ECALL Emergency Response System Project

Introduction and Goals

1. Requirement overview

Functional Requirements

1. Accident Detection

• ID: ECALL-01

• Description: The system must detect accidents based on sudden deceleration (e.g., rapid speed drop to zero).

2. Emergency Call Initiation

• ID: ECALL-02

• Description: Once an accident is detected, the system must automatically initiate an emergency call.

1. Vehicle Data Transmission

• ID: ECALL-03

• Description: The system must send vehicle-related data (speed, location, acceleration, time of accident) to the emergency response center.

2. Notification Handling

• ID: ECALL-04

• Description: The system must support multiple notification methods (e.g., SMS, email) to contact emergency services.

3. Vehicle State Management

• ID: ECALL-05

• Description: The vehicle class must encapsulate state information such as speed, location, and acceleration with private attributes and public getters/setters.

4. Vehicle Type Differentiation

• ID: ECALL-06

• Description: The system must support different vehicle types (e.g., Car, Truck), each potentially having unique accident detection parameters.

5. Error Handling

• ID: ECALL-07

• Description: The system must handle invalid input data (e.g., negative speed values) and simulate network failures when sending notifications.

6. Accident Data Logging

• ID: ECALL-08

• Description: Each accident event must be logged, including vehicle ID, speed, location, acceleration, and timestamp in a structured format (JSON/CSV).

7. Historical Data Analysis

• ID: ECALL-09

• Description: The system must allow reloading past accident data for analysis and review.

10. User Input via Terminal

• ID: ECALL-10

• Description: The system must provide a terminal-based interface for manually entering vehicle speed, location, and acceleration to simulate accident scenarios.

11. Automated Simulation Mode (Optional)

• ID: ECALL-11

• Description: The system should allow users to input a sequence of values for speed and acceleration over time to simulate driving conditions.

12. Networking Support (Optional)

• ID: ECALL-12

• Description: The system should support basic networking features to send accident data to a

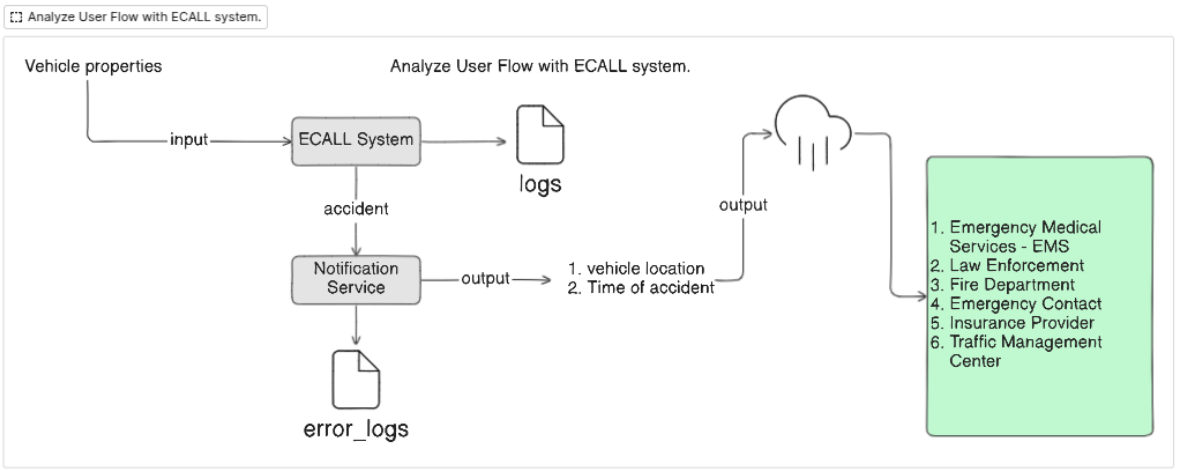
remote server for further processing.

13. Observer Pattern for Notifications (Optional)

• ID: ECALL-13

• Description: The system should use the Observer design pattern to notify components whenever an accident occurs.

Context and Scope



User Flow Diagram for the **ECALL Emergency Response System**, illustrating how accident detection and emergency notifications function within the system.

1. Diagram Breakdown:

1. **Input (Vehicle Properties)**:
    a. The system receives input from the vehicle, such as **speed, location, and acceleration**.
2. **ECALL System**:
    a. This component processes vehicle data.
    b. If an accident is detected (e.g., sudden deceleration), it triggers an emergency response.
    c. Logs normal system activities.

