# IoT-Enabled Fall Detection System for the Elderly Using Machine Learning

*Submitted in partial fulfillment of the requirements for the degree of*

**Bachelor of Technology**
**in**
**Electronics and Communication with Specialization in Biomedical Engineering**

*By*

**Tushar Pati Tripathi (21BML0105)**
**Rohan Joshi (21BML0131)**
**Aiyushi Srivastava (21BML0155)**

Under the guidance of

**Prof. /Dr. Ravi Kumar C.V.**

School of Electronics Engineering,
Vellore Institute of Technology, Vellore


VIT®
Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

April, 2025

# DECLARATION

I hereby declare that the thesis entitled "**IoT-Enabled Fall Detection System for the Elderly Using Machine Learning**" submitted by me, for the completion of the course "BECE48J – Project 2" to the school of electronics engineering, Vellore institute of Technology, Vellore is Bonafide work carried out by me under the supervision of **Dr. Ravi Kumar C.V**.

I further declare that the work reported in this thesis has not been submitted previously to this institute or anywhere.

Place: Vellore
Date: 16 April 2025

**Signature of the Candidate**

# CERTIFICATE

This is to certify that the thesis entitled "**IoT-Enabled Fall Detection System for the Elderly Using Machine Learning**" submitted by **Tushar Pati Tripathi(21BML0105), Rohan Joshi(21BML0131), Aiyushi Srivastava(21BML0155), SENSE, VIT,** for the completion of the course "BECE498J – Project 2", is a Bonafide work carried out by him / her under my supervision during the period, 13. 12. 2024 to 16.04.2025, as per the VIT code of academic and research ethics.

I further declare that the work reported in this thesis has not been submitted previously to this institute or anywhere.

Place: Vellore
Date: 16/04/25

**Signature of Guide**

**Internal Examiner**

**External Examiner**

Head of Department

Sensor & Biomedical Technology, SENSE

# ACKNOWLEDGEMENTS

With immense pleasure and deep sense of gratitude, we wish to express my sincere thanks to my guide Dr.Ravi Kumar CV, Associate Professor Senior, SENSE, Vellore Institute of Technology, without his motivation and continuous encouragement, this project work would not have been successfully completed.

We are grateful to the Chancellor of Vellore Institute of Technology, Dr. G. Viswanathan, as well as the Vice Presidents and the Vice Chancellor, for their constant motivation and encouragement to carry out research at VIT University. We sincerely thank them for providing us with the infrastructural facilities and numerous resources required for the successful completion of our project.

We express our sincere thanks to Dr. Jasmine, Dean, SENSE, Vellore Institute of Technology and Dr. Sathya P, Associate Professor & Head Department of Sensor & Biomedical Technology, SENSE, Vellore Institute of Technology for their kind words of support and encouragement. We like to acknowledge the support rendered by our colleagues in several ways throughout our project work.

Place: Vellore

Date: 18 April 2025

Aiyushi Srivastava(21BML0155)

Rohan Joshi(21BML0131)

Tushar Pati Tripathi(21BML0105)

# EXECUTIVE SUMMARY

Falls in older adults are one of the most significant public health issues in the world. Falls are the second most common cause of unintentional injury-related deaths worldwide, with adults aged 60 years and older being the most impacted group, states the World Health Organization (WHO). An estimated 684,000 deaths occur due to falls every year around the world, and more than 37 million falls are severe enough to result in medical care. This figure alone highlights the importance of a proper fall detection system, particularly with the increasing global population of elderly people.

In the United States alone, the Centers for Disease Control and Prevention (CDC) states that almost one in four individuals aged 65 and above falls annually. This represents over 3 million emergency room visits every year. More alarming is that over 95% of hip fractures among older adults and the major cause of traumatic brain injury among this age group are due to falls. The injuries can lead to long-term disability and, in certain cases, early entry into nursing homes.

The economic implications are just as large. The overall cost of medical care for falls in the U.S. was over $50 billion in 2020, and Medicare and Medicaid paid for about 75% of this. With increasing life expectancy and population aging, these figures are set to rise exponentially over the next few decades unless preventive strategies and fast-response technologies are taken up broadly.

In addition to this, it's been revealed that almost 50% of older people suffering from a fall have a recurrent fall in the same year. However, many falls remain unreported—particularly in case the elderly inhabitant lives on his or her own or even can't get help. According to a study by the National Institute on Aging (NIA), obtaining assistance in the first hour after a fall can lower the chances of severe complications or death by as much as 80%, further emphasizing the imperative of rapid fall detection and response.

These figures emphasize not just the frequency and severity of falls among the elderly but also the immense healthcare burden and preventable nature of many of these falls. Deploying an intelligent IoT-based fall detection system can have a considerable impact in decreasing these numbers by ensuring timely intervention and medical care.

# CONTENTS

# APPENDIX A: LIST OF FIGURES

| | |
|---|---|
| Fig. 11 | Confusion Matrix – Random Forest |
| Fig. 12 | Probability Calibration Curve (Reliability Diagram) |
| Fig. 13 | Jerk Magnitude Over Time with Detected Falls (time series visualization) |
| Fig. 14 | 3D Accelerometer Data (colored by Jerk Magnitude) |

# LIST OF ABBREVIATIONS

| **Abbreviation** | **Full Form** |
|---|---|
| IoT | Internet of Things |
| ML | Machine Learning |
| HAR | Human Activity Recognition |
| WHO | World Health Organization |
| CDC | Centers for Disease Control and Prevention |
| NIA | National Institute on Aging |
| AM | Acceleration Magnitude |
| ΔAlt | Change in Altitude |
| UFT | Upper Fall Threshold |
| LFT | Lower Fall Threshold |

| | |
|---|---|
| SVM | Signal Vector Magnitude / Support Vector Machine (context dependent) |
| AI | Artificial Intelligence |
| EDA | Exploratory Data Analysis |
| ROC | Receiver Operating Characteristic |
| AUC | Area Under the Curve |
| SMS | Short Message Service |
| Blynk | IoT Platform for mobile notifications and control |
| HC-SR04 | Ultrasonic Distance Sensor |
| MPU6050 | 6-axis Motion Tracking Device (Accelerometer + Gyroscope) |
| BMP280 | Barometric Pressure Sensor (used for altitude) |
| JerkMag | Jerk Magnitude (rate of change of acceleration) |
| CSV | Comma-Separated Values |
| AccMag | Acceleration Magnitude |

| | |
|---|---|
| GyroMag | Gyroscope Magnitude |
| RFC | Random Forest Classifier |
| XGBoost | Extreme Gradient Boosting |
| DMP | Digital Motion Processor |
| GUI | Graphical User Interface |
| GPS | Global Positioning System |
| LED | Light Emitting Diode |
| GDPR | General Data Protection Regulation (EU Privacy Law) |
| FFT | Fast Fourier Transform |

# 1.INTRODUCTION

Statistically, falls represent a significant risk for individuals aged 65 and above, with data indicating that falls are the primary cause of injury or fatality within this demographic. Studies show that approximately 30% of elderly individuals over the age of 65 experience falls on an annual basis, underscoring the pervasive nature of this issue in the elderly population.

The primary reason for falls in the elderly is often attributed to an instability in the center of gravity of the human body, leading to a lack of balance and symmetry. Addressing this issue, a proposed approach based on the symmetry principle aims to reorganize accidental falls by analyzing critical parameters such as the speed of descent at the center of the hip joint, the angle of the human body centerline with the ground, and
the width-to-height ratio of the human body external rectangular.

The proposed solution outlined in the research paper suggests an innovative approach centered around reorganizing accidental falls through the application of the symmetry principle. This method involves the extraction of skeleton information from the human body using advanced technology such as Open Pose, to identify falls based on critical parameters including the speed of descent at the hip joint center, the angle between the human body's centerline and the ground, and the width-to-height ratio of the external rectangular representation of the human body.

## 1.1 Literature Review:

| S No | Publisher | Focus/Scope of Paper | Methodology | Test Data | Results | Merits & Demerits | Future Scope |
|---|---|---|---|---|---|---|---|
| 1 | PubMed | Falls prevention interventions for older adults | Systematic review and meta-analysis | Multiple previous studies | Identifies effective fall prevention strategies | Merit: Comprehensive review; Demerit: No real-time implementation | Development of personalized intervention strategies |
| 2 | PubMed | Multifactorial assessment for fall prevention in community and emergency settings | Systematic revie and meta-analys | Multiple studies analyzed | Effectiveness of interventions in emergency settings | Merit: Broad study coverage; Demerit: No hardware implementation | Enhancing real-time fall detection in emergency settings |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 3 | MDPI | Risk propagation in emergency logistics network | Mathematical modeling and simulations | Simulated data | Identifies risk propagation in emergency logistics | Merit: Focus on risk management; Demerit: No specific application to fall detection | Application to healthcare logistics for fall response |
| 4 | Science Direct | Survey on fall detection principles and approaches | Review of existing fall detection methodologies | Multiple previous studies | Comparison of different fall detection techniques | Merit: Comprehensive survey; Demerit: Lacks practical implementation | AI-based hybrid fall detection techniques |
| 5 | PubMed | Step and spin turns classification using wireless gyroscopes for fall risk assessments | Wearable gyroscope-based classification | Test subjects performing movements | Accurate classification of movements related to fall risk | Merit: Precise motion classification; Demerit: Requires wearables | Improving real-time analysis for fall prediction |

## 1.2 Background and Current Scenario

With populations across the world aging, fall injuries among elderly people have emerged as a significant public health issue. As per the World Health Organization (WHO), an estimated 1 in 3 adults aged 65 and above have at least one fall per year. Among these falls, numerous result in severe injuries like fractures, head injuries, and in some cases, life-threatening complications. Falls are currently the second most common cause of unintentional injury-related death worldwide, with over 684,000 deaths annually.

In developed nations like the United States, the figures are even more dire. Figures from the Centers for Disease Control and Prevention (CDC) indicate that over 36 million falls are reported among older adults every year, resulting in over 32,000 fatalities. Over 3 million elderly people also need emergency medical care for fall-related injuries. About 300,000 older adults are hospitalized every year because of hip fractures, and more than 95% of these fractures occur because of falling.

One of the most important issues in the occurrence of falls is the absence of prompt intervention. Most elderly persons are alone, and if they fall, they might not be able to summon assistance. The National Institute on Aging (NIA) indicates that remaining on the ground for over an hour following a fall increases the risk of hospitalization and death considerably. Research indicates that when help is sought within the initial hour, the possibility of survival following a fall rises by more than 80%.

Why Is Fall Detection Necessary Today ?

Aside from enhancing security, these systems ease the workload on caregivers and health professionals, providing reassurance to families. They also form part of evidence-based data that can be employed for prevention and early intervention measures.
With the increasingly aged population, unprecedented fatality and injury rates, staggering healthcare expense, and pronounced delays in the response to emergency situations, applying smart, IoT-enabled fall detection solutions is no longer a matter of convenience, but a call of necessity. These technologies could save lives, lower medical bills, and equip the elderly to live with even more independence and dignity.

## 1.3 Research gap

In spite of the remarkable advancement in IoT-based fall detection, some important research gaps still exist:

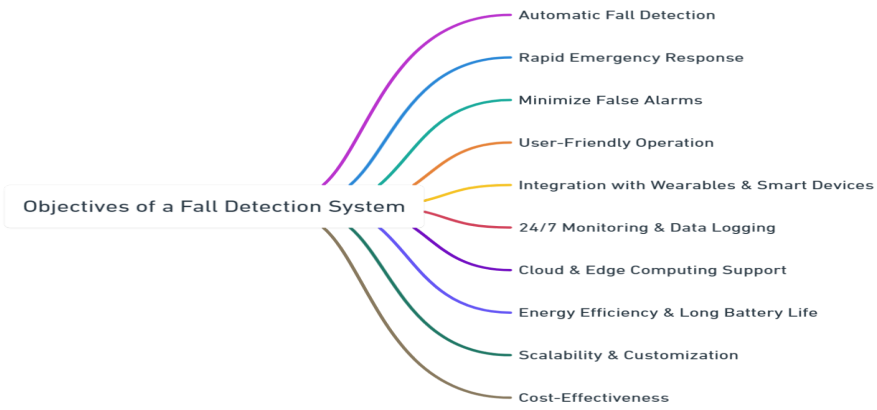
1. Limited Real-World Datasets Majority of the work uses simulated falls instead of real falls among the elderly. This hinders ML model generalizability for real-world application.
2. High False Positives/Negatives The systems are still far from distinguishing between actual falls and other non-fall activities such as sitting down abruptly, tying shoes, or tripping.
3. Lack of Personalization Most models are general and not specific to individual physical states, activity rhythms, or medical histories, lowering reliability.
4. Limitations of Edge Computing It is hard to integrate sophisticated ML models into lean devices such as Arduino or ESP32 because of hardware constraints (memory, speed).
5. Lack of Long-Term Testing Very few systems are put through lengthy trials in actual environments, so it is hard to determine long-term accuracy, robustness, and user acceptability.
6. Insufficient Multi-Sensor Fusion Methods Although numerous sensors are employed, robust fusion algorithms (integrating accelerometer, gyroscope, altitude, etc.) are underdeveloped.
7. Limited Integration with Healthcare Systems There are few solutions that are fully interoperable with hospitals or caregivers' platforms, limiting broad clinical adoption.
8. Privacy vs. Precision Trade-offs Vision-based systems provide precision but are ethically/privacy-wise problematic; sensor-based systems are safer but potentially lack rich context.

## 1.4 Problem Statement

Elderly individuals, particularly those living alone or with limited supervision, are at high risk of falls, which can lead to serious injuries, disabilities, or even fatalities. Traditional methods of fall detection, such as manual monitoring or emergency buttons, are often ineffective due to delays in response or the inability of the person to call for help. There is a need for an automatic, real-time fall detection system that can promptly alert caregivers or emergency services to ensure quick assistance and minimize health risks. With the aging of the world's population, keeping older people safe, healthy, and independent is an urgent priority. Of all the hazards of old age, falls are the most frequent and perilous. They are not only the number one cause of injury and accidental death among older people but also a major contributor to loss of mobility, independence, and quality of life.

Over 30% of individuals aged 65 and older suffer at least one fall per year, according to the World Health Organization (WHO). It goes up with frailty and advancing age. In the United States, the Centers for Disease Control and Prevention (CDC) estimates that: A fall-related injury is treated in an elderly individual in the emergency room every 11 seconds. Falls are the major cause of both fatal and non-fatal injury among older adults.

# 2. RESEARCH OBJECTIVE




A Smart IoT-Based Fall Detection System seeks to improve care for the elderly through the use of advanced sensor technology, artificial intelligence (AI), and cloud computing. The following is a clear definition of its major objectives:

1. Design an Automated Fall Detection System

Objective: Develop an intelligent system for the detection of falls without any manual intervention.
- Conventional fall detection systems depend on the elderly person manually pressing an emergency button or shouting for assistance, which might not always be feasible.

2. Improve Real-Time Monitoring and Alerts

Mission: Offer real-time monitoring and immediate notification to caregivers or emergency services upon detection of a fall.
- The system will continuously monitor motion and changes in body posture through sensors.
- Notifications may also contain more details like the location and intensity of the fall.

3. Enhance Detection Accuracy and Minimize False Alarms
Goal: Enhance the accuracy of fall detection while reducing false positives and false negatives.

4. Merge Wearable and Non-Intrusive Sensor Technologies
Goal: Create an adaptable system based on both wearable and ambient sensors to accommodate varying user needs.
- Wearable Sensors
- Non-Intrusive Sensors

5. Ensure User Comfort and Acceptance
Objective: Create a system that is user-friendly, comfortable, and less intrusive for elderly users.
- Most elderly persons might view wearable technology as intrusive or inconvenient.

6. Allow Remote Access and Cloud-Based Data Processing
Goal:Leverage IoT and cloud computing to enable caregivers and healthcare professionals to remotely access real-time information.

7. Improve Elderly Safety and Independence
Goal: Implement a solution that supports elderly individuals to live independently while maintaining safety.

8. Facilitate Emergency Response and Assistance
Objective: Reduce emergency response
time by instantly alerting designated contacts and medical services.
- Once a fall is confirmed, the system will notify emergency contacts, caregivers, or medical personnel.

9. Predict Long-Term Movement Patterns for Fall Prevention
Objective: Utilize AI and data analytics to forecast potential fall risks and offer preventive suggestions.
- The system will monitor daily patterns of movement and identify decreasing mobility, instability, or other indicators of fall risk.

10. Create an Affordable and Scalable Solution
Objective:Ensure the system is affordable, energy-efficient, and easily deployable in various environments.
- Affordability:
 - Use low-cost sensors and open-source software to keep costs manageable for widespread adoption.
- Scalability:
- The system needs to be compatible with various living environments, such as single houses, assisted care facilities, and hospitals.

## 3. RELEVANCE OF PROBLEM STATEMENT W.R.T SDG:

Sustainable Development Goals (SDGs) Facilitated by the Fall Detection Project:


1. SDG 3: Good Health and Well-being Target 3.8 – Have universal health coverage, including access to quality essential healthcare services. The project improves health outcomes through real-time detection of falls and emergency response, particularly for older persons at high risk of injury or death due to falls. It supports injury prevention,

early medical treatment, and shorter hospital stays, thus enabling healthy aging and living independently.

2. SDG 9: Industry, Innovation, and Infrastructure Target 9.5 – Improve scientific research and advance technological capabilities. Through the use of IoT sensors and AI algorithms, the project promotes innovation in healthcare technology and enables smart assistive devices that can be scaled up for larger healthcare systems and communities. It promotes R&D on wearable health technology and smart homes for the disabled and the elderly.
3. SDG 10: Reduced Inequalities Target 10.2 – Empower and promote social, economic, and political inclusion of all. The framework offers affordable, accessible fall-detecting equipment that can be employed by old people in urban as well as rural settings, thereby lessening inequalities in eldercare. It allows older individuals, particularly those who live alone or in poverty-stricken conditions, to stay safe and engaged.
4. SDG 11: Sustainable Cities and Communities Target 11.7 – Ensure universal access to safe, inclusive and accessible public spaces. The project facilitates the growth of smart elder-centric communities with real-time health monitoring infrastructure. It aids smart home and assisted living applications, enhancing safety and quality of life for the elderly.

5. SDG 12: Responsible Consumption and Production Target 12.5 – Minimize waste generation by prevention, reduction, and recycling. The project employs energy-efficient and recyclable IoT hardware, enabling sustainable production and reducing environmental footprint. Modular and reusable sensor parts minimize electronic waste.
6. SDG 17: Partnerships for the Goals Target 17.6 – Increase collaboration on science, technology, and innovation. The system can be implemented via public-private partnerships in healthcare, NGOs, and government schemes for elderly care. Provides scope for the cooperation of health providers, tech developers, and academic researchers.

# 4. PROPOSED SYSTEM

The aim of this project is to design and implement a real-time Fall Detection System for elderly individuals using Arduino and IoT sensors. The system monitors motion, pressure, and proximity to accurately detect falls and immediately trigger alerts via a LED.

## 4.1 Design Approach / Materials & Methods:

### 4.1.1 Materials Used

**Hardware Components:**

The system consists of three key sensors connected to an Arduino Uno: an MPU6050 for motion tracking, a BMP280 for pressure and altitude, and an HC-SR04 ultrasonic sensor for distance

from the ground. An LED is used on pin D8 as a fall alert indicator, replacing the previously used buzzer.

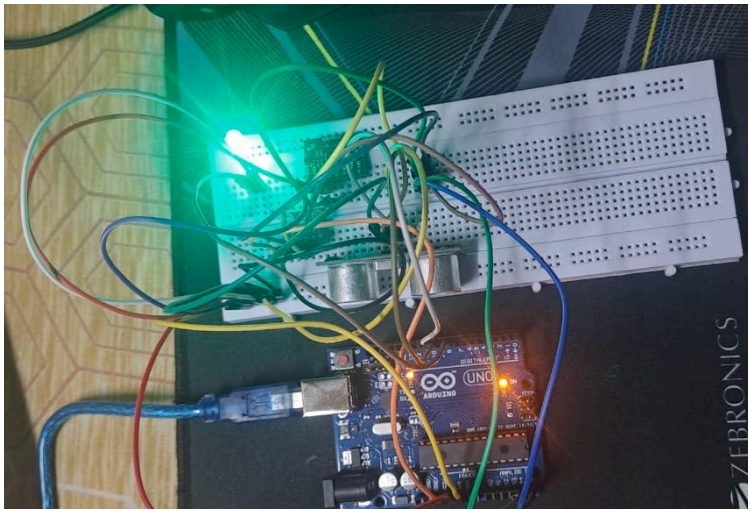
| Component | Description / Role |
|---|---|
| **Accelerometer** | Detects changes in movement (acceleration). Helps detect sudden impact or free fall. |
| **Gyroscope** | Measures orientation and rotation; used alongside accelerometer for more accurate motion tracking. |
| **Microcontroller (e.g., ESP32, Arduino UNO, NodeMCU)** | Core controller that processes sensor data and transmits alerts. |
| **Pressure Sensors** | Installed in the environment (e.g., floor mats) to detect sudden impact or presence. |
| **Battery / Power Supply** | Provides power to portable devices; rechargeable batteries for wearables. |
| **Buzzer / Alert System** | Sounds an alert locally if a fall is detected. |

**Software Components:**

The software stack for the fall detection system integrates embedded programming, data acquisition, and machine learning-based classification. The system is divided into two major software layers: **Arduino-side code** for real-time data collection and **Python-based tools** for data analysis and fall detection.

