, and safety triggers activated during emergencies.
6. **Firebase Machine Learning Kit (ML Kit)**
    - May utilise ML Kit's pre-built models, such as the Face Detection API, for efficient and reliable on-device analysis during the user verification process.
    - To host and deploy the custom-trained TensorFlow Lite model for the audio distress detection system. This allows the model to be updated remotely, while the actual audio processing and analysis occur on the user's device for low latency and offline functionality, reducing the network dependency for recognition service in DriverAuth.

3.2 System Safety Overview and Pipeline

The core novelty of the UniRide platform is its Hierarchical Multimodal Safety Framework. Rather than relying on static, reactive safety measures, the system integrates AI across the entire lifecycle of a carpool trip.

To provide a clear understanding of the system's operational flow, Figure 3.1 illustrates the integrated safety pipeline. The architecture is explicitly arranged by trip stage: Pre-Trip, During-Trip, and Post-Trip, demonstrating how visual biometric gatekeeping seamlessly transitions into continuous acoustic and kinematic distress monitoring.

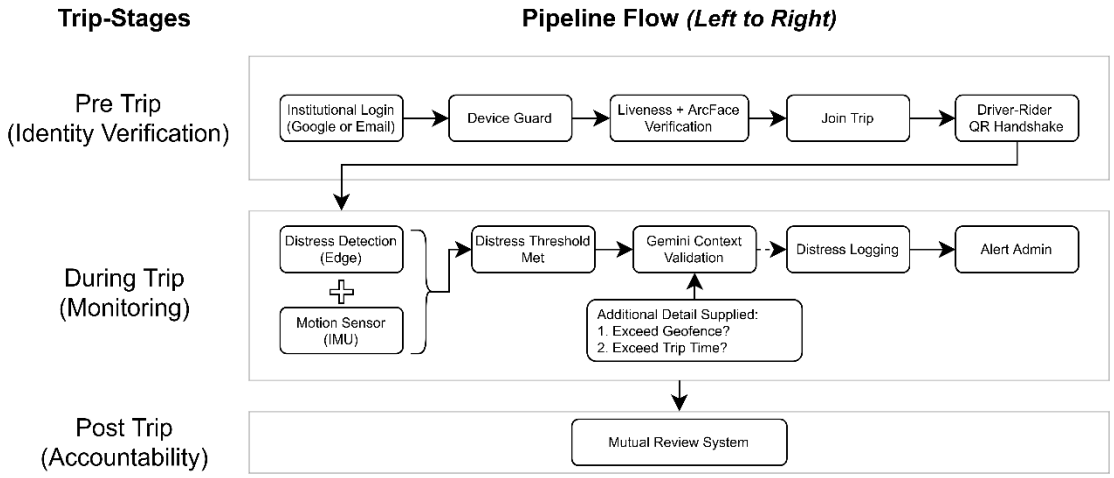


Figure 3.1 Trip safety lifecycle pipeline

3.2.1 Phase 1: Pre-Trip Vision Verification Pipeline

The Pre Trip phase operates as a Zero-Trust gatekeeper to strictly prevent unauthorised access and impersonation before a vehicle is ever boarded. As mapped in the pipeline, this phase follows a sequential authentication flow:

1. **Institutional Login:** The **User (Driver or Rider)** first authenticates using a verified institutional email domain, ensuring they belong to the closed university ecosystem.
2. **Device Guard:** Before proceeding, the **User** must pass an edge-based liveness test (Google ML kit) to prevent spoofing using static images or digital screens.
3. **ArcFace Verification:** Once liveness is confirmed, the captured image is routed securely to the localised Flask API via Cloudflare, where the ArcFace model generates a 512-dimensional embedding and verifies the identity against the institutional database.
4. **Join Trip:** With identity assured, the **Driver** can submit a carpool listing, and the **Rider** can safely browse available listings and join a desired session.

5. **Driver-Rider QR Handshake:** Upon physical meeting, the **Rider** generates a secure QR token. The **Driver** scans this code to confirm attendance and transition the system into the active monitoring phase (During Trip).

3.2.2 Phase 2: During Trip (Monitoring)

Once the QR handshake initiates the trip, the system shifts to continuous, privacy-preserving environmental monitoring to protect participants during the ride.

1. **Distress Detection & Motion Sensor:** Sensors and model operating concurrently on the edge device, the smartphone microphone continuously samples the cabin environment using the highly optimised YAMNet audio model, while the IMU tracks the vehicle for severe kinetic anomalies (e.g. hard braking or impacts).
2. **Distress Threshold Met:** If the acoustic or kinetic readings exceed the predefined safety thresholds, the system flags a potential emergency event.
3. **Gemini Context Validation:** To filter out false positives (such as loud music or road bumps), the anomalous data is sent to the Gemini API. Crucially, the LLM is supplied with additional contextual trip parameters; specifically, whether the vehicle has Exceeded the Geofence or Exceeded the Trip Time. This allows the AI to make a highly accurate, context-aware assessment of the situation.
4. **Alert Admin:** If the Gemini validation confirms a genuine threat, an automated distress signal containing the AI summary and location data is conditionally dispatched to the university administration.

3.2.3 Phase 3: Post Trip (Accountability)

For trips that conclude without severe anomalies, the safety pipeline transitions into a community accountability phase.

1. **Mutual Review System:** Upon reaching the destination, the trip is finalised with an optional rating and feedback system. This allows **User** to review each other, feeding data into the platform's long-term reputation metrics and maintaining a high standard of community trust.

3.3 System Design Diagram

3.3.1 System Architecture Diagram

The UniRide system is built upon a Hybrid Cloud Architecture that strategically distributes computational tasks between the mobile client, managed cloud services, and a high-performance local backend. As illustrated in Figure 3.2 architecture diagram, the system is divided into three distinct operational layers: Client Mobile Device, External Services, and the Local Host Backend.

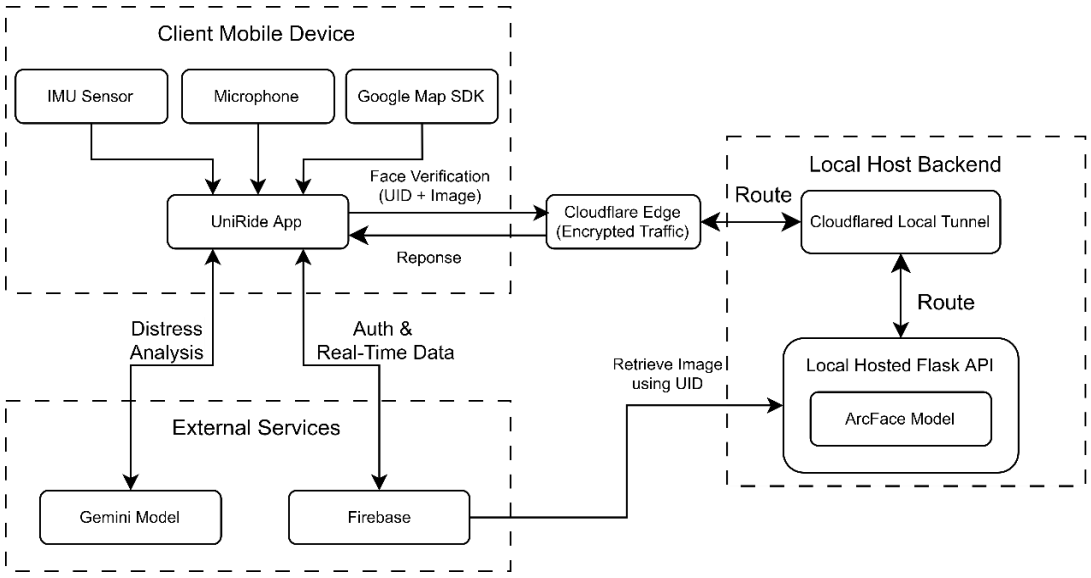


Figure 3.2 System architecture integration diagram

Client-Side Mobile Device

The mobile application serves as the primary data ingestion endpoint. It interfaces directly with device hardware, including the IMU Sensor for kinetic monitoring and the microphone for recording. These inputs are processed locally via TFLite models for immediate triggers, while the Google Map SDK facilitates spatial navigation and real-time route visualisation.

External Services and Multi-Modal Analysis

To maintain data persistence and advanced safety features, the system leverages specialised cloud providers:

- **Firebase:** Acts as the central hub for user authentication, real-time carpool data synchronisation, and secure document storage .

- **Gemini Model:** Provides a sophisticated "Safety Verification" layer. When the app detects potential distress, audio and sensor data are offloaded to the Gemini API for multi-modal analysis to confirm the emergency context.

Local Host Backend (ArcFace)

The most computationally intensive task, generating high-dimensional face embeddings is offloaded to a local server to avoid the latency and cost constraints of serverless environments.

3.3.2 Sequence of Hybrid Face Verification

The interaction lifecycle for a single verification event is detailed in Figure 3.3. This sequence demonstrates how the Cloudflare Tunnel acts as a secure "Zero-Trust" bridge between the public internet and the private local host.

1. **Initiation:** The Mobile App captures a face image and initiates a POST request to a secure endpoint (*face.uniride.cv*).
2. **Traffic Routing:** The request hits the Cloudflare Edge, which recognises the domain and routes the traffic through an Encrypted Stream to the local cloudflared daemon.
3. **Local Processing:** The tunnel forwards the request to localhost:8080, where the **Flask API** utilises the **ArcFace** model to process the face embedding.
4. **Database Validation:** The local API performs a secondary check against **Cloud Firestore** to retrieve the user's stored identity data for comparison.
5. **Secure Response:** Once the verification is complete, a JSON response (Success/Fail) is sent back through the tunnel to the mobile app, completing the handshake in approximately 3 seconds.

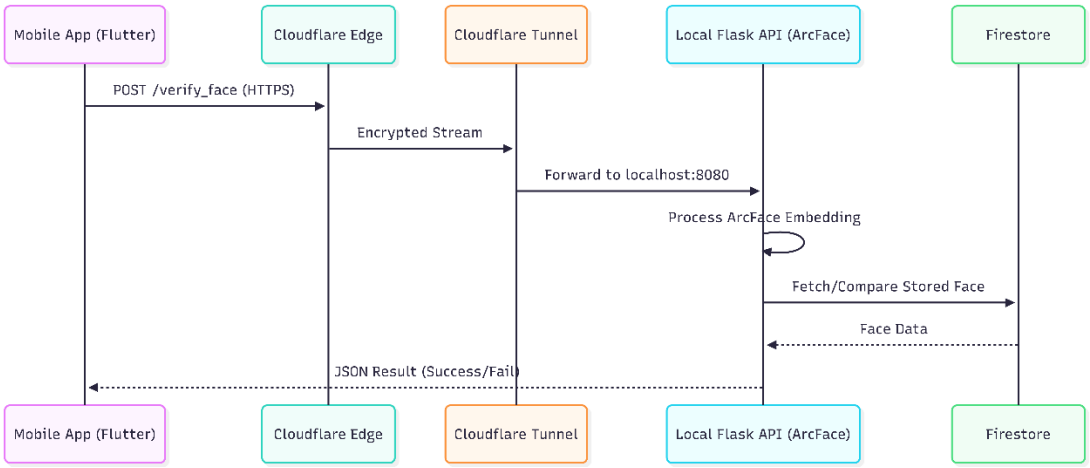


Figure 3.3 Sequence diagram for hybrid cloud face verification

3.3.3 Use Case Diagram and Description

The user-centric architecture of UniRide is designed around a secure "Trip Lifecycle" As illustrated in Figure 3.4, both Riders and Drivers share core foundational use cases, such as Login and Face Identity Verification, ensuring a high-trust environment before any transaction occurs.

User Role: Rider and Driver

- **Pre-Trip Workflow:** Riders engage in carpool discovery via the Search & Join Carpool use case, while Drivers utilise Create Carpool Listing. These flows converge at the View Pre-Trip Screen, where a mandatory QR Scanner handshake (Rider QR Scanner) initiates the physical trip.
- **Safety and Distress Monitoring Logic:** A critical component of the During-Trip Screen is the optional extension to the Distress Trigger. This use case is a composite logic gate:
    - It includes real-time monitoring via the YAMNet based audio distress detector and the Motion Sensor (IMU).
    - The Gemini Analysis use case acts as a secondary verification layer, analysing data from both sensors to contextualise the event.
    - If a genuine emergency is confirmed, the system initiates the Send Distress Alert extension.

- **Post-Trip Operations:** Upon Trigger Trip Completion, the system transitions to the Post-Trip Screen, allowing users to Submit Feedback, which feeds into the system's reputation and safety records.

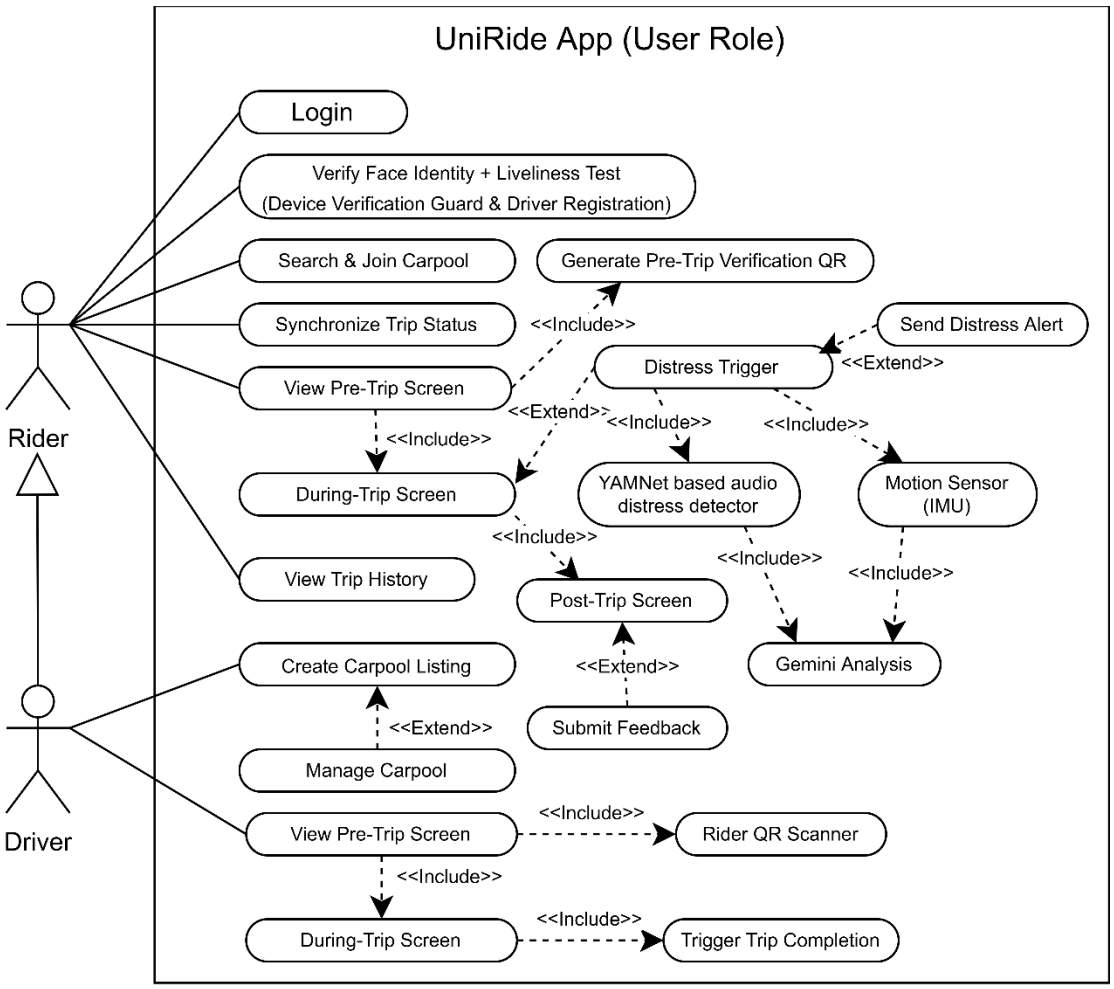


Figure 3.4 Use case diagram for mobile user roles (Rider and Driver)

Use case description:

Shared Role (Rider and Driver)

1. **Verify Face Identity + Liveliness Test:** A mandatory entry-level use case where both roles must undergo biometric verification using the ArcFace model to ensure account security.
2. **Login:** The primary access point for both actors to enter the UniRide ecosystem.
3. **Distress Trigger:** An automated safety logic extends the During-Trip Screen for both roles, monitoring anomalies using YAMNet feature extraction with a classification head for audio distress detection, alongside IMU sensor data.

4. **Gemini Analysis:** A critical processing layer included by the sensor detectors to contextualise potential distress events using the Gemini 2.5 Flash/Lite model.
5. **Send Distress Alert:** A conditional extension that transmits an emergency signal to the Admin dashboard if Gemini confirms a genuine threat.
6. **Submit Feedback:** An optional extension of the Post-Trip Screen allowing both actors to provide reputation and safety data.
7. **Search & Join Carpool:** The primary discovery workflow where Riders browse available listings to join a trip.
8. **Generate Pre-Trip Verification QR:** A mandatory includes of the Rider's View Pre-Trip Screen used to present a secure identity token to the Driver.
9. **Synchronise Trip Status:** A function used by the Rider to ensure their local application state matches the real-time progress of the carpool.
10. **View Trip History:** Allows Riders to review their past completed journeys and transaction records.

Driver-Specific Roles

12. **Create Carpool Listing**: The foundational task for Drivers to publish carpool ride with important information and ride fare per Rider based on EOD.
13. **Manage Carpool**: An optional extension of the listing process, allowing Drivers to manually take Rider attendance or mark them as absent.
14. **Rider QR Scanner**: A mandatory task included in the Driver's View Pre-Trip Screen used to scan and verify the Rider before the trip begins.
15. **Trigger Trip Completion:** A required task included at the end of the During-Trip Screen to formally close the ride session. However, it is geofence-locked, requiring Rider to be within 80m from the destination point to end the trip.

Administrative Role: Admin, Superadmin, and Creator

While the mobile application handles the operational carpooling lifecycle, the back-office governance and system integrity are managed through a multi-tiered administrative hierarchy. As illustrated in Figure 3.5, this separation of concerns ensures that sensitive system configurations and user biometric data are managed with appropriate levels of authority.

- **Administrator (Admin):** The foundational administrative role responsible for high-frequency maintenance tasks. This includes Manage Face Database,

where the admin oversees the lifecycle of ArcFace embeddings, and Manage Stops, which involves updating physical pickup and drop-off points within the university campus.

- **Incident Oversight:** A critical responsibility of the Admin is monitoring the View Distress Alert Dashboard. This base use case is extended by View Incident Detail, a conditional action triggered when a specific distress event validated by the Gemini Analysis layer requires admin end human review.
- **Superadmin:** This role inherits all permissions of the standard Admin but adds strategic governance capabilities. The Superadmin is responsible for Manage University Information and Geofence Area, ensuring the system boundaries align with campus policies, as well as Manage Organisation Users to oversee departmental access.
- **Creator (System Root):** Representing the highest level of the hierarchy, the Creator role manages the global user base through the Manage All Users use case. This role ensures that the underlying Firebase security rules and cloud-to-local tunnel configurations remain synchronised across all organisational tiers.

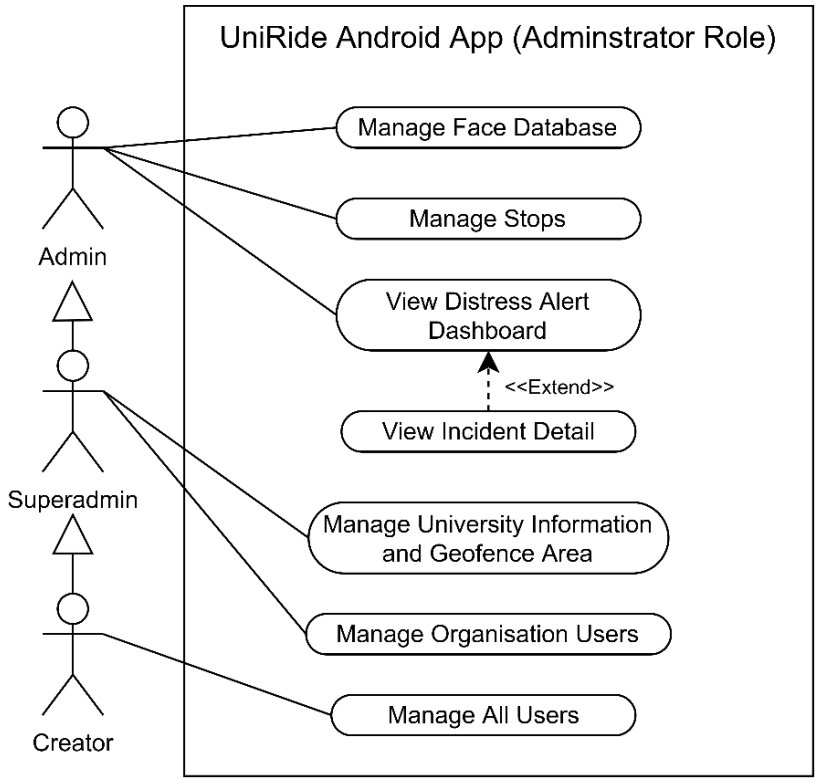


Figure 3.5: Use case diagram system administration (Admin, Superadmin, Creator)

Use case description:

Foundational Administrator Role (Admin)

1. **Manage Face Database:** A core administrative task involving the maintenance of the biometric repository. This includes adding, updating, deleting face to database.
2. **Manage Stops:** Allows the Admin to configure and update official university pickup and drop-off points within the campus geofence.
3. **View Distress Alert Dashboard:** The primary monitoring interface for safety events. It provides a real-time overview of anomalies detected.
4. **View Incident Detail:** A conditional extension of the dashboard. This use case is triggered when an Admin requires deep-dive access into specific event data, including Gemini-contextualised logs and acoustic analysis.

Superadmin Role

5. **Inherited Admin Privileges:** The Superadmin automatically inherits all capabilities of the standard Admin, including database and stop management.
6. **Manage University Information and Geofence Area:** A higher-level management task used to define the geographical boundaries of the UniRide service and update institutional information.
7. **Organisation Users:** Focused on internal access control, allowing the Superadmin to oversee the accounts and permissions of administrative staff within the organisation.

Creator Role

8. **Inherited Superadmin Privileges:** As the highest tier in the hierarchy, the Creator inherits all functions from both the Admin and Superadmin roles.
9. **Manage All Users:** The most expansive administrative use case, providing root-level oversight of the entire user base, including both the mobile users (Riders/Drivers) and the administrative tiers.

3.3.4 Activity Diagram

As illustrated in Figure 3.6, the activity diagram maps the dynamic workflow and chronological sequence of actions within the UniRide system, tracking the user journey from the start of a session to its completion. To clearly depict the separation of responsibilities and parallel processes, the diagram is divided into four distinct swim

lanes: Driver, Rider, System/Safety Logic, and Admin. This visualises the multi-layered interaction between human users and the automated safety framework.

The workflow is categorised into three primary phases: Pre-Trip, During-Trip (with Safety Logic), and Post-Trip.

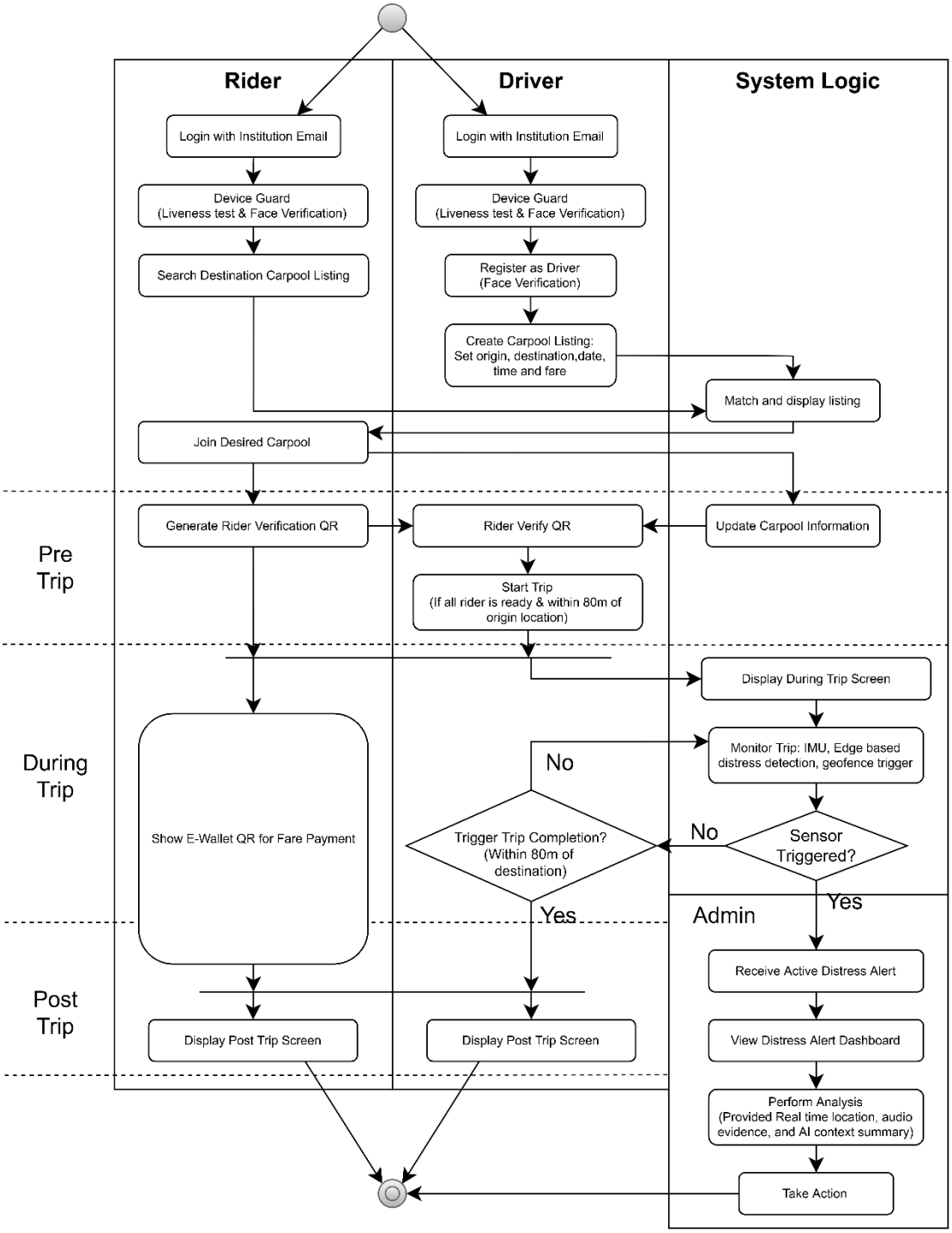


Figure 3.6 Activity diagram with Rider, Driver, System Logic and Admin Swimlane

1. **Pre-Trip Phase (Authentication and Ride Matching)**
    - **User Onboarding:** Both the Driver and Rider initiate their journey by logging in using their verified institutional email. This is immediately followed by a strict "Face Verification & Liveliness Test" to ensure identity authenticity and prevent spoofing.
    - **Driver Preparation:** A user wishing to become a driver must complete a registration process (providing phone number, car details, and an e-wallet QR code for payment) and undergo a secondary face verification. Once registered, the driver creates a carpool listing by setting the fare, origin, destination, and departure time.
    - **Ride Discovery:** The Rider searches for a destination. The System receives the search query alongside the Driver's submitted listing, successfully matching and displaying the available options to the Rider.
    - **Pre-Trip Handshake:** The Rider selects a listing, joins the carpool, and generates a secure "Rider Verification QR". The Driver scans this QR code to confirm the Rider's identity and attendance before officially starting the trip.

