### *Raptee\_based Statement :<Open Innovation>* **Predictive Maintenance System for EVs Using AI**

A predictive maintenance system leverages AI and IoT to analyze sensor data, predict failures, and ensure proactive maintenance. Below are the key requirements and flowmap for designing and implementing this system.

### **Hardware Requirements**

**IoT Sensors**

Temperature sensors (battery, motor, brakes)

Vibration sensors (suspension, wheels)

Voltage and current sensors (battery health)

Tire pressure sensors

GPS for location tracking

OBD-II (On-Board Diagnostics) adapter for vehicle data

**Embedded System**

Microcontroller (ESP32, Arduino, or Raspberry Pi) for real-time data collection

Edge AI device for on-board analysis

**Communication Module**

4G/5G or Wi-Fi module for cloud connectivity

### 

### 

### 

### **Software Requirements**

**Data Collection Layer**

Real-time data logging from EV sensors

Data preprocessing for noise reduction

**AI Model for Prediction**

**ML Algorithms:** Random Forest, XGBoost, or LSTM (for time-series data)

**DL Algorithms:** CNN + RNN (for complex pattern recognition)

**Data Storage & Management**

Cloud platform like AWS IoT, Azure IoT Hub, or Firebase

Database system for structured data (e.g., MongoDB, MySQL)

**Notification System**

Mobile app for user alerts

SMS or Email integration for instant alerts

**Dashboard Interface**

Visual representation of system health, battery status, motor condition, etc.

Maintenance reminders and alerts

### 

### 

### 

### 

### **AI Model Training Requirements**

**Dataset Sources:**

EV telematics data

Historical breakdown records

Manufacturer's failure pattern data

**Feature Engineering:**

Temperature trends, voltage fluctuations, vibration patterns

**Training Tools:**

TensorFlow, PyTorch, or Scikit-learn

**Deployment:**

For real-time edge prediction, consider TensorFlow Lite or OpenVINO

## **Flowmap for Design & Implementation**

### **Step 1: Requirement Analysis**

Identify key EV components that require predictive maintenance (battery, motor, brakes, etc.).

Define sensor data types, sampling rates, and critical failure thresholds.

### **Step 2: Hardware Integration**

Install IoT sensors in the EV for continuous data monitoring.

Use an OBD-II adapter for accessing vehicle diagnostics.

Connect sensors to a microcontroller for data transmission.

### **Step 3: Data Acquisition & Preprocessing**

Collect real-time sensor data (e.g., temperature, voltage, vibrations).

Clean the data (handle missing values, noise reduction).

Normalize data to improve model accuracy.

### **Step 4: Model Development**

Train an AI model using historical data:

**Features:** Temperature spikes, voltage drops, vibration intensity.

**Labels:** Predict failures (e.g., battery degradation, motor overheating).

Implement anomaly detection for unexpected patterns.

### **Step 5: System Deployment**

Deploy the trained model on:

**Edge AI Device** for real-time prediction.

**Cloud Platform** for periodic updates and remote monitoring.

### **Step 6: Notification System**

Develop a mobile app or web dashboard to:

Alert users about upcoming maintenance needs.

Provide actionable insights (e.g., "Battery overheating detected, reduce load").

### **Step 7: Testing & Validation**

Simulate real-world conditions with test vehicles.

Evaluate model accuracy, false positives, and latency.

Fine-tune the model based on real-time performance.

### **Step 8: Maintenance & Improvement**

Continuously update the model using new sensor data.

Add adaptive learning to improve prediction accuracy over time.

## **Tech Stack Recommendations**

**IoT Framework:** AWS IoT, Azure IoT, or Google Cloud IoT

**AI Framework:** TensorFlow, PyTorch, or Scikit-learn

**Database:** Firebase, MongoDB, or InfluxDB (ideal for time-series data)

**Dashboard:** Flutter (for mobile apps) or React.js (for web UI)

## **Sample Use Case**

**Scenario:** Detecting motor overheating in an EV.

Sensor data shows rising motor temperature.AI model predicts that prolonged driving may cause a failure within 20 minutes.The system alerts the driver: **"Motor temperature high! Reduce speed to avoid damage."**

## 

## 

## 

## 

## 

## **Predictive Maintenance System for EVs - Architecture & Implementation**

### **1. System Architecture Diagram**

**[ EV Sensors ] --> [ Microcontroller (ESP32/Raspberry Pi) ] --> [ Cloud Platform (AWS IoT/Firebase) ]**

**| |**

**| |**

**--> [ Edge AI Model ] --> [ AI Model for Prediction ]**

**| |**

**--> [ Notification System (App/Email/SMS) ]**

### 

### EV Sensors: Collect data such as battery temperature, voltage, and vibration.

### Microcontroller (ESP32/Raspberry Pi): Acts as a mini-computer that receives sensor data, processes it locally, and sends it to the cloud for deeper analysis.