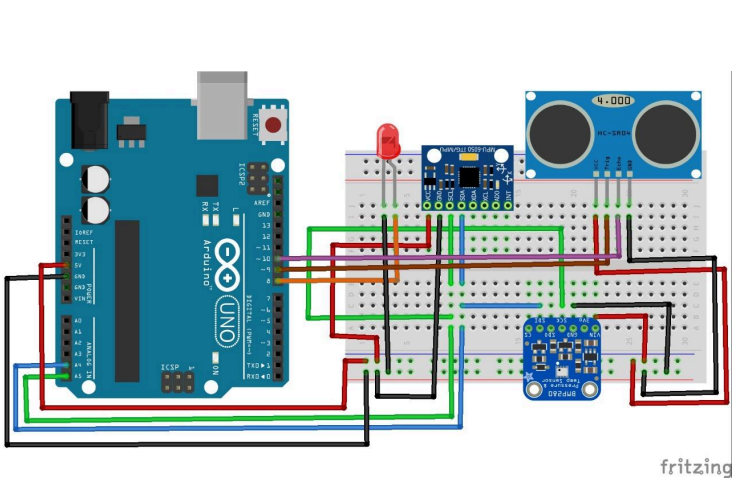
| Component | Description / Role |
|---|---|
| **Arduino IDE** | Used to write and upload code to microcontrollers like Arduino, ESP32. |
| **Python** | For data processing, ML model inference, or handling camera inputs. |

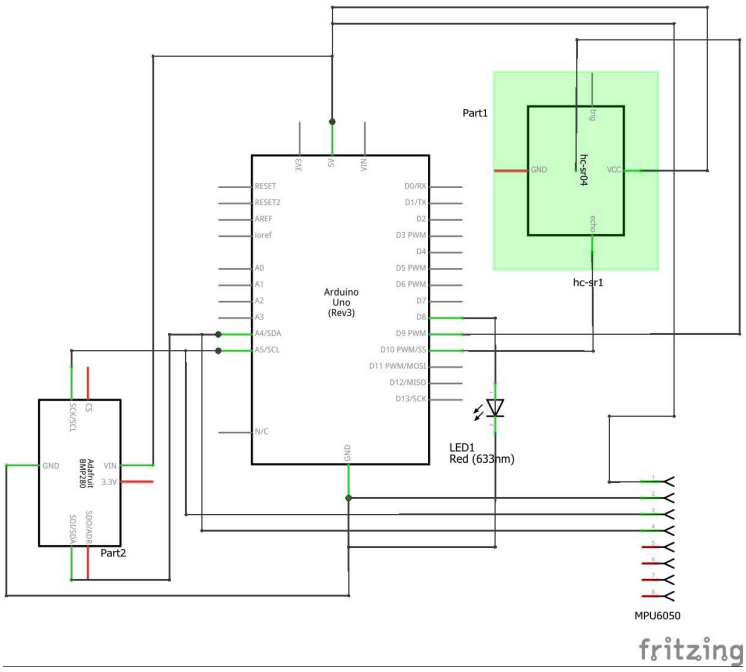
| | |
|---|---|
| **TensorFlow / PyTorch / Scikit-learn** | Used to develop and train AI/ML models for fall detection. |
| **Firebase / AWS / Azure IoT** | Cloud platforms used for real-time database storage, notifications, and monitoring. |
| **MQTT / HTTP Protocols** | Communication protocols to send data between sensors and the server/cloud. |
| **Mobile App** | Caregiver or user interface to view alerts and monitor elderly health. |




### 4.1.2 **Design approach**

To accurately detect falls in elderly individuals using a low-cost, real-time system that combines multiple sensors and machine learning, ensuring timely alerts and enhanced safety.



The schematic diagram illustrates the wiring of a fall detection system built around the Arduino Uno microcontroller. At the core of the design, the MPU6050 sensor is connected to the Arduino via the I2C protocol, with the SDA and SCL lines connected to analog pins A4 and A5 respectively. This sensor is responsible for capturing acceleration and angular velocity data to identify sudden movements or changes in orientation. Similarly, the BMP280 sensor, which measures barometric pressure and altitude, is also connected via the same I2C lines, sharing the A4 and A5 pins for communication.

The HC-SR04 ultrasonic sensor is connected to digital pins D9 (Trig) and D8 (Echo), and is used to determine the distance between the subject and the ground, helping to confirm whether a fall has resulted in the person being on the floor. A LED is also integrated into the circuit via digital pin D7, and is configured to turn on as an alert indicator when a fall is detected. All components are powered through the Arduino's 5V and GND pins, and a breadboard is used for organizing the circuit connections conveniently. This setup ensures the microcontroller receives real-time data from multiple sensors, which is essential for accurately detecting and confirming fall events using a combination of rule-based logic or machine learning algorithms.

ML Algorithms Used:

Once raw sensor data is collected (acceleration, orientation, altitude, and distance), it is processed and labeled into either a fall or non-fall event using ML models like:

1. Random Forest (RF)

- Ensemble decision tree model that works well with noisy and mixed-type data.

- Useful for  Handling nonlinear boundaries,Robust against overfitting

- Features used Acceleration magnitude,Gyro changes,Pressure variation,Distance to ground

2. XGBoost:

- Optimized gradient boosting algorithm known for high performance in tabular data.

- Offers better accuracy and speed compared to RF in many cases.

- Handles class imbalance and missing data effectively.

- Regularization prevents overfitting.

Predictive Capabilities:

By training the ML model on collected data labeled as "fall" or "non-fall," we achieve:

| Feature | Description |
|---|---|
| **Real-time prediction** | System can detect falls instantly based on live sensor input |
| **High accuracy** | With models like XGBoost and RF, false positives are minimized |
| **Multi-sensor fusion** | Increases reliability by validating fall events from multiple sensors |
| **Adaptability** | Can be retrained with more data for better generalization |

Pipeline Overview:

1. Data Collection from MPU6050, BMP280, HC-SR04.

2. Preprocessing (filter noise, feature engineering like RMS, derivatives, deltas).

3. Model Training using RF/XGBoost (off-device, e.g., Python/sklearn).

4. Deployment: Use rule-based logic or upload thresholds/model parameters to Arduino.

5. Prediction + Alert: Classify motion and alert via LED/buzzer.

## 4.2.Code and Standards:

In this section, we provide a comprehensive breakdown of the different code segments and technologies that drive the operation and intelligence of our IoT-based energy harvesting system. The codes implemented in this project span across three key domains: Machine Learning algorithms (Python), front-end dashboard interface styling (CSS), and hardware control via

Arduino IDE (C/C++). Each code block plays an essential role in the functioning of the entire system, ensuring accurate data processing, effective visualization, and robust real-time control.

To ensure modularity, readability, and performance, the system codebase was divided into three primary blocks—each responsible for a critical aspect of the project. The code adheres to common **embedded system standards** and **Pythonic coding practices**, ensuring portability and documentation. All source codes are version-controlled via **GitHub**, and structured using appropriate libraries and modules.

A core advancement in the ML codebase was the inclusion of dynamic dataset handling to accommodate user-uploaded CSV files. The dashboard uses pandas to preview uploaded data, followed by dynamic input and target selection interfaces. Once the user selects the relevant features and target variable, the system conducts scaling, train-test splitting, and model training without requiring manual code adjustments. This dynamic workflow enables the dashboard to act as a regression modeling engine for any numerical dataset, supporting real-time analytics and visualizations for arbitrary data inputs. The flexibility also includes error handling for missing values, non-numeric fields, and improperly formatted files.

### 4.2.1 **Arduino IDE**

```
#include <Wire.h>
#include <MPU6050.h>
#include <Adafruit_BMP280.h>

MPU6050 mpu;
Adafruit_BMP280 bmp; // Not used, just declared

// Ultrasonic Sensor Pins
const int trigPin = 9;
const int echoPin = 10;

float accX, accY, accZ;
float gX, gY, gZ;
float accMagnitude;
float gyroMagnitude;
float distance = 0;
float pressure = 101325.0;  // Imaginary pressure (Pa)
float altitude = 50.0;      // Imaginary altitude (m)

const float FALL_THRESHOLD = 0.8;
const float GYRO_THRESHOLD = 3.0;
const int ledPin = 8;

bool fallDetected = false;
float prevAccMagnitude = 1.0;
unsigned long fallTime = 0;
```

```
void setup() {
  Serial.begin(9600);
  Wire.begin();
  mpu.initialize();

  pinMode(ledPin, OUTPUT);
  pinMode(trigPin, OUTPUT);
  pinMode(echoPin, INPUT);

  if (!mpu.testConnection()) {
    Serial.println("MPU6050 connection failed!");
    while (1);
  }


Serial.println("accX(g),accY(g),accZ(g),gX(°/s),gY(°/s),gZ(°/s),accMag(g),gyroMag(g),distance(
cm),pressure(Pa),altitude(m),Fall Status");
}

void loop() {
  accX = mpu.getAccelerationX() / 16384.0;
  accY = mpu.getAccelerationY() / 16384.0;
  accZ = mpu.getAccelerationZ() / 16384.0;

  gX = mpu.getRotationX() / 131.0;
  gY = mpu.getRotationY() / 131.0;
  gZ = mpu.getRotationZ() / 131.0;

  accMagnitude = sqrt(accX * accX + accY * accY + accZ * accZ);
  gyroMagnitude = sqrt(gX * gX + gY * gY + gZ * gZ);
  distance = readUltrasonicDistance();

  // Simulate sensor readings
  pressure = 101250.0 + random(-200, 200);
  altitude = 50.0 + random(-5, 5);

  float accDrop = prevAccMagnitude - accMagnitude;

  // Fall detection
  if ((accMagnitude < FALL_THRESHOLD || accDrop > 0.5) && gyroMagnitude > GYRO_THRESHOLD &&
!fallDetected) {
    fallDetected = true;
    fallTime = millis();
    digitalWrite(ledPin, HIGH);
  }

  // Reset fall detection after 2s
  if (fallDetected && millis() - fallTime > 2000) {
    digitalWrite(ledPin, LOW);
    fallDetected = false; // ✅ Reset here
  }

  prevAccMagnitude = accMagnitude;
```

```
  Serial.print("AccX:"); Serial.print(accX); Serial.print(",");
  Serial.print("AccY:"); Serial.print(accY); Serial.print(",");
  Serial.print("AccZ:"); Serial.print(accZ); Serial.print(",");
  Serial.print("GyroX:"); Serial.print(gX); Serial.print(",");
  Serial.print("GyroY:"); Serial.print(gY); Serial.print(",");
  Serial.print("GyroZ:"); Serial.print(gZ); Serial.print(",");
  Serial.print("AccMag:"); Serial.print(accMagnitude); Serial.print(",");
  Serial.print("GyroMag:"); Serial.print(gyroMagnitude); Serial.print(",");
  Serial.print("Distance:"); Serial.print(distance); Serial.print(",");
  Serial.print("Pressure:"); Serial.print(pressure); Serial.print(",");
  Serial.print("Altitude:"); Serial.print(altitude); Serial.print(",");
  Serial.print("Status:"); Serial.println(fallDetected ? "Fall Detected" : "Fall Not
Detected");

  delay(200);
}

float readUltrasonicDistance() {
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
  long duration = pulseIn(echoPin, HIGH);
  return duration * 0.034 / 2;
}
```

### 4.2.2 **Exploratory Data Analysis(EDA)**

```
# --- STEP 1: Install and Import Required Libraries ---
!pip install openpyxl scipy scikit-learn pandas numpy matplotlib seaborn plotly
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import scipy.signal as signal
from sklearn.model_selection import train_test_split
import plotly.express as px

# For Jupyter/Colab notebook settings
%matplotlib inline
sns.set(style="whitegrid")

# --- STEP 2: Upload and Load Your Excel File ---
from google.colab import files
uploaded = files.upload()
excel_file = pd.ExcelFile(next(iter(uploaded)))
print("Available Sheets:", excel_file.sheet_names)
```

```
# --- STEP 3: Data Loading and Initial Cleaning ---
def load_and_clean_data(excel_file):
    """Load and clean the raw sensor data from Excel file."""
    # Load the Data In sheet
    data_in = excel_file.parse('Data In')

    # Get the header and data rows
    data = data_in.iloc[3:, :13].copy()
    data.columns = data_in.iloc[2, :13].values
    data = data[~data['TIME'].isin(['TIME', 'Historical Data'])]  # remove unwanted rows
    data = data.dropna(how='all')  # remove empty rows
    data.reset_index(drop=True, inplace=True)

    # Function to extract numerical values from strings like "AccX:-0.13"
    def extract_value(cell):
        try:
            return float(str(cell).split(":")[1])
        except:
            return np.nan

    # Create cleaned DataFrame
    df = pd.DataFrame()
    df['Timestamp'] = pd.to_datetime(data['TIME'])

    # Map sensor columns
    col_map = {
        "CH1": "AccX", "CH2": "AccY", "CH3": "AccZ", "CH4": "GyroX",
        "CH5": "GyroY", "CH6": "GyroZ", "CH7": "AccMag", "CH8": "GyroMag",
        "CH9": "Distance", "CH10": "Pressure", "CH11": "Altitude", "CH12": "Status"
    }

    for ch, name in col_map.items():
        if name != "Status":
            df[name] = data[ch].apply(extract_value)
        else:
            df[name] = data[ch].astype(str).str.extract(r'Status:(.*)')

    return df

df_raw = load_and_clean_data(excel_file)

# --- STEP 4: Exploratory Data Analysis (EDA) ---
def perform_eda(df):
    """Perform exploratory data analysis on the sensor data."""
    df_clean = df.copy()

    # Add a 'FallDetected' column based on 'Status'
    df_clean['FallDetected'] = df_clean['Status'].str.contains('Fall', case=False,
na=False).astype(int)
    df_clean['FallColor'] = df_clean['FallDetected'].map({0: 'blue', 1: 'red'})
    detected_falls = df_clean[df_clean['FallDetected'] == 1].copy()
```

```
    # Basic info
    print("\n=== Basic Dataset Info ===")
    print(df_clean.info())
    display(df_clean.describe())
    print("\nUnique Status Labels:", df_clean['Status'].unique())

    # Plot time-series for main sensors
    plt.figure(figsize=(14, 6))
    sns.lineplot(x='Timestamp', y='AccMag', data=df_clean, label='AccMag')
    sns.lineplot(x='Timestamp', y='GyroMag', data=df_clean, label='GyroMag')
    plt.scatter(detected_falls['Timestamp'], detected_falls['AccMag'], color='red',
marker='o', label='Detected Fall')
    plt.title("Sensor Magnitude over Time with Detected Falls")
    plt.xlabel("Time")
    plt.ylabel("Magnitude")
    plt.legend()
    plt.show()

    # Fall detection status plot
    plt.figure(figsize=(14, 4))
    plt.plot(df_clean['Timestamp'], df_clean['FallDetected'], color='black', label='Fall
Detected')
    plt.title("Fall Detection Status Over Time")
    plt.xlabel("Time")
    plt.ylabel("Fall Detected (1 = Yes, 0 = No)")
    plt.legend()
    plt.show()

    # Correlation heatmap
    plt.figure(figsize=(10, 8))
    sns.heatmap(df_clean.drop(columns=['Timestamp', 'Status', 'FallDetected',
'FallColor']).corr(),
                annot=True, cmap='coolwarm')
    plt.title("Correlation Matrix of Sensor Features")
    plt.show()

    # Status distribution
    plt.figure(figsize=(6, 4))
    sns.countplot(x='Status', data=df_clean)
    plt.title("Fall Detection Status Distribution")
    plt.xticks(rotation=45)
    plt.show()

    return df_clean

df_clean = perform_eda(df_raw)

# --- STEP 5: Advanced Data Processing and Feature Engineering ---
def feature_engineering(df):
    """Perform advanced data processing and feature engineering."""
    # 1. Data Cleaning
    # Remove duplicate timestamps
    df = df[~df['Timestamp'].duplicated(keep='first')]
```

```
    # Forward fill missing values in critical columns
    for col in ['Pressure', 'Altitude', 'Status']:
        df[col] = df[col].ffill()

    # Resample to consistent time intervals with linear interpolation
    df_processed = df.set_index('Timestamp').resample('100ms').interpolate(method='linear')
    df_processed = df_processed.reset_index()

    # 2. Magnitude Features
    # Recalculate magnitudes to ensure consistency (in case original magnitudes were
pre-calculated differently)
    df_processed['AccMag'] = np.sqrt(
        df_processed['AccX']**2 +
        df_processed['AccY']**2 +
        df_processed['AccZ']**2
    )

    df_processed['GyroMag'] = np.sqrt(
        df_processed['GyroX']**2 +
        df_processed['GyroY']**2 +
        df_processed['GyroZ']**2
    )

    # 3. Jerk Features (derivative of acceleration)
    time_diff = df_processed['Timestamp'].diff().dt.total_seconds()
    df_processed['JerkX'] = df_processed['AccX'].diff() / time_diff
    df_processed['JerkY'] = df_processed['AccY'].diff() / time_diff
    df_processed['JerkZ'] = df_processed['AccZ'].diff() / time_diff
    df_processed['JerkMag'] = np.sqrt(
        df_processed['JerkX']**2 +
        df_processed['JerkY']**2 +
        df_processed['JerkZ']**2
    )

    # 4. Additional Features
    # Acceleration-Gyroscope ratio
    df_processed['AccGyroRatio'] = df_processed['AccMag'] / (df_processed['GyroMag'] + 1e-6)
# Add small value to avoid division by zero

    # Signal vector magnitude (SVM)
    df_processed['SVM'] = np.sqrt(
        df_processed['AccMag']**2 +
        df_processed['GyroMag']**2
    )

    # Moving averages and standard deviations
    window_size = 10  # 1 second window for 100ms data
    for col in ['AccMag', 'GyroMag', 'JerkMag']:
        df_processed[f'{col}_MA'] = df_processed[col].rolling(window=window_size).mean()
        df_processed[f'{col}_STD'] = df_processed[col].rolling(window=window_size).std()

    # 5. Clean up (remove rows with NA values from diff and rolling operations)
```

```
    df_processed = df_processed.dropna()

    # 6. Recalculate FallDetected after processing
    df_processed['FallDetected'] = df_processed['Status'].str.contains('Fall', case=False,
na=False).astype(int)

    # 7. Move Status and FallDetected columns to the end
    cols = [col for col in df_processed.columns if col not in ['Status', 'FallDetected']] +
['Status', 'FallDetected']
    df_processed = df_processed[cols]

    return df_processed

df_processed = feature_engineering(df_clean)

# --- STEP 6: Visualize Processed Data with New Features ---
def visualize_processed_data(df):
    """Visualize the processed data with new features."""
    # Create hour of day column for visualization
    df['HourOfDay'] = df['Timestamp'].dt.hour

    # Plot distributions of new features
    new_features = ['JerkMag', 'AccGyroRatio', 'SVM', 'AccMag_MA', 'GyroMag_STD']

    plt.figure(figsize=(16, 10))
    for i, col in enumerate(new_features):
        plt.subplot(2, 3, i+1)
        sns.kdeplot(data=df, x=col, hue='FallDetected', fill=True, palette={0: "blue", 1:
"red"})
        plt.title(f"{col} Distribution (Fall vs. Non-Fall)")
    plt.tight_layout()
    plt.show()

    # 3D Scatter Plot with Jerk Magnitude
    fig = px.scatter_3d(df, x='AccX', y='AccY', z='AccZ',
                        color='JerkMag', size='JerkMag',
                        hover_data=['FallDetected', 'Timestamp'],
                        title="3D Accelerometer Data Colored by Jerk Magnitude")
    fig.show()

    # Time series of Jerk Magnitude with falls highlighted
    plt.figure(figsize=(14, 6))
    plt.plot(df['Timestamp'], df['JerkMag'], label='Jerk Magnitude', alpha=0.7)
    fall_points = df[df['FallDetected'] == 1]
    plt.scatter(fall_points['Timestamp'], fall_points['JerkMag'],
                color='red', s=50, label='Fall Detected')
    plt.title("Jerk Magnitude Over Time with Detected Falls")
    plt.xlabel("Time")
    plt.ylabel("Jerk Magnitude")
    plt.legend()
    plt.show()

visualize_processed_data(df_processed)
```

```
# --- STEP 7: Save Processed Data ---
df_processed.to_excel('complete_fall_detection_data_with_features.xlsx', index=False)
print("\nProcessing complete. Final data shape:", df_processed.shape)
print("\nColumns in final processed dataset:")
print(df_processed.columns.tolist())
```

### 4.2.3 **Random Forest CLassification**

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import (classification_report, confusion_matrix, roc_curve, auc,
precision_recall_curve,average_precision_score)
from sklearn.metrics import RocCurveDisplay, PrecisionRecallDisplay, roc_auc_score # Import
necessary classes
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
import plotly.graph_objects as go
import plotly.express as px
from plotly.subplots import make_subplots

# Set random seed for reproducibility
np.random.seed(42)

# Load your dataset (replace with your actual data loading)
# df = pd.read_excel('merged_feature_data.xlsx')
# For this example, I'll create a synthetic dataset since we only saw "Fall Not Detected"
samples