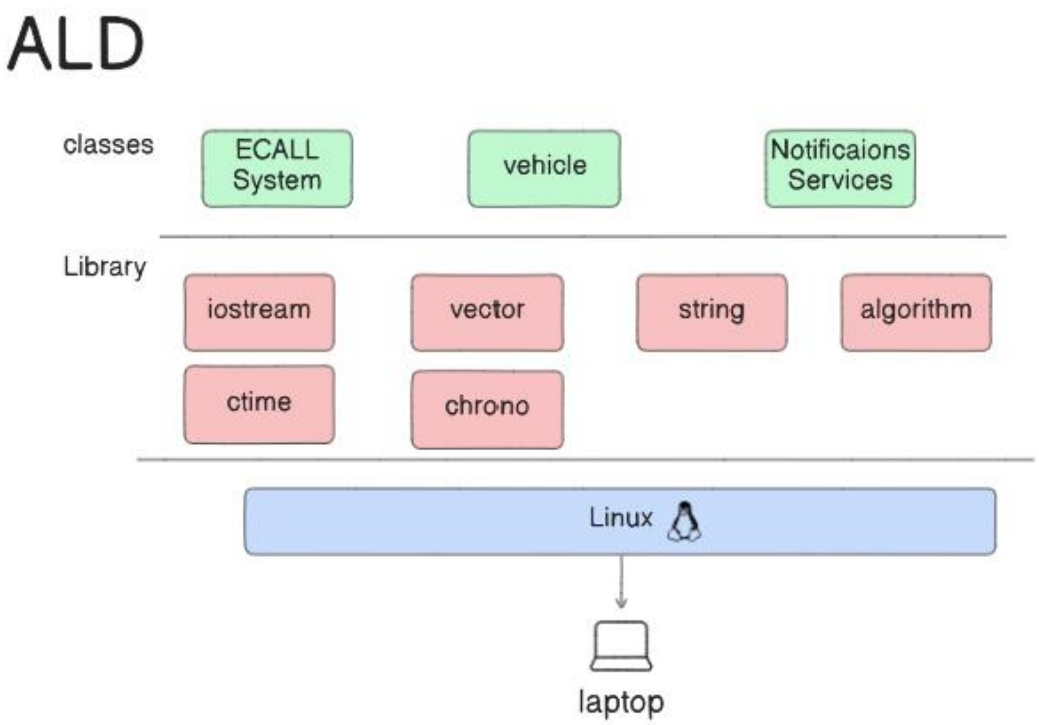
3. **Notification Service**:
    a. If an accident occurs, the ECALL System sends the accident details to the Notification Service.
    b. The Notification Service generates error logs if issues arise during the process.
4. **Data Sent to Cloud**:
    a. The Notification Service sends critical data to emergency responders via a cloud system.
    b. The transmitted data includes:
        i. **Vehicle location**
        ii. **Time of accident**
5. **Emergency Response Recipients** (Green Box):
    a. The cloud system forwards emergency data to multiple entities:
        i. **Emergency Medical Services (EMS)**
        ii. **Law Enforcement**
        iii. **Fire Department**
        iv. **Emergency Contacts** (e.g., family members)
        v. **Insurance Provider**
        vi. **Traffic Management Center**

2. Relation to Project Requirements:

- **Accident Detection**: The ECALL system detects crashes based on sudden deceleration.
- **Data Logging**: Logs normal activity and errors.
- **Notification System**: Sends alerts with vehicle location and accident time.
- **Emergency Services Contact**: Cloud-based distribution to relevant authorities.

Building block view

1.ALF (Architecture Layered Diagram)



1. Classes (Green Boxes)

These are the main objects in the system:

- **ECALL System**
  - Detects accidents and triggers emergency response actions.
  - Collects vehicle data (e.g., speed, location, time of impact).
- **Vehicle**
  - Represents the state of the vehicle, including its speed, position, and emergency status.
- **Notification Services**
  - Manages sending alerts to emergency contacts, responders, and cloud services.