2. **During-Trip Phase (Continuous Monitoring and Safety Logic)**
    - **Active Monitoring:** Once the Driver triggers "Start Trip" the System begins continuous background monitoring utilising the smartphone's IMU sensors (for kinetic anomalies like hard braking or crashes) and the edge-deployed YAMNet audio detector (for distress sounds).
    - **Distress Evaluation:** The system continuously evaluates if an anomaly is detected. If the ride proceeds normally (No Anomaly), the flow continues directly to the destination.
    - **AI Contextual Analysis:** If an anomaly is detected (Yes), the system triggers the Gemini AI Contextual Analysis. Gemini evaluates the multimodal data to determine if it is a genuine distress event. If false, the trip continues normally.
    - **Admin Intervention:** If genuine distress is confirmed, the system autonomously logs the evidence (audio and location data) and routes an alert to the Admin swim lane. The admin receives the active distress

alert, accesses the dashboard, monitors the live distress map and AI summary, and coordinates an emergency response.

3. **Post-Trip Phase (Completion and Accountability)**
    - **Geofenced Completion:** To finalise the journey, the Driver must be physically within 80 meters of the destination point. Once this condition is met, the Driver can trigger the trip completion.
    - **Payment and Review:** Upon completion, the Rider is prompted to pay the Driver via the E-Wallet QR code provided. Both the Driver and Rider then proceed to the final step of submitting feedback and reviews, ensuring community accountability, before the activity flow reaches its termination point.

CHAPTER 4

System Design

4.1 Safety Carpool Framework

Figure 4.1 represents the comprehensive safety framework for the carpool application, detailing the mechanisms implemented across the Pre-Trip, During-Trip, and Post-Trip to protect both rider and driver mutual safety.

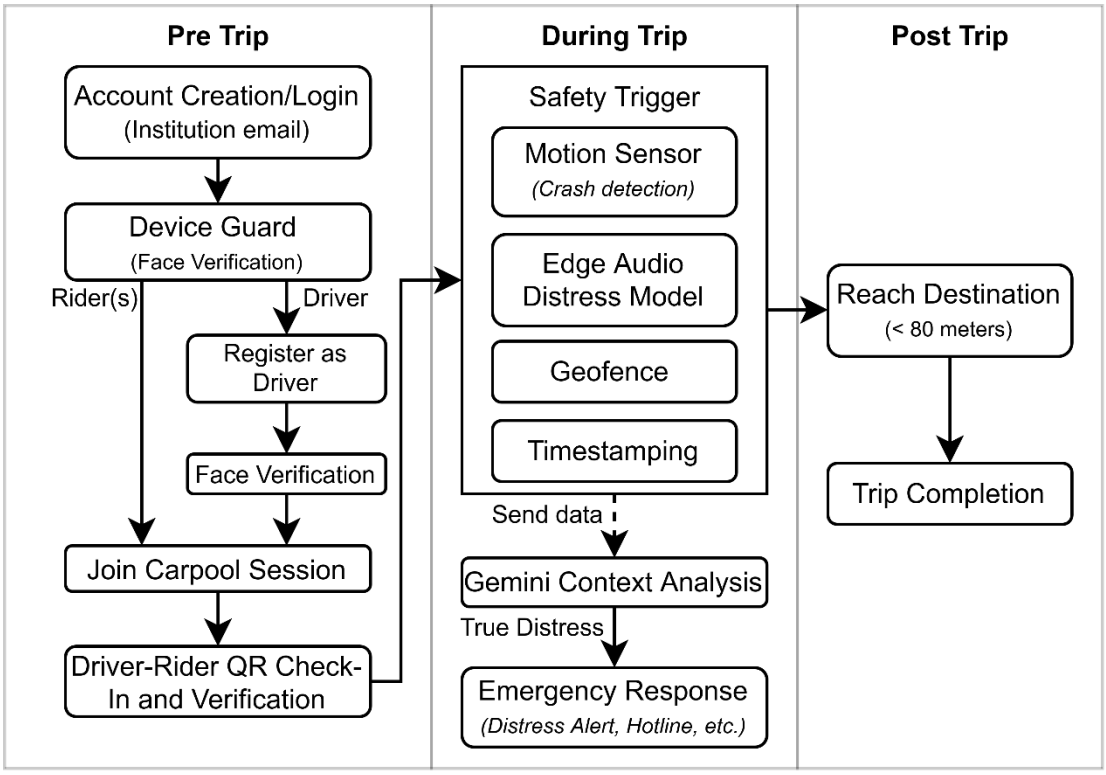


Figure 4.1 Safety mechanism model block diagram

To satisfy the strict requirements of a closed-loop institutional environment, UniRide is engineered primarily as a proactive safety framework rather than a standard ride-matching application. The system acts as a continuous, automated guardian. Table 4.1 outlines the technical logic governing each module within this framework, detailing the precise inputs, models, thresholds, and fallback actions required to maintain zero-trust security and proactive incident response.

Table 4.1: Technical logic of safety framework modules

Safety Module	**Processing Mechanism (Input → Model/Rule)**	**Evaluation (Output & Threshold)**	**Exception Handling (Failure & Fallback)**
Face Liveness Test	**Input:** Smartphone Camera Video Stream **Model:** ML Kit Face Detection (Yaw tracking)	**Output:** Boolean (Pass/Fail) **Threshold:** Yaw angle variance must match direction prompt.	**Failure Case:** Flat image spoofing or static digital screen. **Fallback:** Unable to proceed to face verification; reject access.
Face Verification	**Input:** Smartphone Camera Image **Model:** ArcFace 512-D Embedding	**Output:** Cosine Similarity Score **Threshold:** Similarity > 70%	**Failure Case:** False Reject from poor lighting **Fallback:** Deny from service; prompt manual re-verification.
Motion Sensor Anomaly	**Input:** IMU (x, y, z axes) **Rule:** Vector Magnitude Calculation	**Output:** Total G-Force (g) **Threshold:** > 4.5g	**Failure Case:** False Positive from dropping phone or deep potholes. **Fallback:** Trigger Gemini multimodal validation.
Edge Audio Distress Model	**Input:** 16kHz PCM Audio (Microphone) **Model:** YAMNet Feature Extractor + Classification Head for Distress Detection	**Output:** Binary Classification (0=Safe, 1=Distress) **Threshold:** Probability > 0.50	**Failure Case:** False Positive from loud music, heavy bass, or laughter. **Fallback:** Trigger Gemini multimodal validation.
Geofence Gating	**Input:** GPS Coordinates & Campus Geofence Setup **Rule:** Haversine Distance Calculation	**Output:** Boolean (Inside/Outside) **Threshold:** Outside Campus Geofence Boundary	**Failure Case:** False Positive from GPS drift or dead zones. **Fallback:** Heighten Gemini validation sensitivity and alert Admin.

The following sections detail the safety measures at each stage of the trip lifecycle:

Pre-Trip Phase

1. **Account Creation/Login:** This module utilises institutional email and Firebase Authentication's Google Sign-In to restrict access exclusively to participating institution communities stored in Firebase Firestore. This effectively filters out all external participants because spoofing an email alone cannot bypass Google's OAuth token verification. The logic flow is illustrated in Figure 4.2. Upon logging in for the first time on any device, users are required to complete face verification before accessing the service.

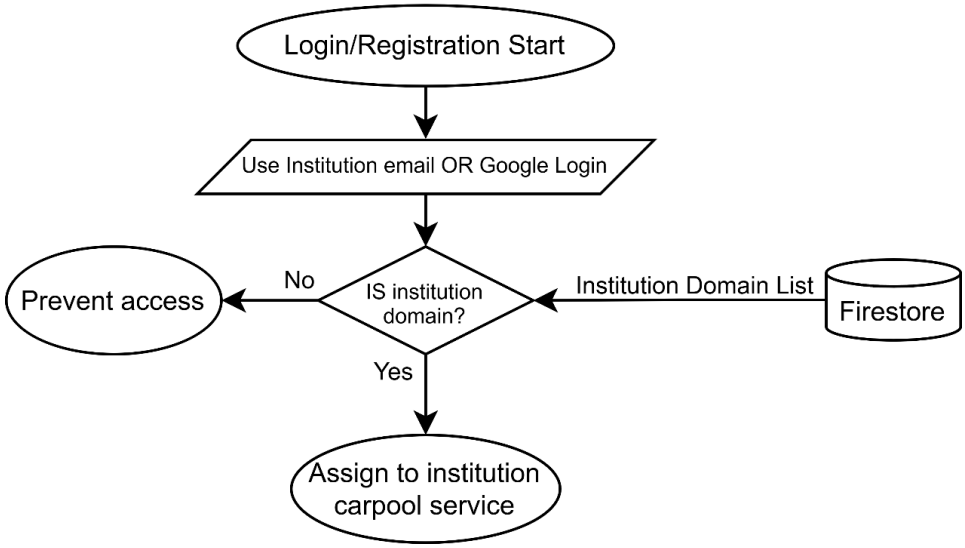


Figure 4.2 Account creation and filter flowchart

2. **Face Verification (Device Guard & Registration):** To establish the baseline biometric identity, a user's facial embedding must first be generated and securely stored within the institutional database. As outlined in Figure 4.3, this initialisation process involves capturing the user's face via the ML Kit detection model, applying standard preprocessing (cropping and upscaling), and performing vectorisation via the ArcFace model to generate a 512-dimensional face vector.

   During subsequent sign-ins or role changes (e.g. Registering as Driver), the system enforces the face liveness test and verification protocol to authenticate the user through a strict sequence (Figure 4.4):

   i. The stored embedding is retrieved from the database (Firestore).

ii. The latest live face input undergoes identical preprocessing and vectorisation using the same model architecture and dimensional parameters.

iii. A similarity comparison is performed using cosine similarity.

iv. If the computed similarity score falls below the predefined threshold (cosine < 0.70), the system determines that the input does not match the original user and rejects access or prompts re-verification.

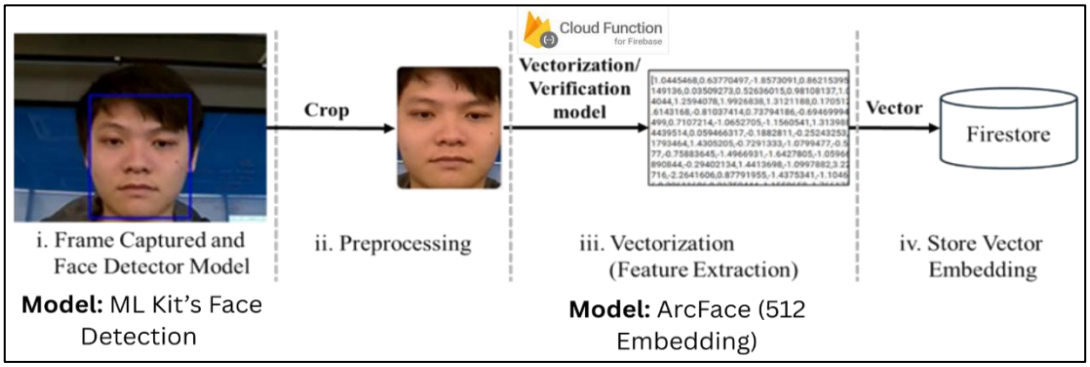


Figure 4.3 Store face embedding/vector

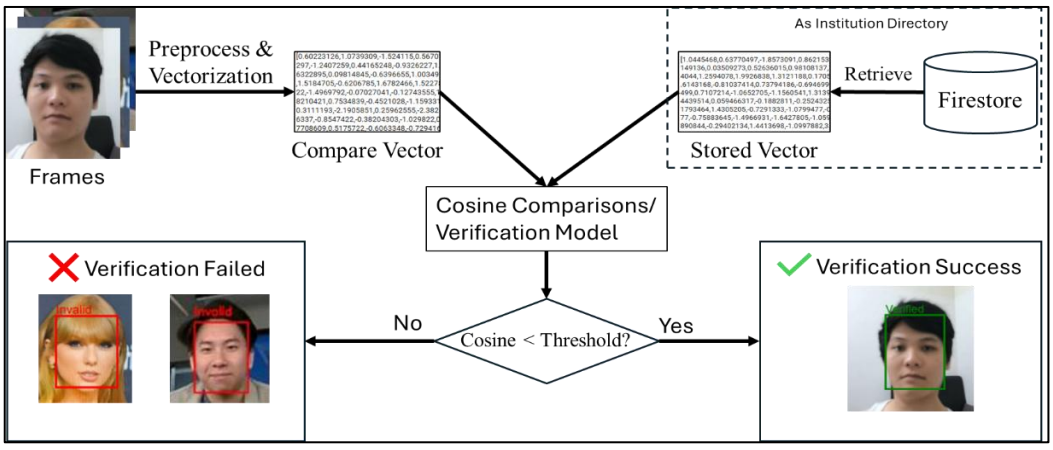


Figure 4.4 Facial verification flow

3. **Driver-Rider QR Code Authentication:** During the pre-trip phase, the rider generates a secure QR token that the driver must physically scan to authenticate the trip, confirming mutual presence and preventing impersonation at the pickup point.

During-Trip Phase

When a trip begins, three safety trigger mechanisms are activated: the driver behaviour detector, audio-based distress detection, and geofence gating rules (Figure 4.5). These

mechanisms are designed to reactively protect participants by initiating an emergency response and automatically lodging distress incidents.

If any trigger is activated, the system captures the last 15 seconds of multimodal data, including IMU readings, distress scores, and audio recordings, and forwards them to Gemini 2.5 Flash/Lite for contextual analysis. If Gemini confirms the event as a true distress situation (rejecting false positives), the Distress Response module is engaged. This module records in-app audio and stores GPS location data in both Firebase Storage and Firestore, ensuring accountability and enabling post-incident analysis. This process fully automates incident handling without requiring user interaction.

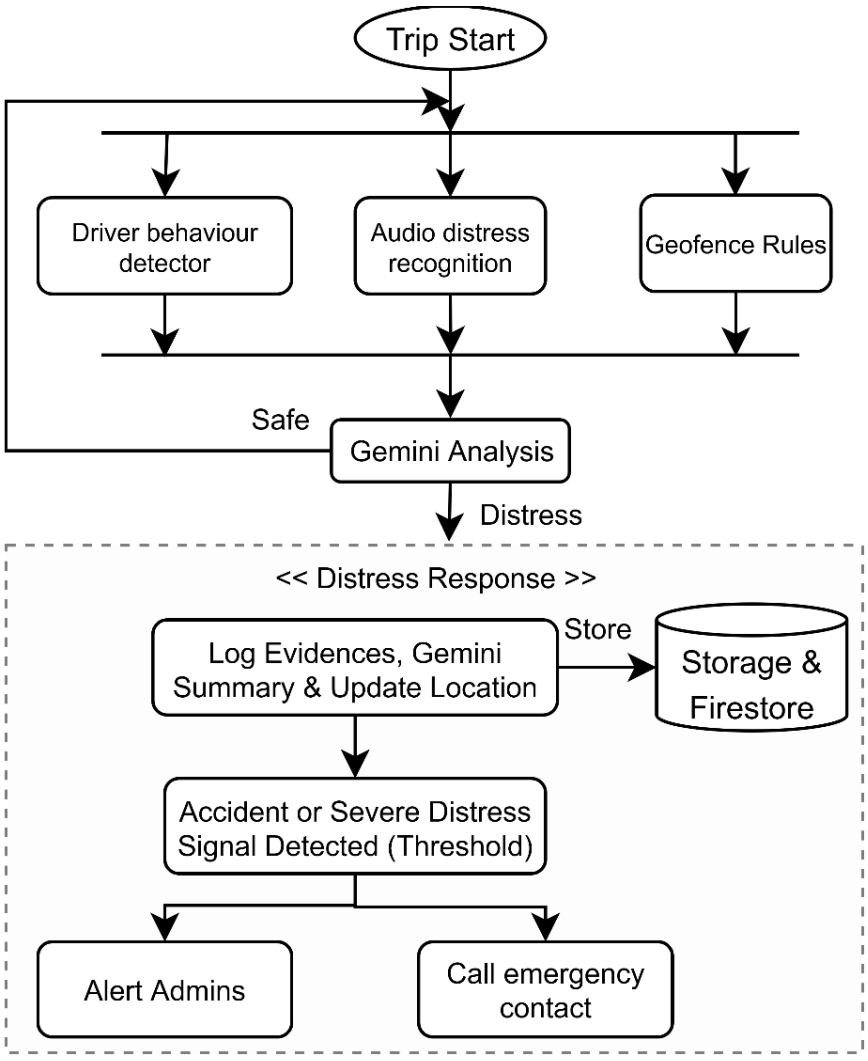


Figure 4.5 During-trip safety mechanism flowchart

Safety Trigger: Driver Behaviours Detector (IMU Motion Sensor)

This module attempts to detect driver driving behaviour by utilising the smartphone's IMU (Figure 4.6):

i. **Sensor inputs:** Accelerometer to identify hard brakes, magnetometer to detect sharp turns, and gyroscope to monitor rotational movement changes.
ii. **Analysis module:** Analyses the sensor data against predefined thresholds to determine possible abnormal or unsafe driving patterns.
iii. **Trigger Decision:** If patterns are detected, it sends a trigger to the Emergency Response Module.

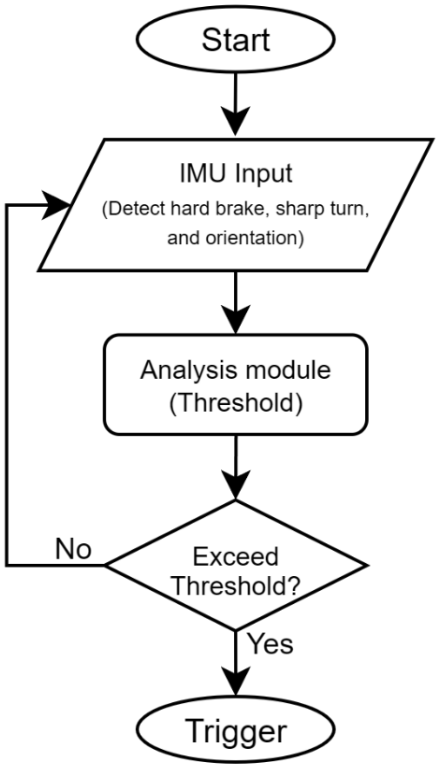


Figure 4.6 Driver behaviours detector flowchart

Safety Trigger: Distress Audio Recognition

This module attempts to recognise distress participants and possible incident from audio to automatically trigger Distress Response module based on logic in Figure 4.7, with following descriptions:

i. **Microphone Input:** Captures real-time audio input,
ii. **Pre-processing:** Applies audio enhancement techniques such as noise reduction, normalisation, and filtering to improve the clarity and accuracy of model detection,
iii. **Distress Detection Model (YAMNet):** Analyses the processed audio for distress-related patterns and output a score,
iv. **Threshold Evaluation:** Compares the score with predefined thresholds,

v. **Trigger Decision:** If the threshold is exceeded, the system sends a trigger to the Distress Response module and passes the trigger value. Otherwise, it loops back to step (i) for continuous audio input monitoring.

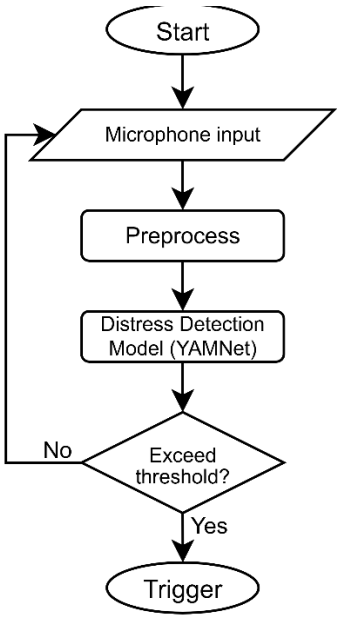


Figure 4.7 Distress audio recognition logic flowchart

Safety Trigger: Geofence Rules Analysis

This module attempts to determine whether the driver is operating within designated carpool service areas provided by universities, as shown in the logic in Figure 4.8, with the following description:

i. **Fetch Geofence Rules:** Retrieves institution-defined geofence parameters, including zone boundaries and service area definitions.

ii. **GPS Input:** Captures the driver's current location using the smartphone's GPS unit.

iii. **Distance Calculation:** the spatial distance from the user's position to the institution location.

iv. **Zone Gating Module:** Evaluates whether the driver's location falls within the defined service area.

v. **Trigger Decision:** If the driver is outside the service area (with certain threshold), the system sends a trigger to the Emergency Response Module. Otherwise, it continues monitoring GPS input, returning to step (ii).

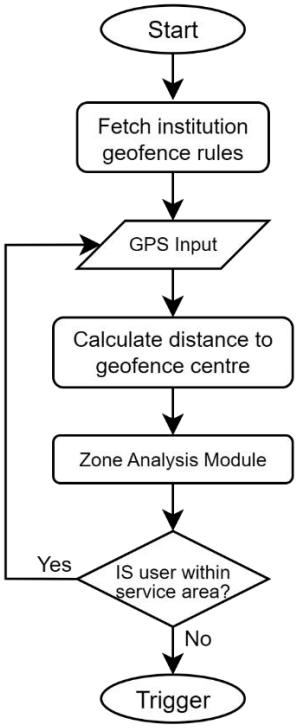


Figure 4.8 Geofence rules analysis flowchart

Post-Trip Phase

This phase is activated when participants reach their desired destination. The user will be prompted to submit a driver review, while the driver can report any rider mischief through a dedicated interface to support the carpool community and feedback cycle. These data points are essential to ensure that no driver or rider is harmed during a trip. Additionally, the records are tied to gamification purposes, awarding verified badges to users who consistently participate safely, thereby improving their trust metric for future trips.

4.2 Carpool Model and EOD

The carpool matching system and Earn-On-Demand (EOD) economic model serve as the operational vehicle for the safety framework (Figure 4.9). This structure enables flexible trip creation while maintaining institutional oversight.

- **Pre-Trip Phase (Trip Creation & Participation):** The driver initiates a trip by setting the origin, destination, and a per-participant rate (RM0-RM1.00) strictly capped by the EOD model. Riders select their destination and join the secure session.

- **During/Post-Trip Phase (Payment):** Upon arrival at the destination, riders' complete payment via an E-Wallet QR code. The system logs the transaction and updates the driver's earnings in real time. This EOD model promotes financial flexibility for drivers and encourages repeated, safe participation within the platform.

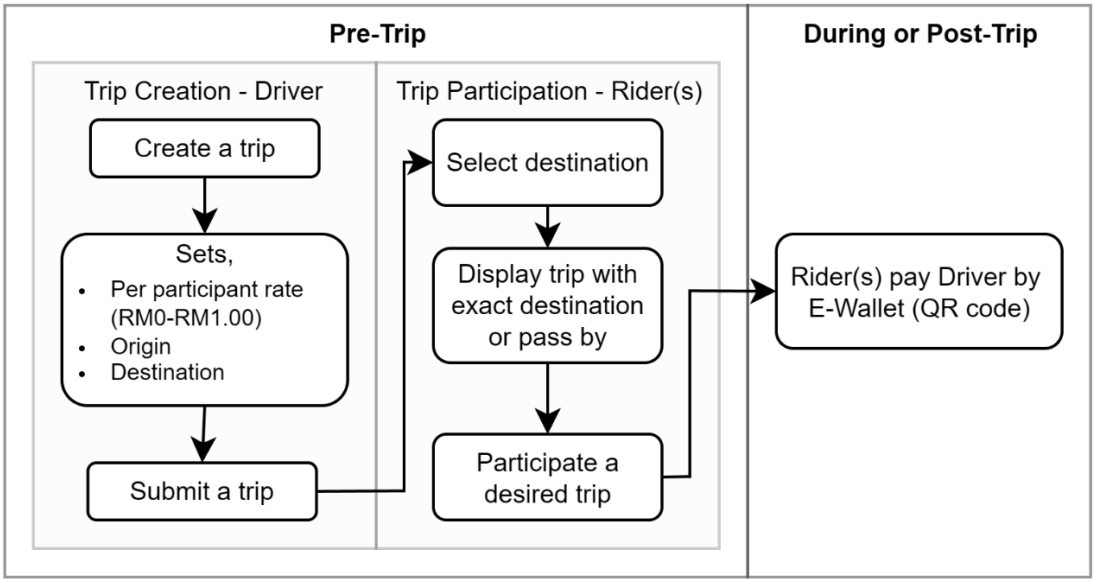


Figure 4.9 Incentivised carpool model

4.3 Distress Audio Recognition Model Architecture

To enable real-time reactive safety monitoring inside the carpool vehicle, an audio distress recognition model was designed using a transfer-learning approach. The system uses YAMNet, a deep convolutional neural network (CNN) pre-trained by Google on the AudioSet-YouTube dataset, acting as a powerful acoustic feature extractor.

Technical Hyperparameters and Configuration

To satisfy the strict demands of edge-deployment while avoiding overfitting, the deep learning pipeline was configured with the following technical details:

Input Data & Preprocessing Configurations:

- **Audio Sampling Rate:** 16,000 Hz (16 kHz) Mono PCM.
- **Chunk Size:** 0.975 seconds (equivalent to 15,600 samples per chunk).

- **Preprocessing:** Audio normalisation and waveform framing prior to mel-spectrogram generation.

Model Architecture & Training Hyperparameters:

- **Feature Extraction:** Frozen MobileNetV1 backbone for extracting a 1024-Dimensional mean-pooled embedding.
- **Classifier Head:** Custom Fully Connected (FC) network incorporating a Gaussian Noise layer (0.1) for regularisation, two hidden Dense layers (512 and 128 units) with ReLU activations, Batch Normalisation, and Dropout (50% and 30%), terminating in a Sigmoid activation function.
- **Optimiser Configurations:** To determine the most effective learning strategy for the highly variable acoustic data, two distinct optimiser configurations were implemented for comparative evaluation:
    - **Model A:** AdamW (utilising a weight_decay=1e-4 to introduce L2 regularisation, aiding in the prevention of model overfitting to the augmented audio chunks).
    - **Model B:** Standard Adam (serving as a baseline to evaluate the necessity of explicit regularisation for this specific dataset mixture).
- **Learning Rate:** 0.0005 (equivalent 5e-4).
- **Batch Size:** 32.
- **Epochs & Patience:** 100 epochs, with Early Stopping callback with a patience of 4 to monitor validation loss and automatically restore the best performing weights.
- **Class Weighting:** Applied dynamically during training to address the inherent dataset imbalance, heavily penalizing misclassifications of the “Safe” class to calibrate the distress threshold.
- **Inference Mode:** TensorFlow Lite (TFLite) executed natively on the mobile smartphone edge.

As illustrated in Figure 4.10, the 1024-D embedding bypasses YAMNet's original classification layer and is fed into the custom-built classifier network to output a binary probability: Safe (0) or Distress (1).