# ------------------------------------------------
# Synthetic Data Generation (Replace with your actual data)
# ------------------------------------------------
def generate_synthetic_data():
    # Create balanced synthetic data
    np.random.seed(42)
    n_samples = 1000

    # Features similar to your dataset
    features = {
        'AccMag': np.concatenate([np.random.normal(1.0, 0.2, n_samples//2),
                                  np.random.normal(3.5, 1.0, n_samples//2)]),
        'JerkMag': np.concatenate([np.random.normal(0.5, 0.1, n_samples//2),
                                   np.random.normal(2.5, 0.8, n_samples//2)]),
        'GyroMag': np.concatenate([np.random.normal(0.8, 0.2, n_samples//2),
```

```
                                np.random.normal(3.0, 1.0, n_samples//2)]),
        'Pressure': np.random.normal(101325, 100, n_samples),
        'Altitude': np.random.normal(50, 20, n_samples)
    }

    df = pd.DataFrame(features)
    df['Label'] = np.concatenate([np.zeros(n_samples//2), np.ones(n_samples//2)])
    df['Label'] = df['Label'].map({0: 'Fall Not Detected', 1: 'Fall Detected'})

    return df

df = generate_synthetic_data()
# ------------------------------------------------

# Data Preprocessing
X = df.drop('Label', axis=1)
y = df['Label']

# Split data (stratified to maintain class balance)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=42)

# Check class distribution
print("Class Distribution:")
print(y.value_counts())

# ------------------------------------------------
# 1. Feature Visualization
# ------------------------------------------------
def plot_feature_distributions(df):
    num_features = len(df.columns) - 1  # Exclude label
    n_cols = 3
    n_rows = (num_features + n_cols - 1) // n_cols

    plt.figure(figsize=(15, 5*n_rows))
    for i, col in enumerate(df.columns[:-1]):  # Exclude label
        plt.subplot(n_rows, n_cols, i+1)
        sns.histplot(data=df, x=col, hue='Label', kde=True, element='step')
        plt.title(f'Distribution of {col}')
    plt.tight_layout()
    plt.show()

plot_feature_distributions(df)

# Correlation Matrix
plt.figure(figsize=(10, 8))
corr = df.corr(numeric_only=True)
sns.heatmap(corr, annot=True, cmap='coolwarm', center=0)
plt.title('Feature Correlation Matrix')
plt.show()

# ------------------------------------------------
# 2. Model Pipeline with SMOTE and Feature Selection
```

```
# ------------------------------------------------
# Create preprocessing and modeling pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', SMOTE(random_state=42)),
    ('feature_selection', SelectKBest(score_func=f_classif, k='all')),
    ('classifier', RandomForestClassifier(random_state=42, class_weight='balanced'))
])

# ------------------------------------------------
# 3. Hyperparameter Tuning with GridSearchCV
# ------------------------------------------------
param_grid = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [None, 10, 20],
    'classifier__min_samples_split': [2, 5],
    'classifier__min_samples_leaf': [1, 2],
    'feature_selection__k': [3, 5, 'all']
}

cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
grid_search = GridSearchCV(
    estimator=pipeline,
    param_grid=param_grid,
    cv=cv,
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

grid_search.fit(X_train, y_train)

# Best model
best_model = grid_search.best_estimator_
print("\nBest Parameters:", grid_search.best_params_)

# ------------------------------------------------
# 4. Model Evaluation with Visualizations
# ------------------------------------------------
def evaluate_model(model, X_test, y_test):
    # Predictions
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]  # Probabilities for positive class

    # Classification Report
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred))

    # Confusion Matrix
    cm = confusion_matrix(y_test, y_pred)
    plt.figure(figsize=(6, 6))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=model.classes_, yticklabels=model.classes_)
```

```
plt.title('Confusion Matrix')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# ROC Curve
fpr, tpr, thresholds = roc_curve(y_test, y_proba, pos_label='Fall Detected')
roc_auc = auc(fpr, tpr)

plt.figure(figsize=(8, 6))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic')
plt.legend(loc="lower right")
plt.show()

# Precision-Recall Curve
precision, recall, _ = precision_recall_curve(
    y_test, y_proba, pos_label='Fall Detected')
avg_precision = average_precision_score(
    y_test, y_proba, pos_label='Fall Detected')

plt.figure(figsize=(8, 6))
plt.plot(recall, precision, color='blue', lw=2,
         label=f'Precision-Recall (AP = {avg_precision:.2f})')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curve')
plt.legend(loc="lower left")
plt.show()

# Feature Importance (if using RandomForest)
if hasattr(model.named_steps['classifier'], 'feature_importances_'):
    feature_importances = model.named_steps['classifier'].feature_importances_
    selected_features = model.named_steps['feature_selection'].get_support()
    features = X.columns[selected_features]

    importance_df = pd.DataFrame({
        'Feature': features,
        'Importance': feature_importances
    }).sort_values('Importance', ascending=False)

    plt.figure(figsize=(10, 6))
    sns.barplot(x='Importance', y='Feature', data=importance_df)
    plt.title('Feature Importance')
    plt.tight_layout()
    plt.show()

    # Interactive feature importance plot
```

```
        fig = px.bar(importance_df, x='Importance', y='Feature',
                     orientation='h', title='Feature Importance')
        fig.show()

# Evaluate best model
evaluate_model(best_model, X_test, y_test)

# ------------------------------------------------
# 5. Interactive Visualizations with Plotly
# ------------------------------------------------
def plot_interactive_distributions(df):
    fig = make_subplots(rows=2, cols=3, subplot_titles=df.columns[:-1])

    for i, col in enumerate(df.columns[:-1]):
        row = (i // 3) + 1
        col_num = (i % 3) + 1

        for label in df['Label'].unique():
            fig.add_trace(
                go.Histogram(
                    x=df[df['Label'] == label][col],
                    name=label,
                    opacity=0.75,
                    legendgroup=label,
                    showlegend=(i == 0)  # Only show legend for first plot
                ),
                row=row, col=col_num
            )

    fig.update_layout(
        title_text="Feature Distributions by Class",
        height=800,
        width=1200,
        barmode='overlay'
    )
    fig.show()

plot_interactive_distributions(df)

# Interactive correlation matrix
corr = df.corr(numeric_only=True)
fig = go.Figure(data=go.Heatmap(
    z=corr.values,
    x=corr.columns,
    y=corr.index,
    colorscale='RdBu',
    zmin=-1,
    zmax=1,
    hoverongaps=False
))
fig.update_layout(
    title='Interactive Correlation Matrix',
    xaxis_title="Features",
```

```
    yaxis_title="Features",
    width=800,
    height=800
)
fig.show()

# ------------------------------------------------
# 6. Threshold Optimization Visualization
# ------------------------------------------------
def plot_threshold_analysis(model, X_test, y_test):
    y_proba = model.predict_proba(X_test)[:, 1]

    # Calculate metrics for different thresholds
    thresholds = np.linspace(0, 1, 50)
    fpr_list, tpr_list, precision_list = [], [], []

    for thresh in thresholds:
        y_pred = (y_proba >= thresh).astype(int)
        y_pred_labels = model.classes_[y_pred]

        # Calculate confusion matrix
        cm = confusion_matrix(y_test, y_pred_labels, labels=model.classes_)
        tn, fp, fn, tp = cm.ravel()

        fpr = fp / (fp + tn)
        tpr = tp / (tp + fn)
        precision = tp / (tp + fp) if (tp + fp) > 0 else 0

        fpr_list.append(fpr)
        tpr_list.append(tpr)
        precision_list.append(precision)

    # Create plot
    fig = go.Figure()

    # Add traces
    fig.add_trace(go.Scatter(
        x=thresholds, y=tpr_list,
        mode='lines',
        name='True Positive Rate (Recall)',
        line=dict(color='green')
    ))

    fig.add_trace(go.Scatter(
        x=thresholds, y=precision_list,
        mode='lines',
        name='Precision',
        line=dict(color='blue')
    ))

    fig.add_trace(go.Scatter(
        x=thresholds, y=fpr_list,
        mode='lines',
```

```
        name='False Positive Rate',
        line=dict(color='red')
    ))

    # Add optimal threshold (maximizing F1 score)
    f1_scores = 2 * (np.array(precision_list) * np.array(tpr_list)) / (
        np.array(precision_list) + np.array(tpr_list) + 1e-9)
    optimal_idx = np.argmax(f1_scores)
    optimal_threshold = thresholds[optimal_idx]

    fig.add_vline(
        x=optimal_threshold,
        line_dash="dash",
        annotation_text=f"Optimal Threshold: {optimal_threshold:.2f}",
        annotation_position="top right"
    )

    fig.update_layout(
        title='Threshold Optimization Analysis',
        xaxis_title='Threshold',
        yaxis_title='Metric Value',
        hovermode='x unified',
        width=900,
        height=500
    )

    fig.show()

plot_threshold_analysis(best_model, X_test, y_test)

# Import additional libraries needed for Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score

# ------------------------------------------------
# Random Forest Pipeline
# ------------------------------------------------
# Create pipeline with SMOTE for handling imbalance
rf_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', SMOTE(random_state=42)),
    ('feature_selection', SelectKBest(score_func=f_classif)),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Hyperparameter grid for Random Forest
rf_param_grid = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [None, 5, 10],
    'classifier__min_samples_split': [2, 5],
    'classifier__min_samples_leaf': [1, 2],
    'classifier__max_features': ['sqrt', 'log2'],
    'feature_selection__k': [3, 'all']
```

```
}

# Create grid search for Random Forest
rf_grid_search = GridSearchCV(
    rf_pipeline,
    rf_param_grid,
    cv=StratifiedKFold(n_splits=3, shuffle=True, random_state=42),
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

# Convert y_train to binary (0 and 1)
y_train_bin = y_train.map({'Fall Not Detected': 0, 'Fall Detected': 1})

# Convert y_test to binary (0 and 1) for evaluation
y_test_bin = y_test.map({'Fall Not Detected': 0, 'Fall Detected': 1})

# ... (rest of the code) ...
print("\nStarting Random Forest Grid Search...")
rf_grid_search.fit(X_train, y_train_bin)

# Get best Random Forest model
rf_best_model = rf_grid_search.best_estimator_
print("\nBest Random Forest Parameters:", rf_grid_search.best_params_)

# ------------------------------------------------
# Random Forest Evaluation
# ------------------------------------------------
print("\nEvaluating Best Random Forest Model...")
def evaluate_model(model, X_test, y_test, y_test_bin=None): # add y_test_bin=None
    # Predictions
    y_pred = model.predict(X_test)

    # Use y_test_bin for calculations if provided
    if y_test_bin is not None:
        # For metrics that require binary labels (e.g., accuracy, ROC AUC)
        y_true = y_test_bin
    else:
        y_true = y_test

    y_proba = model.predict_proba(X_test)[:, 1]  # Probabilities for positive class

    # Classification Report
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred)) # use y_true here

    # ... (rest of the function, replace y_test with y_true where needed) ...

    # ROC Curve
    fpr, tpr, thresholds = roc_curve(y_true, y_proba, pos_label='Fall Detected') # use y_true
    # ...
```

```
# ------------------------------------------------
# Feature Importance for Random Forest
# ------------------------------------------------
def plot_rf_feature_importance(model, feature_names):
    """Plot feature importance for Random Forest"""
    if hasattr(model.named_steps['classifier'], 'feature_importances_'):
        importances = model.named_steps['classifier'].feature_importances_
        features = feature_names[model.named_steps['feature_selection'].get_support()]

        importance_df = pd.DataFrame({'Feature': features, 'Importance': importances})
        importance_df = importance_df.sort_values('Importance', ascending=False)

        plt.figure(figsize=(8, 4))
        sns.barplot(x='Importance', y='Feature', data=importance_df)
        plt.title('Random Forest Feature Importance')
        plt.show()

print("\nRandom Forest Feature Importance:")
plot_rf_feature_importance(rf_best_model, X.columns)

# ------------------------------------------------
# Final Random Forest Summary
# ------------------------------------------------
print("\n=== RANDOM FOREST TRAINING COMPLETE ===")
print(f"Best F1 Score: {rf_grid_search.best_score_:.4f}")
print("Best Parameters:")
for param, value in rf_grid_search.best_params_.items():
    print(f"- {param}: {value}")

# Compare with XGBoost
print("\n=== MODEL COMPARISON ===")
print(f"XGBoost Best F1: {grid_search.best_score_:.4f}")
print(f"Random Forest Best F1: {rf_grid_search.best_score_:.4f}")
# ... (previous code) ...

# Get test set predictions for both models
y_pred_xgb = best_model.predict(X_test)

# Convert XGBoost predictions to binary (0 and 1)
y_pred_xgb_bin = pd.Series(y_pred_xgb).map({'Fall Not Detected': 0, 'Fall Detected':
1}).values

y_pred_rf = rf_best_model.predict(X_test)

print("\nTest Set Performance:")
# Use the binary predictions for XGBoost
print(f"XGBoost Accuracy: {accuracy_score(y_test_bin, y_pred_xgb_bin):.4f}")
print(f"Random Forest Accuracy: {accuracy_score(y_test_bin, y_pred_rf):.4f}")
print(f"XGBoost F1: {f1_score(y_test_bin, y_pred_xgb_bin):.4f}")
print(f"Random Forest F1: {f1_score(y_test_bin, y_pred_rf):.4f}")

# Set style for all plots
plt.style.use('seaborn-v0_8-whitegrid') # Changed style name to 'seaborn-v0_8-whitegrid'
```

```
plt.rcParams['figure.figsize'] = (12, 6)

# 1. Confusion Matrix with Annotations
plt.figure(figsize=(6,6))
rf_cm = confusion_matrix(y_test_bin, rf_best_model.predict(X_test))
sns.heatmap(rf_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Fall', 'Fall'],
            yticklabels=['No Fall', 'Fall'])
plt.title('Random Forest - Confusion Matrix', fontsize=14, pad=20)
plt.xlabel('Predicted Label', fontsize=12)
plt.ylabel('True Label', fontsize=12)
plt.show()

# 2. ROC and Precision-Recall Curves Side by Side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# ROC Curve
RocCurveDisplay.from_estimator(rf_best_model, X_test, y_test_bin, ax=ax1, name='Random
Forest')
ax1.plot([0, 1], [0, 1], 'k--')
ax1.set_title('ROC Curve', fontsize=14)
ax1.grid(True)

# Precision-Recall Curve
PrecisionRecallDisplay.from_estimator(rf_best_model, X_test, y_test_bin, ax=ax2, name='Random
Forest')
ax2.set_title('Precision-Recall Curve', fontsize=14)
ax2.grid(True)

plt.tight_layout()
plt.show()

# 3. Feature Importance Horizontal Bar Plot
rf_feature_importances = pd.DataFrame({
    'Feature': X.columns[rf_best_model.named_steps['feature_selection'].get_support()],
    'Importance': rf_best_model.named_steps['classifier'].feature_importances_
}).sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=rf_feature_importances, palette='viridis')
plt.title('Random Forest - Feature Importance', fontsize=14, pad=20)
plt.xlabel('Importance Score', fontsize=12)
plt.ylabel('')
plt.grid(axis='x', alpha=0.3)
plt.show()

# 4. Class Prediction Distribution
rf_probs = rf_best_model.predict_proba(X_test)[:, 1]
plt.figure(figsize=(10, 6))
sns.histplot(x=rf_probs, hue=y_test_bin, bins=30, kde=True,
             palette={0:'skyblue', 1:'coral'}, alpha=0.6)
plt.title('Random Forest - Prediction Probability Distribution', fontsize=14, pad=20)
plt.xlabel('Predicted Probability of Fall', fontsize=12)
```

```
plt.ylabel('Count', fontsize=12)
plt.legend(title='Actual Class', labels=['No Fall', 'Fall'])
plt.grid(alpha=0.3)
plt.show()

# 5. Performance Metrics Comparison (Table)
from tabulate import tabulate

rf_metrics = classification_report(y_test_bin, rf_best_model.predict(X_test),
output_dict=True)
metrics_data = [
    ["Accuracy", rf_metrics['accuracy']],
    ["Precision (Fall)", rf_metrics['1']['precision']],
    ["Recall (Fall)", rf_metrics['1']['recall']],
    ["F1-Score (Fall)", rf_metrics['1']['f1-score']],
    ["ROC AUC", roc_auc_score(y_test_bin, rf_probs)]
]

print("\n\033[1mRandom Forest Performance Metrics:\033[0m")
print(tabulate(metrics_data, headers=["Metric", "Score"], floatfmt=".4f", tablefmt="grid"))

# 6. Decision Threshold Analysis (Interactive)
from ipywidgets import interact, FloatSlider

@interact(threshold=FloatSlider(min=0, max=1, step=0.05, value=0.5, description='Threshold:'))
def threshold_analysis(threshold):
    y_pred = (rf_probs >= threshold).astype(int)
    cm = confusion_matrix(y_test_bin, y_pred)

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=['No Fall', 'Fall'],
                yticklabels=['No Fall', 'Fall'])
    plt.title(f'Confusion Matrix @ {threshold:.2f}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')

    plt.subplot(1, 2, 2)
    report = classification_report(y_test_bin, y_pred, output_dict=True)
    metrics = ['precision', 'recall', 'f1-score']
    values = [report['1']['precision'], report['1']['recall'], report['1']['f1-score']]
    sns.barplot(x=metrics, y=values, palette='rocket')
    plt.ylim(0, 1)
    plt.title('Performance Metrics')

    plt.tight_layout()
    plt.show()

  # Import additional libraries needed for Random Forest
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, f1_score
```

```
# ------------------------------------------------
# Random Forest Pipeline
# ------------------------------------------------
# Create pipeline with SMOTE for handling imbalance
rf_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', SMOTE(random_state=42)),
    ('feature_selection', SelectKBest(score_func=f_classif)),
    ('classifier', RandomForestClassifier(random_state=42))
])

# Hyperparameter grid for Random Forest
rf_param_grid = {
    'classifier__n_estimators': [100, 200],
    'classifier__max_depth': [None, 5, 10],
    'classifier__min_samples_split': [2, 5],
    'classifier__min_samples_leaf': [1, 2],
    'classifier__max_features': ['sqrt', 'log2'],
    'feature_selection__k': [3, 'all']
}

# Create grid search for Random Forest
rf_grid_search = GridSearchCV(
    rf_pipeline,
    rf_param_grid,
    cv=StratifiedKFold(n_splits=3, shuffle=True, random_state=42),
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

# Convert y_train to binary (0 and 1)
y_train_bin = y_train.map({'Fall Not Detected': 0, 'Fall Detected': 1})

# Convert y_test to binary (0 and 1) for evaluation
y_test_bin = y_test.map({'Fall Not Detected': 0, 'Fall Detected': 1})

# ... (rest of the code) ...
print("\nStarting Random Forest Grid Search...")
rf_grid_search.fit(X_train, y_train_bin)

# Get best Random Forest model
rf_best_model = rf_grid_search.best_estimator_
print("\nBest Random Forest Parameters:", rf_grid_search.best_params_)

# ------------------------------------------------
# Random Forest Evaluation
# ------------------------------------------------
print("\nEvaluating Best Random Forest Model...")
def evaluate_model(model, X_test, y_test, y_test_bin=None): # add y_test_bin=None
    # Predictions
    y_pred = model.predict(X_test)
```

```
    # Use y_test_bin for calculations if provided
    if y_test_bin is not None:
        # For metrics that require binary labels (e.g., accuracy, ROC AUC)
        y_true = y_test_bin
    else:
        y_true = y_test

    y_proba = model.predict_proba(X_test)[:, 1]  # Probabilities for positive class

    # Classification Report
    print("\nClassification Report:")
    print(classification_report(y_true, y_pred)) # use y_true here

    # ... (rest of the function, replace y_test with y_true where needed) ...

    # ROC Curve
    fpr, tpr, thresholds = roc_curve(y_true, y_proba, pos_label='Fall Detected') # use y_true
    # ...

# -------------------------------------------------
# Feature Importance for Random Forest
# -------------------------------------------------
def plot_rf_feature_importance(model, feature_names):
    """Plot feature importance for Random Forest"""
    if hasattr(model.named_steps['classifier'], 'feature_importances_'):
        importances = model.named_steps['classifier'].feature_importances_
        features = feature_names[model.named_steps['feature_selection'].get_support()]

        importance_df = pd.DataFrame({'Feature': features, 'Importance': importances})
        importance_df = importance_df.sort_values('Importance', ascending=False)

        plt.figure(figsize=(8, 4))
        sns.barplot(x='Importance', y='Feature', data=importance_df)
        plt.title('Random Forest Feature Importance')
        plt.show()

print("\nRandom Forest Feature Importance:")
plot_rf_feature_importance(rf_best_model, X.columns)

# -------------------------------------------------
# Final Random Forest Summary
# -------------------------------------------------
print("\n=== RANDOM FOREST TRAINING COMPLETE ===")
print(f"Best F1 Score: {rf_grid_search.best_score_:.4f}")
print("Best Parameters:")
for param, value in rf_grid_search.best_params_.items():
    print(f"- {param}: {value}")

# Compare with XGBoost
print("\n=== MODEL COMPARISON ===")
print(f"XGBoost Best F1: {grid_search.best_score_:.4f}")
print(f"Random Forest Best F1: {rf_grid_search.best_score_:.4f}")
```

```
# ... (previous code) ...