2. Library (Red Boxes)

These are standard **C++ libraries** chosen for their specific functionalities:

- **#include <iostream>**
  - Manages input/output operations for logging, debugging, and displaying messages.
- **#include <vector>**
  - Provides dynamic array storage for handling lists of emergency contacts, logged events, or past alerts.
- **#include <string>**
  - Handles text-based data, such as vehicle status messages and notifications.
- **#include <algorithm>**
  - Offers sorting and searching functions for processing vehicle and alert data efficiently.
- **#include <ctime>**
  - Provides functions for handling **timestamps**, allowing the system to log when an accident occurs.
- **#include <chrono>**
  - Enables **high-precision timing** to measure delays, event durations, or system response times.

3. Linux (Blue Box)

- The **Linux operating system** is the execution environment for the ECALL system.
- It provides:
  - **Process management** to handle multiple system operations.
  - **Networking capabilities** for sending notifications over the internet.
  - **File system support** for logging accident records.

4. Laptop (Bottom)

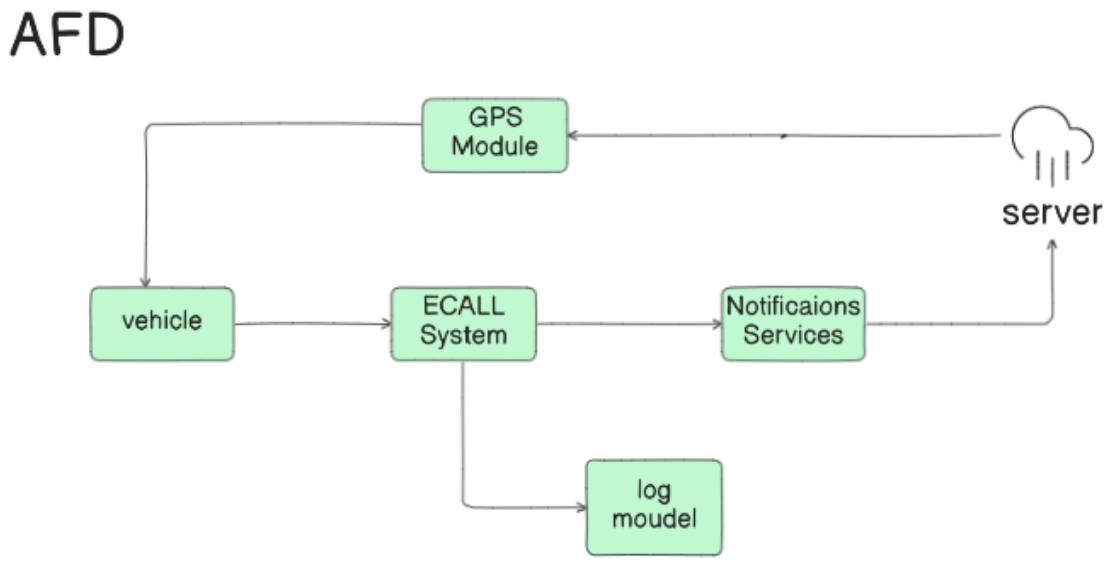
- Represents the **hardware platform** running the ECALL system.
- The **Linux-based environment** enables testing and development.

Why Were ctime and chrono Added?

- **ctime** helps with **date/time tracking**, ensuring that accident logs and notifications have accurate timestamps.

- **chrono** provides **precise timing** for performance measurement and tracking system delays.

2.AFD (Architecture Flow Diagram)



1. Components and Their Roles

- **Vehicle**
  - Represents the car's real-time status, including motion, speed, and accident detection.
  - Feeds data into the **ECALL System** and **GPS Module**.
- **GPS Module**
  - Collects **location data** from the vehicle.
  - Sends this data to both the **ECALL System** and the **server**.
- **ECALL System**
  - Acts as the **core unit** that processes accident detection.
  - Takes inputs from the **vehicle** and **GPS Module**.
  - Sends alerts to both the **Notification Services** and the **Log Module**.
- **Notification Services**
  - Handles sending emergency alerts.
  - Notifies relevant contacts, emergency responders, or cloud services via the **server**.

- **Log Module**
  - Stores accident-related data (e.g., timestamps, location, vehicle status).
  - Helps in system monitoring, debugging, and post-accident analysis.
- **Server**
  - Acts as the **cloud backend** for external processing and storage.
  - Receives data from both the **GPS Module** and **Notification Services**.
  - Can be used for further analysis, remote monitoring, and emergency response coordination.