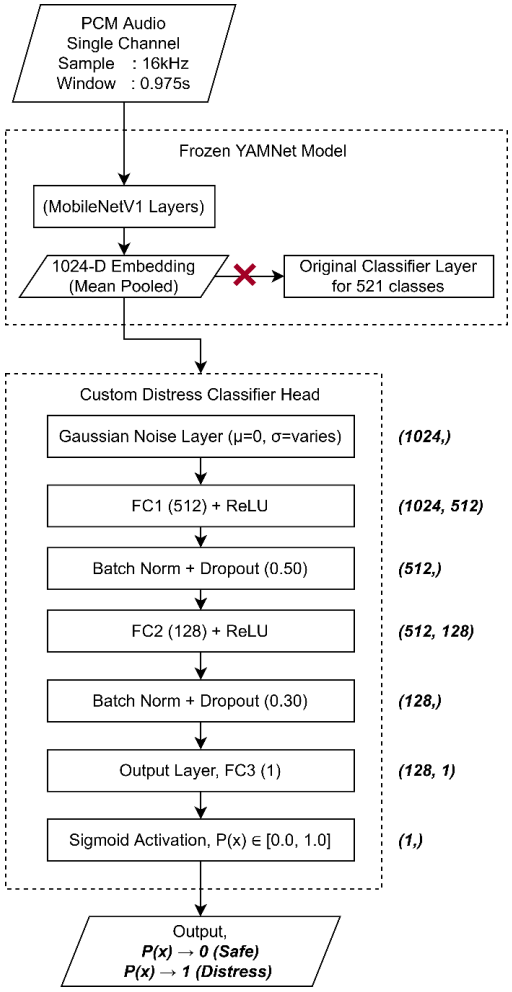


Figure 4.10 Architecture for YAMNet transfer-learning for distress detection

The custom top-layer classification head is structured sequentially. The precise parameter breakdown for the trainable layers is detailed in Table 4.2:

Table 4.2 Parameters for YAMNet transfer-learning for distress detection

Layer Type	Output Shape	Trainable Params	Non-trainable Params	Total Params
Input (YAMNet Mean)	(1024,)	0	0	0
Gaussian Noise	(1024,)	0	0	0
FC1 (ReLU)	(512,)	$(1024 \times 512) + 512$ **524,800**	0	524,800
Batch Normalisation	(512,)	512×4 **1,024**	1,024	2,048

Dropout (50%)	(512,)	0	0	0
FC2 (ReLU)	(128,)	(512 × 128) + 128 **65,664**	0	65,664
Batch Normalisation	(128,)	128 × 4 **256**	256	512
Dropout (30%)	(128,)	0	0	0
FC3 (Sigmoid)	(1,)	(128 × 1) + 1 **129**	0	129
Total Params		**591,873**	**1,280**	**593,153**

When fully exported as a single, unified *.tflite* payload for mobile edge-deployment, the entire framework incorporates both the frozen YAMNet base architecture (~3.7M parameters) and the custom classification head, resulting in a highly optimised final model footprint of approximately 4.29MB (Float16).

4.4 Dataset Composition and Specifications

To ensure the YAMNet model generalises well across various real-world scenarios, a diverse mixture of publicly available and custom-curated datasets was utilised. This comprehensive dataset serves as the foundation for training the model to distinguish between normal ambient noise and genuine distress signals.

A critical challenge in training a safety model for a vehicle interior is the acoustic similarity between benign high-energy sounds and genuine emergencies. To illustrate the necessity of deep feature extraction over a simple volume-threshold approach, the following sample figures provide waveform and spectrogram comparisons of the dataset attributes.

- **Analysis of Safe Audio (Figure 4.11):** The visual representation of standard car-cabin noise (e.g. human conversation coupled with minor wind noise) displays a consistent, low-amplitude waveform. The spectrogram reveals dense, steady low-frequency bands (0-2000 Hz) representing the natural formants of human speech, alongside faint, continuous broadband noise in the higher registers (4000-8000 Hz) typical of wind shear. The absence of sudden, chaotic energy bursts allows the model to reliably classify this as a baseline safe state.

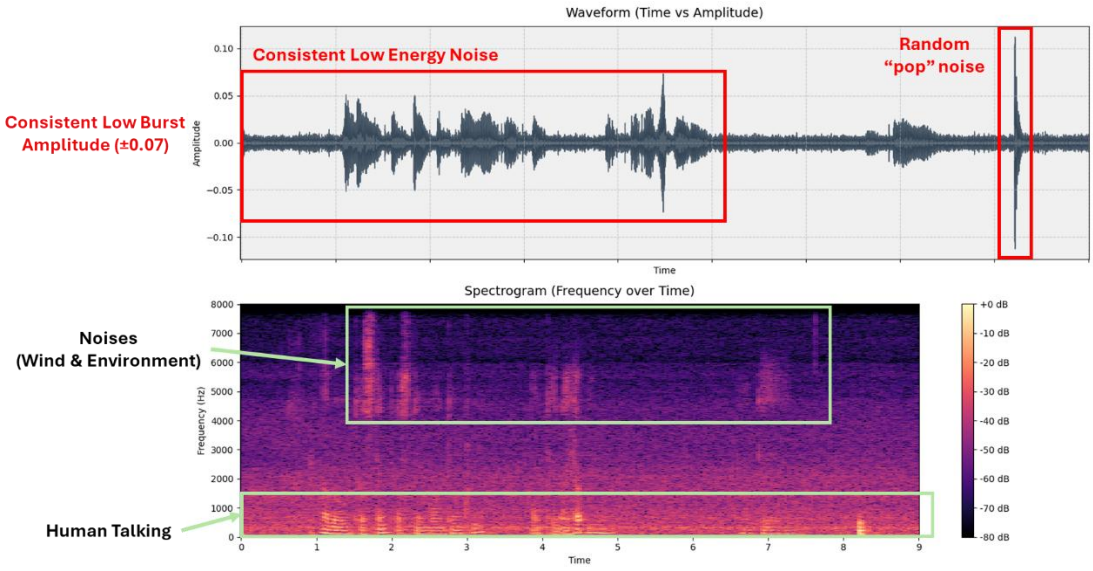


Figure 4.11 Sample Waveform and Spectrogram of Safe Audio

Analysis of Distress Audio (Figure 4.12): True distress events (e.g. individuals aggressively arguing) exhibit sharp, erratic spikes in the waveform amplitude, occasionally peaking near the system maximum (+0dB). The corresponding spectrogram reveals sudden, vertical bursts of high-frequency energy across multiple bands, disrupting the baseline noise floor. These chaotic, high-intensity acoustic signatures prompt an immediate distress classification.

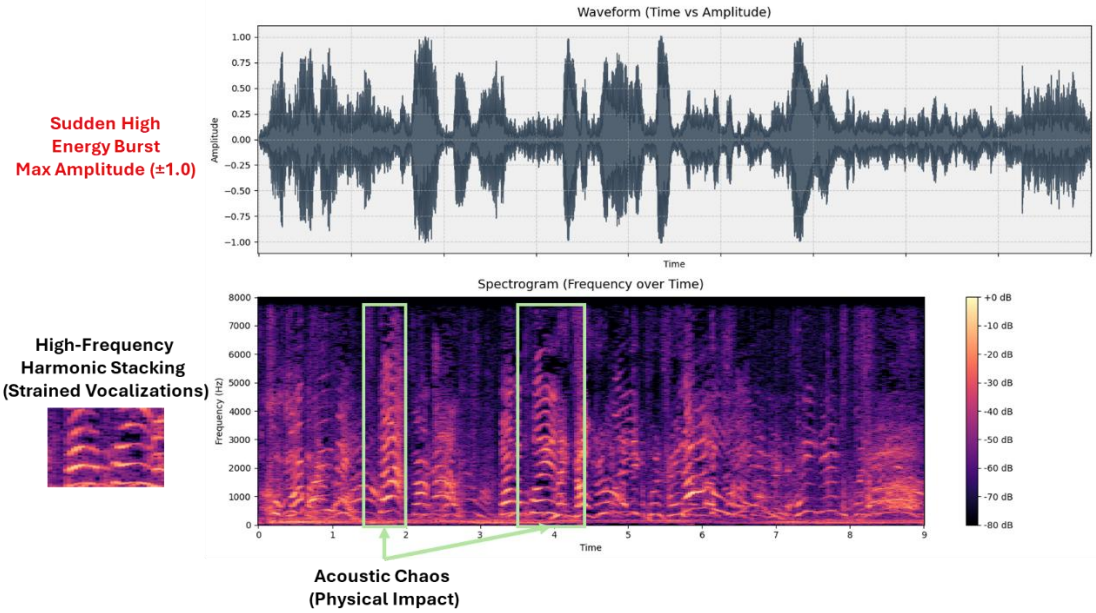


Figure 4.12 Sample Waveform and Spectrogram of Distress Audio

- **Analysis of False-Positive Triggers (Figure 4.13):** Noisy but safe environments, such as loud radio playback, present a highly volatile waveform

that superficially mimics distress through pure amplitude (volume). However, the spectrogram reveals structured, rhythmic frequency patterns and continuous horizontal harmonic lines rather than chaotic energy. Providing the model with this specific data ensures it learns to differentiate structural, musical noise from genuine distress.

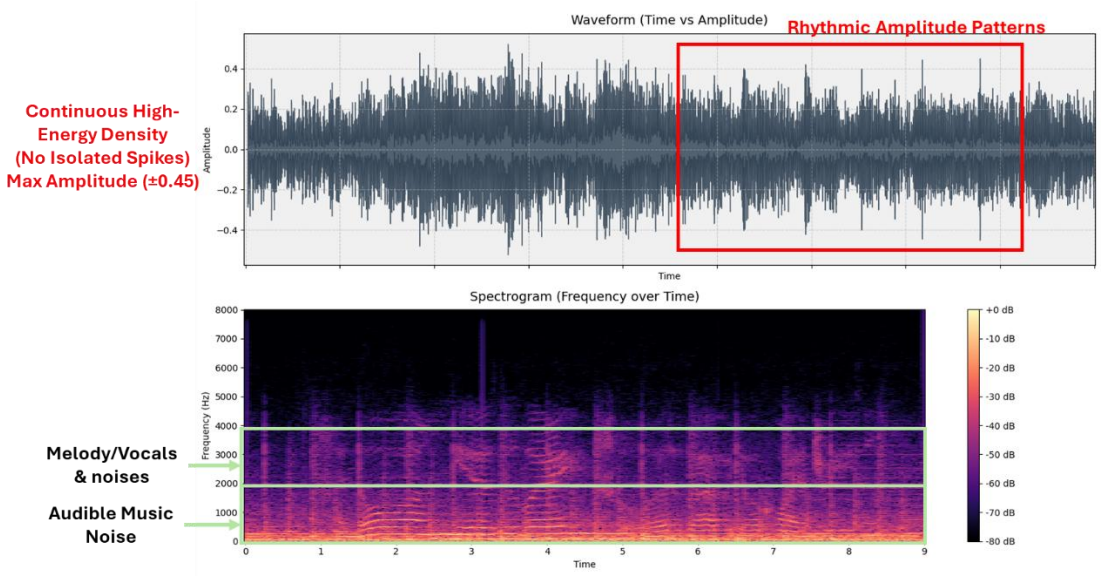


Figure 4.13 Sample Waveform and Spectrogram of Noisy Car-Cabin

This comprehensive dataset, summarised in Table 4.3, serves as the foundation for training the model:

Table 4.3 Summary of YAMNet training datasets

Dataset	Role in UniRide	Data Type	Safe/Distress Samples
ESC-50 Kaggle	**Safe/Background**: Benchmarks the model to ignore non-threatening noise (Animals, Weather, Car Ambience).	Environmental	6,860 / 0
RAVDESS Kaggle	**Mixed Baseline**: Provides emotional speech patterns to distinguish normal talk from high-stress yelling.	Human Vocal	1,132 / 1,320
VSD (Violent Scenes) Mandeley	**Distress Baseline**: Captures cinematic physical violence, explosions, and screaming.	Cinematics with high pitch	4 / 337

Violence Detection [Kaggle]	**Distress Detection**: Highly specialised incident patterns for binary classification.	Incident Audio Only	0 / 127
Custom Dataset ***Recording, Freedsound, Pixabay, Mixkit, Youtube (NCS, Car Crash, Crying)***	**Extra Class Handling Includes, Safe:** Radio and music, and animal sound, hand clapping, crowd sound, sirens **Distress:** Car Crash, Glass Breaking, Riot, Crying, Screaming	Self-Labelled	58 / 86
TOTAL			**8,054 / 1,870 (9,924)**

CHAPTER 5

System Implementation

5.1 Setup

5.1.1 Hardware

The development of this project requires a computer and at least two Android devices. The computer, with specifications listed in Table 5.1, is utilised for application development, model training, and backend configuration, while the Android devices serve for testing and deploying the carpooling application to ensure real-world usability and edge-processing performance.

Table 5.1 Specifications of computers used for development

Description	Specifications
Model (Motherboard)	ASRock B760M Pro RS/D4 Wi-Fi
Processor	Intel Core i5-14600KF (Not Overclocked)
Operating System	Windows 11 Home
Graphic	NVIDIA GeForce RTX 4060TI 8GB GDDR6 (Utilised for CUDA-accelerated model training)
Memory	32GB DDR4 RAM
Storage	2TB SSD

5.1.2 Software

Before commencing the development of the carpooling application, seven essential software tools must be installed and configured on the development computer.

1. Visual Studio Code (VS) 1.106.3
    - Install Flutter extension for VS.
2. Flutter SDK: 3.38.3 (Came with Dart: 3.10.1)
3. Android Studio (For Android development tools): 2025.2.2
4. Java Development Kit (JDK) 17
5. Firebase CLI (For Firebase Function Deployment)
6. Python 3.12 + Cloudflared for self-hosting

Python 3.12 was selected to ensure compatibility with the high-performance *insightface* and *onnxruntime-gpu* libraries. The decision to use a Local AI Server instead of a cloud-hosted one was driven by the computational cost of running ArcFace models and the need for GPU acceleration, which is more cost-effective on local hardware during the development phase.

7. A Windows Subsystem for Linux (WSL) environment was configured with Python 3.10.12 and CUDA support enabled, for YAMNet transfer-learning.

5.1.3 Architectural Integration and Configuration

To transition the system from a localised development environment into a production-ready prototype, several integrations were established across the mobile client, Firebase, and a local Zero-Trust backend. *(Note: Complete setup instructions, including Cloudflare domain registration, nameserver delegation, and API key generation, have been documented in the and Appendix 2).*

- **Firebase Backend-as-a-Service (BaaS):** The *com.fyp.uniride* package was integrated with Firebase to handle institutional Google Sign-In authentication, real-time Firestore database synchronisation (for carpool matching), and Cloud Storage for biometric image and incident audio backup.
- **Environment Modularity:** API credentials (Gemini, Google Maps) and the backend routing URLs were decoupled from the source code using *.env* files (*flutter_dotenv* and *python-dotenv*). This "Configuration as Code" approach secures sensitive keys and allows dynamic threshold tuning without recompiling the mobile application.
- **Zero-Trust Local API Tunnelling:** To bypass the high latency and egress costs associated with running heavy computer vision models (ArcFace) on serverless cloud functions, a hybrid cloud bridge was implemented. The system utilises cloudflared daemon to establish an encrypted, reverse-proxy tunnel. This securely exposes the local Python Flask server to the public internet via the *uniride.cv* domain, granting the mobile application sub-second access to GPU-accelerated biometric processing without exposing the host machine's IP address.

- **Edge-Sensor Integration:** The *AndroidManifest.xml* was configured with stringent hardware permissions (*RECORD_AUDIO, CAMERA, ACCESS_FINE_LOCATION*). The Flutter application utilises native channels to poll the IMU sensors. If the vector magnitude of the 3-axis accelerometer exceeds the 4.5g threshold, the system autonomously initiates the YAMNet audio sampling and distress detection pipeline.

5.1.4 Data Privacy and Network Security

Given the sensitive nature of biometric data (facial images) and audio recordings, the system implementation prioritises two layers of security:

1. **Encryption in Transit**: By delegating the domain to Cloudflare, a managed SSL/TLS certificate is automatically applied. All traffic between the mobile app and the local server is encrypted using HTTPS, preventing "Man-in-the-Middle" attacks.

2. **Network Isolation**: The use of cloudflared tunnelling avoids the need for traditional port forwarding. This keeps the hosting machine's public IP address hidden, effectively shielding the development environment from direct network probes and unauthorised access.

5.2 System Operation

The application defines two major role categories to represent sub-roles for system operations. These major categories are not mutually exclusive, but they follow specific exclusivity rules:

User Group

- **Sub-roles:** Rider, Driver
- Not mutually exclusive within the group: A single account may hold both sub-roles concurrently.
- A User Group can contain Admin Group privilege.

Admin Group

- **Sub-roles:** Admin, Superadmin, Creator

- Mutually exclusive within the group: A single account cannot hold more than one Admin sub-role simultaneously.

Table 5.2 represent all the main features implemented in UniRide mobile application,

Table 5.2 Feature set implemented

Role	Features Implemented
User (Rider)	• Device verification guard (U) • Radius based carpool searching and listing (R) • Carpool ride history (R) • Driver registration form/screen (U) • Facial verification based on dummy institution face database (R) • Face verification liveness test for face anti-spoofing • **Pre Trip Screen** with carpool detail, bidding (RU) • **Pre Trip QR Generation** for Rider Verification • **During Trip Screen** with QR payment and trip details (R) • **Post Trip Screen** with Feedback & Review System (CRU) • **Pre, During, Post Trip screens** of all participants are synchronised (U)
User (Driver)	• Inclusive of all **Rider features** • Carpool listing creations with EOD model (CRD) • Carpool creation history (R/D) • Bidding management (RU) • **Pre Trip Screen** with session initiation button (U) • **Pre Trip QR Scanning** to verify rider and take attendance (RU) • **Pre Trip Manual Attendance Taking** (RUD) • **During Trip Screen** with session completion button that allows driver to toggle within 80m from the **Destination Point** (U)
Admin	• Manage specific universities and their campus(es) (CRUD) • University email domain registration (CRUD) • Geofence configurations (CRUD) • Manage institution face database: Include face registration (CRUD) • Stop/Marker creation (CRUD) • Distress Alerts dashboard & AI assessment review (RU) • Manage user (Superadmin and Creator: CRUD)

Additional features that enhance user experience include an interactive location selector with geofence overlay for gatekeeping, map integration with route and ETA, and email login activation through password creation.

With these features, the system provides a secure, usable, and commercialisable carpool platform with baseline safety measures to safeguard against misuse and impersonation. The following section discusses the facial verification module and its reliability in preventing impersonation.

5.2.1 Admin: Login

Users with an admin, superadmin or creator account can login the app bypassing the gated face verification for normal user (student and staff) as shown in Figure 5.1. This allows universities and admins to perform managerial tasks including University Details, Face Database, Stops Management, Manage Users and view Distress Alert log while unable to use the carpooling services.

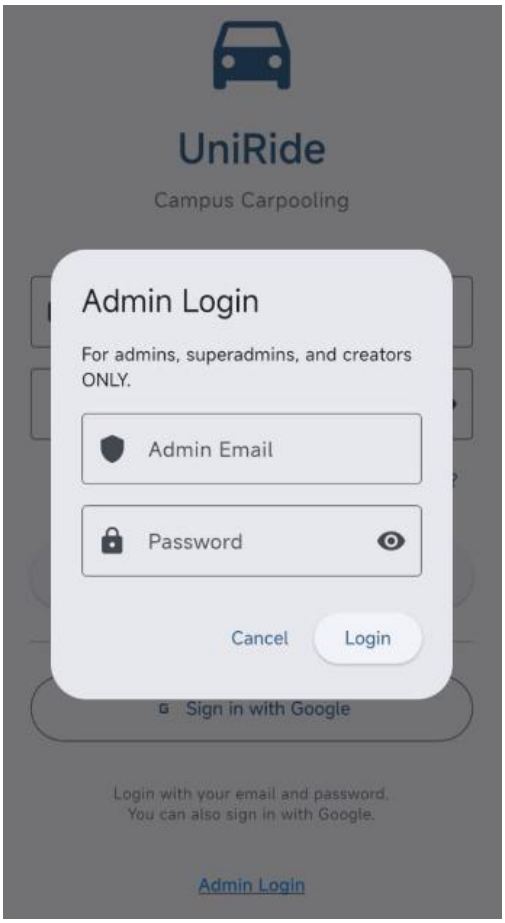


Figure 5.1 Admin Login interface

5.2.2 Admin: Monitoring and Management Tools

Participating universities are provided with a dedicated administrative interface at Admin Tab as shown in Figure 5.2.

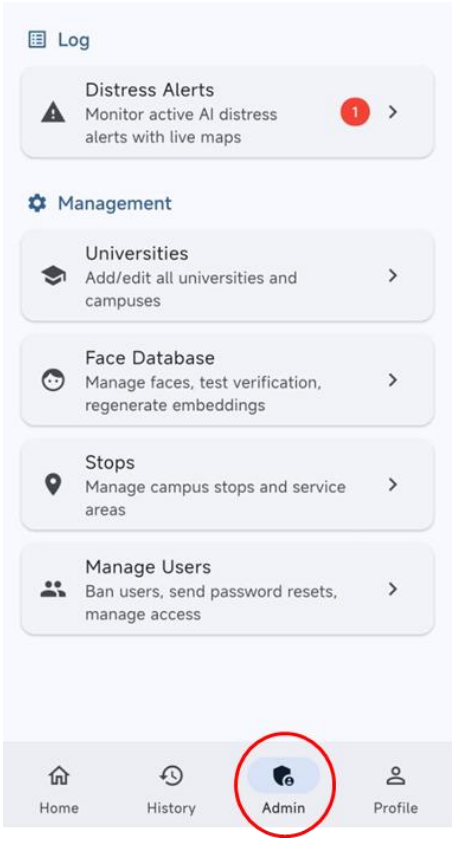


Figure 5.2 Admin tab and tools listing

The Admin tab contains 2 categories of tools (Log and Management):

1. **Log: Distress Alerts (Figure 5.3)**

   Used for monitoring the distress signal triggered containing real time location tracking, context-based AI summary and audio evidence.

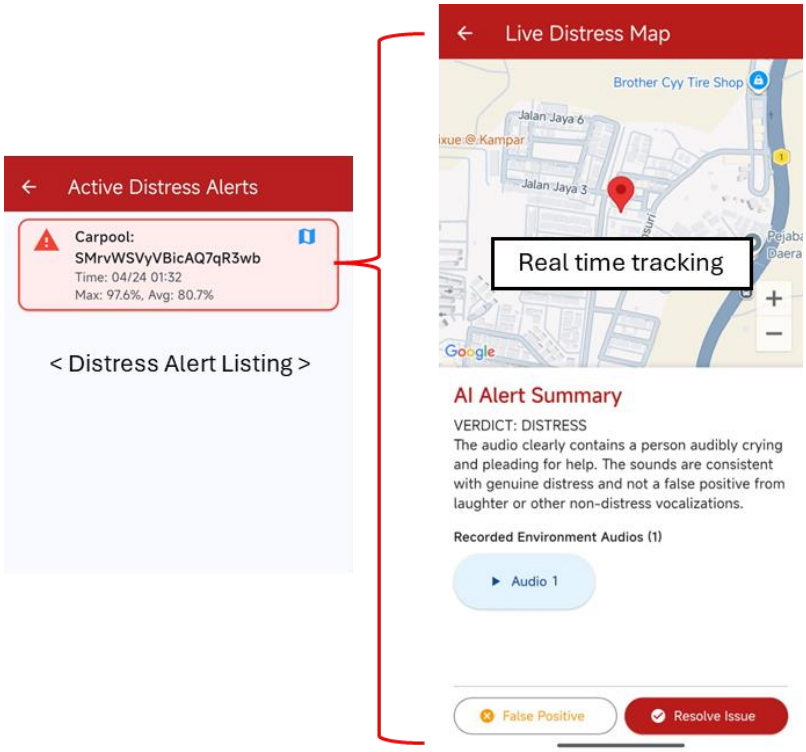


Figure 5.3 Distress Alert listing and distress details

Management

1. **Manage Universities (Figure 5.4)**

   Used for managing university logo, email domain and the campus Geofence setup.

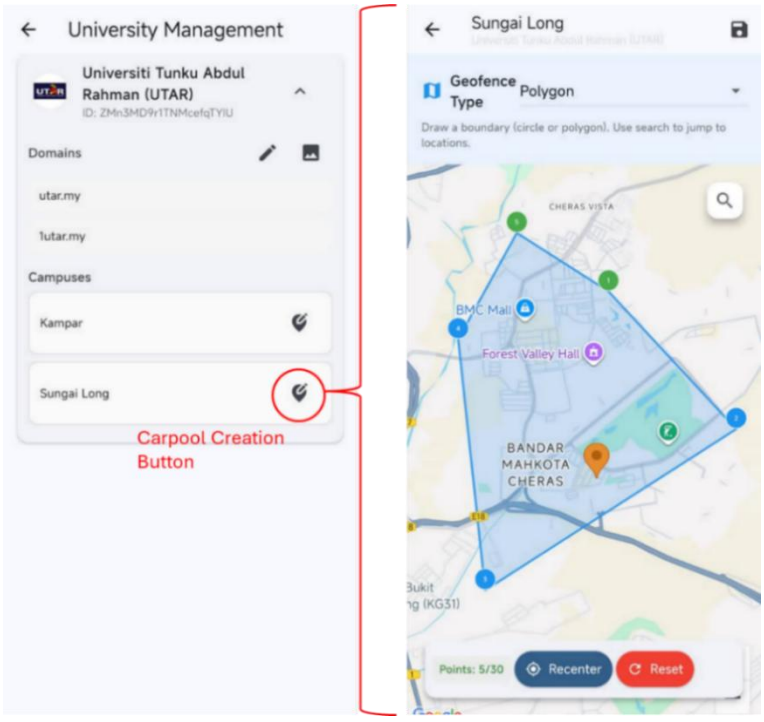


Figure 5.4 University domain and campus management with geofence modification

2. **Face Database (Figure 5.5)**

Manage student face data, providing features to add and delete embeddings, verify faces with a score, regenerate embeddings from stored images, and edit student information. This is the core management tool required for face verification, as device changes or first sign in are guarded by face verification. Without successful verification, users are restricted from fully accessing the app for security consideration.

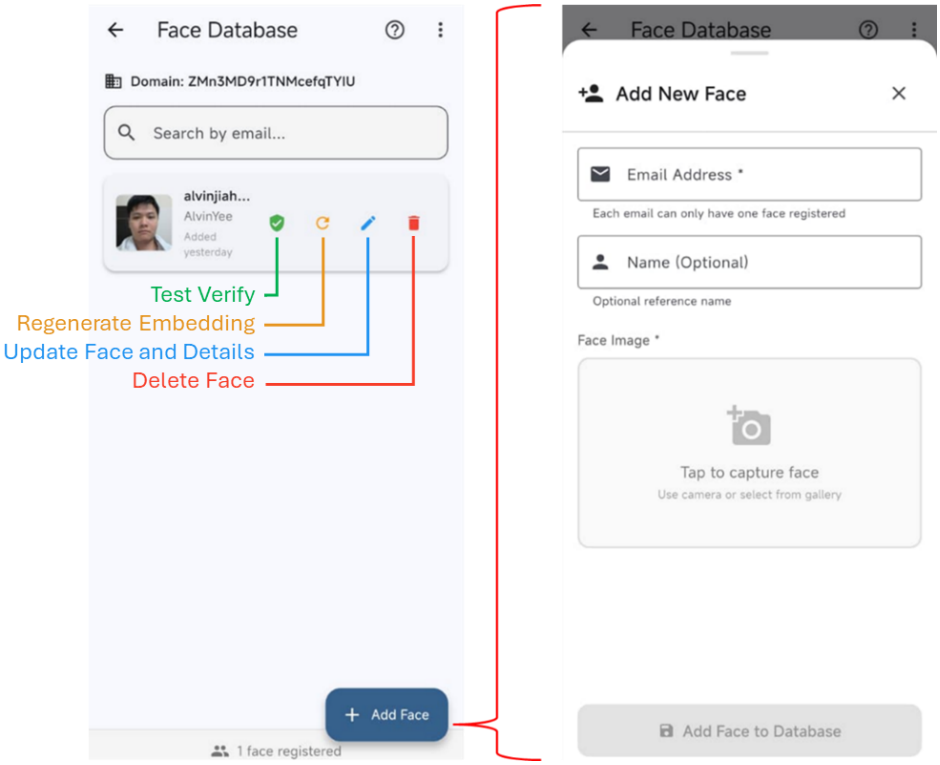


Figure 5.5 Face database tools and creation

3. **Manage Stops (Figure 5.6)**

Used for managing stops available around the campuses area.