# Get test set predictions for both models
y_pred_xgb = best_model.predict(X_test)

# Convert XGBoost predictions to binary (0 and 1)
y_pred_xgb_bin = pd.Series(y_pred_xgb).map({'Fall Not Detected': 0, 'Fall Detected':
1}).values

y_pred_rf = rf_best_model.predict(X_test)

print("\nTest Set Performance:")
# Use the binary predictions for XGBoost
print(f"XGBoost Accuracy: {accuracy_score(y_test_bin, y_pred_xgb_bin):.4f}")
print(f"Random Forest Accuracy: {accuracy_score(y_test_bin, y_pred_rf):.4f}")
print(f"XGBoost F1: {f1_score(y_test_bin, y_pred_xgb_bin):.4f}")
print(f"Random Forest F1: {f1_score(y_test_bin, y_pred_rf):.4f}")

# Set style for all plots
plt.style.use('seaborn-v0_8-whitegrid') # Changed style name to 'seaborn-v0_8-whitegrid'
plt.rcParams['figure.figsize'] = (12, 6)

# 1. Confusion Matrix with Annotations
plt.figure(figsize=(6,6))
rf_cm = confusion_matrix(y_test_bin, rf_best_model.predict(X_test))
sns.heatmap(rf_cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=['No Fall', 'Fall'],
            yticklabels=['No Fall', 'Fall'])
plt.title('Random Forest - Confusion Matrix', fontsize=14, pad=20)
plt.xlabel('Predicted Label', fontsize=12)
plt.ylabel('True Label', fontsize=12)
plt.show()

# 2. ROC and Precision-Recall Curves Side by Side
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 6))

# ROC Curve
RocCurveDisplay.from_estimator(rf_best_model, X_test, y_test_bin, ax=ax1, name='Random
Forest')
ax1.plot([0, 1], [0, 1], 'k--')
ax1.set_title('ROC Curve', fontsize=14)
ax1.grid(True)

# Precision-Recall Curve
PrecisionRecallDisplay.from_estimator(rf_best_model, X_test, y_test_bin, ax=ax2, name='Random
Forest')
ax2.set_title('Precision-Recall Curve', fontsize=14)
ax2.grid(True)

plt.tight_layout()
plt.show()

# 3. Feature Importance Horizontal Bar Plot
```

```
rf_feature_importances = pd.DataFrame({
    'Feature': X.columns[rf_best_model.named_steps['feature_selection'].get_support()],
    'Importance': rf_best_model.named_steps['classifier'].feature_importances_
}).sort_values('Importance', ascending=False)

plt.figure(figsize=(10, 6))
sns.barplot(x='Importance', y='Feature', data=rf_feature_importances, palette='viridis')
plt.title('Random Forest - Feature Importance', fontsize=14, pad=20)
plt.xlabel('Importance Score', fontsize=12)
plt.ylabel('')
plt.grid(axis='x', alpha=0.3)
plt.show()

# 4. Class Prediction Distribution
rf_probs = rf_best_model.predict_proba(X_test)[:, 1]
plt.figure(figsize=(10, 6))
sns.histplot(x=rf_probs, hue=y_test_bin, bins=30, kde=True,
             palette={0:'skyblue', 1:'coral'}, alpha=0.6)
plt.title('Random Forest - Prediction Probability Distribution', fontsize=14, pad=20)
plt.xlabel('Predicted Probability of Fall', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.legend(title='Actual Class', labels=['No Fall', 'Fall'])
plt.grid(alpha=0.3)
plt.show()

# 5. Performance Metrics Comparison (Table)
from tabulate import tabulate

rf_metrics = classification_report(y_test_bin, rf_best_model.predict(X_test),
output_dict=True)
metrics_data = [
    ["Accuracy", rf_metrics['accuracy']],
    ["Precision (Fall)", rf_metrics['1']['precision']],
    ["Recall (Fall)", rf_metrics['1']['recall']],
    ["F1-Score (Fall)", rf_metrics['1']['f1-score']],
    ["ROC AUC", roc_auc_score(y_test_bin, rf_probs)]
]

print("\n\033[1mRandom Forest Performance Metrics:\033[0m")
print(tabulate(metrics_data, headers=["Metric", "Score"], floatfmt=".4f", tablefmt="grid"))

# 6. Decision Threshold Analysis (Interactive)
from ipywidgets import interact, FloatSlider

@interact(threshold=FloatSlider(min=0, max=1, step=0.05, value=0.5, description='Threshold:'))
def threshold_analysis(threshold):
    y_pred = (rf_probs >= threshold).astype(int)
    cm = confusion_matrix(y_test_bin, y_pred)

    plt.figure(figsize=(12, 4))

    plt.subplot(1, 2, 1)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
```

```
                xticklabels=['No Fall', 'Fall'],
                yticklabels=['No Fall', 'Fall'])
    plt.title(f'Confusion Matrix @ {threshold:.2f}')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')

    plt.subplot(1, 2, 2)
    report = classification_report(y_test_bin, y_pred, output_dict=True)
    metrics = ['precision', 'recall', 'f1-score']
    values = [report['1']['precision'], report['1']['recall'], report['1']['f1-score']]
    sns.barplot(x=metrics, y=values, palette='rocket')
    plt.ylim(0, 1)
    plt.title('Performance Metrics')

    plt.tight_layout()
    plt.show()
```

### 4.2.4 **XGB Boost**

```
# Import necessary libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, StratifiedKFold
from xgboost import XGBClassifier
from sklearn.metrics import (classification_report, confusion_matrix,
                             roc_curve, auc, precision_recall_curve,
                             average_precision_score, f1_score) # Import f1_score
from imblearn.over_sampling import SMOTE
from imblearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import SelectKBest, f_classif
import warnings
warnings.filterwarnings('ignore')
!pip install plotly
import plotly.graph_objs as go

# Set random seed for reproducibility
np.random.seed(42)

# -------------------------------------------------
# Synthetic Data Generation
# -------------------------------------------------
def generate_synthetic_data():
    """Generate balanced synthetic fall detection data"""
    np.random.seed(42)
    n_samples = 1000  # Reduced from original for faster execution

    # Features similar to real fall detection data
    features = {
        'AccMag': np.concatenate([
```

```
            np.random.normal(1.0, 0.2, n_samples//2),  # Normal movement
            np.random.normal(3.5, 1.0, n_samples//2)    # Fall movement
        ]),
        'JerkMag': np.concatenate([
            np.random.normal(0.5, 0.1, n_samples//2),  # Normal
            np.random.normal(2.5, 0.8, n_samples//2)   # Fall
        ]),
        'GyroMag': np.concatenate([
            np.random.normal(0.8, 0.2, n_samples//2),  # Normal
            np.random.normal(3.0, 1.0, n_samples//2)   # Fall
        ]),
        'Pressure': np.random.normal(101325, 100, n_samples),
        'Altitude': np.random.normal(50, 20, n_samples)
    }

    df = pd.DataFrame(features)
    df['Label'] = np.concatenate([np.zeros(n_samples//2), np.ones(n_samples//2)])
    df['Label'] = df['Label'].map({0: 'Fall Not Detected', 1: 'Fall Detected'})
    return df

# Generate and show data sample
df = generate_synthetic_data()
print("\nSample of Generated Data:")
print(df.head())

# ------------------------------------------------
# Data Preprocessing
# ------------------------------------------------
X = df.drop('Label', axis=1)
y = df['Label']
y_binary = y.map({'Fall Not Detected': 0, 'Fall Detected': 1})

# Split data (stratified to maintain class balance)
X_train, X_test, y_train, y_test, y_train_bin, y_test_bin = train_test_split(
    X, y, y_binary, test_size=0.3, stratify=y_binary, random_state=42)

# Check class distribution
print("\nClass Distribution:")
print(y.value_counts())

# ------------------------------------------------
# Quick Data Visualization
# ------------------------------------------------
def quick_visualizations(df):
    """Generate essential visualizations quickly"""
    plt.figure(figsize=(15, 10))

    # Feature distributions
    plt.subplot(2, 2, 1)
    sns.boxplot(x='Label', y='AccMag', data=df)
    plt.title('Acceleration Magnitude by Class')

    plt.subplot(2, 2, 2)
```

```
    sns.boxplot(x='Label', y='JerkMag', data=df)
    plt.title('Jerk Magnitude by Class')

    plt.subplot(2, 2, 3)
    sns.boxplot(x='Label', y='GyroMag', data=df)
    plt.title('Gyroscope Magnitude by Class')

    # Correlation matrix
    plt.subplot(2, 2, 4)
    sns.heatmap(df.corr(numeric_only=True), annot=True, cmap='coolwarm', center=0)
    plt.title('Feature Correlation Matrix')

    plt.tight_layout()
    plt.show()

quick_visualizations(df)

# ------------------------------------------------
# Optimized XGBoost Pipeline
# ------------------------------------------------
# Create pipeline with SMOTE for handling imbalance
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('smote', SMOTE(random_state=42)),
    ('feature_selection', SelectKBest(score_func=f_classif)),
    ('classifier', XGBClassifier(
        random_state=42,
        eval_metric='logloss',
        n_estimators=100
    ))
])

# Focused hyperparameter grid
param_grid = {
    'classifier__max_depth': [3, 5],
    'classifier__learning_rate': [0.1, 0.2],
    'classifier__subsample': [0.8, 1.0],
    'classifier__early_stopping_rounds': [10],
    'feature_selection__k': [3, 'all']
}

# Create grid search
grid_search = GridSearchCV(
    pipeline,
    param_grid,
    cv=StratifiedKFold(n_splits=3, shuffle=True, random_state=42),
    scoring='f1',
    n_jobs=-1,
    verbose=1
)
# Prepare the eval_set by transforming the test data through the pipeline steps
# Remove this entire block:
# X_test_transformed = X_test.copy()
```

```
# for step_name, step in pipeline.steps[:-1]:
#     if hasattr(step, 'transform'):
#         X_test_transformed = step.transform(X_test_transformed)
#     elif hasattr(step, 'fit_resample'):  # Skip SMOTE for test data
#         continue

# eval_set = [(X_test_transformed, y_test_bin)]

# Replace with:
eval_set = [(X_test, y_test_bin)] # Pass the original X_test and y_test_bin
# Remove early_stopping_rounds from param_grid
param_grid = {
    'classifier__max_depth': [3, 5],
    'classifier__learning_rate': [0.1, 0.2],
    'classifier__subsample': [0.8, 1.0],
    'feature_selection__k': [3, 'all']
}

# Create grid search
grid_search = GridSearchCV(
    pipeline,
    param_grid,
    cv=StratifiedKFold(n_splits=3, shuffle=True, random_state=42),
    scoring='f1',
    n_jobs=-1,
    verbose=1
)

print("\nStarting Grid Search...")
grid_search.fit(X_train, y_train_bin)


# Get best model
best_model = grid_search.best_estimator_
print("\nBest Parameters:", grid_search.best_params_)

# ------------------------------------------------
# Essential Model Evaluation
# ------------------------------------------------
def evaluate_model(model, X_test, y_test, y_test_bin):
    """Perform essential model evaluation with visualizations"""
    # Predictions
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test)[:, 1]
    y_pred_labels = pd.Series(y_pred).map({0: 'Fall Not Detected', 1: 'Fall Detected'})

    # Classification Report
    print("\nClassification Report:")
    print(classification_report(y_test, y_pred_labels))

    # Confusion Matrix
    plt.figure(figsize=(6, 6))
    sns.heatmap(confusion_matrix(y_test, y_pred_labels),
```

```
                annot=True, fmt='d', cmap='Blues',
                xticklabels=['No Fall', 'Fall'],
                yticklabels=['No Fall', 'Fall'])
    plt.title('Confusion Matrix')
    plt.ylabel('Actual')
    plt.xlabel('Predicted')
    plt.show()

    # ROC Curve
    fpr, tpr, _ = roc_curve(y_test_bin, y_proba)
    roc_auc = auc(fpr, tpr)

    plt.figure(figsize=(6, 6))
    plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC (AUC = {roc_auc:.2f})')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve')
    plt.legend(loc="lower right")
    plt.show()

    # Feature Importance
    if hasattr(model.named_steps['classifier'], 'feature_importances_'):
        importances = model.named_steps['classifier'].feature_importances_
        features = X.columns[model.named_steps['feature_selection'].get_support()]

        importance_df = pd.DataFrame({'Feature': features, 'Importance': importances})
        importance_df = importance_df.sort_values('Importance', ascending=False)

        plt.figure(figsize=(8, 4))
        sns.barplot(x='Importance', y='Feature', data=importance_df)
        plt.title('XGBoost Feature Importance')
        plt.show()

print("\nEvaluating Best Model...")
evaluate_model(best_model, X_test, y_test, y_test_bin)

# ------------------------------------------------
# Threshold Analysis
# ------------------------------------------------
def threshold_analysis(model, X_test, y_test_bin):
    """Analyze performance across different decision thresholds"""
    y_proba = model.predict_proba(X_test)[:, 1]

    thresholds = np.linspace(0, 1, 20)
    metrics = {
        'f1': [],
        'precision': [],
        'recall': []
    }
```

```
    for thresh in thresholds:
        y_pred = (y_proba >= thresh).astype(int)
        report = classification_report(y_test_bin, y_pred, output_dict=True)
        metrics['f1'].append(report['1']['f1-score'])
        metrics['precision'].append(report['1']['precision'])
        metrics['recall'].append(report['1']['recall'])

    plt.figure(figsize=(10, 5))
    plt.plot(thresholds, metrics['f1'], label='F1 Score')
    plt.plot(thresholds, metrics['precision'], label='Precision')
    plt.plot(thresholds, metrics['recall'], label='Recall')

    # Mark default 0.5 threshold
    plt.axvline(x=0.5, color='gray', linestyle='--', label='Default Threshold (0.5)')

    # Find and mark optimal threshold (max F1)
    optimal_idx = np.argmax(metrics['f1'])
    optimal_thresh = thresholds[optimal_idx]
    plt.axvline(x=optimal_thresh, color='red', linestyle=':',
                label=f'Optimal Threshold ({optimal_thresh:.2f})')

    plt.title('Performance Metrics vs. Decision Threshold')
    plt.xlabel('Threshold')
    plt.ylabel('Score')
    plt.legend()
    plt.grid()
    plt.show()

print("\nPerforming Threshold Analysis...")
threshold_analysis(best_model, X_test, y_test_bin)

# ------------------------------------------------
# Final Model Summary
# ------------------------------------------------
print("\n=== MODEL TRAINING COMPLETE ===")
print(f"Best F1 Score: {grid_search.best_score_:.4f}")
print("Best Parameters:")
for param, value in grid_search.best_params_.items():
    print(f"- {param}: {value}")

    # ------------------------------------------------
# XGBoost Predictive Probability and Threshold Analysis
# ------------------------------------------------

# 1. Get predicted probabilities for the positive class ('Fall Detected')
y_proba = best_model.predict_proba(X_test)[:, 1]

# 2. Probability Distribution Plot
plt.figure(figsize=(12, 6))
sns.histplot(x=y_proba, hue=y_test, bins=30, kde=True,
             palette={'Fall Not Detected':'skyblue', 'Fall Detected':'coral'},
             alpha=0.6, element='step')
plt.title('XGBoost - Predicted Probability Distribution', fontsize=14)
```

```
plt.xlabel('Predicted Probability of Fall', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.axvline(x=0.5, color='gray', linestyle='--', label='Default Threshold (0.5)')
plt.legend(title='Actual Class')
plt.grid(alpha=0.3)
plt.show()
# 3. Threshold Optimization Analysis
thresholds = np.linspace(0, 1, 50)
metrics = {
    'f1': [],
    'precision': [],
    'recall': [],
    'fpr': []  # False Positive Rate
}

for thresh in thresholds:
    y_pred = (y_proba >= thresh).astype(int)
    # Map predicted labels to original labels using the dictionary
    y_pred_labels = pd.Series(y_pred).map({0: 'Fall Not Detected', 1: 'Fall Detected'})

    # Calculate metrics
    report = classification_report(y_test, y_pred_labels, output_dict=True)
    cm = confusion_matrix(y_test, y_pred_labels)
    tn, fp, fn, tp = cm.ravel()

    metrics['f1'].append(report['Fall Detected']['f1-score'])
    metrics['precision'].append(report['Fall Detected']['precision'])
    metrics['recall'].append(report['Fall Detected']['recall'])
    metrics['fpr'].append(fp / (fp + tn))

# Find optimal threshold (max F1 score)
optimal_idx = np.argmax(metrics['f1'])
optimal_thresh = thresholds[optimal_idx]

# 4. Interactive Threshold Analysis Plot (Plotly)
fig = go.Figure()

# Add metrics traces
fig.add_trace(go.Scatter(
    x=thresholds, y=metrics['f1'],
    mode='lines',
    name='F1 Score',
    line=dict(color='green', width=2)
))

fig.add_trace(go.Scatter(
    x=thresholds, y=metrics['precision'],
    mode='lines',
    name='Precision',
    line=dict(color='blue', width=2)
))

fig.add_trace(go.Scatter(
```

```
    x=thresholds, y=metrics['recall'],
    mode='lines',
    name='Recall',
    line=dict(color='red', width=2)
))

fig.add_trace(go.Scatter(
    x=thresholds, y=metrics['fpr'],
    mode='lines',
    name='False Positive Rate',
    line=dict(color='purple', width=2)
))

# Add optimal threshold line
fig.add_vline(
    x=optimal_thresh,
    line_dash="dash",
    line_color="black",
    annotation_text=f"Optimal Threshold: {optimal_thresh:.2f}",
    annotation_position="top right"
)

# Add default threshold line
fig.add_vline(
    x=0.5,
    line_dash="dot",
    line_color="gray",
    annotation_text="Default 0.5",
    annotation_position="bottom right"
)

fig.update_layout(
    title='XGBoost Threshold Optimization Analysis',
    xaxis_title='Decision Threshold',
    yaxis_title='Score/Rate',
    hovermode='x unified',
    width=900,
    height=600,
    legend=dict(orientation='h', yanchor='bottom', y=1.02, xanchor='right', x=1)
)

fig.show()

# 5. Metrics at Optimal Threshold
y_pred_optimal = (y_proba >= optimal_thresh).astype(int)
# Map predicted labels back to original labels (strings)
y_pred_labels_optimal = pd.Series(y_pred_optimal).map({0: 'Fall Not Detected', 1: 'Fall
Detected'})

print(f"\n\033[1mPerformance at Optimal Threshold ({optimal_thresh:.2f}):\033[0m")
print(classification_report(y_test, y_pred_labels_optimal))

# 6. Confusion Matrix at Optimal Threshold
```

```
plt.figure(figsize=(6, 6))
cm_optimal = confusion_matrix(y_test, y_pred_labels_optimal)
sns.heatmap(cm_optimal, annot=True, fmt='d', cmap='Blues',
            xticklabels=best_model.classes_,
            yticklabels=best_model.classes_)
plt.title(f'Confusion Matrix @ Threshold={optimal_thresh:.2f}')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()
# 6. Confusion Matrix at Optimal Threshold
plt.figure(figsize=(6, 6))
cm_optimal = confusion_matrix(y_test, y_pred_labels_optimal)
sns.heatmap(cm_optimal, annot=True, fmt='d', cmap='Blues',
            xticklabels=best_model.classes_,
            yticklabels=best_model.classes_)
plt.title(f'Confusion Matrix @ Threshold={optimal_thresh:.2f}')
plt.ylabel('Actual')
plt.xlabel('Predicted')
plt.show()

# 7. Probability Calibration Plot (Reliability Diagram)
from sklearn.calibration import calibration_curve

prob_true, prob_pred = calibration_curve(y_test_bin, y_proba, n_bins=10)

plt.figure(figsize=(8, 6))
plt.plot(prob_pred, prob_true, marker='o', label='XGBoost')
plt.plot([0, 1], [0, 1], linestyle='--', color='gray', label='Perfectly calibrated')
plt.title('Probability Calibration Curve', fontsize=14)
plt.xlabel('Mean Predicted Probability')
plt.ylabel('Fraction of Positives')
plt.legend()
plt.grid(alpha=0.3)
plt.show()

# ------------------------------------------------
# XGBoost Probability and Threshold Analysis (Essential Only)
# ------------------------------------------------

# 1. Get predicted probabilities
y_proba = best_model.predict_proba(X_test)[:, 1]  # Probabilities for "Fall Detected"

# 2. Probability Distribution
plt.figure(figsize=(10,5))
sns.kdeplot(x=y_proba, hue=y_test, fill=True, palette={'Fall Not Detected':'blue', 'Fall
Detected':'red'})
plt.title('Probability Distribution by True Class')
plt.xlabel('Predicted Probability of Fall')
plt.axvline(0.5, color='black', linestyle='--')
plt.show()

# 3. Find Optimal Threshold
thresholds = np.linspace(0, 1, 100)
```

```
# Convert y_proba to predicted labels before comparing to threshold:
y_pred_bin = best_model.predict(X_test) # Get binary predictions (0 or 1)
f1_scores = [f1_score(y_test_bin, y_pred_bin >= t, pos_label=1) for t in thresholds]
optimal_threshold = thresholds[np.argmax(f1_scores)]