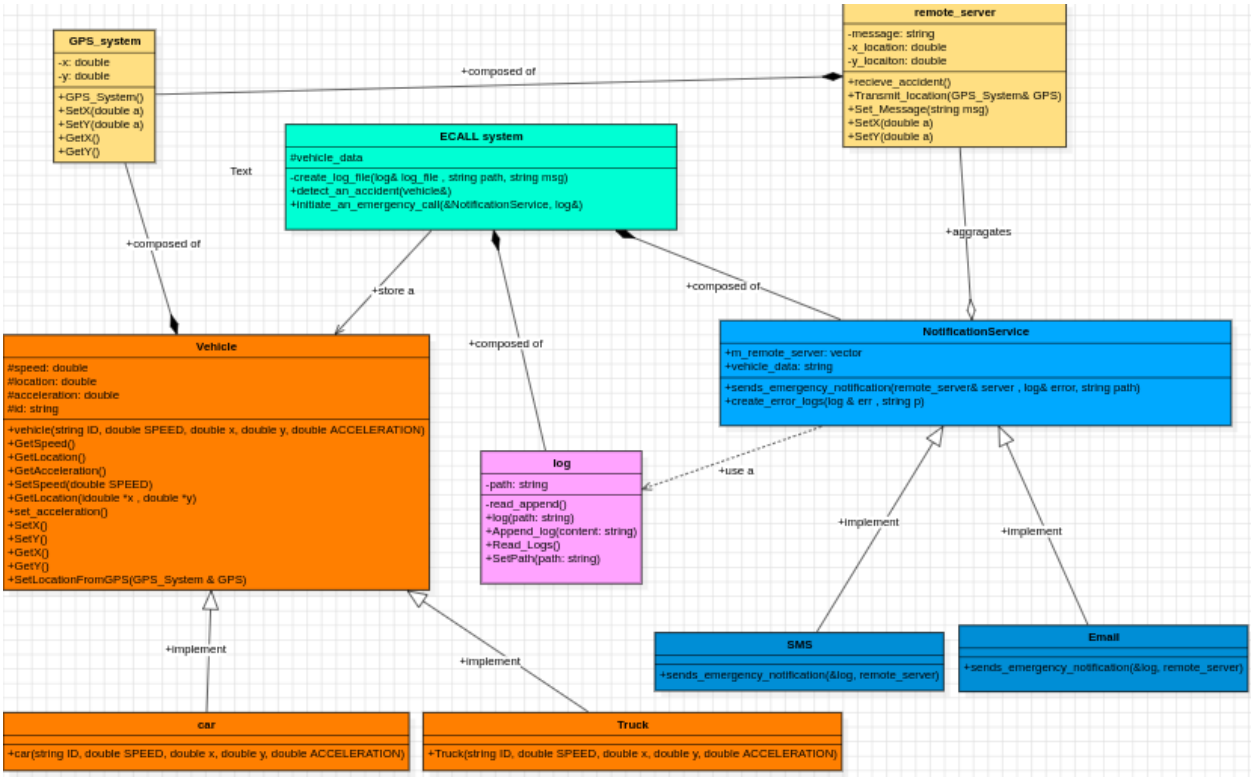
2. Interaction Between Modules

1. The **vehicle** provides data to the **ECALL System** and **GPS Module**.
2. The **GPS Module** sends location data to the **server**.
3. The **ECALL System** processes accident detection and sends:
   a. Alerts to **Notification Services** → which then communicates with the **server**.
   b. Logs to the **Log Module** for storage.

Key Takeaways

- The **ECALL System** is the **main processing unit**, integrating vehicle status, GPS data, and notifications.
- The **server** plays a crucial role in **cloud storage and remote alerting**.
- The **Log Module** ensures that accident history and system events are recorded for analysis.

3.UML (Class Diagram Description)



1. Key Classes and Their Roles

📌 ECALL System (Main Processing Unit)

- Responsible for handling vehicle emergency calls.
- Composed of:
  - **Vehicle Data**
  - **Notification Service**
  - **Log System**
- **Key Functions:**
  - create_log(): Stores logs of incidents.
  - detect_an_accident(): Determines if an accident has occurred.
  - Initiate_an_emergency_call(): Triggers an emergency notification.

📌 Vehicle (Data Source)

- Represents a **car or truck** with attributes:
  - **Speed**
  - **Location**

    - **Acceleration**
    - **Vehicle ID**
- **Key Functions:**
    - GetLocationFromGPS(): Fetches location data from the **GPS system**.
    - UpdateSpeed(), SetX(), SetY(): Updates movement parameters.