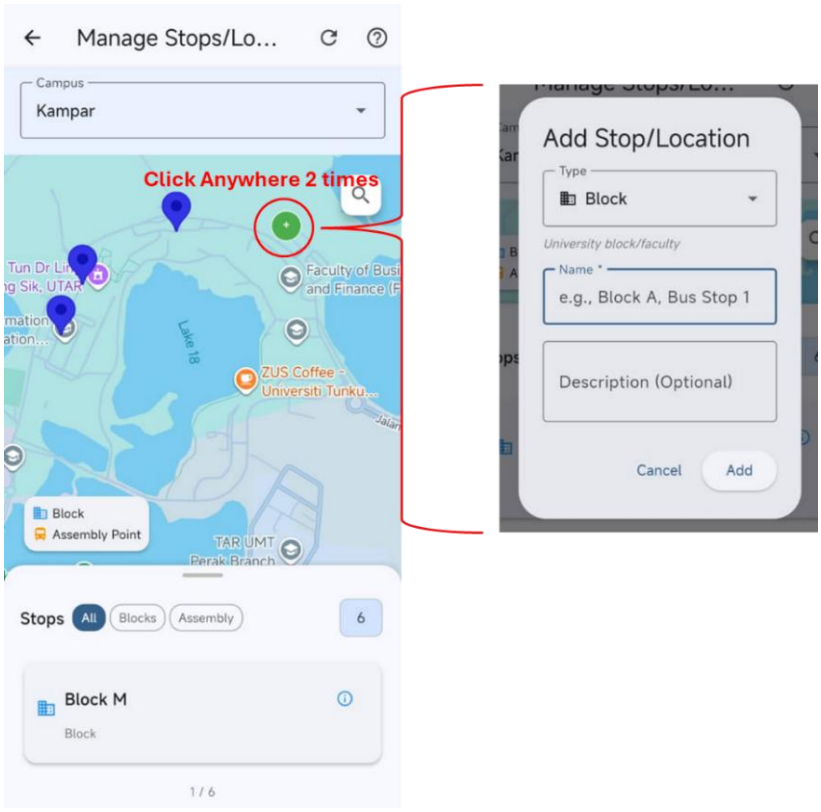


Figure 5.6 Stops creation screen

4. **Manage Users (Figure 5.7)**

The Manage Users module controls user accounts within the application. Operations are restricted based on hierarchical roles:

Admin (within own university domain):

- Preview user feedback
- Send password reset emails
- Ban or unban accounts

Superadmin (within own university domain):

- All Admin privileges
- Change user roles (Normal, Admin)
- Delete user accounts

Creator (across all university domains):

- Full Superadmin privileges
- Access and manage users across all domains

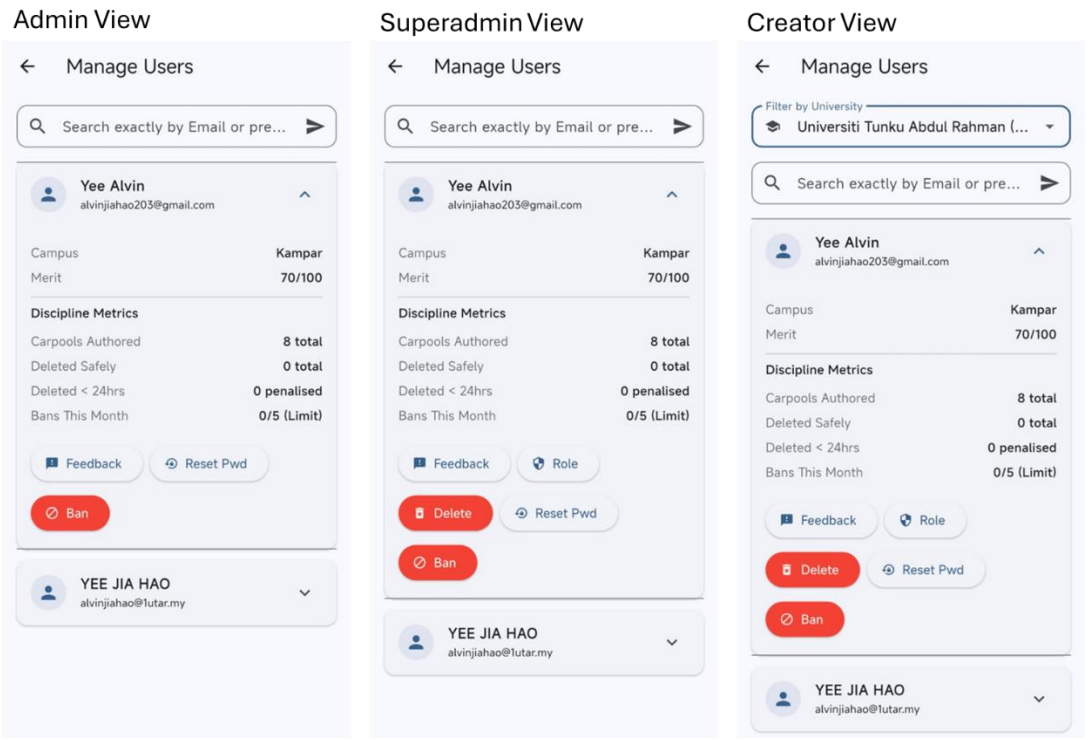


Figure 5.7 Admin (Left), Superadmin (Centre) and Creator (Right) manage user view

5.2.3 User: Login

For student users, it is assumed that the university has already populated the database with specific student face data, either directly or based on student consent. During first-time sign-in and whenever a device change is detected, users are required to verify their face to access the application as shown in Figure 5.8. This verification process ensures that only authorised individuals can utilise the system and maintain the integrity of account and community security.

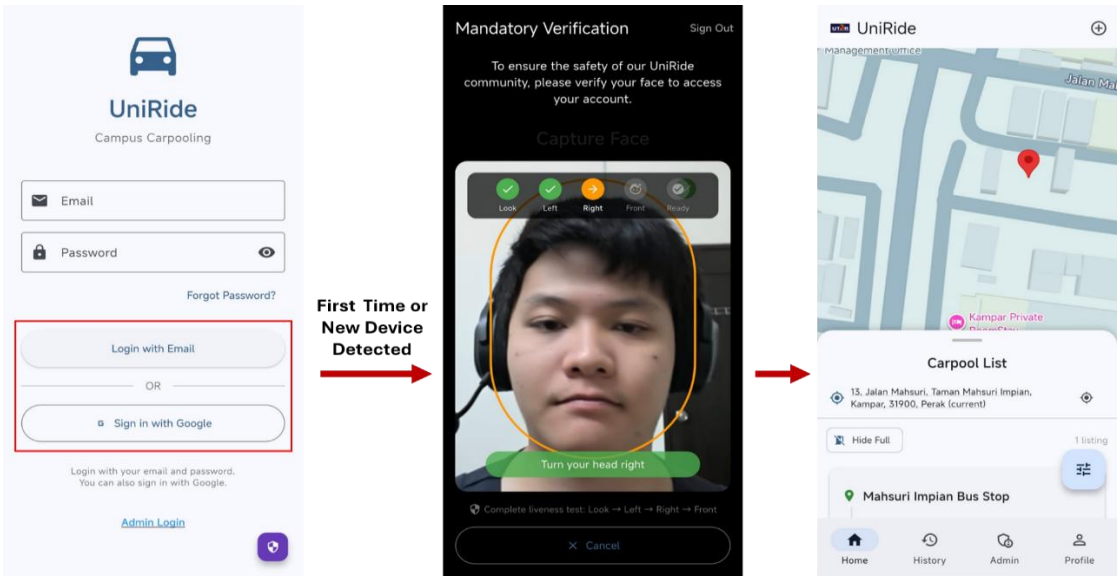


Figure 5.8 User login with verification guard sequence diagram

For every successful verification, the system generates a random ID that is stored locally using Shared Preferences and simultaneously saved in Firestore. Whenever a device does not contain the locally stored ID, or the ID differs from the one stored in Firebase, the verification guard is triggered, and the user is required to verify their identity to access the service. This approach ensures that the service remains uninterrupted as long as the user continues to use the same mobile device.

5.2.4 User: Navigations and Screens

After logging in, user will see the Home screen. Following, we will discuss all the available tabs and their functionality, and what users can do:

1. **Home Screen (Figure 5.9)**

   This screen represents the Carpool Listing interface. Riders use this screen to search for available carpool listings in selected locations, while Drivers can create a new carpool listing using the designated ⊕ button.

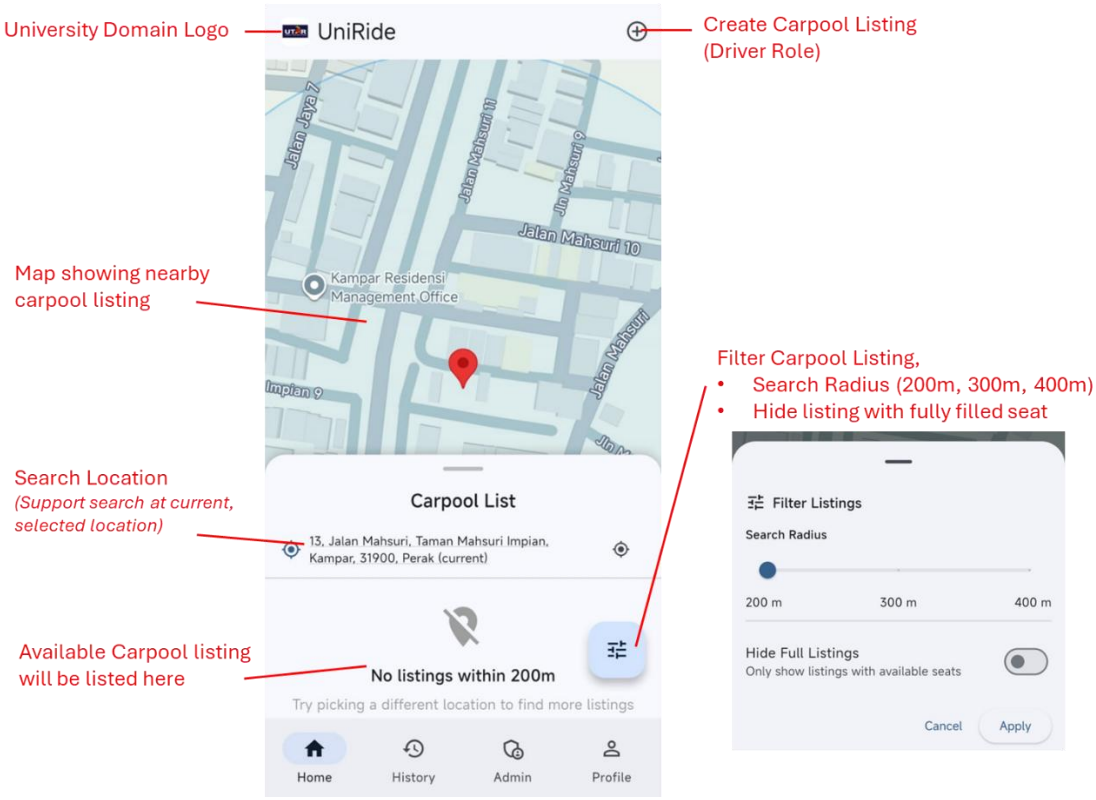


Figure 5.9 UniRide Home Screen

2. History Screen (Figure 5.10)

This screen provides two categories, Rider and Driver, which separate the carpool listings according to the user's role. It displays all carpool listings that the user has either participated in or created, enabling them to keep track of their activity. Basic filtering and sorting tools are also available to assist users in managing and reviewing their listings.

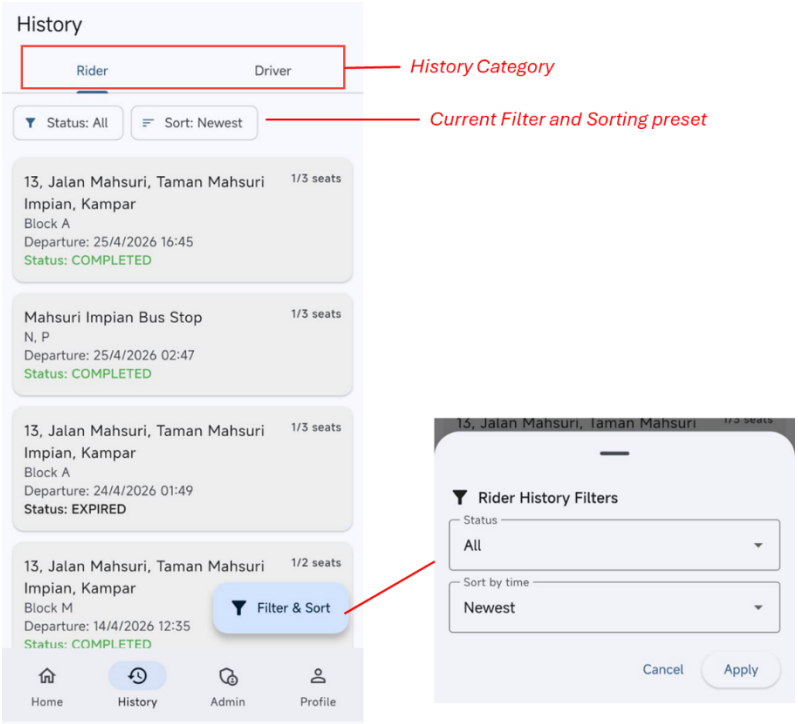


Figure 5.10 UniRide History Listing Screen

3. Profile Screen (Figure 5.11)

This screen allows users to view their information, change password, register as a driver, toggling the During trip distress detection and to sign out of the account.

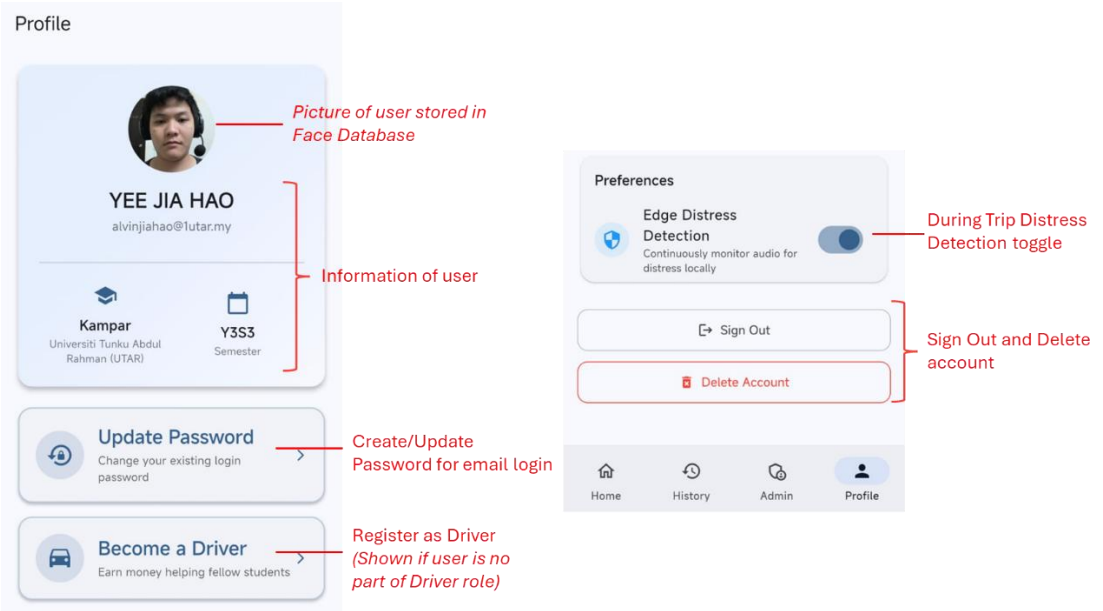


Figure 5.11 UniRide Profile Screen with tools

5.2.5 User: Set and Update Password

Based on Figure 5.12, the "Set Password" feature allows users to create a new password for non-Google login (if none exists), while the "Update Password" feature is provided for existing accounts.

- Password creation requires a chosen password and confirmation and additionally requires email verification before the service can be used.
- Password updates require the old password, new password, and confirmation.

Passwords must be at least nine characters long and include uppercase, lowercase, and numeric characters; symbols are optional. A strength bar is provided to help users select a secure password.

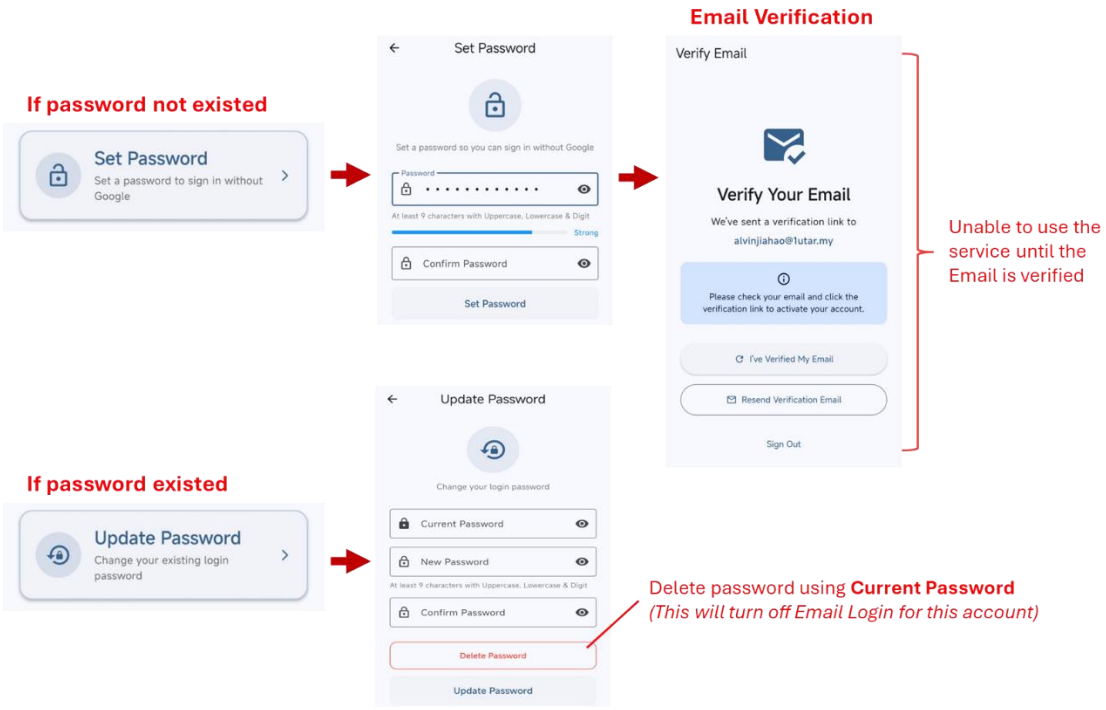


Figure 5.12 Set/Update password process

5.2.6 User: Driver Registration

By selecting the Become a Driver option in Figure 5.12, users may register as Drivers by completing the following steps, as illustrated in Figure 5.13:

1. Enter Phone number and Car Details
2. Upload QR code (DuitNow specifically) for receiving payment.
3. Perform Face Verification

4. Successfully registered as Driver

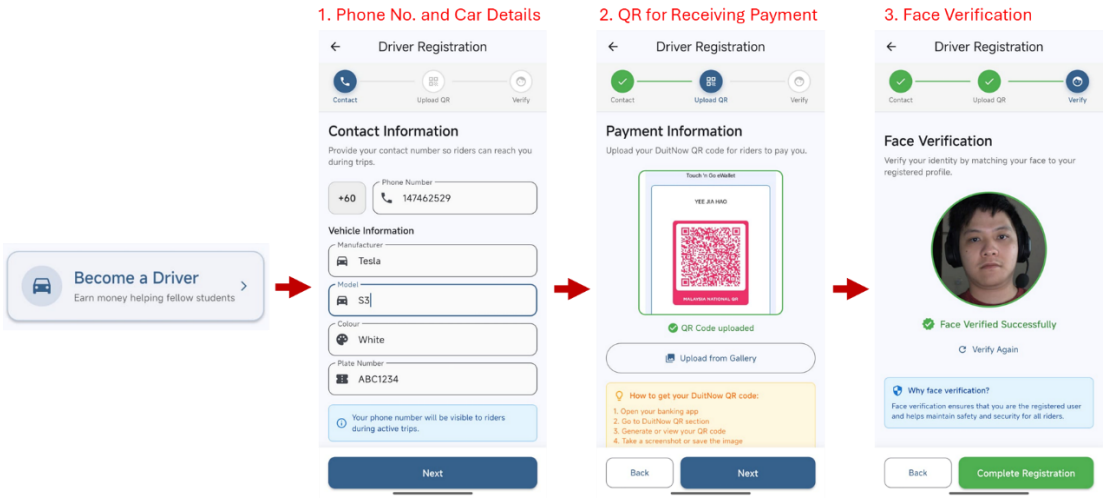


Figure 4.13 Driver registration process

5.2.7 User (Driver): Carpool Listing Creation

Carpool listing is one of the core functionalities of the application, serving as the primary interface for users upon launching the app. Figure 5.14 illustrates this interface. For users with the driver role, a carpool can be created by tapping the ⊕ button, which opens the Create Carpool Listing screen. In this screen, drivers are required to specify key trip details, including origin, destination, departure date and time, and fare. To enhance accuracy and user experience, an interactive map displaying the route is provided, ensuring that the selected origin and destination are correctly set.

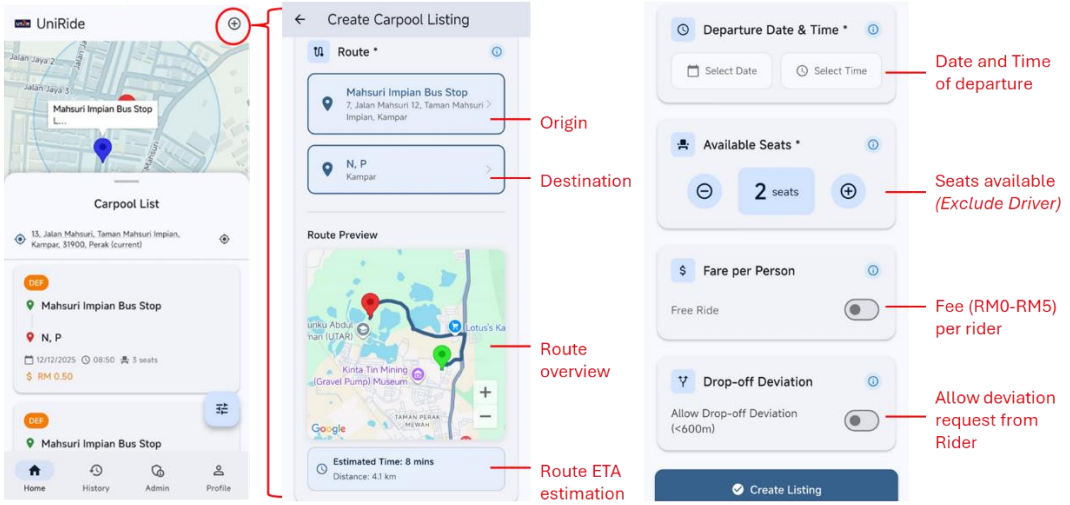


Figure 5.14 Carpool listing creation

5.2.8 User (Rider): Joining Carpool Session - Pre-Trip

After a Driver submits a carpool session, nearby Riders within the default 100-meter radius can detect the session, as illustrated in Figure 5.15. To join, the Rider selects the specific carpool session to open the Carpool Detail screen and then clicks the Join Carpool button.

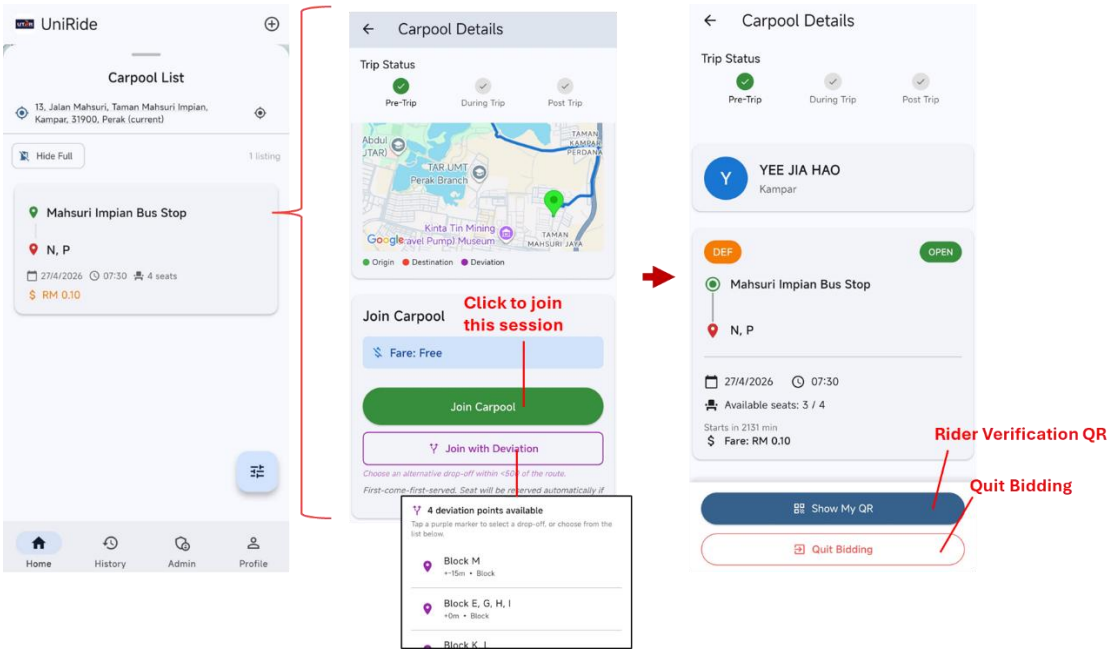


Figure 5.15 Carpool detail and joining session process

In cases where the Driver has enabled "Drop-off Deviation" toggle, an additional interaction layer is presented to the Rider. A second button labeled "Join With Deviation" appears directly below the standard "Join Carpool button". Upon selecting this option, the Rider is presented with a list of available alighting clusters (e.g. Block B, C or Block N, P) that fall within the driver's allowed deviation threshold. To assist in decision-making, the interface explicitly displays the deviation penalty for each point, calculated as:

$$ExtraDistance = (DeviatedRoute - OriginalRoute)$$

Once the Rider has successfully joined the carpool session, two additional options become available:

Show My QR: Displays a unique QR code for the Driver to verify the Rider during the pre-trip phase.

Quit Bidding: Allows the Rider to withdraw from the session. However, quitting less than 24 hours before the scheduled departure may result in a penalty, where the user is restricted from joining any other carpool sessions for the subsequent 24-hour period.