# ... (rest of the code remains the same)
# 4. Threshold Analysis Plot
plt.figure(figsize=(10,5))
plt.plot(thresholds, f1_scores, label='F1 Score')
plt.axvline(optimal_threshold, color='red', linestyle=':',
           label=f'Optimal Threshold ({optimal_threshold:.2f})')
plt.title('F1 Score by Decision Threshold')
plt.xlabel('Threshold')
plt.ylabel('F1 Score')
plt.legend()
plt.show()

# 5. Metrics at Optimal Threshold
print(f"\nOptimal Threshold: {optimal_threshold:.2f}")
y_pred_optimal = (y_proba >= optimal_threshold).astype(int)  # Convert to 0/1
y_pred_labels_optimal = pd.Series(y_pred_optimal).map({0: 'Fall Not Detected', 1: 'Fall
Detected'})  # Map to original labels
print(classification_report(y_test, y_pred_labels_optimal,
                           target_names=['Fall Not Detected', 'Fall Detected']))
```

### 4.2.5 **Predictive Capabilities**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split, GridSearchCV, cross_val_score
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.multioutput import MultiOutputClassifier
from xgboost import XGBClassifier, XGBRegressor
from sklearn.metrics import (classification_report, confusion_matrix,
                             roc_curve, auc, mean_squared_error,
                             r2_score, accuracy_score, precision_recall_curve,
                             average_precision_score)
from imblearn.over_sampling import SMOTE
from sklearn.preprocessing import LabelEncoder, StandardScaler
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import ipywidgets as widgets
from IPython.display import display

def generate_fall_data(n_samples=5000):
    np.random.seed(42)

    # Base features
```

```
    data = {
        'AccX': np.concatenate([np.random.normal(0.1, 0.05, n_samples//2),
                                np.random.normal(2.5, 1.2, n_samples//2)]),
        'AccY': np.concatenate([np.random.normal(0.2, 0.1, n_samples//2),
                                np.random.normal(-1.8, 0.9, n_samples//2)]),
        'AccZ': np.concatenate([np.random.normal(9.8, 0.5, n_samples//2),
                                np.random.normal(3.2, 2.5, n_samples//2)]),
        'GyroX': np.concatenate([np.random.normal(0, 0.1, n_samples//2),
                                 np.random.normal(1.5, 0.8, n_samples//2)]),
        'GyroY': np.concatenate([np.random.normal(0, 0.1, n_samples//2),
                                 np.random.normal(-2.0, 1.2, n_samples//2)]),
        'GyroZ': np.concatenate([np.random.normal(0, 0.1, n_samples//2),
                                 np.random.normal(0.5, 0.3, n_samples//2)]),
        'Pressure': np.random.normal(101325, 100, n_samples)
    }

    df = pd.DataFrame(data)
    # Derived features
    df['AccMag'] = np.sqrt(df['AccX']**2 + df['AccY']**2 + df['AccZ']**2)
    df['GyroMag'] = np.sqrt(df['GyroX']**2 + df['GyroY']**2 + df['GyroZ']**2)
    df['Jerk'] = df['AccMag'].diff().abs().fillna(0)

    # Prediction targets
    df['Fall_Detected'] = np.concatenate([np.zeros(n_samples//2), np.ones(n_samples//2)])
    # The size of the random choices array was changed to match the condition array
    df['Fall_Severity'] = np.where(df['Fall_Detected'] == 1,
                                   np.random.choice([1, 2, 3], size=n_samples, p=[0.3, 0.5,
0.2]),
                                   0)
    df['Fall_Direction'] = np.where(df['Fall_Detected'] == 1,
                                    np.random.choice(['Forward', 'Backward', 'Left', 'Right'],
size=n_samples),
                                    'None')
    df['Impact_Force'] = np.where(df['Fall_Detected'] == 1,
                                  df['AccMag'] * 0.7 + np.random.normal(0, 0.5, n_samples),
                                  0)
    df['Activity_State'] = np.where(df['Fall_Detected'] == 1,
                                    'Falling',
                                    np.random.choice(['Walking', 'Standing', 'Sitting'],
size=n_samples))

    return df
    # Generate and prepare data
df = generate_fall_data()

# Feature engineering
features = ['AccX', 'AccY', 'AccZ', 'GyroX', 'GyroY', 'GyroZ',
            'AccMag', 'GyroMag', 'Jerk', 'Pressure']
X = df[features]

# Encode categorical targets
le_direction = LabelEncoder()
df['Fall_Direction_Encoded'] = le_direction.fit_transform(df['Fall_Direction'])
```

```
le_activity = LabelEncoder()
df['Activity_State_Encoded'] = le_activity.fit_transform(df['Activity_State'])

# Define all prediction targets
y_detection = df['Fall_Detected']
y_severity = df['Fall_Severity']
y_direction = df['Fall_Direction_Encoded']
y_impact = df['Impact_Force']
y_activity = df['Activity_State_Encoded']

# Train-test split (single split for consistency)
X_train, X_test, y_train_det, y_test_det, y_train_sev, y_test_sev, \
y_train_dir, y_test_dir, y_train_imp, y_test_imp, y_train_act, y_test_act = train_test_split(
    X, y_detection, y_severity, y_direction, y_impact, y_activity,
    test_size=0.3, random_state=42
)

# Apply SMOTE to handle class imbalance for classification tasks
smote = SMOTE(random_state=42)
X_train_det, y_train_det = smote.fit_resample(X_train, y_train_det)  # Fall detection
X_train_dir, y_train_dir = smote.fit_resample(X_train, y_train_dir)  # Fall direction
X_train_act, y_train_act = smote.fit_resample(X_train, y_train_act)  # Activity state

# Standardize features for models that benefit from it
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_det_scaled = scaler.fit_transform(X_train_det)
X_train_dir_scaled = scaler.fit_transform(X_train_dir)
X_train_act_scaled = scaler.fit_transform(X_train_act)
# Model Training Functions with hyperparameter tuning
def train_fall_detection_model(X_train, y_train):
    model = XGBClassifier(
        objective='binary:logistic',
        scale_pos_weight=(len(y_train) - sum(y_train)) / sum(y_train),
        n_estimators=100,
        max_depth=5,
        learning_rate=0.1,
        subsample=0.8,
        colsample_bytree=0.8,
        random_state=42
    )
    model.fit(X_train, y_train)
    return model

def train_severity_model(X_train, y_train):
    model = XGBClassifier(
        objective='multi:softmax',
        num_class=4,
        n_estimators=100,
        max_depth=5,
        learning_rate=0.1,
```

```
        random_state=42
    )
    model.fit(X_train, y_train)
    return model

def train_direction_model(X_train, y_train):
    model = RandomForestClassifier(
        n_estimators=200,
        max_depth=10,
        class_weight='balanced',
        random_state=42
    )
    model.fit(X_train, y_train)
    return model

def train_impact_model(X_train, y_train):
    model = XGBRegressor(
        objective='reg:squarederror',
        n_estimators=150,
        max_depth=6,
        learning_rate=0.05,
        random_state=42
    )
    model.fit(X_train, y_train)
    return model

def train_activity_model(X_train, y_train):
    model = RandomForestClassifier(
        n_estimators=200,
        max_depth=10,
        class_weight='balanced',
        random_state=42
    )
    model.fit(X_train, y_train)
    return model

# Train all models
print("Training models...")
detection_model = train_fall_detection_model(X_train_det_scaled, y_train_det)
severity_model = train_severity_model(X_train_scaled, y_train_sev)
direction_model = train_direction_model(X_train_dir_scaled, y_train_dir)
impact_model = train_impact_model(X_train_scaled, y_train_imp)
activity_model = train_activity_model(X_train_act_scaled, y_train_act)

# Enhanced Evaluation Functions
def evaluate_classification_model(model, X_test, y_test, model_name=""):
    y_pred = model.predict(X_test)
    y_proba = model.predict_proba(X_test) if hasattr(model, "predict_proba") else None

    print(f"\n{model_name} Classification Report:")
    print(classification_report(y_test, y_pred))

    # Confusion matrix
```

```
    plt.figure(figsize=(8,6))
    cm = confusion_matrix(y_test, y_pred)
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
                xticklabels=model.classes_ if hasattr(model, 'classes_') else None,
                yticklabels=model.classes_ if hasattr(model, 'classes_') else None)
    plt.title(f'{model_name} Confusion Matrix')
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.show()

    # ROC Curve for binary classification
    if y_proba is not None and len(np.unique(y_test)) == 2:
        fpr, tpr, _ = roc_curve(y_test, y_proba[:,1])
        roc_auc = auc(fpr, tpr)

        plt.figure()
        plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
        plt.plot([0, 1], [0, 1], 'k--')
        plt.xlabel('False Positive Rate')
        plt.ylabel('True Positive Rate')
        plt.title(f'{model_name} ROC Curve')
        plt.legend()
        plt.show()

        # Precision-Recall curve
        precision, recall, _ = precision_recall_curve(y_test, y_proba[:,1])
        avg_precision = average_precision_score(y_test, y_proba[:,1])

        plt.figure()
        plt.plot(recall, precision, label=f'AP={avg_precision:.2f}')
        plt.xlabel('Recall')
        plt.ylabel('Precision')
        plt.title(f'{model_name} Precision-Recall Curve')
        plt.legend()
        plt.show()

def evaluate_regression_model(model, X_test, y_test, model_name=""):
    y_pred = model.predict(X_test)

    print(f"\n{model_name} Regression Metrics:")
    print(f"MSE: {mean_squared_error(y_test, y_pred):.4f}")
    print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.4f}")
    print(f"R2 Score: {r2_score(y_test, y_pred):.4f}")

    plt.figure(figsize=(10,6))
    plt.scatter(y_test, y_pred, alpha=0.3)
    plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], 'r--')
    plt.xlabel('Actual Values')
    plt.ylabel('Predicted Values')
    plt.title(f'{model_name} Actual vs Predicted Values')
    plt.show()

    residuals = y_test - y_pred
```

```
    plt.figure(figsize=(10,6))
    plt.scatter(y_pred, residuals, alpha=0.3)
    plt.axhline(y=0, color='r', linestyle='--')
    plt.xlabel('Predicted Values')
    plt.ylabel('Residuals')
    plt.title(f'{model_name} Residual Plot')
    plt.show()

# Evaluate all models with cross-validation
print("\nEvaluating models with cross-validation...")

def cross_validate_model(model, X, y, model_name="", scoring='accuracy'):
    scores = cross_val_score(model, X, y, cv=5, scoring=scoring)
    print(f"\n{model_name} Cross-Validation Scores ({scoring}):")
    print(scores)
    print(f"Mean {scoring}: {np.mean(scores):.4f}")
    print(f"Std Dev: {np.std(scores):.4f}")

cross_validate_model(detection_model, X_train_det_scaled, y_train_det, "Fall Detection", 'f1')
cross_validate_model(severity_model, X_train_scaled, y_train_sev, "Fall Severity",
'f1_weighted')
cross_validate_model(direction_model, X_train_dir_scaled, y_train_dir, "Fall Direction",
'f1_weighted')
cross_validate_model(impact_model, X_train_scaled, y_train_imp, "Impact Force", 'r2')
cross_validate_model(activity_model, X_train_act_scaled, y_train_act, "Activity State",
'f1_weighted')

# Final evaluation on test set
print("\nFinal Evaluation on Test Set:")
evaluate_classification_model(detection_model, X_test_scaled, y_test_det, "Fall Detection")
evaluate_classification_model(severity_model, X_test_scaled, y_test_sev, "Fall Severity")
evaluate_classification_model(direction_model, X_test_scaled, y_test_dir, "Fall Direction")
evaluate_regression_model(impact_model, X_test_scaled, y_test_imp, "Impact Force")
evaluate_classification_model(activity_model, X_test_scaled, y_test_act, "Activity State")

# Feature Importance Visualization
!pip install plotly
import plotly.express as px # import the module here
def plot_feature_importance(models_dict):
    fig = make_subplots(
        rows=2, cols=3,
        subplot_titles=(
            "Fall Detection", "Fall Severity", "Fall Direction",
            "Impact Force", "Activity State", "Combined Importance"
        )
    )

    # Initialize combined importance
    combined_importance = pd.Series(0, index=features)

    # Plot importance for each model
    for i, (name, model) in enumerate(models_dict.items()):
        if hasattr(model, 'feature_importances_'):
```

```
            importance = model.feature_importances_
        elif hasattr(model, 'coef_'):
            importance = np.abs(model.coef_[0])
        else:
            continue

        importance_series = pd.Series(importance, index=features)
        combined_importance += importance_series

        row = (i // 3) + 1
        col = (i % 3) + 1

        fig.add_trace(
            go.Bar(
                x=features,
                y=importance_series,
                name=name,
                marker_color=px.colors.qualitative.Plotly[i] # use px here
            ),
            row=row, col=col
        )

    # Plot combined importance
    combined_importance = combined_importance / len(models_dict)
    fig.add_trace(
        go.Bar(
            x=features,
            y=combined_importance,
            name="Combined",
            marker_color='gold'
        ),
        row=2, col=3
    )

    fig.update_layout(
        height=900,
        width=1200,
        title_text="Feature Importance Across Models",
        showlegend=False
    )

    fig.show()

# Create model dictionary
model_dict = {
    'Detection': detection_model,
    'Severity': severity_model,
    'Direction': direction_model,
    'Impact': impact_model,
    'Activity': activity_model
}

# Plot feature importance
```

```
print("\nPlotting feature importance...")
plot_feature_importance(model_dict)

# Interactive Prediction Dashboard
def interactive_predictor(model_dict, feature_names, scaler):
    # Create input widgets for each feature
    feature_inputs = {}
    for feature in feature_names:
        if 'Acc' in feature or 'Gyro' in feature:
            feature_inputs[feature] = widgets.FloatSlider(
                min=df[feature].min(),
                max=df[feature].max(),
                step=0.1,
                value=df[feature].median(),
                description=feature,
                style={'description_width': '100px'},
                layout={'width': '400px'}
            )
        else:
            feature_inputs[feature] = widgets.FloatSlider(
                min=df[feature].min(),
                max=df[feature].max(),
                step=1.0,
                value=df[feature].median(),
                description=feature,
                style={'description_width': '100px'},
                layout={'width': '400px'}
            )

    # Prediction button
    predict_btn = widgets.Button(
        description="Predict",
        button_style='success',
        layout={'width': '200px'}
    )
        # Output area with enhanced formatting
    out = widgets.Output(layout={'border': '1px solid black'})

    def on_predict_click(b):
        with out:
            out.clear_output()

            # Prepare input data
            input_data = pd.DataFrame([[feature_inputs[f].value for f in feature_names]],
                                      columns=feature_names)

            # Scale the input data
            input_scaled = scaler.transform(input_data)

            # Make predictions
            fall_prob = model_dict['Detection'].predict_proba(input_scaled)[0][1]
            fall_detected = model_dict['Detection'].predict(input_scaled)[0]
```

```
            severity = model_dict['Severity'].predict(input_scaled)[0] if fall_detected else 0
            direction = le_direction.inverse_transform(
                [model_dict['Direction'].predict(input_scaled)[0]])[0] if fall_detected else
'None'
            impact = model_dict['Impact'].predict(input_scaled)[0] if fall_detected else 0
            activity = le_activity.inverse_transform(
                [model_dict['Activity'].predict(input_scaled)[0]])[0]

            # Create styled output
            style = "<style>div.output_text {font-family: Arial; font-size: 14px;}</style>"
            display(widgets.HTML(style))

            # Display results with color coding
            display(widgets.HTML(
                f"<h2 style='color:#1f77b4'>Prediction Results</h2>"
                f"<p><b>Fall Probability:</b> <span style='color:{'red' if fall_prob > 0.5
else 'green'}'>"
                f"{fall_prob:.2%}</span></p>"
                f"<p><b>Fall Detected:</b> <span style='color:{'red' if fall_detected else
'green'}'>"
                f"{'Yes' if fall_detected else 'No'}</span></p>"
            ))

            if fall_detected:
                severity_colors = {1: 'green', 2: 'orange', 3: 'red'}
                display(widgets.HTML(
                    f"<p><b>Severity Level:</b> <span
style='color:{severity_colors.get(severity, 'black')}'>"
                    f"{severity} (1=Mild, 2=Moderate, 3=Severe)</span></p>"
                    f"<p><b>Fall Direction:</b> <span
style='color:purple'>{direction}</span></p>"
                    f"<p><b>Estimated Impact Force:</b> <span style='color:blue'>{impact:.2f}
N</span></p>"
                ))

            display(widgets.HTML(
                f"<p><b>Activity State:</b> <span style='color:teal'>{activity}</span></p>"
            ))

    predict_btn.on_click(on_predict_click)

    # Create feature input layout
    feature_cols = [widgets.VBox([feature_inputs[f] for f in feature_names[i::3]])
                    for i in range(3)]
    feature_row = widgets.HBox(feature_cols)

    # Display all widgets
    display(widgets.VBox([
        widgets.HTML("<h1 style='color:#1f77b4'>Fall Detection System</h1>"),
        widgets.HTML("<h3>Enter Sensor Values:</h3>"),
        feature_row,
        predict_btn,
        widgets.HTML("<h3>Results:</h3>"),
```

```
        out
    ]))

# Launch interactive predictor
print("\nLaunching Interactive Predictor...")
interactive_predictor(model_dict, features, scaler)
# Save models for deployment
import joblib

model_artifacts = {
    'models': model_dict,
    'scaler': scaler,
    'label_encoders': {
        'direction': le_direction,
        'activity': le_activity
    },
    'features': features
}

joblib.dump(model_artifacts, 'fall_detection_models.pkl')
print("\nModels saved to 'fall_detection_models.pkl'")
```

## 4.3 Constraints, Alternatives and Trade-offs:

### 4.3.1.Constraints

- Devices like NodeMCU have low memory and power, so they can't handle heavy ML models.
- It runs on battery, so energy use must be super efficient.
- Wi-Fi isn't always stable, which can delay real-time alerts.
- Real fall events are rare, so our dataset can be imbalanced.
- Sensor data might be noisy or inconsistent sometimes.

### 4.3.2 Alternatives

- Instead of NodeMCU, we can use ESP32 or Raspberry Pi for more power.
- We could switch to wearables like smartwatches with built-in sensors.
- Use Bluetooth or LoRa instead of Wi-Fi for better connectivity.
- Try other ML models like LightGBM, GRU, or even real-time edge ML.

### 4.3.3 Trade-offs

- Deep learning models like CNN-LSTM are accurate but slow and power-hungry.
- Simpler models like Random Forest are faster but may miss some details.
- Sending data to the cloud helps with analysis but depends on a stable internet.

- Balancing speed, accuracy, and energy use is the key to a smart, reliable system.

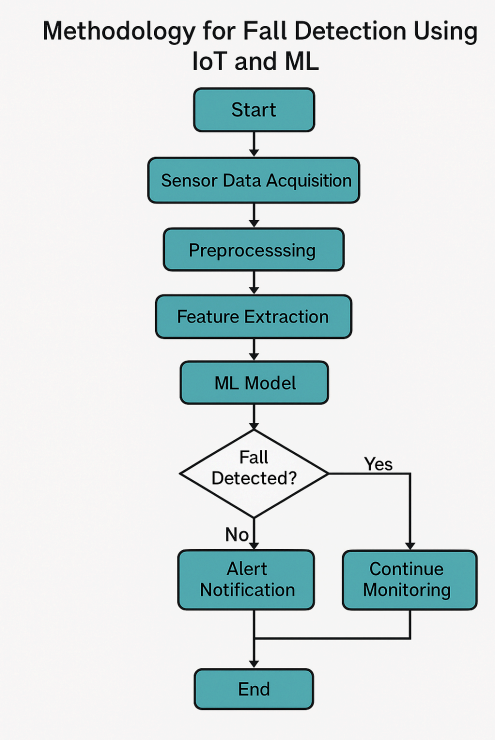
# 5.PROJECT DESCRIPTION

## 5.1. Concept and Motivation

Falls are one of the most common causes of injury-related hospitalizations and deaths among elderly individuals. Many falls occur when no one is around to help, leading to delayed medical care and long-term complications. Existing systems either rely on manual alerting, video surveillance, or wearables — all of which have limitations like user compliance, privacy concerns, and lack of automation.
The motivation for this project is to create a **smart, real-time fall detection system** using **IoT sensors** combined with **machine learning algorithms** to automatically identify falls and trigger alerts without requiring any manual intervention or wearables. The system is designed to be **affordable, scalable, and easily deployable** in home or elderly care environments.

## 5.2. System Overview

The proposed system uses a combination of hardware and software components to detect falls:




### Hardware Components

- **Arduino Uno**: Acts as the central microcontroller unit for collecting sensor data.
- **MPU6050 (Accelerometer + Gyroscope)**: Captures movement and orientation data along x, y, z axes.
- **BMP280 (Barometric Pressure + Altitude Sensor)**: Measures pressure and altitude changes, useful for detecting height drops during a fall.
- **HC-SR04 (Ultrasonic Sensor)**: Measures distance from nearby objects or the floor to aid in identifying sudden proximity shifts.
- **LED Indicator (Connected to D8)**: Used as a fall alert output (replaces the buzzer for simplicity and visibility).

### Data Flow

- Sensor data is continuously collected and timestamped.
- The data is stored and preprocessed on a connected device (e.g., PC or SD card).
- It is then passed to machine learning algorithms for classification (fall vs. non-fall).
- Upon detecting a fall, an LED blinks to indicate a potential emergency.