🚗 Car & Truck (Subclasses)

- Inherit from the **Vehicle** class but may have specific implementations.

📌 GPS System (Location Provider)

- Tracks the vehicle's position.
- **Key Attributes:**
    - x (longitude)
    - y (latitude)
- **Key Functions:**
    - SetX(), SetY(): Updates coordinates.
    - GetX(), GetY(): Retrieves current location.

📌 Log System (Data Storage)

- Stores accident records.
- **Key Attributes:**
    - path: File path for logs.
- **Key Functions:**
    - append(logContent): Adds a new log entry.
    - Read_Logs(): Retrieves previous logs.

📌 Remote Server (Cloud Integration)

- Collects and processes emergency data.
- **Key Attributes:**
    - message: Stores emergency messages.
    - x, y: Coordinates of the incident.
- **Key Functions:**
    - create_accident(): Logs an accident event.
    - Transmit_location(): Sends GPS location to the server.

📌 Notification Service (Emergency Alerts)

- Aggregates emergency notifications.
- **Key Functions:**
    - sends_emergency_notification(): Sends alerts using different channels (SMS, Email).

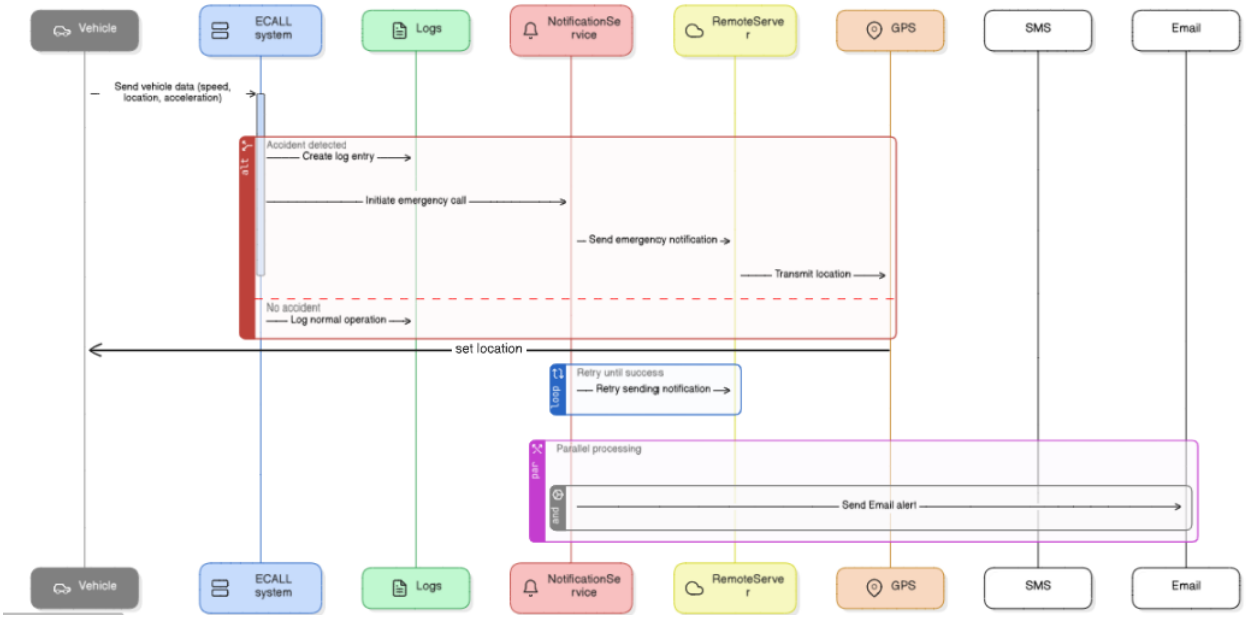
📨 Email & 📱 SMS (Notification Methods)

- Both implement sends_emergency_notification().
- Provide different ways to notify emergency responders.

2. UML Class Relationships

- **Composition (♦)**
    - ECALL system is composed of **Vehicle Data**, **Notification Service**, and **Log System**.
    - Vehicle uses **GPS System** for location tracking.
- **Aggregation (◇)**
    - Notification Service aggregates multiple **Remote Server** objects.
- **Inheritance (→)**
    - Car and Truck inherit from Vehicle

Run time view



This **sequence diagram** illustrates the runtime interactions between various components of the **ECALL emergency response system** when an accident occurs.