5.2.9 User (Rider & Driver): Pre, During and Post-Trip Screen

These screens are essential for participants as they represent the core stages of the carpooling experience: before trip, during trip, and after trip, which are synchronised for all participants.

1. **Pre-Trip:**

   Displays driver and rider information and trip details, while allowing riders to verify the driver and confirm the booking as shown in Figure 5.16.

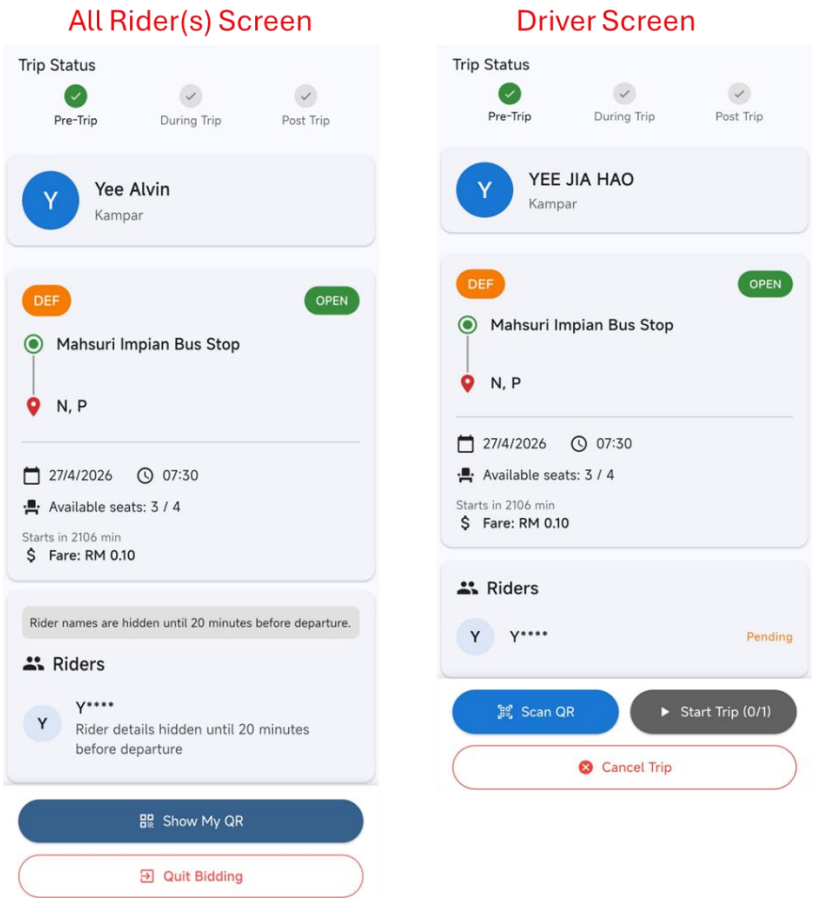


Figure 5.16 Pre-Trip Rider and Driver Screen

The Driver must verify each Rider by scanning the Rider's "Show My QR" code before the trip can begin. This process serves as attendance and identity confirmation.

Alternatively, the Driver may manually mark a Rider as Absent (only after three minutes past the scheduled departure time) or Arrived (to confirm presence manually). Importantly, the Driver cannot start the session until all Riders have been marked as ready.

2. **During Trip:**

This screen displays the navigation map, payment QR, and trip information, as shown in Figure 5.17. At this stage, the Distress Detection system is activated to provide reactive protection. It utilises IMU sensors (to detect crashes, speeding, and sharp turns), the microphone (to detect shouting or distress sounds), and geofencing (to monitor if the trip moves outside the designated service area).

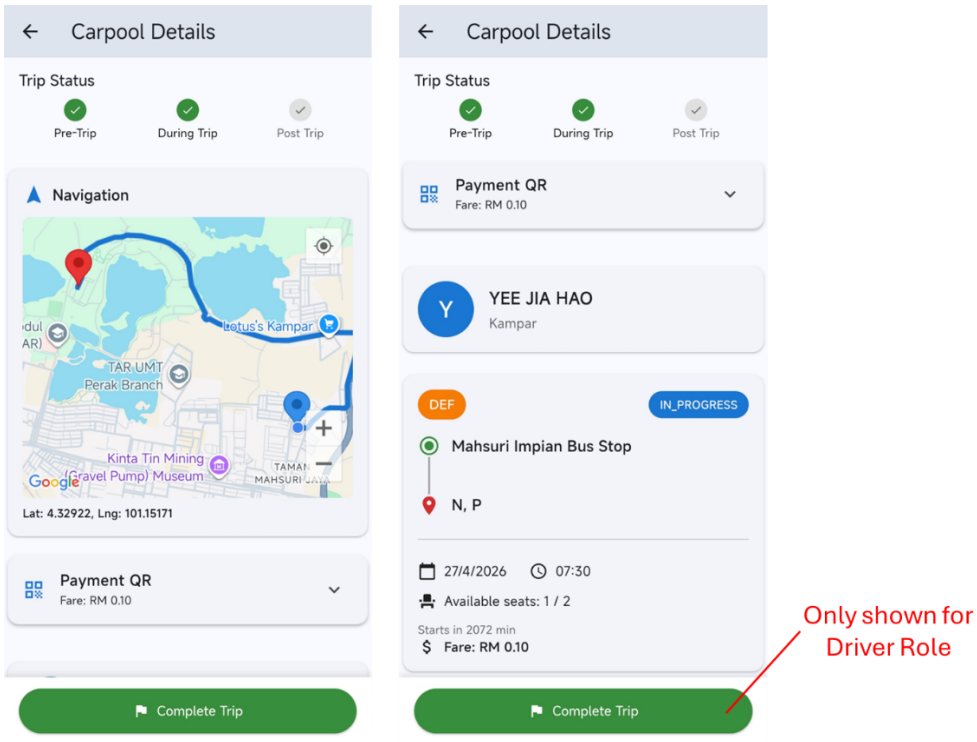


Figure 5.17 During Trip Screen

To complete the trip, the Driver's position must be within 80 metres of the destination point. This requirement acts as a safeguard to ensure accurate trip completion and proper session closure; otherwise, the Driver will be unable to end the trip, as illustrated in Figure 5.18.

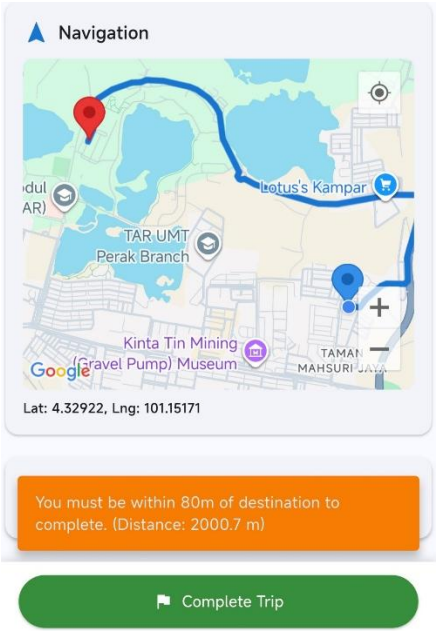


Figure 5.18 Completing trip but yet to reach destination (>80m)

3. **Post Trip:**

As shown in Figure 5.19, the Post-Trip screen provides carpool details and includes a simple review system that allows Riders to rate the Driver and submit feedback. This mechanism supports trust-building and enhances community accountability within the platform.

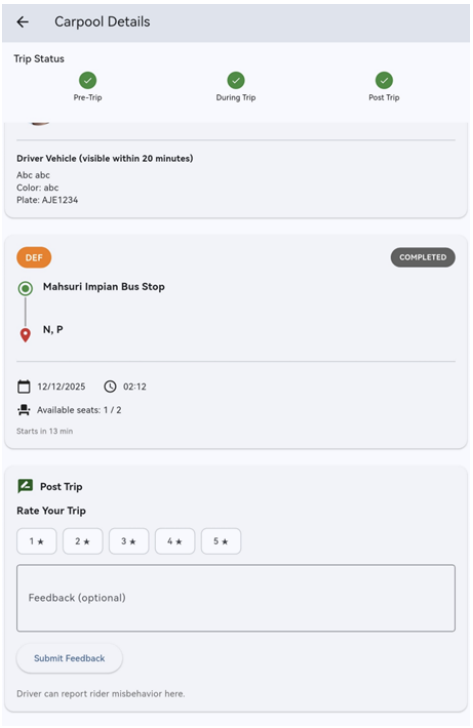


Figure 5.19 Post-Trip Review System

5.2.10 Admin and Users: During Trip Distress Triggering and Logging

During the trip, if a distress situation is detected, for example, someone in the car screaming for help, the YAMNet model processes the audio input and passes its verdict to Gemini for confirmation. When a distress event is verified, the Rider and Driver screens display a warning pop-up, as shown in Figure 5.20. On the Admin side, the system records the incident and displays it in the log as a Distress Alert, enabling administrators to monitor and respond accordingly.

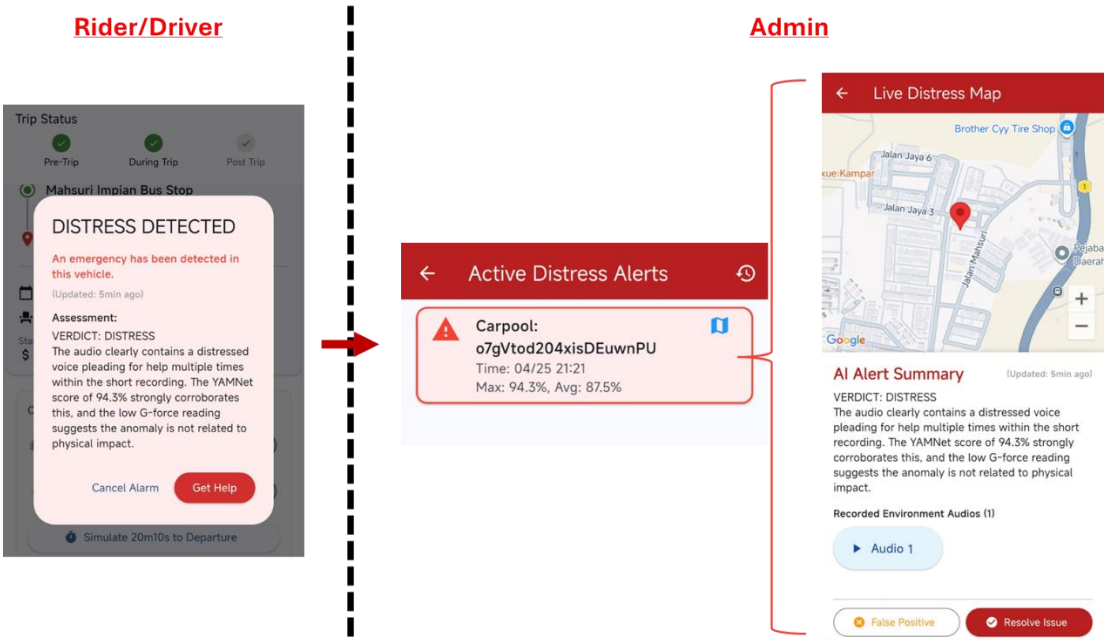


Figure 5.20 Distress Trigger and Admin Monitoring Interface

5.3 YAMNet Distress Detection Implementation

5.3.1 Data Preprocessing Pipeline

As detailed in the system design, the raw dataset comprising 9,924 audio files (8,054 Safe vs 1,870 Distress) was heavily imbalanced. To prevent data leakage and prepare the data for the TensorFlow model, the files were split into Training (80%), Validation (10%), and Testing (10%) sets at the file level before any processing occurred.

Through targeted augmentation (dynamic up-sampling of distress sounds) applied strictly to the training set and overlapping window segmentation (0.975-second chunks with a 50% stride), the preprocessing pipeline successfully transformed the

imbalanced raw files into hundreds of thousands of proportionate analytical chunks. As detailed in Table 5.3, the testing and validation sets achieved a near 1:1 ratio of Safe to Distress chunks, ensuring an unbiased model evaluation.

Table 5.3 Dataset preprocessing and transformation summary

Dataset Split	# Raw Samples (Safe / Distress)	# Samples after Augmentation (Safe / Distress)	# Final 0.975s Chunks (Safe / Distress)
Training (80%)	6,443 / 1,496	5,610 / 1,740	304,831 / 202,877
Validation (10%)	805 / 187	No Augmentation	4,835 / 4,793
Testing (10%)	806 / 187	No Augmentation	5,280 / 4,885
Total	**8,054 / 1,870**		**314,946 / 212,555**

Mathematical Derivation of Chunk Generation

The exponential expansion of the dataset from a few thousand raw files to hundreds of thousands of final samples is a direct result of the overlapping window technique applied during preprocessing. To ensure the YAMNet model effectively captures transient distress sounds, continuous audio files are segmented into **static 0.975s chunks with a 50% overlap**.

The number of chunks generated from a single audio file is mathematically determined by the file's total sample length, the chunk size, and the stride. Given the target sample rate, f_s = **16,000Hz**, the variables are defined as follows:

- S_{total} : Total number of samples in the augmented audio file
- S_{chunk} : Number of samples per chunk, calculated as,

$$S_{chunk} = 0.975 * 16000 = 15600$$

- S_{step} : The step size between chunks. With a 50% overlap,

$$S_{step} = S_{chunk} * 0.5 = 7800$$

The base number of full chunks N_{full} extracted from a single audio file can be calculated using the following equation:

$$N_{full} = \left\lfloor \frac{S_{total} - S_{chunk}}{S_{step}} \right\rfloor + 1$$

To prevent the loss of critical audio data at the tail end of the recording, the preprocessing algorithm includes a conditional padding mechanism. If the remaining tail of the audio file exceeds half the chunk size (> 7800 samples), an additional chunk with zeros is padded until input shape is 15600 chunks,

$N_{total} = N_{full} + N_{padded}$ *(where* N_{padded} *is either 0 or 1 based on the tail length)*

For a detailed, step-by-step mathematical demonstration of this expansion process using a sample audio file, please refer to Appendix 3.

5.3.2 Model Training Configurations

The transfer-learning process of the YAMNet model was conducted using two distinct optimiser configurations (AdamW and Adam) to determine the most effective learning strategy. Class weights were computed dynamically to heavily penalise misclassifications of the "Safe" class, calibrating the model specifically for the high false-positive environment of a vehicle cabin. The model was configured for 100 epochs with an Early Stopping callback (patience = 4) monitoring validation loss to automatically restore the optimal weights.

5.3.3 Training and Optimisation Metrics

To establish the scientific validity of the acoustic distress model, empirical training evidence was recorded throughout the transfer-learning process. The implementation utilised the AdamW optimiser (Model A) on the CUDA-accelerated local machine.

- **Loss Curve Analysis (Figure 5.21):** The loss curve for Model A demonstrates rapid convergence in the early epochs. The early stopping callback successfully halted training at Epoch 26 to prevent overfitting, restoring the optimal weights where validation loss stabilised at approximately 0.29. Comparatively, Model A achieved a lower and more stable final validation loss than the unregularized Model B, confirming that the explicit L2 regularisation (*weight_decay=1e-4*) was necessary to handle the augmented dataset.

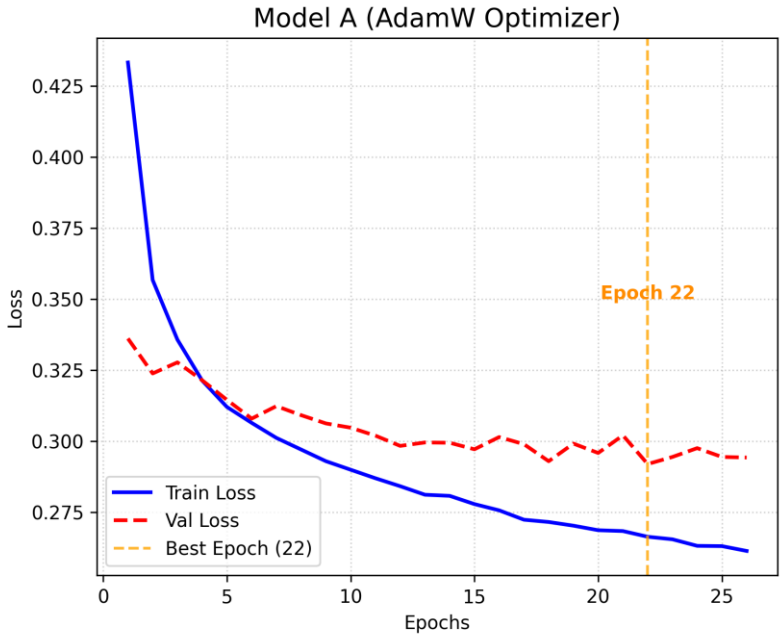


Figure 5.21 Model Training and Validation Loss Curve

- **Accuracy Curve Analysis (Figure 5.22):** The training accuracy steadily climbed to approximately 93%, while the validation accuracy plateaued effectively near 85%. This convergence indicates the custom classification head successfully learned to generalise the acoustic features of distress without simply memorizing the training data. In final comparative testing, Model A slightly outperformed Model B in validation accuracy across the closing epochs.

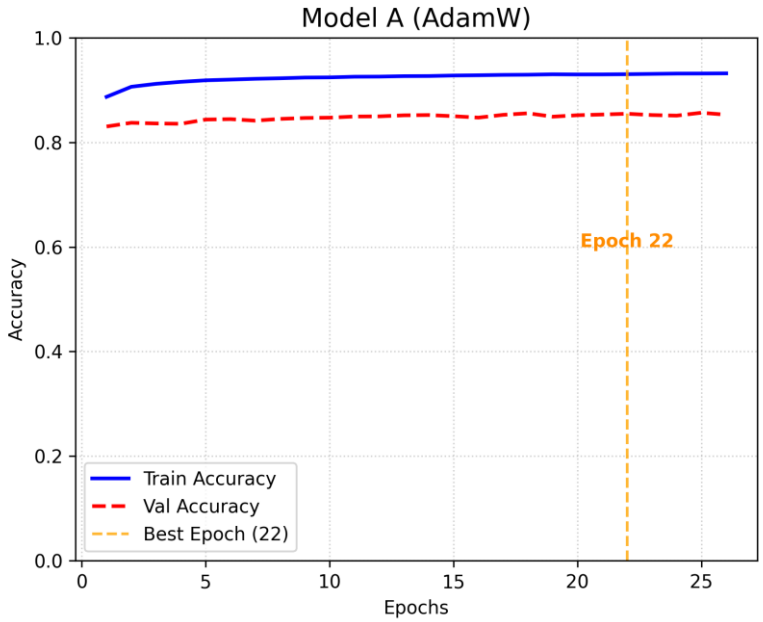


Figure 5.22 Model Training and Validation Accuracy Curve

Following training, the model was evaluated for mobile edge deployment. The physical performance metrics and inference speeds on the device are detailed in Table 5.4:

Table 5.4 YAMNet Deployment and Edge Inference Metrics

Performance Metric	Recorded Value	Notes
Total Training Time	26 minutes	Stopped at Epoch 26 (Processed on GPU)
Optimal Epoch	Epoch 22	Restored best model weights before callback halt
Custom Classifier Head Size	2.25MB	Extracted classification head payload
Combined Pipeline Size	14.53MB	Full Float32 unquantised model *(YAMNet + New Distress Classification Head)*
Quantisation	None	Maintained full precision (Float32) for maximum accuracy
Avg. Classifier Latency	0.0358ms	Latency of the custom dense layers
Avg. Pipeline Inference Time	**PC GPU:** 1.77ms **Mobile:** 360ms	End-to-end processing from Audio to Score output

5.3.4 Quantisation Trade-off Analysis

To further optimise the neural network for mobile edge deployment, Post-Training Quantisation (PTQ) was evaluated on Model A using the same validation set. The goal was to reduce the model's physical footprint and memory utilisation without compromising the safety-critical acoustic accuracy. Table 5.5 details the comparative performance of the model under different quantisation constraints:

Table 5.5 TFLite Quantisation Performance Comparison

Model Quantisation	Validation Accuracy	Model Size	Avg. Latency(ms)	
			PC (CPU)	Mobile (Edge)
None (Float32)	86.20%	14.53 MB	1.9	360
Float16 (FP16)	86.22%	7.29 MB	1.8	341
Integer8 (INT8)	57.12%	3.96 MB	0.8	164

As demonstrated in the comparison, aggressive 8-bit Integer (INT8) quantisation successfully reduced the file size and latency to the absolute minimum, but it caused a catastrophic degradation in model accuracy (dropping to 57.12%), rendering it unviable for a life-safety application.

Conversely, the 16-bit floating-point (FP16) quantisation proved to be the optimal engineering compromise. It successfully halved the combined pipeline size from 14.53 MB to a highly efficient 7.29 MB and slightly improved baseline latency, all while preserving the maximum validation accuracy (86.22%).

Therefore, the FP16 quantised model was selected as the final payload for the UniRide application. At 7.29MB, it is easily bundled into the Flutter application assets. Most notably, the end-to-end mobile edge inference time of 341ms (0.341 seconds) remains highly efficient. Because the FP16 model evaluates a 0.975 second audio chunk in roughly one-third of a second, the system operates significantly faster than real-time. This ensures UniRide can execute continuous, continuous background distress monitoring without causing CPU thermal throttling or severe battery drain on the user's mobile device.

5.4 Implementation Challenges and Resolutions

Throughout the development of the UniRide system, several critical implementation challenges were encountered, primarily centred on the computational demands of the face verification module and the environmental sensitivity of the distress detection pipeline.

5.4.1 Latency and Scaling in Serverless Environments

The ArcFace model for face verification was initially deployed via Firebase Functions. While accuracy was within acceptable parameters, generating a face embedding frequently incurred a latency of up to 30 seconds. This delay was prohibitive for real-time authentication and database indexing.

Migrating ArcFace face verification into AWS Lambda as an alternative revealed significant deployment constraint. The Python dependencies and weight files required for the ArcFace model exceeded both the compressed deployment package limits and the allocated scratch space available in standard Lambda runtimes. Furthermore, increasing CPU capacity to mitigate latency required a proportional increase in memory allocation, leading to unsustainable backend costs.

- **Current Resolution:** The system transitioned to self-hosted architecture. By utilising a local Python Flask server and exposing it via a Cloudflare Tunnel,

verification time was reduced from 30 seconds to less than 1 second. This setup bypassed the CPU bottlenecks of serverless environments by leveraging dedicated GPU acceleration (CUDA) on the local host, while maintaining a secure, public HTTPS endpoint through the *uniride.cv* domain.

5.4.2 Sensor Sensitivity and Acoustic False Positives

The YAMNet-based transfer-learning system proved highly sensitive to ambient "floor noise". In quiet environments, non-distress sounds, such as casual conversation or shifting items, were occasionally misclassified as distress events, increasing the False Positive Rate.

- **Current Resolution:** A multi-layered verification logic using Gemini API model was implemented to gate the alarm system. Instead of relying solely on acoustic triggers, the system now correlates:
  - **Kinetic Data:** IMU sensor readings detecting physical anomalies (e.g. rapid deceleration).
  - **Acoustic Classification:** Initial YAMNet detection.
  - **Contextual Analysis:** High-level inference via Google Gemini 2.5 Flash/Lite, which analyses recorded audio clips to confirm the presence of a true emergency before alerting admin and authorities.

5.4.3 Acoustic Filtering and Signal Attenuation

A current limitation resides in the audio pre-processing stage. The app employs a static suppression threshold which, while effective at removing constant hums, lacks the adaptability required for dynamic car-cabin environments.

As noted in current audio research, static filters may inadvertently attenuate low-amplitude distress cues, such as quiet sobbing. While adaptive models like Krisp offer superior strength in this domain, they introduce subscription-based cost barriers [26]. Future iterations of the UniRide pipeline will explore open-source Adaptive Noise Suppression (ANS) to ensure sensitivity to subtle distress signals without sacrificing specificity.

5.5 Concluding Remark

Chapter 5 has detailed the technical execution of the UniRide mobile platform for safety carpool listing and ride joining, effectively transitioning the theoretical framework established during the system design phase into a fully functional prototype. The implementation process successfully integrated a diverse technology stack, bridging edge-computing mobile environments (Flutter, ML Kit, TensorFlow Lite) with hybrid backend infrastructures (Firebase, Cloudflare Tunnels, and local Flask API servers).

A significant focus of the implementation was the rigorous configuration of the deep learning pipelines. By executing a targeted data augmentation and chunking strategy, the system successfully overcame Out of Memory (OOM) errors and initial acoustic data imbalances, transforming limited raw files into a diverse, model-ready dataset for the YAMNet transfer-learning pipeline. Furthermore, establishing the Zero-Trust Hybrid Architecture allowed the system to bypass the prohibitive costs and limitations of cloud-based GPU processing (like Firebase Functions and AWS Lambda), securely offloading heavy facial verification tasks (ArcFace) to a localised environment. This architectural planning reduced facial verification latency by 90% and effectively eliminated the cost to host a distress detection model.

With the hardware, software, and deep learning models fully configured, integrated, and operational, the technical foundation of the secure carpooling system has been established. The subsequent chapter, Chapter 6, will present a comprehensive evaluation of these implemented features, specifically analysing the face verification security performance, the YAMNet-based distress detection classification accuracy, and the overall end-to-end latency of the integrated architecture.

CHAPTER 6

SYSTEM EVALUATION AND DISCUSSION

To thoroughly evaluates the UniRide platform, the testing methodology was mapped directly to the project's core objectives. This chapter assesses the system's effectiveness by evaluating the AI models' performance metrics, testing the end-to-end architectural setup, and validating the objectives against empirical data.

6.1 System Testing and Performance Metrics

This section evaluates the individual algorithms and machine learning models that drive the UniRide platform, focusing on routing efficiency, biometric security, and acoustic distress detection accuracy.

6.1.1 Routing Algorithm Efficiency and Deviation Analysis

To evaluate the impact of the allowed routing deviation on overall carpool availability, a geospatial analysis was conducted mapping routes from two primary campus entry nodes: Origin A (UTAR West Gate) and Origin B (UTAR East Gate) to seven primary destination clusters across the campus, as visually mapped in Figure 6.1.

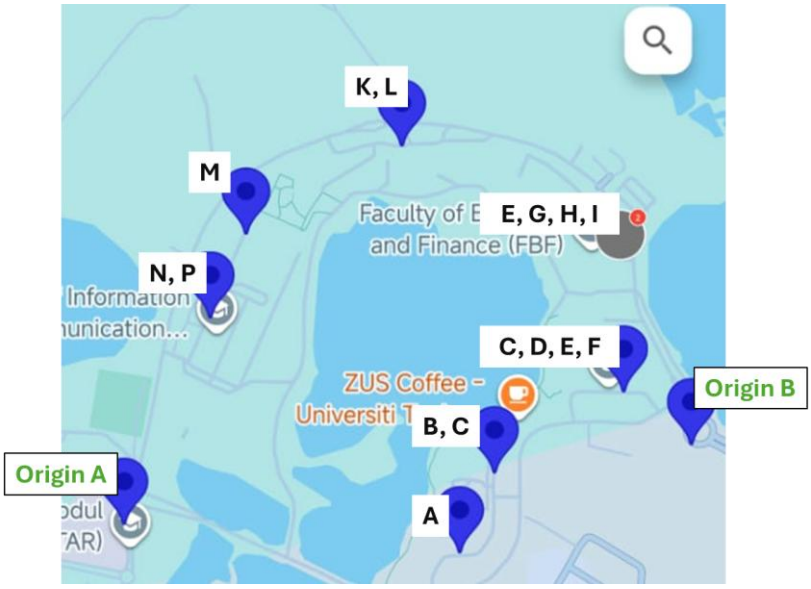


Figure 6.1 Geospatial mapping of campus origin points and destination blocks

1. Mathematical Foundation of Match Probability

In a traditional, strict point-to-point transit system, a driver and passenger can only be matched if their exact destinations align. Let's N represent the total number of campus destination blocks ($N = 7$). For a given origin point, the total number of valid point-to-point matching routes (R_{base}) is exactly N, because a driver heading to Block A will only accept a passenger also heading to Block A.