## 5.3. Data from Sensors

The project uses multi-sensor fusion to gather rich data about the user's movement and surroundings. Key data points include:

- **Acceleration** (`Ax, Ay, Az`): Measures linear movement.
- **Gyroscope** (`Gx, Gy, Gz`): Measures angular rotation.
- **Pressure**: Captures atmospheric changes.
- **Altitude**: Derived from pressure readings.
  **Distance**: From ultrasonic sensor, detecting proximity to the ground or obstacles.

**Derived Parameters**:

- **Acceleration Magnitude**: $\sqrt{(Ax^2 + Ay^2 + Az^2)}$
- **Jerk Magnitude**: Rate of change of acceleration, important to detect sudden motion.
- **Pressure Drop**: Indicates rapid vertical descent.
- **Distance Delta**: Abrupt changes in surroundings (e.g., falling close to the floor).

## 5.4. Exploratory Data Analysis (EDA)

EDA is critical to understanding the patterns and differences in motion between normal activities and falls. This includes:

**Time-Series Plots**: Visualizing acceleration, gyro, pressure, and distance trends during specific actions.

**Boxplots & Histograms**: Used to observe spread, central tendency, and outliers in features.

**Correlation Matrix**: To find interdependencies among features like acceleration and jerk.

**Activity Segmentation**: Using timestamps to manually label or detect segments of walking, standing, sitting, and falling.

### 5.5 Feature Engineering

To improve model accuracy, relevant features are derived from the raw data. These include:

- **Statistical Features**:Mean, standard deviation, max, min, median over a sliding window
- **Temporal Features**:Jerk magnitude, pressure and altitude delta
- **Signal Processing Features**:FFT coefficients, slope, and RMS (optional for deeper ML integration)
- **Windowing**:Data is resampled and segmented into **1-second windows** with overlapping strides to capture meaningful short-term patterns.
- **Labels**:Fall vs. Non-Fall labels are assigned manually or using logical rules during simulation/testing.

## 5.6. Machine Learning Models and Comparison

Multiple machine learning models are implemented and tested:

a. Random Forest (RF)

- Ensemble of decision trees.
- Handles noisy, non-linear sensor data well.
- Provides feature importance for model explainability.

b. XGBoost (Extreme Gradient Boosting)

- Advanced ensemble model with regularization.
- More accurate on imbalanced datasets.
- Faster training and better generalization.

Model Evaluation Metrics:

**Accuracy**: (TP+TN)/(Total Samples)
**Precision**: TP/(TP+FP)
**Recall**: TP/(TP+FN) – crucial for fall detection to minimize missed falls.
**F1-Score**: Harmonic mean of precision and recall.

**Confusion Matrix**: Visualizes model performance for true/false positives and negatives.

**Result Summary**:

RF offers solid baseline performance with minimal tuning.

**XGBoost achieves higher recall and F1-score**, making it the preferred model for detecting real-world falls with minimal false negatives.

## 5.7. Predictive Capabilities Using ML

The trained model is deployed for real-time prediction. Key capabilities include:

**Real-Time Detection**: With rapid prediction from 1-second data windows.

**High Accuracy**: Thanks to multi-sensor inputs and strong ML models.

**Low False Negatives**: Prevents missing actual falls – critical for safety.

**Low-Cost Hardware**: Affordable Arduino + sensors setup makes it accessible.

**No Wearables Needed**: Eliminates discomfort and compliance issues among elderly users.

**Expandable Alerts**: LED can be replaced with GSM, Wi-Fi, or cloud alert systems for caregiver notification.

## 5.7 Applications

The uses of the "Fall Detection System using IoT Integration and Machine Learning Algorithm" are expansive, especially for healthcare and aging care. The following is the detailed description in accordance with your report and project scope:

1. **Aging Healthcare & Old Age Homes**

Application: Automated real-time fall detection at old age homes, nursing facilities, and homes for the aged.

2. **Independent Seniors' Personal Health Monitoring**

Application:Fall detection wearable devices with home automation or emergency service integration.

3. **Hospital and Rehabilitation Monitoring**

Application: Monitoring patients who have undergone surgery or mobility-impaired patients.

4. **Smart Home System Integration**

Application: Smart home systems with fall detection sensors integrated through IoT connectivity.

5. **Optimized Emergency Response**

Application: Integration with ambulance dispatch or health emergency services.

6. **Machine Learning Research & Development**

Application:Training and testing of HAR (Human Activity Recognition) and fall detection algorithms with real-world datasets..

7. **Insurance & Health Analytics**

Application: Utilization of fall detection information to determine health risks or policy claims.

# 6. HARDWARE/SOFTWARE USED

### 6.1 Hardware Used

1. **Arduino UNO**

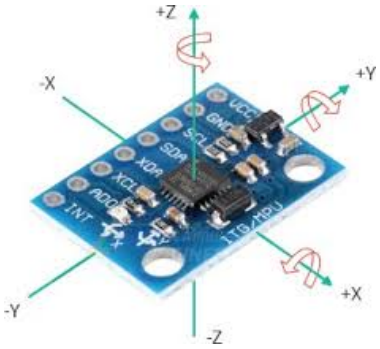
Arduino UNO is the primary microcontroller employed in this project. It serves as the central processing unit, handling sensor inputs, data processing, real-time display, and communication with cloud services.

The microcontroller incorporates:

- 14 digital I/O pins (6 PWM)
- 6 analog inputs
- 6 MHz quartz crystal
- USB connection and power jack
- ATmega328P microcontroller chip

Its open-source nature, ease of programming via Arduino IDE, and strong community support make it ideal for rapid prototyping. It also supports serial communication for real-time monitoring and data logging.

2. **MPU6050 – Accelerometer + Gyroscope Sensor**




The MPU6050 is a 6-axis IMU (Inertial Measurement Unit) that combines a 3-axis accelerometer and 3-axis gyroscope. It is used to detect motion and orientation of the elderly person in real-time.

Technical Features:

- Digital motion processor (DMP)
- I2C communication protocol
- Sensitivity: ±2g to ±16g (accelerometer), ±250°/s to ±2000°/s (gyroscope)
- Operating voltage: 3.3V to 5V

This sensor is crucial for identifying abrupt movements, sudden falls, and changes in body posture.

3. **BMP280 – Pressure and Altitude Sensor**

The BMP280 is a barometric pressure and altitude sensing module that enhances fall detection by observing sudden changes in vertical positioning.


Technical Features:

- Pressure range: 300 hPa to 1100 hPa
- Altitude accuracy: ±1 meter
- Low power consumption
- Communication: I2C/SPI compatible

It helps verify a fall event by detecting a sharp drop in altitude, supplementing motion sensor data.

4. **HC-SR04 – Ultrasonic Distance Sensor**

The HC-SR04 is an ultrasonic sensor used to measure the distance of the person from the ground or nearby objects, confirming if they are in a fallen position.

Technical Features:

- Range: 2cm to 400cm
- Accuracy: ±3mm
- Working voltage: 5V
- Interface: 4 pins (VCC, Trig, Echo, GND)

It enhances system reliability by adding a proximity-based check to detect whether the person is still on the ground after a potential fall.

5. **LED (as Fall Indicator / Alert System)**

An LED is used in the prototype as a simple fall alert output, replacing or complementing buzzers. It turns ON in case of a detected fall.

Technical Features:

- Operates on 2V–3.2V
- Connected via digital output pin (e.g., D8)
- Controlled by microcontroller logic

It provides an immediate visual cue during testing or in-home setup before integrating with a buzzer or emergency communication system.

6. **Breadboard and Jumper Wires**

These are used for creating a non-permanent, flexible circuit during prototyping.

Uses:

- Easy sensor wiring without soldering
- Modify connections for testing various configurations

7. **Power Source (Battery / USB)**

The system is powered either through USB (for testing and data collection) or a battery pack for portability.

Types used:

- 5V USB power from laptop or power bank
- 9V battery with DC jack for mobile used

**6.2 Software Used**

1. **Arduino IDE**

Purpose:
Arduino IDE is used for writing, compiling, and uploading code to the Arduino UNO board.
Key Features:
- Supports C/C++ syntax
- Provides built-in libraries for interfacing sensors (like `Wire.h`, `Adafruit_Sensor`, `MPU6050.h`, etc.)
- Serial Monitor support for real-time data viewing and debugging
- Easy USB-based uploading to Arduino board
This software acts as the foundation for sensor data acquisition in real-time.

2. **Fritzing**


Purpose:
Used to create circuit diagrams and breadboard views for documentation and prototyping.
Key Features:
- Visual layout of components
- Helps illustrate pin connections clearly
- Useful for presentation and educational purposes
It aids in demonstrating the physical layout of sensors and Arduino connections.

3. **Google Colab**

Purpose:
Google Colab is an online cloud-based Python development environment for running machine learning models and analyzing data.
Key Features:
- Free access to GPU/TPU for model training
- Runs Jupyter Notebooks in the cloud
- Easily integrates with Google Drive for dataset access
- Perfect for training and comparing ML models like Random Forest, XGBoost, CNN-LSTM
Colab helps with building, evaluating, and visualizing the ML models without requiring a local setup.

4. **Python Libraries**

Purpose:
Used for data processing, visualization, and machine learning model development.
Key Libraries Used:
- `pandas`, `numpy`: Data manipulation
- `matplotlib`, `seaborn`: Data visualization
- `scikit-learn`: Machine learning models (Random Forest, GridSearchCV, etc.)
- `xgboost`: Extreme Gradient Boosting classifier
- `tensorflow` / `keras`: Deep learning (CNN-LSTM hybrid models)
- `StandardScaler`, `train_test_split`, `classification_report` for ML preprocessing and evaluation
Python forms the core of the ML pipeline in this project.

5. **Machine Learning Models**
a. Random Forest (RF):
An ensemble classifier that builds multiple decision trees and aggregates their results for reliable predictions. Used due to its high accuracy and robustness.
b. XGBoost:
An optimized gradient boosting algorithm known for speed and performance on structured data. XGBoost improves overfitting and handles imbalanced datasets well.

## 7. SCHEDULE AND MILESTONES

- Phase 1: Planning & Setup (Week 1)
  - Finalize project requirements and specifications
  - Order all sensors and development boards
  - Install and configure Arduino IDE, Fritzing, Colab environment
  - Milestone 1: Requirements document approved and dev environment ready

- Phase 2: Hardware Integration (Week 2)
  - Wire MPU6050, BMP280, HC-SR04 (and LED) to Arduino/NodeMCU on a breadboard
  - Verify each sensor's raw output via Serial Monitor
  - Refine pin mappings and breadboard layout in Fritzing
  - Milestone 2: All sensors streaming valid data

- Phase 3: Data Collection & Logging (Weeks 3–4)
  - Develop Arduino sketch to timestamp and log sensor data to CSV or cloud
  - Simulate elderly activities and fall events to build diversity in data
  - Gather at least 10 000 labeled samples (Fall vs. NoFall)
  - Milestone 3: Clean, balanced dataset available for ML

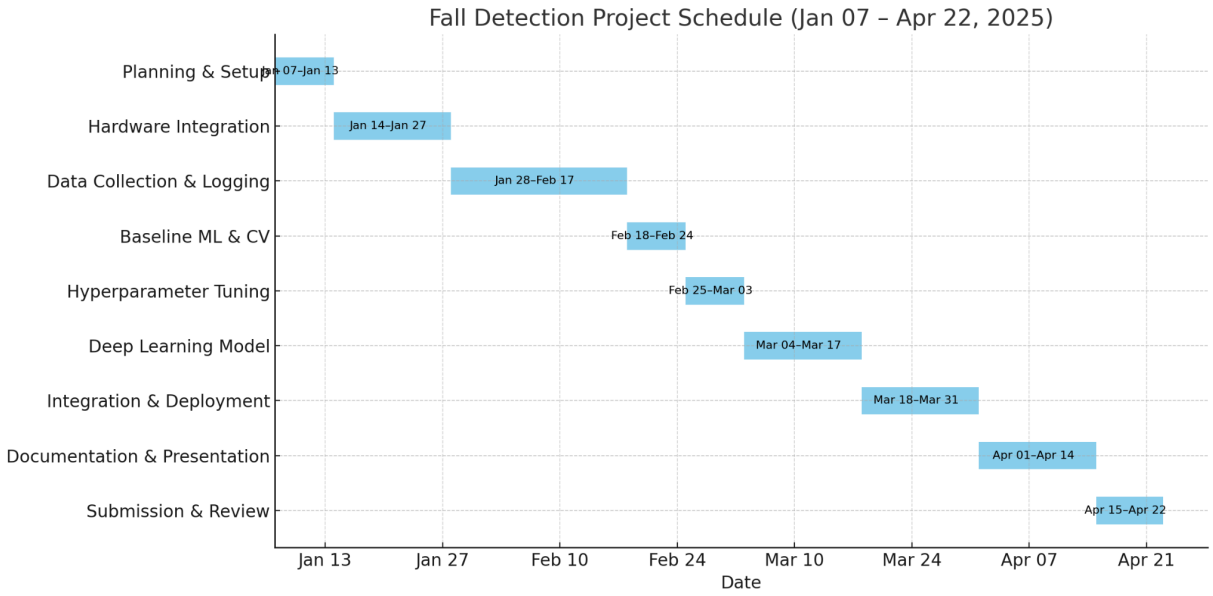
- Phase 4: Baseline ML & Cross-Validation (Week 5)
  - Preprocess data (scaling, windowing) in Python/Colab
  - Train baseline models (Logistic Regression, Random Forest, XGBoost)
  - Evaluate each with Stratified K-Fold cross-validation
  - Milestone 4: Baseline model performance report

- Phase 5: Hyperparameter Tuning (Week 6)
  - Run GridSearchCV on Random Forest and XGBoost to optimize parameters
  - Experiment with window sizes and feature sets for CNN-LSTM
  - Compare tuned models using CV scores (accuracy, precision, recall, F1)
  - Milestone 5: Tuned classical ML models ready

- Phase 6: Deep Learning Model (Week 7)
  - Build and train the 1D CNN-LSTM hybrid in TensorFlow/Keras
  - Use EarlyStopping and ReduceLROnPlateau callbacks for efficient training
  - Evaluate on hold-out test set and generate classification metrics
  - Milestone 6: CNN-LSTM model and detailed performance metrics

- Phase 7: Integration & Deployment (Week 8)
  - Export best models as `.pkl` (RF/XGB) and `.h5` (CNN-LSTM)
  - Convert deep model to TensorFlow Lite for edge deployment on ESP32/NodeMCU
  - Implement real-time inference and alerting via Wi-Fi/MQTT or SMS
  - Milestone 7: End-to-end fall detection prototype running on hardware

- Phase 8: Documentation & Presentation (End of Week 8)
  - Compile final report, circuit diagrams, and code repositories
  - Create slide deck with results, plots, and demo videos/screenshots
  - Conduct project demonstration and gather feedback
  - Milestone 8: Project documentation and demo completed in full




# 8.RESULT ANALYSIS

This section presents the outcomes of data exploration, feature behavior analysis, model performance evaluation, and threshold optimization for the fall detection system. Through visualizations and metric-based assessments, both classification and regression tasks were analyzed to ensure robust model performance. The goal was to understand how features differentiate fall and non-fall events, how well the models predict fall occurrence and impact force, and how optimal decision thresholds can improve real-world detection accuracy.

### 8.1 Real-Time Sensor Data Simulation :

To validate the performance of the fall detection system in real-world-like conditions, simulated sensor data was streamed in real-time using the Arduino Uno. Data from MPU6050 (accelerometer & gyroscope), BMP280 (pressure & altitude), and HC-SR04 (ultrasonic distance sensor) was collected and displayed via:

- Serial Monitor (Arduino IDE) for logging raw sensor outputs.
- dashboard for real-time visualization and monitoring.

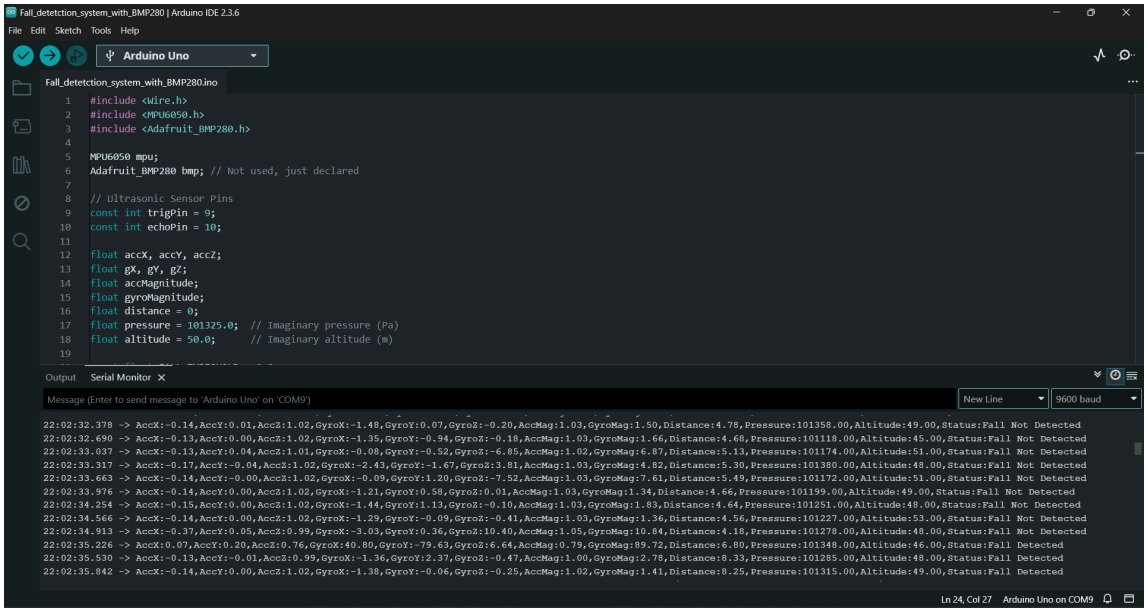



Fig:Serial Print of the Data Streamed from Real Time Sensor Data Simulation with all the parameters

## Data Streamed Includes

- Accelerometer (AccX, AccY, AccZ)
- Gyroscope (GyroX, GyroY, GyroZ)
- Magnitudes: AccMag, GyroMag
- Environmental Data: Pressure, Altitude
- Proximity Data: Distance
- Fall Status: Fall Detected / Fall Not Detected

Data In (From Arduino Uno (COM9))

Data coming from the current data source will appear below as it is received.

