SMART DAMAGE/LEAKAGE DETECTION SYSTEM

1.Aim

Oil and gas pipelines are critical but essential structures. A damage/leakage in such pipeline may lead to accidents.

2.Motivation

A. Any damage or leakage in pipelines can lead to dangerous accidents, environmental hazards, and financial losses.

B. Manual inspection and traditional monitoring methods are often slow, costly, and inefficient.

C. Early detection and timely alerts can prevent large-scale disasters and ensure safety.

3.Dataset

The dataset consists of simulated and/or real sensor data representing oil and gas pipeline conditions. Leakage and damage events are labeled in the dataset to train and test the detection model. The dataset is used to train a machine learning model capable of classifying normal vs. leakage/damage conditions.The model is then converted into TFLite format for deployment on embedded devices for real-time monitoring.

4.Exploratory Data Analysis (EDA) – Code

```
import pandas as pd

import numpy as np

import tensorflow as tf

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import LabelEncoder, StandardScaler


# Load the dataset

df = pd.read_csv("Gas_Sensors_Measurements.csv")


# Drop Serial Number as it's not a feature
```

```
df.drop(columns=["Serial Number"], inplace=True)

# Separate features and target variable
X = df.drop(columns=["Gas"])  # Sensor readings
y = df["Gas"]  # Target variable (Gas presence)

# Encode categorical target variable
label_encoder = LabelEncoder()
y = label_encoder.fit_transform(y)  # Convert labels to numerical values

# Normalize sensor readings
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Split dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.2,
random_state=42)

# Define neural network model
model = tf.keras.Sequential([
    tf.keras.layers.Dense(16, activation='relu', input_shape=(X_train.shape[1],)),
    tf.keras.layers.Dense(16, activation='relu'),
    tf.keras.layers.Dense(len(np.unique(y)), activation='softmax')  # Output layer
#Model Used : feedforward neural network
# Compile the model
model.compile(optimizer='adam', loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

# Train the model
model.fit(X_train, y_train, epochs=50, batch_size=32, validation_data=(X_test, y_test))
```

```
# Convert model to TensorFlow Lite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save TFLite model
with open("gas_leak_model.tflite", "wb") as f:
    f.write(tflite_model)

print("Model training & conversion complete! Saved as 'gas_leak_model.tflite'.")
```

****

Convertion

```
import numpy as np

# Load trained TFLite model
tflite_model_path = "gas_leak_model.tflite"
with open(tflite_model_path, "rb") as f:
    model_data = f.read()

# Convert model to C array format
hex_array = ",".join(f"0x{b:02x}" for b in model_data)

c_code = f"""#include <cstdint>

const unsigned char gas_leakage_model[] = {{
  {hex_array}
}};
"""

# Save as header file
with open("gas_leakage_model.h", "w") as f:
```

```
    f.write(c_code)

print("gas_leakage_model.h has been generated successfully")\
```
****