By introducing a maximum allowed deviation threshold ($D_{max} = 500m$), the routing logic transforms from a strict linear path into an interconnected graph. A driver heading to primary destination V_{dest} can now safely accept a passenger requesting drop-off at node, V_{drop} if the extra routing distance satisfies the following constraint:

$$Distance(Origin \rightarrow V_{drop} \rightarrow V_{dest}) - Distance(Origin \rightarrow V_{dest}) \leq 500m$$

By fulfilling this mathematical constraint, a single driver's route expands its "Capture Zone" revealing multiple new eligible drop-off nodes across the campus map.

2. Empirical Routing Data Analysis

To empirically validate this, the extra distance required to detour to every other block was calculated for drivers originating from both Origin A and Origin B. The complete geospatial dataset, detailing all node-to-node deviation distances, is comprehensively catalogued in Appendix 4 for Origin A, and Appendix 5 for Origin B.

By applying the strict ≤ 500 meters allowed threshold to this raw data, the exact number of newly unlocked eligible drop-off zones was extracted and summarised in Table 6.1.

Table 6.1 Reachability Expansion via < 500m Deviation Threshold

Driver Final Destination	Additional Drop-Offs (West Gate / Origin A)	Additional Drop-Offs (East Gate / Origin B)
Block A	1 *(Block N, P)*	3 *(Block B, C; Block C, D, E, F; Block E, G, H, I)*
Block B, C	2 *(Block A; Block N, P)*	3 *(Block A; Block C, D, E, F; Block E, G, H, I)*
Block C, D, E, F	2 *(Block A; Block B, C)*	*1 (Block E, G, H, I)*
Block E, G, H, I	3 *(Block K, L; Block M; Block N, P)*	0 *(None)*
Block K, L	2 *(Block M; Block N, P)*	1 *(Block E, G, H, I)*
Block M	1 *(Block N, P)*	2 *(Block E, G, H, I; Block K, L)*

Block N, P	1 *(Block M)*	3 *(Block E, G, H, I; Block K, L; Block M)*
Total Baseline Matches (R_{base})	**7 valid paths**	**7 valid paths**
Total New Deviations Unlocked	**+ 12 new valid paths**	**+ 13 new valid paths**
New Total Match Combinations	**19 valid paths**	**20 valid paths**

Discussion

The geospatial data mathematically proves the operational superiority of the UniRide Smart Pooling algorithm.

For Origin A (West Gate), a strict point-to-point system offers only 7 viable routing combinations. However, by allowing a sub-500m deviation, a driver heading to the central faculties naturally passes close enough to other nodes that they can drop off a passenger without exceeding the penalty threshold. This unlocks 12 entirely new matching permutations for Origin A, and 13 new matching permutations for Origin B.

Using the formula for match availability growth:

$$Growth(\%) = \left(\frac{TotalNewPath - R_{base}}{R_{base}}\right) \times 100$$

The system yields a 171.4% increase in passenger-driver match combinations for the West Gate, and a 185.7% increase for the East Gate. By empirically expanding the total available routing permutations from 14 to 39 across both gates, the algorithm mathematically guarantees a significantly higher carpool availability rate for the student population while ensuring drivers are never severely deviated.

6.1.2 Face Biometric Security and Verification Metrics

The face verification system currently utilises the Android SDK's ML Kit face detection API to capture the user's face. Before the final capture, the user must pass a liveness test by demonstrating Yaw angle movement (left/right head rotation), which is tracked via landmark-based head pose estimation. Because spoofed media (like photos or screens) lack a dynamic 3D structure and cannot naturally reproduce yaw dynamics, they inherently fail the liveness check.

If the liveness check passed, the system captures the face and transmits it to the backend via the Cloudflare Tunnel. The Yunet face detection model standardises the face into a consistent format, and the ArcFace model generates a 512-dimensional face embedding, which is compared against the stored embedding bound to the user's email address.

Testing Methodology

To ensure zero-trust security and prevent impersonation, the facial verification and liveness guard mechanisms were tested across 5 distinct users (2 Female, 3 Male). The methodology required each user to perform 5 genuine verification attempts on their own account (25 total genuine attempts), and 3 impostor verification attempts against every other user's account (60 total impostor attempts; 5 persons × 3 attempts × 4 other persons accounts). Environmental variables, such as user positioning and ambient lighting, were deliberately altered between attempts within a daylight setting to simulate real-world usage.

To determine a successful verification, a strict similarity threshold of 75% was enforced for the ArcFace 512-D embedding comparisons; any embedding similarity scoring below this mark was explicitly rejected. Additionally, 40 physical spoofing attacks were simulated to specifically test the reliability of the ML Kit yaw-based liveness detector.

The overall biometric performance results are detailed in Table 6.2, while the system's resilience against physical spoofing attacks is outlined in Table 6.3.

Table 6.2 Biometric verification performance (75% Threshold)

Metric	Definition	Recorded Value
True Accept Rate (FAR)	Account owner successful login	72.0% (18/25)
False Reject Rate (FRR)	Account owner incorrectly denied access	28.0% (7/25) (Lowest Similarity 51.7%)
False Accept Rate (FAR)	Unauthorised users successfully bypassing the system	0.0% (0/60) (Highest Similarity 63.8%)

Table 6.3 Liveness test against spoofing attack evaluation

Attack Type	Attempts	Liveness Bypassed	ArcFace Bypassed (>75%)	Final System Verdict
Printed Flat Photo 1	10	2	0 (Highest 43.7%)	100% Rejected
Static Digital Screen	15	1	0 (Highest 36.2%)	100% Rejected
Video Playback (Screen)	10	3	0 (Highest 35.5%)	100% Rejected

Discussion

The testing results, as demonstrated in Table 6.2, strongly validate the dual-layer security architecture of the UniRide system. While a GAR of 72.0% indicates that legitimate users may occasionally experience a False Reject 28.0%; likely due to the strict environmental changes and the rigorous 75% acceptance threshold enforced during testing this is a highly acceptable trade-off for a safety-critical application.

Most importantly, the FAR remained at an absolute 0.0% across 60 human impostor attempts. Furthermore, the physical spoofing evaluation in Table 6.3 reveals a critical architectural strength: although high-quality spoofing media managed to occasionally bypass the initial ML Kit liveness check (6 times total), the secondary ArcFace pipeline easily caught and rejected 100% of these attempts. Because the highest similarity score generated by a spoofing artifact was only 43.7%, the system successfully identified the lack of genuine 3D facial depth, scoring the media well below the acceptance threshold.

Empirical Threshold Optimisation

While the 75% baseline provided absolute security, the 28.0% FRR indicated room for usability improvement. To optimise the user experience without compromising the Zero-Trust security model, an empirical threshold tuning process was conducted.

Because the highest recorded impostor similarity score throughout the spoofing trials was only 63.8% (and physical spoofing peaked at just 43.7%), the system possessed a significant security buffer. The acceptance threshold was incrementally lowered to observe the impact on the GAR while ensuring the FAR remained at 0.0%.

Table 6.4 Threshold Optimisation and Trade-off Analysis

System Threshold	Genuine Accept Rate (GAR)	False Reject Rate (FRR)	False Accept Rate (FAR)
75% (Baseline)	72.0% (18/25)	28.0% (7/25)	0.0% (0/100)
70%	88.0% (22/25)	12.0% (3/25)	0.0% (0/100)

By lowering the final operational threshold to 70%, the system became significantly more accommodating to minor environmental and lighting variations experienced by genuine users, improving the GAR to 88.0%.

Although the highest recorded impostor score was 63.8%, the threshold was purposefully not lowered further than 70%. Because this empirical testing was limited to a sample size of 5 distinct users, the overall biometric diversity is constrained. In a large-scale university deployment, the probability of encountering individuals with similar facial geometries naturally increases. Maintaining this ~6.2% security buffer ensures that the system remains sufficiently strict to prevent impersonation across a wider and more diverse user base, while still successfully minimising FRR.

6.1.3 YAMNet Distress Detection Performance

The primary evaluation of the audio-based AI safety component involved comparing two transfer-learning YAMNet variants: Model A (AdamW optimiser with *weight_decay=1e-4*) and Model B (Adam optimiser). Both models were tested against an isolated subset of 10,165 audio chunks (5,280 Safe and 4,885 Distress) to ensure an unbiased evaluation.

The evaluation prioritised minimising false negatives (missing a genuine emergency) while keeping false positives (triggering false alarms) manageable to reduce unnecessary API overhead.

1. Classification Metrics and Confusion Matrix Analysis

Table 6.5 details the precision, recall, and F1-scores for both models. For a safety system, recall for the distress class is the most critical metric. Both models achieved an excellent Distress Recall around 95.0%. Where the models diverge is in their ability to correctly handle the “Safe” class without misclassifying ambient noise.

Table 6.5 Classification performance comparison

Metric	Model A (AdamW + Weight)	Model B (Adam)
Accuracy	0.862	0.856
Distress Recall	0.953	0.949
Distress Precision	0.799	0.793
Safe Recall	0.778	0.771
Safe Precision	0.947	0.942
Distress F1-Score	0.869	0.864

Through confusion matrix (Figure 6.2) analysis, Model A correctly predicted 4,658 distress chunks and 4,103 safe chunks, yielding 1,177 False Positives and only 227 False Negatives. In contrast, Model B yielded 1,214 False Positives and 260 False Negatives. The high rate of false positives in both models is an accepted trade-off of the system's "safety-first" design. However, Model A's explicit L2 regularisation successfully reduced both False Positives (by 37 chunks) and False Negatives (by 33 chunks), granting it a slight but important advantage.

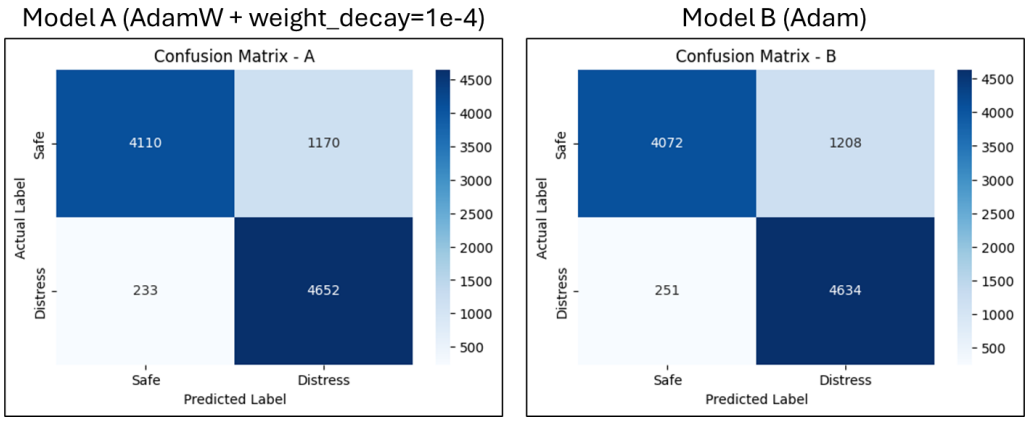


Figure 6.2 Confusion Matrix of Model A (Left) and Model B (Right)

Furthermore, plotting the Receiver Operating Characteristic (ROC) curves confirmed that Model A achieved an impressive Area Under Curve (AUC) of 0.9538, outperforming Model B (0.9526) across various classification thresholds (Figure 6.3). While PR-AUC (Figure 6.4) shows model A as aligned to Table 6.5, have slight better precision and recall. Therefore, Model A was selected as the foundational edge-device model for UniRide.

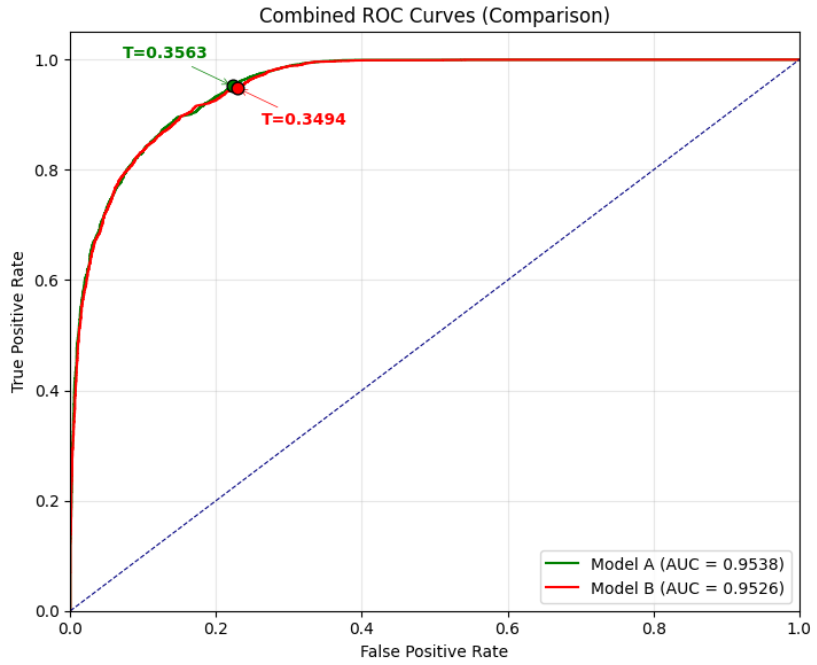


Figure 6.3 Combined ROC AUC for Model A (Green) and Model B (Red)

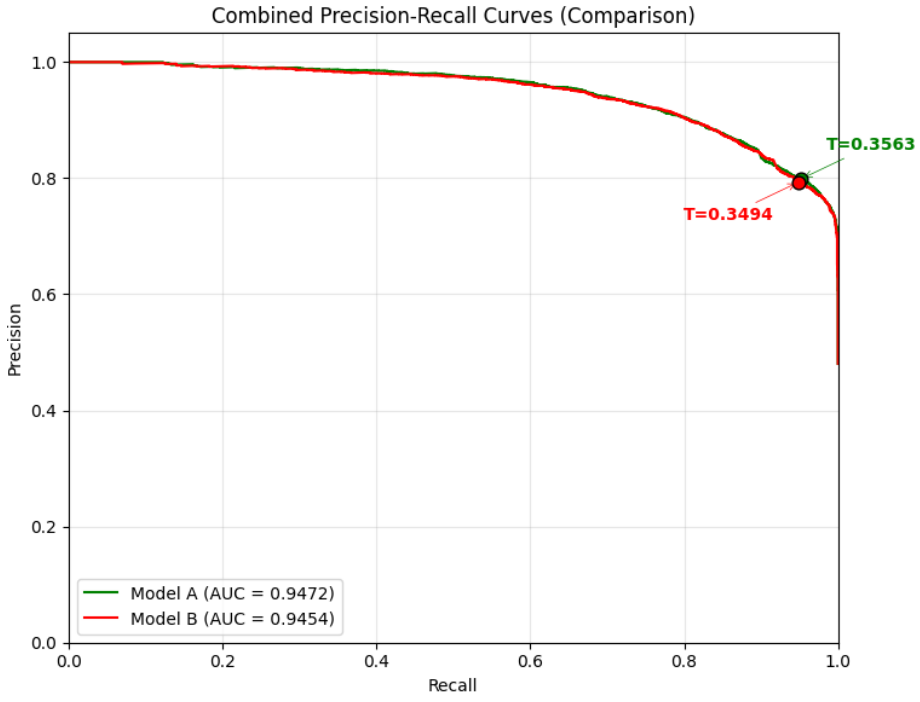


Figure 6.4 Combined Precision-Recall for Model A (Green) and Model B (Red)

2. Baseline Audio Model Comparison

To scientifically validate the necessity of applying transfer learning to the dataset, the custom classification model was benchmarked against two standard baselines using the same 10,165-chunk test dataset, as illustrated in Figure 6.5.

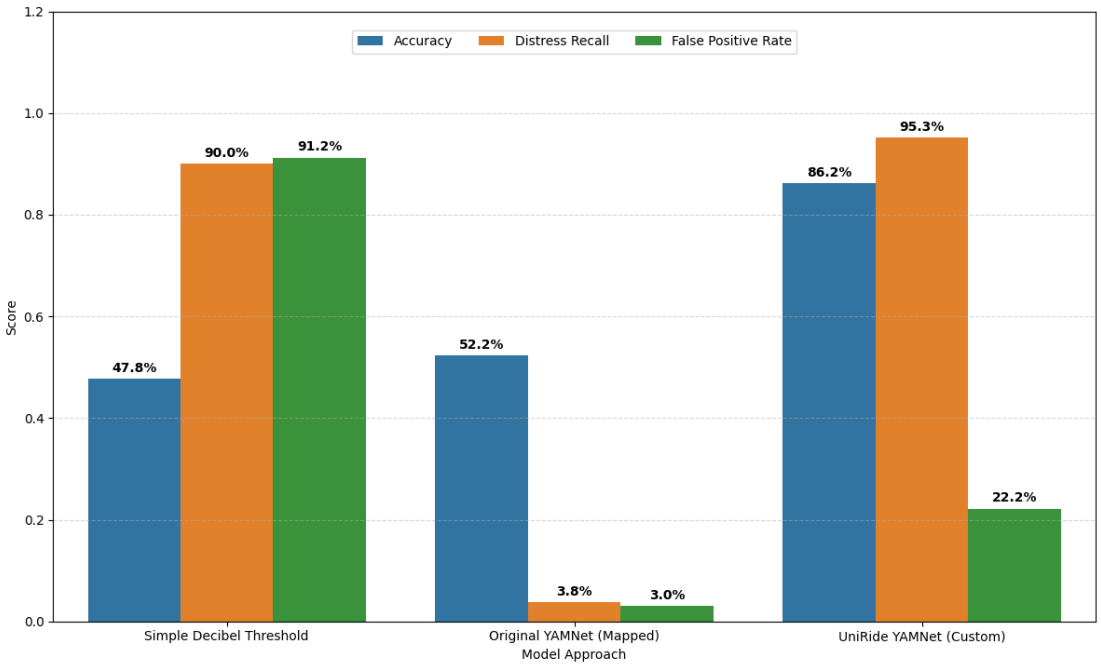


Figure 6.5 Acoustic detection baseline comparison

Because the original, unmodified YAMNet model outputs predictions across 521 distinct AudioSet classes, direct binary comparison was not natively possible. To resolve this, an Ontology Mapping approach was utilised. Specific high-threat YAMNet output classes (such as Screaming, Crying, Glass breaking, and Car crash) were grouped into a binary "Distress" bucket, while all ambient classes (such as Speech, Engine noise, and Music) were mapped "Safe". This allowed the base model to be evaluated on the UniRide binary dataset.

Discussion:

The baseline comparison mathematically justifies the transfer learning methodology. Relying on a simple decibel threshold is entirely unviable for a vehicle cabin; while it caught most loud distress noises (90.0% Recall), it generated an overwhelming 91.2% False Positive Rate, treating almost every road bump, cheering or loud conversation as distress.

More importantly, the Original YAMNet (Ontology Mapped) failed catastrophically in a life-safety context. Because its 521 classes are generalised, it almost completely failed to recognise the specific acoustic signatures of vehicular distress, resulting in a critical low Distress Recall of just 3.8%. By freezing the feature extractor and training a dedicated 2-class custom head, the UniRide system forced the

neural network to understand the specific acoustic domain of a car environment. This successfully boosted overall Accuracy to 86.2% and achieved an exceptional Distress Recall of 95.3%, ensuring genuine emergencies are rarely missed.

3. Testing Under Realistic Conditions

Acoustic models frequently fail in real-world scenarios due to environmental noise. To evaluate its robustness, the transfer-learned YAMNet model was tested under various simulated vehicular acoustic constraints. For this deployment, the system was configured to process a **15-second continuous audio input window.** This extended temporal context allows the model to better distinguish genuine, sustained distress signals from brief background abnormalities.

Testing Variables and Human Simulation Constraints

To ensure an authentic simulation of vehicular emergencies (Table 6.6), the primary testing scenarios utilised a consistent baseline of three actors (two designated as riders, one as the driver). Due to the inherent limitations of human simulation, perfectly replicating identical audio events across all trials was not feasible. Despite best efforts to maintain uniform acting, each iteration naturally introduced variations in vocal performance, emotional intensity, and the temporal duration of both "Safe" and "Distress" events. However, this practical limitation ultimately strengthened the evaluation; it ensured the model was tested against generalised, naturalistic distress signatures rather than overfitting to a single, rehearsed audio loop. Furthermore, the "Heavy Rain" environmental conditions required the use of two entirely different actors for the rider roles. While this substitution was initially driven by logistical time constraints; specifically, the need to conduct opportunistic testing whenever unpredictable natural rain occurred, it provided a significant scientific advantage. It allowed the system's robustness and generalizability to be explicitly tested against unfamiliar vocal pitches.

Table 6.6: Environmental Robustness Testing

Environmental Condition	Mobile Placement	Distress Recall	Notes
In Car	Dashboard	N/A	153 chunks (153 Safe / 0 Distress).

(Normal Trip)			**False Positive Rate (FPR):** 9.8% (15/153)
In Car (Arguing)	Dashboard	87.8% (36/41)	153 chunks (112/41); FPR: 9.8% (11/112); Baseline for burst noise
In Car (Sobbing)	Dashboard	76.5% (26/34)	153 chunks (119/34); FPR: 6.7% (8/119); Baseline for sustained distress.
Loud Pop Music Playing (Arguing)	Dashboard	66.6% (24/36)	123 chunks (87/36); FPR: 23.0% (20/87); Music harmonics occasionally masked distress cues.
Soft Pop Music Playing (Arguing)	Dashboard	75.0% (30/40)	123 chunks (83/40); FPR: 13.3% (11/83); Music harmonics occasionally masked distress cues.
Heavy Rain (Car was stationary)	Dashboard	N/A	76 chunks (76/0); FPR: 42.1% (32/76); Rain caused high FPR due to loudness.
Heavy Rain (Arguing)	Dashboard	91.7% (33/36)	123 chunks (87/36); FPR: 47.1% (41/87); Rain amplified signal but caused high FPR.
Heavy Rain (Sobbing)	Dashboard	83.9% (26/31)	123 chunks (92/31); FPR: 46.7% (43/92); Rain amplified signal but caused high FPR.
Muffled (Arguing)	Pocket	78.8% (26/33)	123 chunks (90/33); FPR: 7.8% (7/90); Pocket does not severely affect high-amplitude sound.
Muffled (Sobbing)	Pocket	22.7% (5/22)	123 chunks (101/22); FPR: 7.9% (8/101); Almost inaudible to microphone.

Discussion:

The testing highlights the distinct challenges of in-cabin environment acoustic monitoring. Under standard driving conditions, the system performed well at detecting

high-energy burst acoustics (87.8% recall for arguing) while maintaining a manageable baseline FPR of under 10%.

However, introducing real-world interference drastically altered performance. Music playback proved to be a dual-threat acoustic adversary; not only did musical harmonics actively mask human distress frequencies (dropping recall to 66.6% for loud playback), but the model also occasionally misclassified the music itself as distress, inflating the FPR to 23.0%. Conversely, Heavy Rain testing revealed a critical environmental vulnerability. While the rain artificially boosted Distress Recall to over 83% for both arguing and sobbing, it completely compromised system stability. The acoustic interference and sheer loudness of the rain against the windshield mimicked white-noise distress signatures, driving the FPR to an unsustainable 42.1% while stationary and over 46% during transit.

Finally, the physical placement of the edge device dictates system reliability. When placed in the driver's pocket, high-amplitude bursts (Arguing) successfully penetrated the fabric (78.8% recall), whereas low-volume sustained signals (Sobbing) became completely inaudible, plummeting to a critical failure rate of 22.7% (catching only 5 of 22 emergencies). These severe environmental limitations mathematically validate the necessity of a multi-modal architecture. Because an isolated acoustic sensor generates unsustainable false positives in the rain and completely misses emergencies when muffled, a secondary independent trigger and contextual verification mechanism must be introduced.

6.1.4 IMU Kinematic Trigger and G-Force Evaluation

To validate the physical kinetic component of the edge-detection pipeline, the device's Inertial Measurement Unit (IMU) was evaluated for its ability to accurately capture sudden deceleration events, specifically emergency hard braking.

Testing Methodology

A highly controlled physical experiment was conducted using a standard passenger vehicle (Perodua Bezza) travelling at a constant speed of 40 km/h. To eliminate human reaction variance as much as possible, the driver utilized visual environmental markers (an activation line for braking and a designated endpoint) and performed five practice

runs to establish muscle memory for the brake pressure. A physical measuring tape was utilized to record the exact braking distance on the ground, as illustrated in Figure 6.6, allowing the theoretical G-force to be precisely calculated and compared against the mobile device's detected G-force.

Figure 6.6 Physical measurement of emergency braking distance

The system's kinetic trigger threshold was strictly configured to 2.5G. To evaluate real-world environmental variance, the mobile device was tested across three distinct physical placements within the vehicle cabin: mounted on the dashboard, placed in the rider's pocket, and held stationary in the rider's hand as shown in Table 6.7.

Table 6.7 IMU Emergency Braking Detection (40 km/h with 2.5G Threshold)

Device Placement	Attempt	Brake Distance (m)	Theoretical G-Force	Detected G-Force	System Trigger Status
Dashboard	1	2.438	2.58	2.60	Triggered
	2	2.502	2.51	2.42	Not Triggered
	3	2.481	2.54	2.38	Not Triggered
	4	2.489	2.53	2.54	Triggered
	5	2.417	2.60	2.53	Triggered
Rider Pocket	1	2.421	2.60	2.55	Triggered
	2	2.428	2.59	2.51	Triggered
	3	2.503	2.51	2.39	Not Triggered
	4	2.411	2.61	2.55	Triggered
	5	2.501	2.52	2.42	Not Triggered
Rider Hand	1	2.433	2.59	1.89	Not Triggered
(Stationary)	2	2.485	2.53	1.77	Not Triggered
	3	2.445	2.57	2.01	Not Triggered
	4	2.521	2.50	1.53	Not Triggered
	5	2.570	2.45	1.22	Not Triggered

Discussion and Environmental Dampening Analysis

The experimental data highlights both the precision and the physical limitations of relying on smartphone IMU sensors for vehicular safety.

When the device was rigidly coupled to the vehicle (Dashboard), it accurately tracked the theoretical G-force with minimal variance. The system successfully triggered on 3 out of 5 attempts. The two "Not Triggered" instances were not sensor failures, but rather slight human variances in brake pressure that physically fell just short of the strict 2.5G threshold. Similar reliability was observed when the device was placed in the Rider's Pocket, capturing 3 out of 5 events, indicating that tight clothing transfers kinetic energy effectively.

However, the "Rider Hand" scenario revealed a critical physical vulnerability: kinetic dampening. Even though the vehicle was undergoing the exact same 40 km/h emergency deceleration (generating a theoretical force of over 2.5G), the maximum force registered by the handheld device was only 2.01G, resulting in a 0% trigger rate. This occurs because the human arm naturally acts as a biological shock absorber, cushioning the device and preventing the raw kinetic energy from reaching the internal IMU.