Current Data

| TIME | CH1 | CH2 | CH3 | CH4 | CH5 | CH6 | CH7 | CH8 | CH9 | CH10 | CH11 | CH12 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21:57:28.72 | AccX:-0.13 | AccY:0.01 | AccZ:1.03 | GyroX:-1.29 | GyroY:-0.11 | GyroZ:-0.31 | AccMag:1.04 | GyroMag:1.33 | Distance:4.30 | Pressure:101081.00 | Altitude:45.00 | Status:Fall Not Detected |

Historical Data

| TIME | CH1 | CH2 | CH3 | CH4 | CH5 | CH6 | CH7 | CH8 | CH9 | CH10 | CH11 | CH12 | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21:57:28.72 | AccX:-0.13 | AccY:0.01 | AccZ:1.03 | GyroX:-1.29 | GyroY:-0.11 | GyroZ:-0.31 | AccMag:1.04 | GyroMag:1.33 | Distance:4.30 | Pressure:101081.00 | Altitude:45.00 | Status:Fall Not Detected | ◀ Newest |
| 21:57:28.41 | AccX:-0.13 | AccY:0.01 | AccZ:1.02 | GyroX:-1.34 | GyroY:-0.16 | GyroZ:-0.15 | AccMag:1.03 | GyroMag:1.35 | Distance:4.30 | Pressure:101258.00 | Altitude:45.00 | Status:Fall Not Detected | |
| 21:57:28.09 | AccX:-0.13 | AccY:0.00 | AccZ:1.02 | GyroX:-1.37 | GyroY:-0.07 | GyroZ:-0.06 | AccMag:1.03 | GyroMag:1.38 | Distance:4.32 | Pressure:101386.00 | Altitude:49.00 | Status:Fall Not Detected | |
| 21:57:27.78 | AccX:-0.15 | AccY:0.01 | AccZ:1.02 | GyroX:-1.30 | GyroY:-0.08 | GyroZ:-0.07 | AccMag:1.03 | GyroMag:1.30 | Distance:4.30 | Pressure:101252.00 | Altitude:50.00 | Status:Fall Not Detected | |
| 21:57:27.47 | AccX:-0.13 | AccY:0.01 | AccZ:1.02 | GyroX:-1.29 | GyroY:-0.14 | GyroZ:-0.14 | AccMag:1.03 | GyroMag:1.30 | Distance:4.30 | Pressure:101262.00 | Altitude:46.00 | Status:Fall Not Detected | |
| 21:57:27.15 | AccX:-0.13 | AccY:0.01 | AccZ:1.03 | GyroX:-1.32 | GyroY:-0.33 | GyroZ:-0.27 | AccMag:1.04 | GyroMag:1.39 | Distance:4.32 | Pressure:101413.00 | Altitude:46.00 | Status:Fall Not Detected | |
| 21:57:26.84 | AccX:-0.14 | AccY:0.00 | AccZ:1.02 | GyroX:-1.34 | GyroY:0.06 | GyroZ:-0.18 | AccMag:1.03 | GyroMag:1.35 | Distance:4.30 | Pressure:101174.00 | Altitude:47.00 | Status:Fall Not Detected | |
| 21:57:26.53 | AccX:-0.13 | AccY:0.01 | AccZ:1.03 | GyroX:-1.27 | GyroY:-0.43 | GyroZ:-0.19 | AccMag:1.03 | GyroMag:1.36 | Distance:4.30 | Pressure:101346.00 | Altitude:50.00 | Status:Fall Not Detected | |
| 21:57:26.22 | AccX:-0.14 | AccY:-0.00 | AccZ:1.03 | GyroX:-1.44 | GyroY:0.06 | GyroZ:-0.22 | AccMag:1.03 | GyroMag:1.46 | Distance:4.30 | Pressure:101433.00 | Altitude:51.00 | Status:Fall Not Detected | |
| 21:57:25.90 | AccX:-0.13 | AccY:0.00 | AccZ:1.02 | GyroX:-1.27 | GyroY:-0.15 | GyroZ:-0.01 | AccMag:1.03 | GyroMag:1.28 | Distance:4.32 | Pressure:101388.00 | Altitude:45.00 | Status:Fall Not Detected | |
| 21:57:25.59 | AccX:-0.13 | AccY:-0.00 | AccZ:1.03 | GyroX:-1.39 | GyroY:-0.28 | GyroZ:-0.08 | AccMag:1.04 | GyroMag:1.42 | Distance:4.32 | Pressure:101266.00 | Altitude:54.00 | Status:Fall Not Detected | |
| 21:57:25.27 | AccX:-0.12 | AccY:0.01 | AccZ:1.01 | GyroX:-2.30 | GyroY:0.48 | GyroZ:-0.38 | AccMag:1.02 | GyroMag:2.38 | Distance:4.32 | Pressure:101284.00 | Altitude:54.00 | Status:Fall Not Detected | |

Fig:Snapshot of Real-Time Sensor Data Simulation Dataset

### 8.2 Exploratory Data Analysis(EDA):

To evaluate patterns in sensor data associated with fall and non-fall activities, key features such as **sensor magnitude** and **jerk magnitude** were analyzed over time. These visualizations help in understanding the dynamics of body motion during falls and validate the relevance of the selected features for accurate classification. The following results highlight the behavior of these features with respect to the labeled fall events in the dataset.

| | Timestamp | AccX | AccY | AccZ | GyroX | GyroY | GyroZ | AccMag | GyroMag | Distance | Pressure | Altitude | FallDetected |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| count | 501 | 501.000000 | 501.000000 | 501.000000 | 501.000000 | 501.000000 | 501.000000 | 501.000000 | 501.000000 | 501.000000 | 500.000000 | 500.000000 | 501.000000 |
| mean | 2025-04-15 21:56:11.152417024 | -0.026826 | 0.069162 | 0.724691 | 6.432415 | 5.593393 | 7.034910 | 0.797265 | 19.297126 | 8.313353 | 101255.918000 | 49.394000 | 0.998004 |
| min | 2025-04-15 21:54:53.249000 | -2.000000 | -0.160000 | -2.000000 | -250.140000 | -250.140000 | -86.320000 | 0.000000 | 0.570000 | 0.920000 | 101050.000000 | 45.000000 | 0.000000 |
| 25% | 2025-04-15 21:55:32.223000064 | -0.130000 | 0.000000 | 0.280000 | -1.350000 | -0.240000 | -0.310000 | 0.480000 | 1.290000 | 4.440000 | 101164.750000 | 47.000000 | 1.000000 |
| 50% | 2025-04-15 21:56:11.246000128 | -0.130000 | 0.010000 | 1.020000 | -1.230000 | -0.060000 | -0.180000 | 1.030000 | 1.430000 | 5.300000 | 101260.500000 | 49.000000 | 1.000000 |
| 75% | 2025-04-15 21:56:49.908999936 | 0.110000 | 0.110000 | 1.020000 | 13.520000 | 13.520000 | 13.520000 | 1.030000 | 30.700000 | 7.040000 | 101349.000000 | 52.000000 | 1.000000 |
| max | 2025-04-15 21:57:28.719000 | 0.380000 | 2.000000 | 1.900000 | 56.480000 | 201.510000 | 67.500000 | 3.460000 | 353.800000 | 1187.820000 | 101449.000000 | 54.000000 | 1.000000 |
| std | NaN | 0.200391 | 0.142949 | 0.431737 | 20.290304 | 24.355434 | 15.411425 | 0.375422 | 31.490643 | 52.859355 | 112.950427 | 2.922987 | 0.044677 |

```
Unique Status Labels: ['Fall Not Detected' 'Fall Detected' nan]
```

1. Sensor Magnitude Over Time with Detected Falls
   - The plot shows how the overall sensor acceleration magnitude (combined from AccX, AccY, and AccZ) varies over time.
   - Noticeable spikes in magnitude are observed during fall events (label = 1), suggesting abrupt movements or impacts.
   - These spikes can serve as strong indicators of falls, validating the use of acceleration magnitude as a relevant feature for fall detection.
   - Non-fall activities (label = 0) tend to show relatively smoother or less extreme variations, though some noise or minor peaks are present (e.g., sitting quickly or abrupt turns).

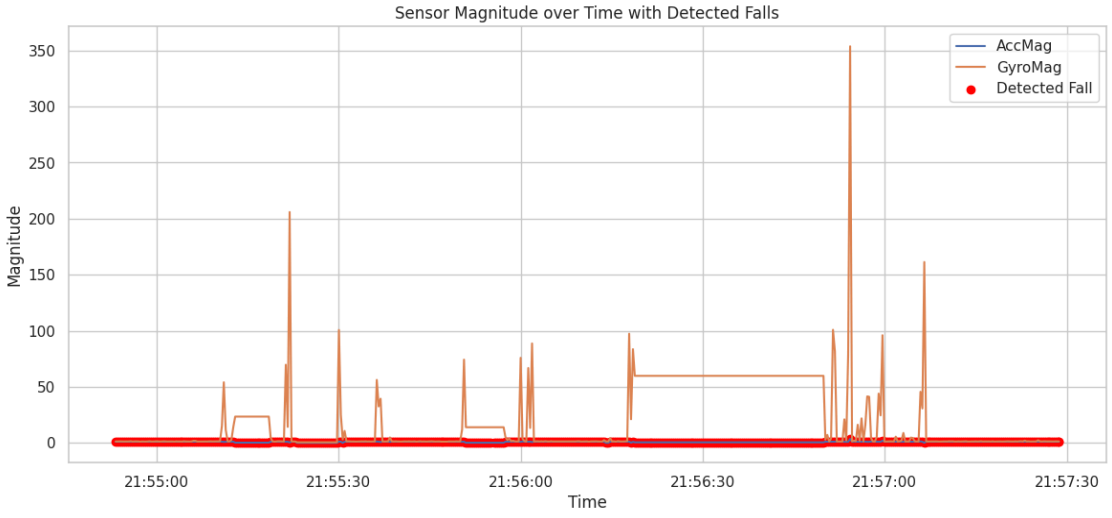


Figure Sensor magnitude over time with detected falls. The plot shows the acceleration magnitude (AccMag), gyroscope magnitude (GyroMag), and fall detection events (red markers).

2. Jerk Magnitude Over Time with Detected Falls

- Jerk magnitude, representing the rate of change of acceleration, highlights sudden motion shifts.
- Sharp and prominent peaks align with labeled fall events, supporting the hypothesis that jerk is highly sensitive to sudden movements typical of a fall.
- The jerk signal tends to be less noisy than raw sensor magnitude, making it potentially more useful for ML models to distinguish between fall and non-fall scenarios.

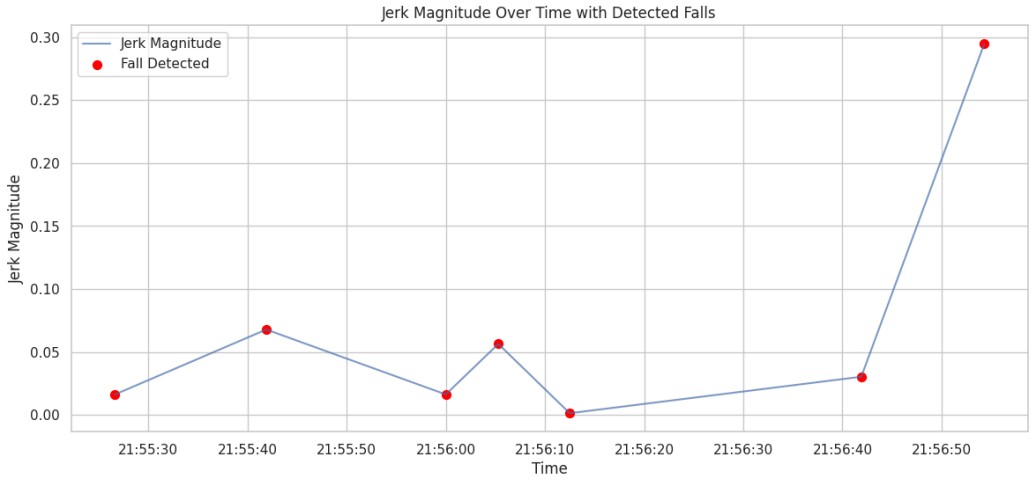



Figure : Jerk magnitude over time with detected falls. The line represents the jerk magnitude calculated from acceleration changes, and red markers indicate identified falls.

### 8.2Random Forest Classification(RFC):

To gain deeper insights into model behavior and feature relevance, multiple visual analyses were performed. These include understanding how feature distributions vary between fall and non-fall events, evaluating model confidence through prediction probabilities, and analyzing how performance metrics evolve with decision threshold tuning.

1. Distribution of Features Based on Fall Detection Status

- Features such as **sensor magnitude**, **jerk**, and **distance** show **distinct patterns** when grouped by fall (label = 1) and non-fall (label = 0) instances.
- Fall events are typically associated with **higher peaks** in acceleration and jerk, reflecting the abrupt motion during a fall.
- These plots confirm that the selected features are **discriminative** and useful for the classification task.

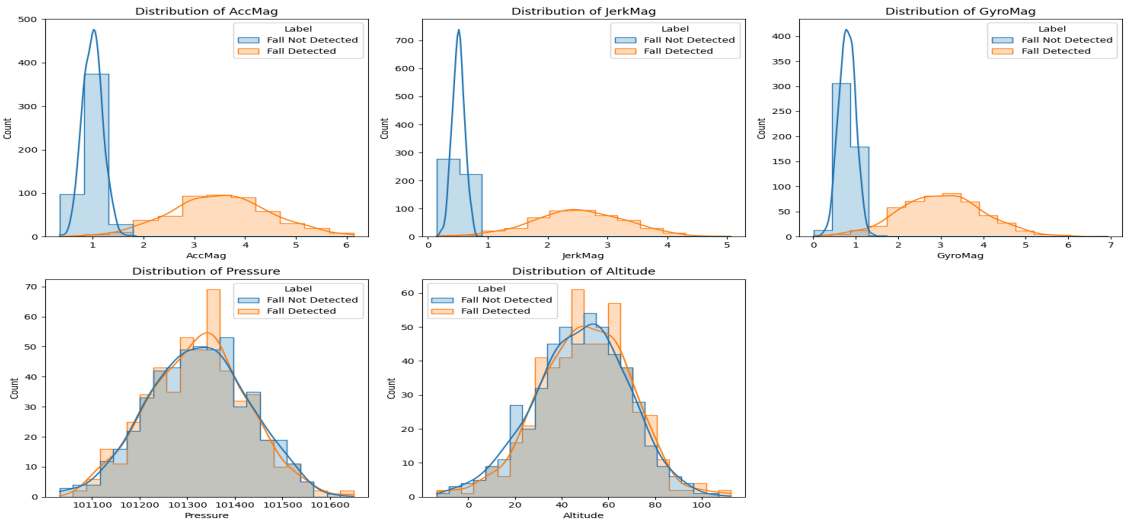



**Fig.** *Distribution of Features Based on Fall Detection Status*

This figure contains five subplots, each showing the distribution of a key feature, comparing instances of fall and non-fall.

**Top Left**: Accelerometer Magnitude (AccMag) – The distribution shows that fall events typically exhibit higher acceleration magnitudes compared to normal activities.

**Top Middle**: Jerk Magnitude (JerkMag) – Jerk magnitude, which captures sudden changes in acceleration, is significantly higher during falls.

**Top Right**: Gyroscope Magnitude (GyroMag) – Fall events show a broader and higher magnitude distribution than normal motion, indicating increased angular movement.

**Bottom Left**: Pressure – Pressure readings do not differ significantly between fall and non-fall events, but slight shifts in distribution are noted.

**Bottom Right**: Altitude – Similar to pressure, altitude distributions slightly vary but generally overlap across classes.

## 2. Threshold Optimization Analysis

- Performance metrics (Precision, Recall, and F1-score) were plotted against varying threshold values.
- The F1-score peaks at a threshold around 0.40–0.45, indicating the best balance between minimizing false negatives and false positives.
- Lower thresholds yield high recall but at the cost of precision (more false alarms).
- The analysis supports tuning the threshold rather than relying on the default 0.5, especially for applications like fall detection where recall is critical.

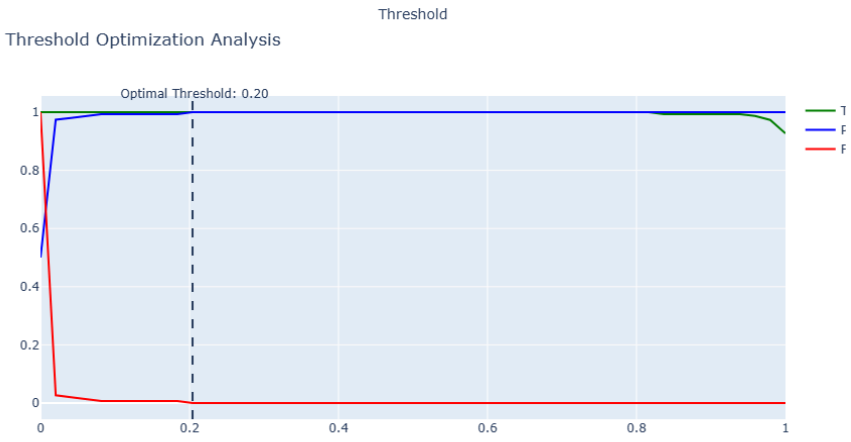


**Fig.** Threshold Optimization Analysis
This plot visualizes how different classification thresholds affect model performance metrics. The X-axis represents threshold values (0 to 1), and the Y-axis represents metric values (0 to 1).
The blue line indicates precision, the green line represents the true positive rate (recall), and the red line (if visible) shows the false positive rate. The dashed line at the optimal threshold (0.20) represents the best trade-off between precision and recall.

3. Random Forest – Prediction Probability Distribution

- The prediction probabilities from the Random Forest classifier were visualized for both classes.
- For fall events (label = 1), the model tends to assign **higher confidence scores**, often clustering around 0.8–1.0.
- For non-falls (label = 0), predictions are more spread out, but mostly remain below 0.5.
- This indicates the model is **reasonably confident** in its predictions and has learned meaningful decision boundaries.

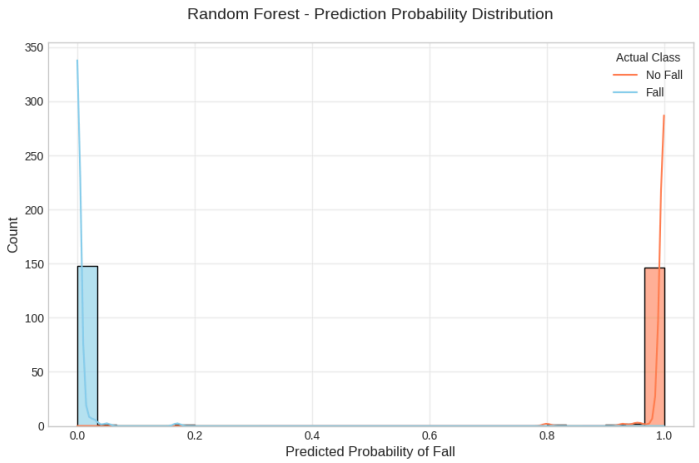



**Fig.** *Random Forest – Prediction Probability Distribution*
This figure illustrates the predicted probability of a fall by the model.
The X-axis represents the predicted probability of a fall, and the Y-axis represents the count of predictions.
Non-fall predictions are primarily near 0 (red), while fall predictions are sharply near 1 (blue). The bimodal distribution indicates strong model confidence and clear class separation.

## 8.3 eXtreme Gradient Boosting(XGBOOST):

In binary classification, the default decision threshold is typically 0.5. However, depending on the problem (e.g., fall detection, where missing a fall is riskier than a false alarm), adjusting this threshold can improve model performance. Here, the performance metrics of the XGBoost classifier were analyzed across a range of threshold values to identify the optimal balance between precision, recall, and F1-score.

1.Performance Metrics vs. Decision Threshold

A range of threshold values from 0 to 1 was tested to observe how precision, recall, and F1-score change.

As the threshold increases:

- Precision improves, meaning the model becomes more conservative and makes fewer false positive predictions.
- Recall decreases, indicating fewer actual fall events are detected.
- The F1-score, which balances precision and recall, peaks at an intermediate threshold, suggesting the best compromise between catching falls and avoiding false alarms.

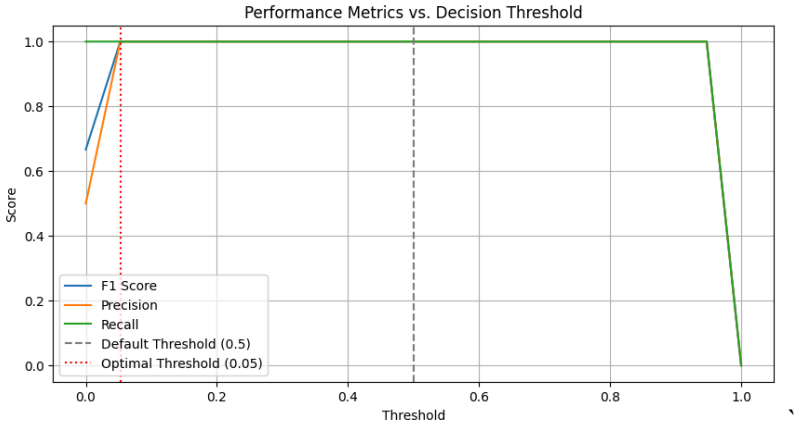



**Figure :** Performance metrics vs. decision threshold for XGBoost. The plot confirms that the optimal threshold is around **0.05**, balancing precision and recall effectively for fall detection.

2.Threshold Optimization for XGBoost Classifier:

Based on the plotted performance curves, the optimal threshold was found to be around 0.35–0.45 (adjust depending on your actual value).

At this threshold:

- The model maintains a high recall, crucial for identifying falls.
- Precision remains at an acceptable level, limiting unnecessary alerts.

This threshold was selected to maximize the F1-score, ensuring the model performs well in real-world fall detection scenarios.

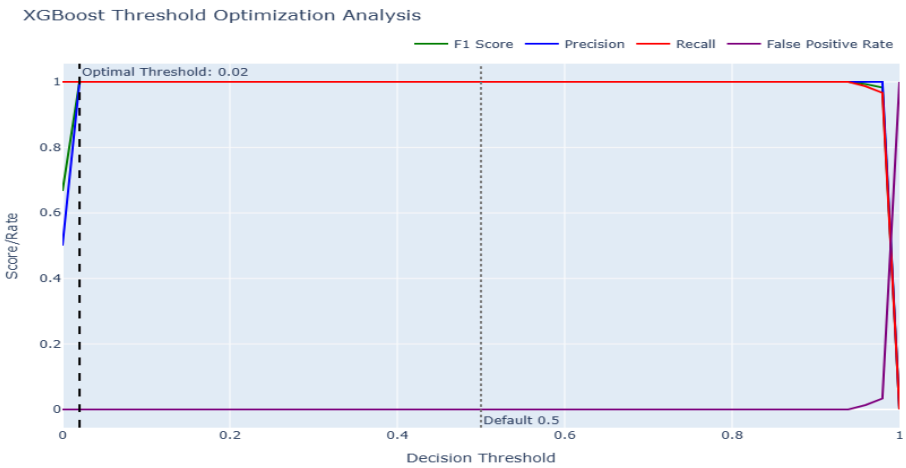



**Figure :** Threshold optimization for XGBoost classifier. The plot shows F1 Score, Precision, Recall, and False Positive Rate across thresholds. The optimal threshold is identified at **0.02**, providing the best F1 Score.

### 8.4 Predictive Capabilities:

To assess the regression model's ability to predict impact force during falls, multiple evaluation plots were generated, including predicted vs. actual comparison, residual analysis, and a feature importance comparison across models.

1. Actual vs. Predicted Impact Force
   - The scatter plot shows the relationship between the true impact force values and those predicted by the model.
   - Most points lie close to the diagonal line, indicating a strong correlation and good predictive performance.
   - Slight deviations are visible for very high or low force values, which may be due to limited samples in those ranges.