5.Arduino

```
#include <Arduino.h>
#include <ArduTFLite.h> // TensorFlow Lite for Arduino
#include "tensorflow/lite/micro/micro_interpreter.h"
#include "tensorflow/lite/micro/all_ops_resolver.h"
#include "tensorflow/lite/schema/schema_generated.h"
#include "gas_leakage_model.h" // Include the generated TFLite model file

// Increase tensor arena size for better memory allocation
constexpr int kTensorArenaSize = 16384;
uint8_t tensor_arena[kTensorArenaSize];

// TensorFlow Lite model variables
tflite::MicroInterpreter* interpreter;
const tflite::Model* model;
tflite::AllOpsResolver resolver;

// Define the gas sensor pin
const int gasSensorPin = A0;

void setup() {
  Serial.begin(115200);
  while (!Serial);
  Serial.println("Initializing TensorFlow Lite model...");

  // Load TensorFlow Lite model
```

```
  model = tflite::GetModel(gas_leakage_model);
  if (model->version() != TFLITE_SCHEMA_VERSION) {
    Serial.println("Model schema version mismatch!");
    while (1);
  }

  // Initialize the interpreter
  static tflite::MicroInterpreter static_interpreter(model, resolver, tensor_arena,
kTensorArenaSize);
  interpreter = &static_interpreter;

  // Allocate memory for model tensors
  if (interpreter->AllocateTensors() != kTfLiteOk) {
    Serial.println("AllocateTensors() failed! Not enough memory.");
    while (1);
  }

  Serial.println("TensorFlow Lite model initialized successfully!");
}

void loop() {
  // Read gas sensor value
  int sensorValue = analogRead(gasSensorPin);
  float normalized_input = sensorValue / 1024.0; // Normalize sensor input

  // Set input tensor
  float* input = interpreter->input(0)->data.f;
  *input = normalized_input;

  // Perform inference
  if (interpreter->Invoke() != kTfLiteOk) {
```

```
    Serial.println("Invoke() failed!");
    return;
  }

  // Retrieve output tensor
  float* output = interpreter->output(0)->data.f;

  // Display results
  Serial.print("Gas Sensor Reading: ");
  Serial.print(sensorValue);
  Serial.print(" | Model Prediction: ");
  Serial.println(*output);

  // Trigger alert if gas level is above threshold
  if (*output > 0.8) { // Threshold for gas leakage detection
    Serial.println("Warning! Gas Leakage Detected!");
    // Add additional actions like buzzer, LED, or wireless alert here
  }

  delay(1000); // Wait before next reading
}
**********************************************************************************
#ifndef GAS_LEAKAGE_MODEL.H
#define GAS_LEAKAGE_MODEL.H
 const unsigned char gas_leakage_model[] = {
const unsigned int GAS_LEAKAGE_MODEL.H= 2468;

#endif // GAS_LEAKAGE_MODEL.H
```

5.ML Model Justification

Detecting damage or leakage in pipelines involves analyzing multiple sensor parameters and identifying patterns that may not be easily visible through manual observation.

Machine Learning models can effectively learn from historical data to recognize complex relationships between pressure, temperature, gas concentration, and flow rate anomalies.

Supervised ML models can classify conditions as "Normal" or "Leakage/Damage" with high accuracy based on trained patterns.

ML-based solutions are more adaptable compared to fixed threshold-based systems, which may produce false alarms under varying environmental conditions.

6. ML Model Code (Example with Random Forest)

```
from sklearn.model_selection import train_test_split

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import classification_report, confusion_matrix


# Data preprocessing

X = df.drop(columns=['Quality'])  # Assuming "Quality" is the target

y = df['Quality']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)


# Model training

model = RandomForestClassifier(n_estimators=100, random_state=42)

model.fit(X_train, y_train)



# Predictions

y_pred = model.predict(X_test)


# Evaluation

print(confusion_matrix(y_test, y_pred))

print(classification_report(y_test, y_pred))
```

7. Metrics for Model Evaluation

Accuracy — To measure how well the model classifies leakage and non-leakage conditions.

Precision — To evaluate how many detected leakages are actually correct (avoid false alarms).

Recall (Sensitivity) — To measure how many actual leakage incidents the model is able to detect (avoid missed detections).

F1-Score — Balance between precision and recall for reliable performance

8. Self Inference

The model successfully detects abnormal patterns (leakages or damages) in pipeline sensor data.

Real-time monitoring and alerts help in quick identification and prevention of accidents.

The system reduces manual inspection effort and enhances safety and reliability.

The deployed TFLite model runs efficiently on embedded devices, making it suitable for field applications.

9. Scope for Enhancement

Integration of more advanced sensors for multi-parameter analysis (vibration, acoustic, humidity).

Development of a mobile app for remote alerts and pipeline monitoring.

Adding cloud-based dashboards for real-time visualization and analytics.

Use of anomaly detection techniques for unknown damage patterns.