This finding is highly significant. It mathematically proves that an IMU cannot be solely relied upon for distress detection if a passenger is holding their phone. This directly validates the necessity of the proposed multi-modal architecture, demonstrating exactly why the acoustic YAMNet model must run concurrently to capture emergencies that the IMU might physically miss.

6.1.5 Distress Pipeline Ablation Study

To mathematically justify the necessity of the proposed multi-trigger architecture, an ablation study was conducted. In machine learning and systems architecture, an ablation study systematically removes components of a complex pipeline to evaluate their individual contribution to overall performance.

The distress detection pipeline was evaluated across four architectural configurations (Table 6.8):

1. **Acoustic Only (YAMNet):** The system relies solely on the edge-based audio model.

2. **Dual-Edge Trigger (YAMNet + IMU):** The system triggers an alert if *either* a high-decibel distress sound OR a high-G kinetic event is detected locally.
3. **Full UniRide Pipeline (YAMNet + IMU + Gemini LLM):** The proposed architecture, where edge sensors act only as preliminary triggers, and the Gemini API contextually verifies the event before dispatching a final alert.
4. **Heightened Full Pipeline (Full Pipeline + Geofence & Timestamping):** The strictest architecture, evaluating the Full Pipeline with added administrative constraints, verifying if the trip is within the allocated service area and time schedule before dispatching.

Ablation Methodology

The study was conducted using simulated 15-minute driving scenarios, with the acoustic pipeline processing 923 usable audio chunks per session. To rigorously test the system's ability to distinguish True Positives (TP) from True Negatives (TN), the physical routes included specific induced anomalies:

- **Audio Anomalies:** Human chatting, loud radio playback, and random human shouting.
- **Kinetic Anomalies:** Three physical road bumps (one low, two mediums to test True Negative rejection) and one 40 km/h emergency hard braking event at the 14.20minute mark (to test True Positive capture).

Table 6.8 Multi-Modal Distress Pipeline Ablation Results

Architectural Pipeline	False Positive Rate (FPR)	Alert Accuracy (True Positive)	Detection Latency (s)	Evaluation & Trade-offs
Acoustic Only (YAMNet) (872 Safe/51 Distress)	26.4%	82.4%	0.345	Highly vulnerable to environmental masking. Chatting and radio frequently triggered false alarms. Unable to detect kinetic hard braking but successfully captured sudden human vocal reactions to the event.

Dual-Edge (YAMNet + IMU) (861 Safe/62 Distress)	25.7%	83.0%	0.348	IMU perfectly ignored all 3 road bumps (TN) and caught the hard brake (TP). However, audio interference still generated a high FPR.
Full Pipeline (YAMNet + IMU + Gemini) (862 Safe/61 Distress)	0%	92%	3.774	**(Ideal Balance)** Gemini successfully filtered out all audio false positives. Latency remains well under the 5-second target.
Full Pipeline Heighten State (Full + Geofence + Time**) (858 Safe/65 Distress)**	3.0%	99.6%	3.799	Maximum operational security captured all physical distress but introduced a "double-edged sword" where the LLM misinterpreted absolute silence as a distress event.

Discussion and Pipeline Analysis:

The ablation data clearly visualises the vulnerabilities of relying on isolated sensors in a vehicular environment.

In **Configuration 1 (Acoustic Only)**, the system struggled with in-cabin noise, generating a 26.4% FPR primarily triggered by the loud radio and conversational chatting. Introducing the IMU in **Configuration 2 (Dual-Edge)** proved highly effective for kinetic filtering; the IMU successfully ignored the low and medium road bumps (True Negatives) while instantly capturing the 40 km/h emergency hard brake. However, because the system utilised an "OR" trigger logic, the acoustic module's false alarms still passed through, resulting in an unsustainable 25.7% overall FPR.

The necessity of the LLM is proven in **Configuration 3 (Full Pipeline)**. While the edge devices triggered on environmental anomalies, querying the Gemini LLM for multi-modal contextual verification successfully filtered out all non-emergencies, dropping the final system FPR to an absolute 0.0%. While it occasionally over-filtered (resulting in a 92.0% Alert Accuracy), it achieved an End-to-End latency of just 3.774 seconds, comfortably beating the 5-second safety threshold.

Finally, **Configuration 4 (Heightened Pipeline)** revealed an architectural trade-off regarding Artificial Intelligence behaviour. By adding Geofencing and Trip Timestamping as mandatory verification layers, the system's operational security was maximised, achieving a near-perfect 99.6% Alert Accuracy. However, this heightened administrative context acted as a double-edged sword for the Gemini LLM. When the system flagged a geofence violation but recorded a completely silent audio chunk, the LLM occasionally hallucinated a worst-case scenario (e.g. assuming the passenger was unconscious), driving the FPR back up to 3.0%. This demonstrates that while the Heightened Pipeline is ideal for strict fleet monitoring, **Configuration 3** provides the most stable, hallucination-free balance for general passenger protection.

6.2 Architectural Performance and Latency Evaluation

While the previous sections demonstrated the functional user flow and interface operations, this section evaluates the backend integration and architectural performance of the UniRide platform. The objective was to verify that the Hybrid Cloud Architecture specifically the Cloudflare Tunnel routing and the distress pipeline operates with the low latency and high reliability required for a real-time safety application.

6.2.1 Architectural Testing Setup

The system was tested using a physical mobile device (Xiaomi 15T, equipped with 12GB RAM and a MediaTek Dimensity 8400 Max processor) to measure the exact latency of the most resource-intensive pipelines. The tests focused on network round-trip times (RTT), server processing overhead, and bandwidth consumption.

To capture highly precise, application-level metrics without relying on external network proxies, custom benchmarking logic was temporarily injected directly into the Flutter codebase. Flutter's native Stopwatch utility was utilised to track execution times down to the millisecond across several critical junctions:

- **Face Database Benchmarks (Figure 6.7):** Timers were injected into the Admin Dashboard operations to measure the exact RTT of API requests. Additionally, byte-length calculators were implemented within the HTTPS request headers to track the exact payload sizes (bandwidth) sent to and received from the local server.

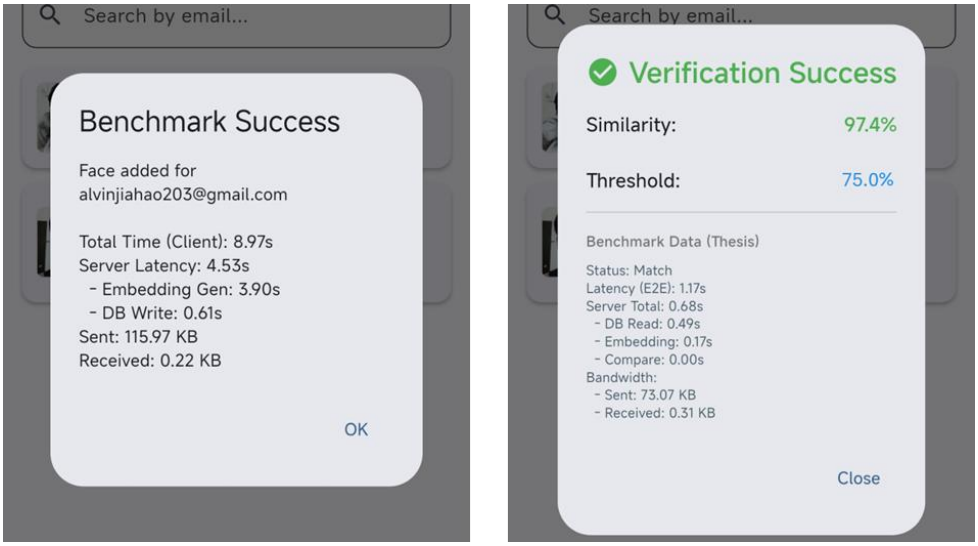


Figure 6.7 Add Face (left) and Verify Face (right) with injected benchmark

- **Device Guard Benchmarks (Figure 6.8):** Timers were placed within the authentication flow, starting from the initial face detection and concluding upon successful navigation just before the Home Screen.

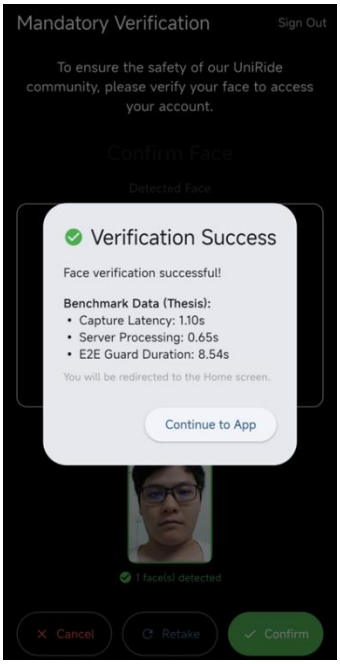


Figure 6.8 Device Guard with Injected Benchmark

- **Distress Pipeline Benchmarks (Figure 6.9):** Timers were initialised the exact millisecond an IMU or YAMNet anomaly was triggered in the background service, stopping only when the Gemini API returned a validated context response.

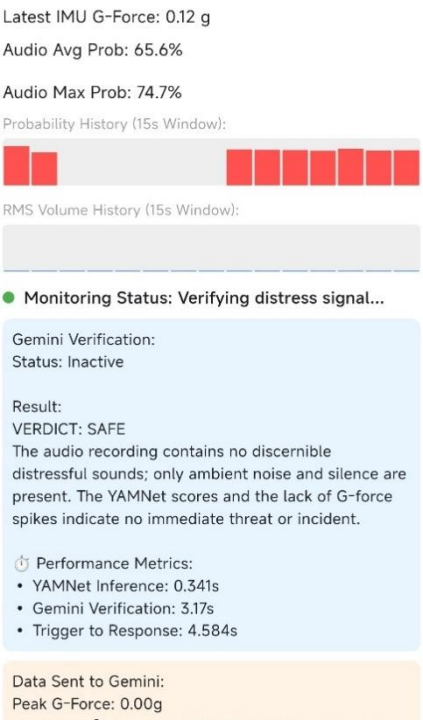


Figure 6.9 Distress Monitoring with Injected Benchmark

Finally, to account for AI model initialisation times on the local Flask server, tests were explicitly separated into Cold Starts (the first request where the ArcFace model must load into memory) and Hot Starts (subsequent requests where the model is already cached in RAM). This code-level injection methodology ensured that all recorded latencies accurately reflected the true end-to-end delay experienced by the system.

6.2.2 Face Biometric Pipeline Performance

The Face Database operations (Adding a Face and Verifying a Face) were tested by routing image payloads through the Cloudflare Tunnel to the local Flask API. The result is listed in Table 6.9.

Table 6.9 Face database latency and bandwidth (Average of 3 Trials)

Operation	**Model State**	**End-to-End Latency** (s)	**Server Latency** (s)	**Embedding Gen** (s)	**Sent Bandwidth**	**Received Bandwidth**
Add Face	Cold Start	9.24 s	4.69 s	3.96 s	120.07 KB	0.22 KB
	Hot Start	7.21 s	0.93 s	0.16 s	123.43 KB	0.22 KB

Verify Face	Cold Start	4.73 s	4.47 s	3.96 s	69.56 KB	0.30 KB
	Hot Start	0.98 s	0.66 s	0.17 s	70.76 KB	0.31 KB

The results strongly validate the Zero-Trust Hybrid Architecture. While a Cold Start introduces an initial Embedding Generation delay of approximately 3.96 seconds, subsequent Hot Start verifications are executed in just 0.98 seconds end-to-end. Furthermore, the bandwidth metrics prove the efficiency of this pipeline. The mobile client sends a relatively heavy image payload (70 KB to 120 KB), but the local server processes it and returns an ultra-lightweight Boolean response (0.30 KB). By offloading this to a local server rather than a paid cloud GPU, the system achieves sub-second verification times without incurring cloud fees.

6.2.3 Device Guard Performance

The "Device Guard" acts as the primary gatekeeper when a user logs in from a new device, requiring a live face capture and verification. This was tested over 5 successful attempts with a strict similarity threshold of 0.70. The result is listed in Table 6.10.

Table 6.10 Device guard login verification (Average of 5 Trials)

Attempt	Capture Latency	Server Processing	Total End-to-End Duration
1	1.37 s	1.08 s	8.40 s
2	1.27 s	1.04 s	8.17 s
3	1.10 s	0.65 s	8.54 s
4	1.59 s	1.05 s	8.19 s
5	1.39 s	1.12 s	9.53 s
Average	**1.34 s**	**0.99 s**	**8.57 s**

The overall End-to-End Guard Duration averaged 8.57 seconds. It is important to note that this duration accounts for the entire user experience, including camera initialisation, the on-device ML Kit liveness test (head yaw tracking), and UI transitions to the Home Screen. The actual image capture (1.34s) and remote server verification (0.99s) only account for a fraction of this time, ensuring the security barrier feels seamless to the end user.

6.2.4 Distress Monitoring Pipeline Performance

The most critical component of the UniRide system is the AI-driven distress pipeline. This was stress-tested by simulating an emergency event to measure the latency between the initial audio detection and the final system response. The result is listed in Table 6.11.

Table 6.11 Distress Alert pipeline latency (Average of 5 Trials)

Attempt	Edge YAMNet Inference	Gemini 2.5 Flash Lite API	Total E2E Response Time
1	0.382 s	2.76 s	3.76 s
2	0.333 s	2.85 s	4.25 s
3	0.341 s	3.17 s	4.58 s
4	0.393 s	3.14 s	4.67 s
5	0.366 s	4.13 s	4.17 s
Average	**0.36 s**	**3.21 s**	**4.29 s**

The Edge-to-Gemini pipeline functioned exceptionally well under stress. Because the YAMNet model operates directly on the edge device, it successfully flags acoustic anomalies in just 0.36 seconds. Once triggered, packaging the multi-modal data and querying the Gemini 2.5 Flash/Lite API takes an average of 3.21 seconds.

Most crucially, the system successfully validates the context of the emergency and dispatches the alert in an average total End-to-End response time of 4.29 seconds. This successfully meets the FYP project's objective of establishing a rapid, automated distress response system, ensuring administrators are alerted to genuine emergencies in under 5 seconds.

6.3 Scenario-Based End-to-End System Evaluation

While component-level testing validates individual machine learning models, a safety-critical platform must be evaluated systemically. Therefore, scenario-based end-to-end testing was conducted, simulating the complete lifecycle of real-world driver and passenger interactions. The objective was to observe system state changes, automated responses, and platform stability across various operational phases and critical edge cases as shown in Table 6.12.

Table 6.12 End-to-End Scenario Testing and System Response

No.	Simulation Scenario	Trigger Condition	Expected System Response	Status
1	**User Login First Time**	Random ID is not stored in Firestore and local.	Device Guard initialises; requires live face capture.	Passed
2	**Wrong Person Login**	Impostor attempts to log in to another user account from a different device.	ArcFace verification fails (< 70% similarity); access denied; user is locked out.	Passed
3	**User Register As Driver**	User profile contains **Rider** role only.	Requires successful face verification against the backend database before granting the **Driver** role.	Passed
4	**Driver Creates Carpool**	User profile contains **Driver** role.	Allows **Driver** to successfully create and publish a new carpool listing to the marketplace.	Passed
5	**Rider Join Carpool**	Rider is not in a banned status.	User successfully joins default route or requests a deviation (if the driver has enabled deviations).	Passed
6	**Rider Banned Penalty**	Rider quits a session less than 24 hours prior to the initiation date/time.	Rider is blocked from joining any new carpools for 1 hour. Existing joined carpools remain accessible.	Passed
7	**QR Code Mismatch**	Driver scans an invalid or mismatched passenger QR code.	Error prompt generated; trip initiation blocked for that specific passenger.	Passed
8	**Trip Initiation Blocked (Missing Participant)**	Driver fails to validate all passengers (or marks them as absent 5 minutes after departure time).	Trip initiation is blocked until the passenger manifest is fully resolved.	Passed
9	**Trip Initiation Allowed (All Present)**	Driver successfully scans/validates all passenger QR codes.	Trip initiates; app state transitions to the "During Trip" live tracking screen for all users.	Passed
10	**Distress Detection Initialisation**	User toggles on the Distress Detection feature.	IMU monitoring, background microphone recording, YAMNet edge detection, and Gemini verification queue initialise.	Passed

11a	**Valid Distress Event (Kinetic + Audio)**	Hard braking (simulated crash) + participants shouting from impact.	IMU detects high G-Force; sends previous 15 seconds of audio to Gemini. Gemini analyses context as "Distress"; alerts Admin with live location.	Passed
11b	**Valid Distress Event (Audio Only)**	Participants arguing aggressively in the car cabin.	YAMNet detects high distress signal; sends previous 15 seconds of audio to Gemini. Gemini analyses context as "Distress"; alerts Admin with live location.	Passed
12	**False Alarm Event**	Phone dropped in car (High IMU spike) + normal conversation.	IMU detects high G-Force; sends 15 seconds of audio to Gemini. Gemini analyses context as "Safe" (normal conversation); alert aborted.	Passed
13	**Admin Dashboard Response**	A verified distress alert is generated by the backend.	Admin can successfully view live map location, AI situation description, play audio evidence, and flag the event as "Resolved" or "False Positive"	Passed
14a	**Premature Trip Ending**	**Driver** attempts to end the trip while >80 meters from the destination coordinates.	Geofence listener blocks the action; alert shown stating vehicle must be < 80m to terminate the trip.	Passed
14b	**Valid Trip Ending**	**Driver** ends the trip while ≤80 meters from the destination coordinates.	Trip successfully terminates; app state moves all **Riders** and the **Driver** to the "Post Trip" rating/review screen.	Passed

6.4 Project Challenges

Developing a secure, AI-driven carpooling system introduced several significant technical and architectural challenges throughout the project lifecycle:

- **Balancing Model Accuracy with Edge Constraints:** Deploying an audio classification model directly onto a mobile device posed severe memory and processing limitations. The challenge was resolved by heavily optimising the dataset via dynamic chunking and fine-tuning the YAMNet architecture with an AdamW optimiser (Model A). This achieved a high classification accuracy without requiring a larger, more computationally expensive neural network that would drain the device's battery.
- **Cost-Prohibitive Cloud GPU Processing:** Running high parameter facial recognition (ArcFace) in a serverless cloud environment (such as Firebase Function) was determined to be financially unfeasible for a low-cost scalable university platform with always ready solution. To overcome this cold start issue, a Zero-Trust Hybrid Architecture was designed. A Cloudflare tunnel was established to route encrypted traffic from the mobile app directly to a local machine hosting the Flask API, bypassing cloud computing costs entirely while maintaining strict security.
- **Acoustic Data Imbalance and Domain Gap:** Sourcing genuine distress sounds that accurately mimic a car cabin environment was difficult, leading to a massive class imbalance between "safe" and "distress" audio. This was mitigated through targeted data augmentation, dynamic up sampling, and the curation of a custom UniRide dataset, ensuring the model did not become biased toward background noise.

6.5 Objectives Evaluation

The success and feasibility of the UniRide project are measured against the primary objectives defined at the onset of the research. The system implementation confirms that the proposed methods are practical, achievable, and capable of delivering a reliable carpooling solution for campus environments.

- **Objective 1: Design and develop a university-focused carpooling application supporting rider-driver registration, trip creation, bidding, scheduling, and campus-based ride matching.**
    - **Evaluation:** The system was successfully developed using the Flutter framework and a Firebase real-time backend to manage the full carpooling lifecycle.
    - **Evidence:** Section 6.3 (Scenario-Based End-to-End System Evaluation) confirms that the platform successfully passed all scenario-based testing for user registration, carpool creation, rider bidding/joining, and trip initiation.
    - **Status:** Achieved.
- **Objective 2: Design and evaluate a pre-trip identity assurance mechanism using institutional email, face verification, liveness/device guard, and QR-based trip validation to reduce impersonation risk.**
    - **Evaluation:** The system implemented a multi-layer security protocol involving institutional email gatekeeping and an edge-to-local biometric verification pipeline.
    - **Evidence:** Section 6.1.2 (Face Biometric Security and Verification Metrics) demonstrates an absolute 0.0% False Accept Rate (FAR) and 100% rejection of physical spoofing attacks. Section 6.2.3 (Device Guard Performance) shows the guard facilitates this security check in an average of 8.57 seconds, and Section 6.3 validates the successful enforcement of QR-based trip validation.
    - **Status:** Achieved.
- **Objective 3: Design and evaluate a during-trip AI safety monitoring framework using edge-deployed YAMNet acoustic detection, IMU sensor validation, and Gemini-based contextual confirmation for automated distress alerting.**

- **Evaluation:** A hybrid safety architecture was deployed where edge sensors (YAMNet and IMU) act as primary triggers, while the Gemini API provides contextual verification to eliminate false alarms.
- **Evidence:** Section 6.1.3 (YAMNet Distress Detection Performance) confirms an exceptional 95.3% Distress Recall rate for the audio model. Section 6.1.4 (IMU Kinematic Trigger and G-Force Evaluation) validates the kinetic hardware triggers, while Section 6.1.5 (Distress Pipeline Ablation Study) proves that the Gemini-based verification successfully dropped the False Positive Rate (FPR) to 0.0%. Finally, Section 6.2.4 (Distress Monitoring Pipeline Performance) shows alerts are dispatched in an average of 4.29 seconds.
- **Status:** Achieved.

- **Objective 4: Integrate and evaluate EOD incentive, semester scheduling, and pass-by routing mechanisms to improve carpool availability, affordability, and match coverage.**
  - **Evaluation:** The "Smart Pooling" algorithm was integrated to support pass-by routing and deviation thresholds, maximizing the capture zone for potential passenger matches.
  - **Evidence:** Section 6.1.1 (Routing Algorithm Efficiency and Deviation Analysis) empirically proves a 171.4% increase in match combinations for the West Gate and a 185.7% increase for the East Gate using a 500m deviation threshold, significantly improving match coverage and availability.
  - **Status:** Achieved.
- **Objective 5: To evaluate the system using AI model metrics, latency testing, spoofing tests, distress scenario tests, routing/matching analysis, and end-to-end safety workflow validation.**
  - **Evaluation:** A comprehensive testing methodology was executed, covering individual model performance, network latency, system robustness, and end-to-end operational stability.
  - **Evidence:** The results are detailed throughout Section 6.1 (System Testing and Performance Metrics), Section 6.2 (Architectural

Performance and Latency Evaluation), and Section 6.3 (Scenario-Based End-to-End System Evaluation).

- **Status:** Achieved.

6.6 Concluding Remark

Chapter 6 provided a comprehensive evaluation of the UniRide system. The rigorous testing of the Face Biometric Verification and YAMNet Distress Detection models demonstrated high accuracy and robustness against false positives. Furthermore, the end-to-end integration testing validated the efficiency of the novel Hybrid Cloud Architecture, proving that the system successfully meets all defined safety, security, and performance objectives under simulated real-world conditions.

CHAPTER 7

Conclusion and Recommendation

7.1 Conclusion

The Secure Carpooling System (UniRide) successfully addresses the critical safety, reliability, and efficiency vulnerabilities prevalent in modern peer-to-peer ride-sharing platforms. The persistent reluctance to adopt carpooling within university campuses stems from a dual-sided market failure: a profound trust deficit regarding passenger safety and the inconsistent availability of drivers driven by weak voluntary incentives. By shifting away from traditional reactive safety measures, such as manual SOS buttons that require physical interaction from a victim, towards a proactive, automated, and continuous monitoring approach, UniRide establishes a new paradigm for institutional transportation.

The successful development of this project proves the technical and economic feasibility of a context-aware, highly secure carpooling framework tailored specifically for academic environments. The realisation of the project's objectives was heavily anchored in the implementation of a novel Zero-Trust Hybrid Cloud Architecture. Solving the classic cost-latency dilemma of AI applications, heavy computational processes, such as facial recognition via the ArcFace model, were deliberately offloaded from expensive serverless cloud environments (e.g. Firebase Functions or AWS Lambda). Instead, the system established an encrypted Cloudflare Tunnel routing traffic directly to a localised Flask API. This architectural innovation not only eliminated prohibitive cloud computing egress costs but also successfully reduced the biometric verification latency to under one second for hot-start requests. This established a robust "Device Verification Guard" that strictly prevents impersonation and spoofing without compromising the user experience.

In parallel with pre-trip gatekeeping, the project successfully designed and integrated a real-time, AI-driven safety monitoring framework to protect users during the trip. By deploying a transfer-learning YAMNet acoustic classification model directly to the mobile edge via TensorFlow Lite, the application achieved continuous, privacy-preserving distress monitoring. Because the inference occurs locally on the device, the system operates with ultra-low latency (flagging anomalies in

approximately 0.36 seconds) and functions independently of internet dead zones. Crucially, the integration of the Gemini 2.5 Flash/Lite API as a secondary, multi-modal validation layer correlating YAMNet acoustic triggers with kinetic IMU sensor data proved remarkably effective at mitigating false alarms. This edge-to-cloud pipeline ensures that university administrators are autonomously alerted with actionable AI summaries and live GPS coordinates during genuine emergencies in under five seconds, completely bypassing the need for manual user intervention.

To resolve the challenge of inconsistent driver availability, the project successfully introduced the Earn-On-Demand (EOD) economic model alongside a Semester-Based Scheduling system. By allowing drivers to set flexible, university-capped fares (e.g. RM0.00 to RM1.00), the platform financially incentivises driver participation while maintaining the affordability that students require. Coupled with flexible routing algorithms that incorporate pass-by drop points rather than rigid Origin-Destination matching, the system maximises seat utilisation and overall match rates.

As a result, the UniRide platform demonstrates that integrating advanced edge AI, hybrid networking, and dynamic economic models can yield a highly secure, cost-effective, and scalable mobility solution. By heavily reducing the reliance on single-occupancy vehicles (SOVs), the project actively supports the United Nations Sustainable Development Goals, specifically aligning with SDG 11 (Sustainable Cities and Communities), SDG 13 (Climate Action), and SDG 16 (Peace, Justice, and Strong Institutions). The preliminary implementation validates that UniRide is primed not only for deployment within Universiti Tunku Abdul Rahman (UTAR) but possesses the modularity required for nationwide commercialisation across other institutions.

7.2 Recommendations for Future Work

While the current iteration of the UniRide system has successfully met its core objectives, several enhancements can be pursued in future developments to scale the platform and further harden its security framework.

1. **Expansion of the Acoustic Dataset and Generalisation**

   Currently, the YAMNet distress detection model demonstrates high accuracy in identifying anomalies within a controlled environment. However, real-world vehicular acoustics are highly dynamic. Future work should focus on

significantly expanding the custom UniRide training dataset to include a wider variety of background noises, such as heavy rain hitting the car roof, wind noise from rolled-down windows, loud radio playback, and overlapping passenger conversations. Implementing advanced, open-source Adaptive Noise Suppression (ANS) models at the pre-processing stage will further refine the model's ability to isolate subtle distress signals, subsequently lowering the False Positive Rate in highly complex environments.