### Cloud Platform (AWS IoT/Firebase): Stores collected data securely and hosts the primary AI model for accurate predictions.

### Edge AI Model: Runs directly on the microcontroller to provide quick alerts for immediate risks — helpful in areas with poor internet connectivity.

### AI Model for Prediction: The cloud-hosted model continuously improves by learning from more data to provide better predictions.

### Notification System: Sends alerts to users via an app, email, or SMS when potential maintenance issues are detected.

### **2. Sample Code for Data Preprocessing**

import pandas as pd

import numpy as np

from sklearn.preprocessing import StandardScaler

from scipy.signal import savgol\_filter

# Load sample sensor data

df = pd.read\_csv('sensor\_data.csv')

# Handle missing values

df.fillna(method='ffill', inplace=True)

# Noise reduction using Savitzky-Golay filter

df['temperature'] = savgol\_filter(df['temperature'], window\_length=11, polyorder=2)

df['voltage'] = savgol\_filter(df['voltage'], window\_length=11, polyorder=2)

# Feature scaling

scaler = StandardScaler()

df[['temperature', 'voltage', 'vibration']] = scaler.fit\_transform(df[['temperature', 'voltage', 'vibration']])

# Save the cleaned data

df.to\_csv('processed\_sensor\_data.csv', index=False)

### 

### **3. AI Model for Predictive Maintenance (LSTM-based)**

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras import layers

# Load processed data

df = pd.read\_csv('processed\_sensor\_data.csv')

# Create sequences for LSTM model

def create\_sequences(data, seq\_length):

sequences, labels = [], []

for i in range(len(data) - seq\_length):

seq = data.iloc[i:i + seq\_length].values

label = data.iloc[i + seq\_length]['failure\_risk'] # Binary target: 0 = Safe, 1 = Risk

sequences.append(seq)

labels.append(label)

return np.array(sequences), np.array(labels)

seq\_length = 30 # 30 data points as a sequence

X, y = create\_sequences(df, seq\_length)

# LSTM Model Definition

model = keras.models.Sequential([

layers.LSTM(64, return\_sequences=True, input\_shape=(X.shape[1], X.shape[2])),

layers.LSTM(32),

layers.Dense(32, activation='relu'),

layers.Dense(1, activation='sigmoid')

])

# Compile Model

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

# Train Model

model.fit(X, y, epochs=20, batch\_size=32, validation\_split=0.2)

# Save the model

model.save('predictive\_maintenance\_model.h5')

### **4. Deployment on Edge Device (ESP32 / Raspberry Pi)**

# Convert to TFLite formatConvert the model to TensorFlow Lite:

**#code:**

tflite\_convert --saved\_model\_dir=predictive\_maintenance\_model --output\_file=maintenance\_model.tflite

**On the microcontroller: (Hardware Input)**

import tflite\_runtime.interpreter as tflite

import numpy as np

interpreter = tflite.Interpreter(model\_path='maintenance\_model.tflite')

interpreter.allocate\_tensors()

input\_details = interpreter.get\_input\_details()

output\_details = interpreter.get\_output\_details()

def predict\_failure(data):

interpreter.set\_tensor(input\_details[0]['index'], np.array([data], dtype=np.float32))

interpreter.invoke()

result = interpreter.get\_tensor(output\_details[0]['index'])[0]

return "Failure Risk" if result > 0.5 else "Safe"

### **Notification System**

import smtplib

from email.message import EmailMessage

def send\_alert(message):

email = EmailMessage()

email['Subject'] = 'EV Maintenance Alert'

email['From'] = 'your\_email@example.com'

email['To'] = 'user\_email@example.com'

email.set\_content(message)

with smtplib.SMTP('smtp.gmail.com', 587) as server:

server.starttls()

server.login('your\_email@example.com', 'your\_password')

server.send\_message(email)

**# Conclude**

### **Final Workflow Summary**

**Sensors collect EV data (Temperature, Voltage, Vibration)  
 Data preprocessed for noise reduction and feature scaling  
 AI Model predicts failure risks using real-time data  
 Edge AI predicts risks locally for immediate response  
 Cloud AI Model updates with newer data for improved accuracy  
 User receives alerts for timely maintenance actions**

**END….**