1. Scenario Overview

- A **vehicle** sends real-time data (**speed, location, acceleration**) to the **ECALL system**.
- The **ECALL system** determines whether an accident has occurred.
- If an accident is detected:
  - A **log entry** is created.
  - An **emergency call** is initiated.
  - A **notification service** sends an alert.
  - A **remote server** transmits the vehicle's location.
  - Alerts are sent via **SMS and Email**.
- If **no accident is detected**, normal logging continues.

2. Key Interactions in the Sequence Diagram

🚗 Vehicle → ECALL System

- The **vehicle** continuously transmits speed, location, and acceleration data.

🟦 ECALL System

- Analyzes incoming data.
- If an **accident is detected**:
    - Logs the incident.
    - Initiates an emergency call to the notification service.
- If **no accident is detected**, normal operations continue.

🟩 Logs

- Stores normal operation logs or emergency event logs.

🔔 Notification Service

- Sends **emergency notifications**.
- If sending fails, it **retries until success**.

☁️ Remote Server

- Receives and processes emergency notifications.
- **Transmits location** of the vehicle to emergency responders.

📍 GPS System

- Provides real-time **location data**.
- Updates coordinates when necessary.

📧 Email & 📱 SMS Services

- **Parallel Processing** ensures notifications are sent via **Email and SMS**.

3. Special Cases & Enhancements

- **Retry Mechanism**: If an emergency notification fails, the system keeps retrying until successful.
- **Parallel Processing**: Ensures both **Email and SMS alerts** are sent efficiently.

Deployment View

1. Board Specifications (Resources)

The **ECALL Emergency Response System** requires a hardware platform capable of real-time accident detection and emergency communication. The selected board must meet the following specifications:

- **CPU**: ARM Cortex-M series (e.g., Cortex-M4 or Cortex-M7) for real-time processing
- **Minimum Frequency Required by Software**: 80 MHz
- **Maximum Frequency Supported**: 500 MHz
- **Peripherals**:
    - **GPS Module**: To provide real-time vehicle location data
    - **Accelerometer & Gyroscope**: For accident detection based on sudden deceleration
    - **Cellular/Wi-Fi Module**: For emergency notification transmission
    - **CAN Bus**: To interface with the vehicle's internal systems
    - **GPIOs**: For external sensor integration
- **Operating Mode**: 32-bit architecture for optimized embedded processing

For the **ECALL Emergency Response System**, the chosen hardware platform is the **Raspberry Pi 4 Model B (8GB RAM)**. This board was selected due to its powerful processing capabilities and extensive connectivity options. Below are the key specifications:

- **CPU**: Quad-core Cortex-A72 (ARM v8, 64-bit)
- **Minimum Frequency Required by Software**: 600 MHz
- **Maximum Frequency Supported**: 1.5 GHz
- **Peripherals**:
    - **GPS Module**: To provide real-time vehicle location data
    - **Accelerometer & Gyroscope**: For accident detection based on sudden deceleration
    - **Cellular Module (4G/5G or LoRa)**: For emergency notification transmission
    - **CAN Bus Interface (via External Adapter)**: To communicate with vehicle systems
    - **GPIOs & I2C/SPI/UART**: For sensor and module integration
    - **HDMI & USB Ports**: Useful for debugging and system updates
- **Operating Mode**: 64-bit architecture for optimized performance

2.Justification for Selecting Raspberry Pi 4 Model B (8GB RAM)

The **Raspberry Pi 4 Model B** was chosen for the following reasons:

✅ **High Processing Power**: The quad-core Cortex-A72 provides ample computational capability for accident detection algorithms.

✅ **64-bit Architecture**: Ensures compatibility with modern software frameworks and high-speed data processing.

✅ **Connectivity & Expansion**: Supports Wi-Fi, Bluetooth, USB, and multiple communication protocols for sensor integration.

✅ **Multi-Threaded Processing**: Can handle concurrent operations such as GPS tracking, sensor data processing, and network communication.

✅ **Large RAM (8GB)**: Enables smooth execution of AI/ML models for advanced accident detection, if needed.

✅ **Flexible Power Supply**: Can operate via USB Type-C power or be adapted for automotive battery integration.

✅ **Strong Community & Support**: Large developer ecosystem ensures availability of libraries, drivers, and troubleshooting resources.

This configuration ensures that the **ECALL System** is powerful, reliable, and capable of real-time accident detection and emergency message transmission.