2. **Transition to Explainable AI (XAI) and Context-Aware Architectures**
   While the current system effectively utilises the Gemini API for secondary contextual validation, the primary edge-filtering model (YAMNet) operates largely as a "black box" classifying audio without detailing its reasoning. For a safety-critical application like UniRide, future iterations should explore the integration of Explainable AI (XAI) frameworks. By utilising techniques such as SHAP (SHapley Additive exPlanations) or gradient-based attention mapping, the edge model could pinpoint the exact acoustic frequencies or temporal frames that triggered an alert. Transitioning to a fully context-aware acoustic model that dynamically adjusts its own sensitivity threshold based on the vehicle's current speed or ambient noise levels would drastically improve distress analysis and ensure unparalleled transparency for campus security administrators.
3. **Hardening of Anti-Spoofing and Liveness Detection Mechanisms**
   While the implemented yaw-based head pose estimation effectively thwarts static image spoofing, integration testing revealed a minor vulnerability to high-quality screen-based attacks (e.g. presenting a video of a moving face). Future iterations should transition to more advanced liveness detection algorithms. Recommendations include integrating Active Flash liveness (which analyses the reflection of randomised screen illumination on a 3D face) or leveraging native hardware Depth-Sensors (Time-of-Flight cameras) to ensure the biometric gatekeeper is completely immune to face spoofing attacks.
4. **Dedicated Edge Server Deployment and IT Infrastructure Integration**
   The current local Flask backend successfully operates as a proof-of-concept for avoiding public cloud GPU latency and costs. For full-scale campus

deployment, this hybrid architecture must be transitioned from a local development machine to a dedicated edge server securely housed within the university's internal IT infrastructure or better cloud. Integrating the system directly into the university's intranet will provide 24/7 uptime, maximise network throughput, and ensure strict compliance with institutional data privacy policies regarding student biometric templates.

5. **Automated Integration with Campus Security Dispatch**

   Currently, the system sends validated emergency alerts to an Admin Dashboard for human review. To drastically reduce emergency response times, the system should be integrated directly with reliable and automated emergency dispatch infrastructure. Future developments should implement API webhooks capable of paging campus patrol vehicles or local authorities immediately, providing them with the live GPS tracking link, the Gemini context summary, and the 15-second audio evidence clip the moment an incident is validated. Realising this integration will require active collaboration with university administration and local law enforcement to ensure the system aligns with official emergency response protocols and data-sharing agreements.

6. **Advanced Graph-Based Routing and Dynamic Pricing**

   To further optimise the availability of carpool listings, the platform's routing capabilities can be upgraded by integrating advanced graph-based pathfinding algorithms, such as Contraction Hierarchies (CH) or GMOMatch. These algorithms would allow the system to calculate complex, many-to-one, and one-to-many carpool clusters dynamically, calculating the most efficient multi-passenger pickup routes in real-time traffic. Furthermore, the Earn-On-Demand (EOD) framework could be upgraded to feature automated dynamic pricing, slightly adjusting the recommended fare caps based on real-time factors like heavy rain or peak campus rush hours, further stimulating driver supply when demand spikes.

7. **Cross-Platform Porting and iOS Optimisation**

   To ensure maximum adoption, the Flutter codebase must be optimised and hardware-tested for the iOS platform. Overcoming OS-level limitations regarding the continuous microphone access required for YAMNet and persistent IMU polling will require implementing iOS-specific native background services to prevent the safety monitoring loop from being terminated by the operating system.

8. **Architectural Decoupling of Client and Administrative Modules**

   For full-scale commercial deployment, student functionalities and administrative tools must be structurally decoupled into two distinct applications (e.g. "UniRide" student app and "UniRide Admin Portal"). From a security perspective, decoupling mitigates risks by drastically reducing the attack surface, ensuring that standard users cannot attempt to reverse-engineer privileged administrative API endpoints.

REFERENCES

[1] J. G. Neoh, M. Chipulu, and A. Marshall, "What encourages people to carpool? An evaluation of factors with meta-analysis," *Transportation*, vol. 44, no. 2, pp. 423–447, Sep. 2017, doi: https://doi.org/10.1007/s11116-015-9661-7.

[2] H. Si, J. Shi, W. Hua, L. Cheng, Jonas De Vos, and W. Li, "What influences people to choose ridesharing? An overview of the literature," *Transport Reviews*, vol. 43, no. 6, pp. 1–26, May 2023, doi: https://doi.org/10.1080/01441647.2023.2208290.

[3] Y. Park, N. Chen, and G. Akar, "Who is Interested in Carpooling and Why: The Importance of Individual Characteristics, Role Preferences and Carpool Markets," *Transportation Research Record: Journal of the Transportation Research Board*, vol. 2672, no. 8, pp. 708–718, Mar. 2018, doi: https://doi.org/10.1177/0361198118756883.

[4] L. Li, H. Zhang, and Z. Gan, "Factors affecting college students' attitudes toward carpooling," *Transportation Safety and Environment*, vol. 6, no. 2, May 2023, doi: https://doi.org/10.1093/tse/tdad025.

[5] B. Chaudhry, A.-U.-H. Yasar, S. El-Amine, and E. Shakshuki, "Passenger Safety in Ride-Sharing Services," *Procedia Computer Science*, vol. 130, no. 1, pp. 1044–1050, Apr. 2018, doi: https://doi.org/10.1016/j.procs.2018.04.146.

[6] R. A. Acheampong, "Societal impacts of smart, digital platform mobility services—an empirical study and policy implications of passenger safety and security in ride-hailing," *Case Studies on Transport Policy*, vol. 9, no. 1, Feb. 2021, doi: https://doi.org/10.1016/j.cstp.2021.01.008.

[7] C. D. A. Icasiano and A. Taeihagh, "Governance of the Risks of Ridesharing in Southeast Asia: an In-Depth Analysis," *Sustainability*, vol. 13, no. 11, p. 6474, Jun. 2021, doi: https://doi.org/10.3390/su13116474.

[8] M. Feeney, "Is Ridesharing Safe?," *Ssrn.com*, Apr. 19, 2019. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=2700891

[9] D. Graziotin, "An Analysis of issues against the adoption of Dynamic Carpooling," *arXiv:1306.0361 [cs]*, pp. 290527 Bytes, 2013, doi: https://doi.org/10.6084/m9.figshare.710636.

[10] Grab Malaysia, "Grab Malaysia Introduces New Safety Innovation - setting standards for preventable incidents," *Grab MY*, Mar. 10, 2023. https://www.grab.com/my/press/others/grab-new-safety-innovation/

[11] J. H. Yee, A. C. Lim, and E. H. L. Chung, "MYSafeZone - Emergency Trigger and Location-Based Response System Using AI Context," presented at the UMPSA Hackathon, Pahang, Malaysia, Nov. 2025. [Online]. Available: https://drive.google.com/file/d/1Sap1H-rL3isf-qyON1OcFetPo9S4-A37/view?usp=sharing

[12] L. Elumalai, D. S, and M. Aadhil, "Campus Carpooling: Optimised Ridesharing for Students Using Hybrid Ridesharing Algorithm HRA," pp. 449–454, Feb. 2025, doi: https://doi.org/10.1109/esic64052.2025.10962610.

[13] L. H. Chang, "Car-pooling application for university students," *UTAR Institutional Repository*, Dec. 29, 2022. http://eprints.utar.edu.my/id/eprint/4767

[14] Grab Malaysia, "Grab partners with Ministry of Transport to implement facial recognition technology in Malaysia | Grab MY," *Grab*, Apr. 11, 2019. [Online]. Available: https://www.grab.com/my/press/social-impact-safety/grab-mot-facial-recognition-technology/

[15] A. J. Oon, "All Grab Passengers Will Have To Take A Selfie To Prove They Are Real From 12 July," *SAYS*, Jun. 04, 2019. [Online]. Available: https://says.com/my/tech/all-grab-passengers-will-have-to-take-a-selfie-for-verification-purposes-by-12-july.

[16] S. Gupta, A. Buriro, and B. Crispo, "DriverAuth: A risk-based multi-modal biometric-based driver authentication scheme for ride-sharing platforms," *Computers & Security*, vol. 83, no. 0167-4048, pp. 122–139, Jun. 2019, doi: https://doi.org/10.1016/j.cose.2019.01.007.

[17] L. Liu, X. Zhang, M. Qiao, and W. Shi, "SafeShareRide: Edge-Based Attack Detection in Ridesharing Services," in *IEEE Xplore*, Oct. 2018, pp. 17–29. doi: https://doi.org/10.1109/SEC.2018.00009.

[18] S. M. Meshkani and B. Farooq, "A generalised ride-matching approach for sustainable shared mobility," *Sustainable Cities and Society*, vol. 76, no. 2210–6707, p. 103383, Jan. 2022, doi: https://doi.org/10.1016/j.scs.2021.103383.

[19] G. S. Shahi, R. S. Batth, and S. Egerton, "A Comparative Study on Efficient Path Finding Algorithms for Route Planning in Smart Vehicular Networks," *International Journal of Computer Networks and Applications*, vol. 7, no. 5, p. 157, Oct. 2020, doi: https://doi.org/10.22247/ijcna/2020/204020.

[20] C. Ignatius, "Grab Launches JustSave, Allows Passengers to Carpool with Other Passengers," *BusinessToday*, May 30, 2023. https://www.businesstoday.com.my/2023/05/30/grab-launches-justsave-allows-passengers-to-carpool-with-other-passengers/

[21] S. Hershey et al., "CNN architectures for large-scale audio classification," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), 2017, pp. 131–135, doi: https://doi.org/10.1109/ICASSP.2017.7952132

[22] Y. Gong, Y. A. Chung, and J. Glass, "AST: Audio Spectrogram Transformer," in Proc. Interspeech, 2021, pp. 571–575, doi: https://doi.org/10.21437/Interspeech.2021-698

[23] A. G. Howard et al., "MobileNets: Efficient convolutional neural networks for mobile vision applications," arXiv preprint arXiv:1704.04861, 2017, doi: https://doi.org/10.48550/arXiv.1704.04861

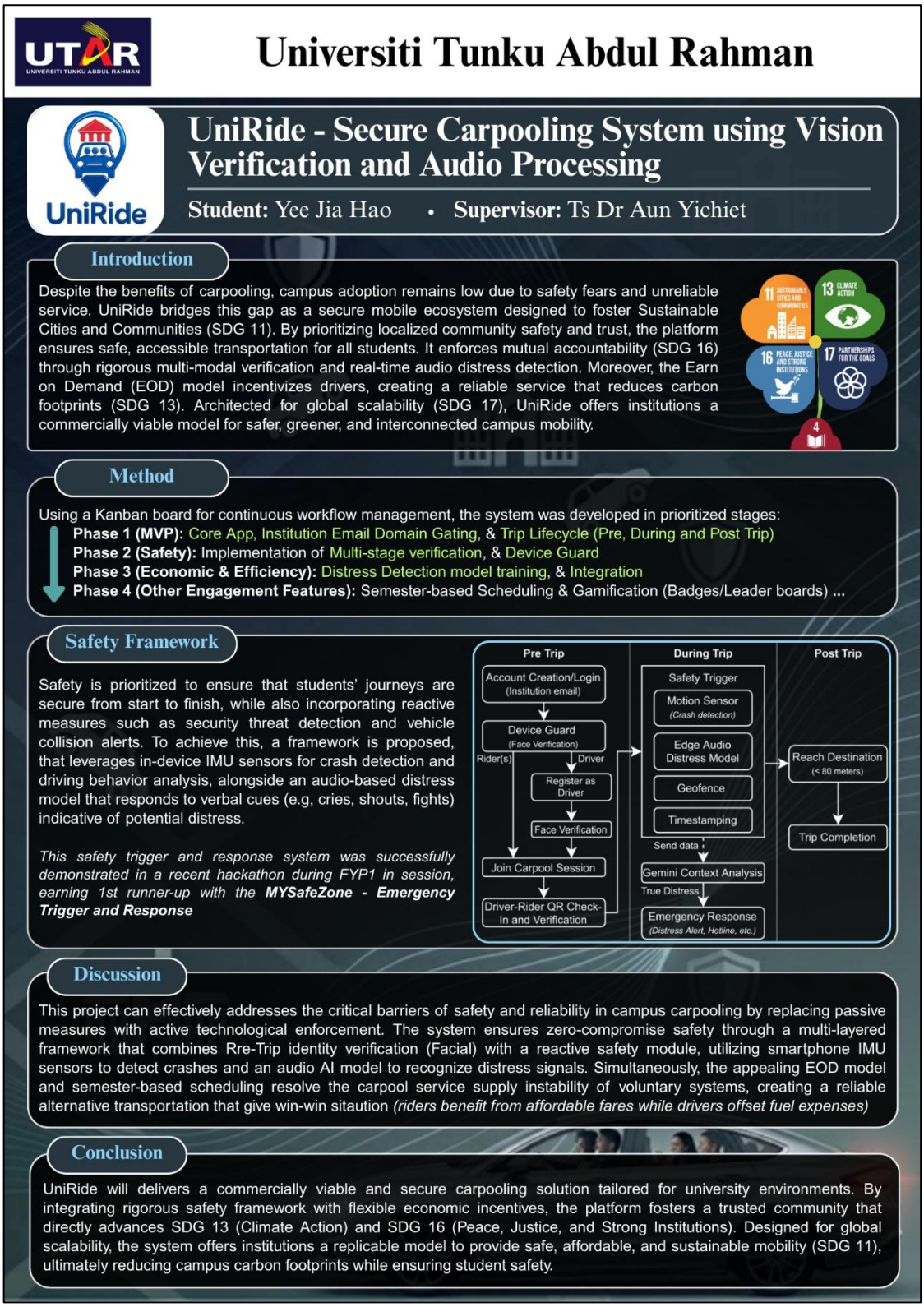
[24] J. F. Gemmeke et al., "Audio Set: An ontology and human-labeled dataset for audio events," in Proc. IEEE Int. Conf. Acoust., Speech, Signal Process. (ICASSP), 2017, pp. 776–780, doi: https://doi.org/10.1109/ICASSP.2017.7952261

[25] Google Research, "YAMNet: A deep net that predicts 521 audio event classes," TensorFlow Hub, 2020. https://tfhub.dev/google/yamnet/1

[26] Krisp, "AI-Powered Noise Cancellation for Enterprise," Krisp Technologies, Inc., 2024. [Online]. Available: https://krisp.ai/

Appendix 1

Poster



Appendix 2

Local Hosted Server and Cloudflared Tunnelling

To support the computationally intensive AI features of UniRide, specifically ArcFace for facial identity verification, a hybrid architecture was implemented. This section details the configuration of the local backend environment and its secure integration with the public internet via a domain, *uniride.cv*.

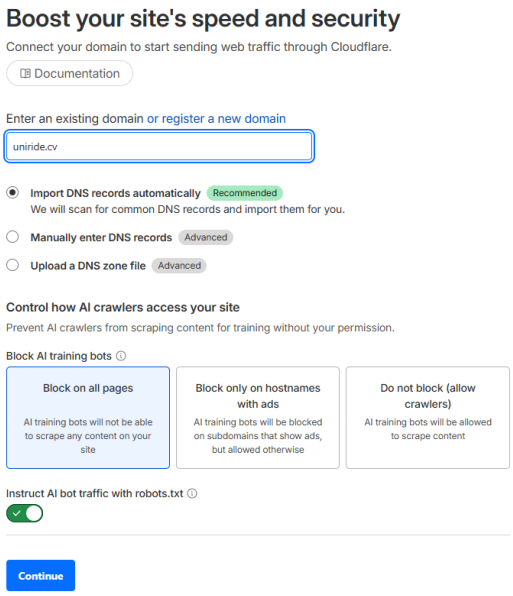
Part 1: Domain Delegation (spaceship.com to Cloudflare)

To leverage Cloudflare's advanced security and Zero Trust tunnelling, the DNS management for the domain purchased from spaceship.com was migrated to Cloudflare's global edge network.

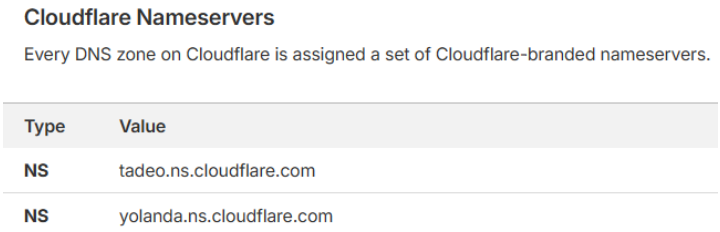
Nameserver Delegation: The default nameservers in the spaceship.com management console was replaced with Cloudflare's specific nameservers. This allows Cloudflare to handle SSL/TLS certificate issuance and DNS record management.

Edge Protection: By delegating the domain, the system benefits from automatic HTTPS encryption, ensuring that all biometric data (face images) sent from the mobile app to the local server is encrypted during transit.

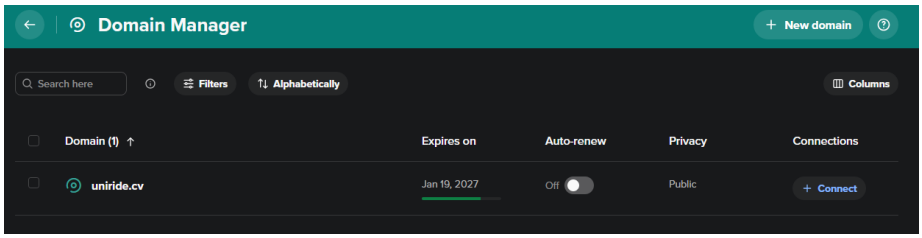
1. Login to *cloudflare.com* > + *Add* > *Connect a domain,*
2. Enter the domain *uniride.cv* and proceed with the default settings as illustrated below,



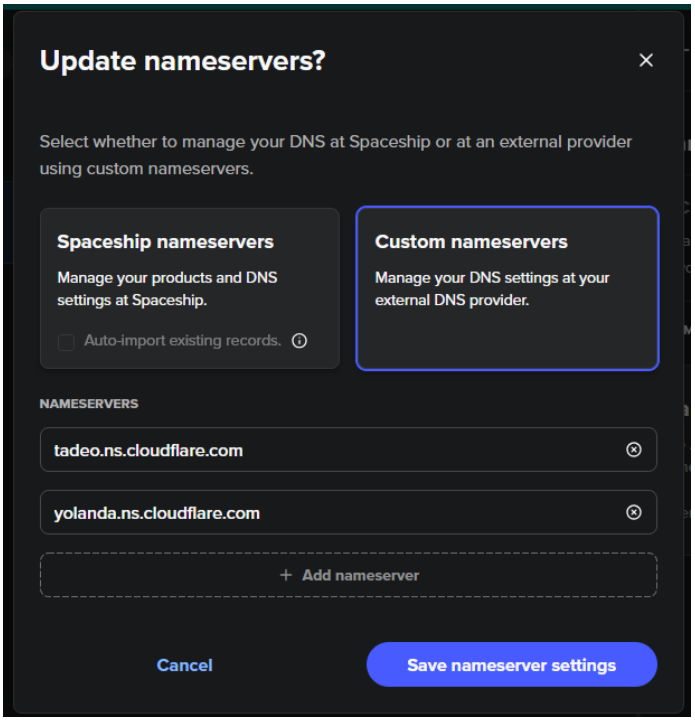
3. Record the two nameservers assigned by Cloudflare,



4. Login to *spaceship.com* > *Library* > *My Account* > Launch *Domain Manager,*



5. Click on the *uniride.cv* domain > Nameservers & DNS
6. Choose *Custom Nameserver* and paste the two-name server,



7. After saving, allow time for DNS propagation. Based on prior experience, this may complete within 2 hours, though up to 24 hours is suggested. Confirm that all DNS records are online before proceeding with Part 3.

Part 2: Install and authenticate *cloudflared*

The *cloudflared* tool is the daemon that creates the bridge between the host computer and the web.

1. Download and install *cloudflared* for hosting Windows, running the following command in PowerShell with Admin,

   winget install --id Cloudflare.cloudflared -e

2. Login *cloudflared* using,

   cloudflared tunnel login

3. A browser will open, login using the Cloudflare account, this will save a *cert.pem* file to "*C:\Users\<USERNAME>\.cloudflared\"*.

Part 3: Tunnel Creation and DNS Routing

Now, create a specific tunnel for the UniRide API.

1. In the PowerShell, run

   cloudflared tunnel create uniride-api

2. Take note and copy the Tunnel ID that appear in the terminal. This will be used to configure the Tunnel.

3. Navigate to *C:\Users\<USERNAME>\.cloudflared*, and create a new file named "*config.yml*" with following content,

```
tunnel: <TUNNEL-ID>
credentials-file: C:\Users\< USERNAME >\.cloudflared\<TUNNEL-ID>.json

ingress:
  - hostname: face.uniride.cv
    service: http://localhost:8080
  - service: http_status:404
```

   - Replace TUNNEL-ID with the ID copied in step 2.
   - *hostname* is the public URL (subdomain) that Cloudflare will expose to.
   - *service* represent the local service endpoint Cloudflared should forward traffic to, typically *http://localhost:8080* or other port containing services endpoint.

- For example, a specific service can be accessed using *https://face.uniride.cv/<specific_service_in_server_endpoint>* with default 8080 port.

4. Running the Local AI Server,

   cloudflared tunnel route dns uniride-api face.uniride.cv

Part 4: Running the Local AI Server

The Flask backend must be active to process the requests coming through the tunnel.

1. Navigate the project directory and to local-server containing the ArcFace hosted. Assume using the following directory, *C:\Users\<USERNAME>\UniRide\local-server*
2. Activate the python environment with the dependencies in *requirements.txt*.
3. Start the server: *python app.py*
4. The server should output that it is running on http://127.0.0.1:8080,

```
  Starting Face Verification Server on 0.0.0.0:8080
  Server will be accessible via Cloudflare Tunnel
  Health check: http://localhost:8080/health
  Using Flask development server (install waitress for production: pip install waitress)
 * Serving Flask app 'app'
 * Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server
 instead.
 * Running on all addresses (0.0.0.0)
 * Running on http://127.0.0.1:8080
 * Running on http://192.168.1.80:8080
Press CTRL+C to quit
```

Part 5: Starting the Secure Bridge

1. In PowerShell, execute the following command to initiate the Cloudflared tunnel and begin routing traffic:

   cloudflared tunnel run uniride-api

2. In a browser, open *"https://localhost:8080/health"*. If the page displays the output "*status: healthy*", this confirms that the server is successfully responding locally.

Part 6: Updating the Flutter Frontend

1. In *C:\Users\<USERNAME>\UniRide\.env* change the BACKEND_URL to "BACKEND_URL=https://face.uniride.cv"
2. In a browser, open *"https://face.uniride.cv/health"*. If the page displays the output "*status: healthy*", this confirms that the server is successfully responding through the Cloudflare Tunnel.

Appendix 3

Demonstration of Dataset Expansion

To illustrate this expansion, consider a standard 5-second ambient noise file from the ESC-50 dataset.

- Total samples, $S_{total} = 5 * 16000 = 80000$ samples
- Applying the formula:

$$N_{full} = \left\lfloor \frac{80000 - 15600}{7800} \right\rfloor + 1 = \lfloor 8.25 \rfloor + 1 = 9 \text{ chunks}$$

Therefore, a single **5 second** audio file yields **9 distinct training samples or chunks**. When this mathematical transformation is applied to longer custom datasets (e.g. 30-second background noise recordings yielding over 50 chunks each) and multiplied by the dynamic augmentation tasks. This explains the dataset's massive expansion to over 300,000 training chunks.

Appendix 4

Origin A (UTAR West Gate) Deviation Data

Primary Destination	Deviated Drop-Off Node	Extra Distance (m)	Status (Threshold < 500m)
Block A	Block B, C	524.00	Invalid
	Block C, D, E, F	1079.00	Invalid
	Block E, G, H, I	1540.00	Invalid
	Block K, L	1738.00	Invalid
	Block M	1001.00	Invalid
	Block N, P	476.00	Valid
Block B, C	Block A	-108.00	Valid
	Block C, D, E, F	651.00	Invalid
	Block E, G, H, I	1112.00	Invalid
	Block K, L	1112.00	Invalid
	Block M	981.00	Invalid
	Block N, P	456.00	Valid
Block C, D, E, F	Block A	431.00	Valid
	Block B, C	431.00	Valid
	Block E, G, H, I	535.00	Invalid
	Block K, L	535.00	Invalid
	Block M	535.00	Invalid
	Block N, P	685.00	Invalid
Block E, G, H, I	Block A	547.00	Invalid
	Block B, C	547.00	Invalid
	Block C, D, E, F	547.00	Invalid
	Block K, L	66.00	Valid
	Block M	66.00	Valid
	Block N, P	67.00	Valid
Block H, I	Block A	791.00	Invalid
	Block B, C	791.00	Invalid
	Block C, D, E, F	791.00	Invalid
	Block E, G, H, I	2.00	Valid
	Block K, L	66.00	Valid
	Block M	66.00	Valid
	Block N, P	67.00	Valid
Block K, L	Block A	1942.00	Invalid

	Block B, C	1888.00	Invalid
	Block C, D, E, F	1888.00	Invalid
	Block E, G, H, I	1341.00	Invalid
	Block M	136.00	Valid
	Block N, P	137.00	Valid
Block M	Block A	1938.00	Invalid
	Block B, C	2462.00	Invalid
	Block C, D, E, F	2820.00	Invalid
	Block E, G, H, I	2273.00	Invalid
	Block K, L	969.00	Invalid
	Block N, P	-14.00	Valid
Block N, P	Block A	1951.00	Invalid
	Block B, C	2475.00	Invalid
	Block C, D, E, F	3012.00	Invalid
	Block E, G, H, I	2715.00	Invalid
	Block K, L	1411.00	Invalid
	Block M	413.00	Valid

Appendix 5

Origin B (UTAR East Gate) Deviation Data

Primary Destination	Deviated Drop-Off Node	Extra Distance (m)	Status (Threshold < 500m)
Block A	Block B, C	94.00	Valid
	Block C, D, E, F	180.00	Valid
	Block E, G, H, I	211.00	Valid
	Block K, L	1553.00	Invalid
	Block M	1553.00	Invalid
	Block N, P	1553.00	Invalid
Block B, C	Block A	416.00	Valid
	Block C, D, E, F	114.00	Valid
	Block E, G, H, I	115.00	Valid
	Block K, L	1451.00	Invalid
	Block M	2057.00	Invalid
	Block N, P	2057.00	Invalid
Block C, D, E, F	Block A	1492.00	Invalid
	Block B, C	968.00	Invalid
	Block E, G, H, I	75.00	Valid
	Block K, L	1411.00	Invalid
	Block M	2148.00	Invalid
	Block N, P	2823.00	Invalid
Block E, G, H, I	Block A	2068.00	Invalid
	Block B, C	1544.00	Invalid
	Block C, D, E, F	1007.00	Invalid
	Block K, L	1336.00	Invalid
	Block M	2073.00	Invalid
	Block N, P	2599.00	Invalid
Block H, I	Block A	2222.00	Invalid
	Block B, C	1698.00	Invalid
	Block C, D, E, F	1161.00	Invalid
	Block E, G, H, I	2.00	Valid
	Block K, L	1312.00	Invalid
	Block M	2049.00	Invalid
	Block N, P	2575.00	Invalid
Block K, L	Block A	2127.00	Invalid

	Block B, C	1549.00	Invalid
	Block C, D, E, F	1012.00	Invalid
	Block E, G, H, I	5.00	Valid
	Block M	873.00	Invalid
	Block N, P	1399.00	Invalid
Block M	Block A	1386.00	Invalid
	Block B, C	1386.00	Invalid
	Block C, D, E, F	1207.00	Invalid
	Block E, G, H, I	200.00	Valid
	Block K, L	232.00	Valid
	Block N, P	511.00	Invalid
Block N, P	Block A	874.00	Invalid
	Block B, C	874.00	Invalid
	Block C, D, E, F	874.00	Invalid
	Block E, G, H, I	117.00	Valid
	Block K, L	149.00	Valid
	Block M	-112.00	Valid