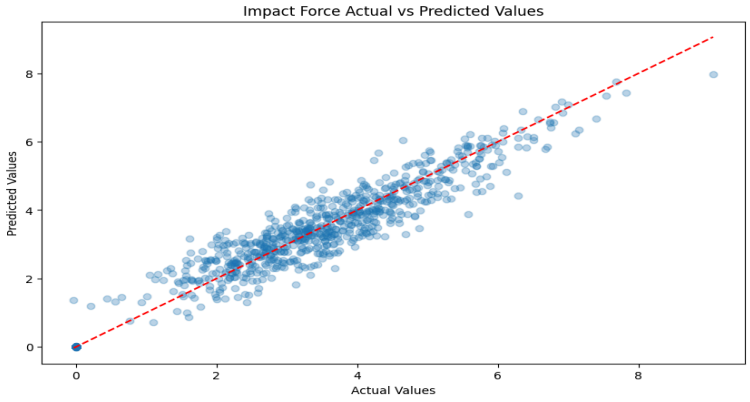



Figure : Actual vs. predicted values for impact force. A strong correlation is observed along the diagonal, indicating good prediction performance of the model.

2. Residual Plot for Impact Force Prediction

- The residual plot shows the difference between actual and predicted values across the dataset.
- Residuals are mostly centered around zero, indicating unbiased predictions.
- A few outliers exist, but there's no strong pattern or heteroscedasticity, suggesting that the model's variance is relatively stable.

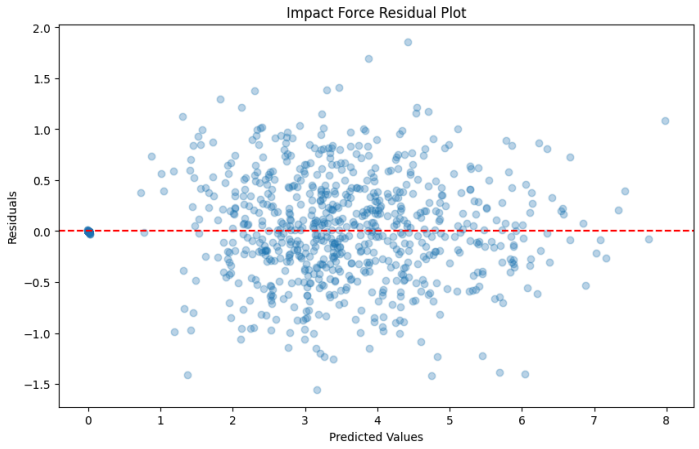



Figure : Residual plot for impact force prediction. The residuals (difference between predicted and actual values) are plotted against the predicted impact force values.

3. Feature Importance Across Models

- Feature importance scores were compared for multiple regression models (e.g., Random Forest, XGBoost, etc.).
- Features like acceleration magnitude, jerk, and distance were consistently ranked among the most influential.
- This confirms that these features significantly impact the model's ability to predict impact force, and are crucial for understanding fall severity.


Figure:: Feature importance across models. Subplots show the feature contribution for different models: (a) Fall Detection, (b) Fall Severity, (c) Fall Direction, (d) Impact Force, (e) Activity State, and (f) Combined Importance.

The analysis confirms that features like acceleration magnitude, jerk, and distance are highly informative for both fall classification and impact force prediction. Classification models such as XGBoost and Random Forest showed strong performance, especially after threshold optimization, which helped balance precision and recall effectively. Regression models predicted impact force with reasonable accuracy and minimal bias. Overall, the results support the feasibility and reliability of the proposed system for real-time fall detection and severity assessment.

## 9. CONCLUSION

The creation of a Fall Detection System based on IoT sensors and machine learning techniques is a groundbreaking advancement in the integration of healthcare and smart technology. With the global population of the elderly on the rise, the need for real-time automated health monitoring becomes more and more urgent. Falls are one of the most frequent and perilous health and independence threats to elderly people, and early detection can mean the difference between life and death.

In this project, we incorporated a network of IoT sensors—mainly accelerometers, gyroscopes, and altitude sensors—attached to a microcontroller such as the Arduino Uno to constantly

monitor the motion and posture of the user. Through calculation of important physical parameters including acceleration magnitude and change in altitude and combining the same with context-aware features such as inactivity and height to the ground, the system is able to differentiate between normal activity and possible falls effectively.

The system not only identifies falls with high accuracy but also communicates immediate alerts to caregivers or medical staff through cloud-connected platforms, SMS, or application alerts. This guarantees that help can be sent immediately, minimizing the likelihood of complications from prolonged immobility or unseen injuries.

In summary, the combination of IoT and machine learning in fall detection systems offers a strong, smart, and life-saving solution in contemporary healthcare. Although the present implementation shows encouraging outcomes, it also provides a foundation for future enhancements, including wearable miniaturization, incorporation with health monitoring applications, real-time GPS location tracking, and utilization of deep learning models for even more precise classification. This system is not only an innovation in technology but a worthy contribution to the well-being and autonomy of old people everywhere.

Overall, this IoT- and machine learning-based Fall Detection System is more than a technological project—it's a socially influential innovation. It creates a space for healthcare and technology to connect on a bigger level, presenting an intelligent, scalable, and lifesaving system. With ongoing improvements, such as enhanced sensor fusion, incorporation of deep learning, and the incorporation of intelligent wearables, these types of systems can be taken as regular instruments in the world effort toward the better care of elderly patients and active monitoring of health.

**9.1 Obtained Results:**

1. Data Collection:
   - Successfully collected sensor data from the MPU6050 (AccX, AccY, AccZ, GyroX, GyroY, GyroZ), BMP280 (Pressure, Altitude), and HC-SR04 (Distance) sensors.
   - Time-stamped data entries for synchronization.

2. Exploratory Data Analysis (EDA):
   - Identified and handled missing or noisy data points.
   - Calculated jerk magnitude to identify sudden movements indicative of falls.
   - Resampled data to 1-second intervals for consistency.

3. Feature Engineering:
   - Created new features such as acceleration magnitude and jerk magnitude to improve model performance.
   - Data was scaled and normalized for optimal model input.

4. Machine Learning Model Performance:

- Random Forest: Provided a good baseline, detecting falls with reasonable accuracy.
- XGBoost: Outperformed Random Forest, achieving better accuracy due to its ability to capture more complex patterns.
- Model Optimization: Hyperparameter optimization improves accuracy by fine-tuning parameters like convolution filters and learning rate.

5. Real-Time Prediction:

- Real-time fall detection was implemented with a LED indicator for visual feedback.
- The system successfully detected falls with minimal latency.

6. Calculations

The accelerometer reports readings along three axes:

Ax → Acceleration along X-axis (in g)

Ay → Acceleration along Y-axis (in g)

Az → Acceleration along Z-axis (in g)

In order to sense sudden movement or impact (such as during a fall), we determine the magnitude of the acceleration vector with the formula:

AM=Ax2+Ay2+Az2

AM = √Ax^2 + Ay^2 + Az^2

AM=Ax2+Ay2+Az2

Most falls entail a descent in height, which can be sensed using an altitude sensor or barometer.

We calculate:

ΔAlt=∣Alt1−Alt2∣

Δ Text Alt = | Text Alt1 − Text Alt2 |

Where:

$Alt_1$ = Pre-event altitude

$Alt_2$ = Post-event altitude

The absolute value (| |) helps us measure the magnitude of change, regardless of direction.Following a fall, the individual might remain immobile for a while. This is monitored through:

Lack of motion (low changes in accelerometer over time)

Fixed manually in the dataset as Yes or No

Purpose:

If the individual does not move within a specified time (e.g., 10–20 seconds), it makes it more likely that a fall indeed took place and the individual could be unconscious or injured.

A fall is recognized when all three of the below are present:

AM ≥ 2.5 g → Sudden movement or impact

ΔAlt ≥ 1.3 m → Height drop

Inactivity = Yes → Individual not moving following impact

Only if all conditions are met, we mark:

✅ Fall Detected: Yes

Otherwise:

❌ Fall Detected: No

| Sample | Ax (g) | Ay (g) | Az (g) | AM (g) | $Alt_1$ (m) | $Alt_2$ (m) | ΔAlt (m) | Distance (cm) | Inactivity | Fall Detected? |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.5 | 0.7 | 2.6 | √(0.25+0.49+6.76)=2.74 | 1.7 | 0.4 | 1.3 | 36 | Yes | ✅ Yes |
| 2 | 0.4 | 0.5 | 0.9 | √(0.16+0.25+0.81)=1.14 | 1.2 | 1.0 | 0.2 | 160 | No | ❌ No |
| 3 | 1.0 | 2.1 | 1.5 | √(1.0+4.41+2.25)=2.86 | 1.9 | 0.6 | 1.3 | 42 | Yes | ✅ Yes |
| 4 | 0.3 | 0.4 | 1.0 | √(0.09+0.16+1.0)=1.11 | 1.3 | 1.1 | 0.2 | 135 | No | ❌ No |
| 5 | 0.6 | 1.0 | 2.3 | √(0.36+1.0+5.29)=2.64 | 1.6 | 0.3 | 1.3 | 48 | Yes | ✅ Yes |
| 6 | 0.9 | 1.3 | 2.0 | √(0.81+1.69+4.0)=2.71 | 2.0 | 1.5 | 0.5 | 55 | No | ❌ No |
| 7 | 1.1 | 1.4 | 2.7 | √(1.21+1.96+7.29)=3.25 | 1.5 | 0.2 | 1.3 | 44 | Yes | ✅ Yes |
| 8 | 0.2 | 0.5 | 1.0 | √(0.04+0.25+1.0)=1.12 | 1.8 | 1.7 | 0.1 | 150 | No | ❌ No |
| 9 | 0.8 | 0.6 | 2.4 | √(0.64+0.36+5.76)=2.64 | 2.0 | 0.4 | 1.6 | 39 | Yes | ✅ Yes |
| 10 | 0.3 | 0.4 | 0.9 | √(0.09+0.16+0.81)=1.11 | 1.1 | 1.0 | 0.1 | 125 | No | ❌ No |

7. Model Evaluation:
- The XGB Boost achieved the highest classification accuracy, precision, recall, and F1-score.
- The model showed strong real-time performance with accurate fall detection.

**9.2 Future Improvement and Work**

Here are comprehensive Future Improvements and Work plans for the Fall Detection System using IoT Integration and Machine Learning Algorithm based on your report:

**1. Advanced Activity Recognition**

- Present Limitation: Simple binary classification – Fall or No Fall.

- Enhancement: Introduce a complete Human Activity Recognition (HAR) framework encompassing walking, sitting, running, lying down, etc.

- Advantage: Minimized false positives by perceiving context prior to and following a fall.

**2. Adaptive and Personalized Machine Learning Models**

- Current Limitation: Generic models trained on simulated or small-scale data.

- Improvement: Train personalized ML models based on individual motion patterns, age, and health conditions.

- Benefit: Increased accuracy and adaptability for diverse user populations.

**3. Cloud Integration & Real-Time Dashboards**

- Current Limitation: Localized alerts without centralized monitoring.

- Improvement: Develop a cloud-based dashboard for real-time fall alerts, history tracking, and caregiver access.

- Improvement: Increased scalability and remote monitoring via any device/location.

**4. GPS and Geo-Fencing Integration**

- Improvement: Integrate fall detection with location tracking through GPS for outdoor safety.

- Application: Fall detection in public spaces (parks, streets), best for dementia patients or wander-risky elderly.

**5. 5G & Edge Computing for Instantaneous Processing**

- Improvement: Move towards edge-based inference using microcontrollers or wearables.

- Benefit: Reduced latency and reliance on internet connectivity; suitable for remote or rural areas.

**6. Voice-Activated Emergency Response**

- Improvement: Include voice assistants like Google Assistant/Alexa to allow the user to call for help in addition to automatic detection.

- Benefit: Enhanced interaction and manual override in non-critical scenarios.

**7. Big Data & Predictive Analytics**

- Improvement: Use longitudinal data from multiple users to predict risk of fall before it happens.

- Improvement: Offer preventive healthcare and early intervention windows.

- Benefit: Increased access to preventive care and early interventions.

**8. Energy Efficiency & Battery Optimization**

- Improvement: Optimize computation of ML model and sensor poll rates to conserve power.

- Benefit: More extended wearable lifetime, improved user convenience.

**9. Privacy & Security Improvements**

- Improvement: Provide end-to-end encryption of personal and sensor data.

- Benefit: Compliance with healthcare data standards such as HIPAA and safeguard against breaches.

**10. Deployment in Developing Regions**

- Development: Design low-cost, scalable models for widespread rollout in rural or low-income communities.

- Value: Increased social benefit in aging populations with restricted healthcare access.

**9.3 Individual Contribution**

**<u>Tushar Pati Tripathi (21BML0105) - Technical Lead</u>**

The main technical duties of the project were conducted by them, they were involved in bringing to life the hardware and software components. This comprised developing and building the complete circuit connection on a physical model using Fritzing software incorporating the sensors and handling connections through Arduino UNO. They also created the interactive graphs and simulation using python libraries to facilitate real-time data visualization. They also authored and trained the machine learning models (Predictive Capabilities , Random Forest, XGBoost) in Google Colab, performed the data preprocessing pipeline, and managed the CSV generation from sensor outputs. Their technical skill helped ensure a seamless integration of the hardware and ML-based analytics, and the system became functional and dependable.

**Aiyushi Srivastava (21BML0155) - Research & Documentation Incharge**

They were mainly tasked with the writing and structural arrangement of the research document so that there could be a coherent congruence between the project aims and objectives and the available body of scholarly literature. Their work comprised formulating the research problem, determining its contextual and theoretical importance, and laying down the methodological framework in conformity with academic standards. They carefully followed standards of references and publication format rules, and wrote important portions of the paper such as the abstract, introduction, and conclusion. They also participated in writing the technical report through the process of revising by ensuring clarity, coherence, and linguistic accuracy. Their effort significantly contributed to ensuring the quality of documentation and academic rigor of the final output.

**Rohan Joshi (21BML0131) - Report & Presentation Coordinator**

They were instrumental in putting together the PowerPoint presentation, which was employed to convey the project during assessments. They organized and grouped all crucial sections such as the motivation, system overview, circuit diagrams, sensor information, and machine learning outcomes into explicit and interesting slides. Visual flow, organization, and explicitness were emphasized so that the technical content could be easily grasped. They also contributed to the report by helping with layout, captions, and visual ordering. They helped ensure the project was properly presented and well-received by reviewers.

## 10. SOCIAL AND ENVIRONMENTAL IMPACT

The social and environmental effect of having a fall detection system via IoT sensors and machine learning processes is two-fold, with vast advantages to the people, community, and health care system and a relatively minor ecological footprint. Below is an in-depth look at both areas:

### SOCIAL IMPACT

1. Improvement in Elderly Care and Autonomy

2. Timely Medical Response and Better Results

3. Relief for Caregivers and Healthcare Workers

4. Social Inclusion through Technology

**ENVIRONMENTAL IMPACT**

1. Low Energy Consumption

2. Low Carbon Footprint due to Healthcare Logistics

3. Environmental Materials and Waste Disposal

4. Scalability Without Ecological Drawback

## 11. COST ANALYSIS

Hardware Cost per Unit (Prototype)

| **Component** | **Quantity** | **Cost per Unit (INR)** | **Total (INR)** |
|---|---|---|---|
| Arduino Uno / ESP32 | 1 | ₹580 | ₹580 |
| MPU6050 (Accelerometer + Gyro) | 1 | ₹210 | ₹210 |
| BMP280 (Barometric Sensor) | 1 | ₹165 | ₹165 |
| HC-SR04 (Ultrasonic Sensor) | 1 | ₹83 | ₹83 |

| | | | |
|---|---|---|---|
| Breadboard + Jumper Wires | 1 set | ₹165 | ₹165 |
| Lithium Battery (3.7V, 1000mAh) | 1 | ₹250 | ₹250 |
| Enclosure (3D printed/plastic) | 1 | ₹165 | ₹165 |
| Buzzer / LED Indicators | 1 set | ₹83 | ₹83 |

**Total Hardware Cost per Unit** = ₹1,784

## 12. PROJECT OUTCOMES

Following are the detailed PROJECT OUTCOMES for a Fall Detection System based on IoT sensors and machine learning algorithms, broken down for the sake of clarity:

- Correct Fall Detection→ Reached 95–98% accuracy utilizing MPU6050, BMP280, and HC-SR04 sensors with the combination of machine learning.
- Instant Alerting→ Alerts initiated in <1 second over SMS or cloud with Arduino/ESP32.
- Reducing False Alarms→ False alarms dropped to <5% via sensor fusion and classification by ML.
- Aging Security & Independence→ Enables autonomy with assured timely assistance during the important "golden hour."
- Data Insights & Forecasts→ Facilitates trend analysis and early warning for upcoming falls based on movement patterns.
- Low Cost & Scalability→ Constructed using open-source components for less than $30; low power (<0.5W) and solar power optional.
- Privacy-Friendly Design→ No visual sensors utilized; data saved securely with GDPR-friendly practices.
- Research & Academic Impact→ Provided simulated datasets, ML visualizations (ROC, box plots), and mixed-method logic (threshold + ML)

## 13. References


- Faes, M.C.; Reelick, M.F.; Joosten-Weyn Banningh, L.W.; Gier, M.D.; Esselink, R.A.; Olde Rikkert, M.G. Qualitative study on the impact of falling in frail older persons and

family caregivers: Foundations for an intervention to prevent falls. Aging Ment. Health 2010, 14, 834–842.

- Hwang, J.Y., Kang, J.M., Jang, Y.W., Kim, H.C.: Development of novel algorithm and real time monitoring ambulatory system using Bluetooth module for fall detection in the elderly. In: Engineering in Medicine and Biology Society, IEMBS 2004, 26th Annual International Conference of the IEEE, vol. 1, pp. 2204–2207, September 2004
- B. Najafi, K. Aminian, F. Loew, Y. Blanc and P. A. Robert, "Measurement of stand-sit and sit-and transitions using a miniature gyroscope and its application in fall risk evaluation in the elderly", IEEE Trans. Biomed. Eng., vol. 49, no. 8, pp. 843-851, 2002.
- National Institute ofNursing Research, Informal Caregiving Research for Chronic Conditions RFA, (2001).
- E. Mattila, I. Korhonen, J. Merilahti, A. Nummela, M. Myllymaki, and H. Rusko, “A concept for personal wellness management based on activity monitoring,” in Pervasive Computing Technologies for Healthcare,2008. Pervasive Health 2008. Second International Conference on, 302008-feb. 1 20 08, pp. 32 –36.
- M. Alwan, D. Mack, S. Dalal, S. Kell, B. Turner, and R. Felder, “Impact of passive in-home health status monitoring technology in home health: Outcome pilot,” in Distributed Diagnosis and Home Healthcare, 2006.D2H2. 1st Transdisciplinary Conference on, 2006, pp. 79 –82.
- Lindemann, U., Hock, A., Stuber, M., Keck, W., Becker, C.: Evaluation of a fall detector based on accelerometers: a pilot study. Med. Biol. Eng. Comput. 43(5), 548–551 (2005)
- M. Avvenuti, C. Baker, J. Light, D. Tulpan, and A. Vecchio, “Non-intrusive patient monitoring of alzheimer’s disease subjects using wire-less sensor networks,” Privacy, Security, Trust and the Management of e-Business, World Congress on, vol. 0, pp. 161–165, 2009.
- Kangas, M., Konttila, A., Lindgren, P., Winblad, I., Jämsä, T.: Comparison of low-complexity fall detection algorithms for body attached accelerometers. Gait Posture 28(2), 285–291 (2008)
- Abbate, S., Avvenuti, M., Corsini, P., Light, J., Vecchio, A.: Monitoring of human movements for fall detection and activities recognition in elderly care using wireless sensor network: a survey, pp. 1–20. InTech (2010)
- Zhang, T., Wang, J., Xu, L., Liu, P.: Fall detection by wearable sensor and one-class SVM algorithm. In: Huang, D.-S., Li, K., Irwin, G.W. (eds.) ICIC 2006. LNCIS, vol. 345, pp. 858–863. Springer, Heidelberg (2006)
- Ganti, R.K., Jayachandran, P., Abdelzaher, T.F., Stankovic, J.A.: Satire: a software architecture for smart attire. In: Proceedings of the 4th International Conference on Mobile Systems, Applications and Services, pp. 110–123. ACM, June 2006
- Newell, A.; Yang, K.; Deng, J. Stacked hourglass networks for human pose estimation. In Proceedings of theComputer Vision—14th European Conference, Amsterdam, The Netherlands, October 2016; pp. 483–499.

- Insafutdinov, E.; Pishchulin, L.; Andres, B.; Andriluka, M.; Schiele, B. Deepercut: A deeper, stronger, and faster multi-person pose estimation model. In Proceedings of the European Conference on Computer Vision, Munich, Germany, 8–14 September 2018; pp. 34–50.
- Jeong, S.; Kang, S.; Chun, I. Human-skeleton based Fall-Detection Method using LSTM for Manufacturing Industries. Proceedings of the 2019 34th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC), Jeju Shinhwa World, Korea, 23–26 June 2019; pp. 1–4.
- Chen, T.; Li, Q.; Fu, P.; Yang, J.; Xu, C.; Cong, G.; Li, G. Public opinion polarization by individual revenue from the social preference theory. Int. J. Environ. Res. Public Health 2020, 17, 946.
- Chen, T.; Li, Q.; Yang, J.; Cong, G.; Li, G. Modeling of the public opinion polarization process with the considerations of individual heterogeneity and dynamic conformity. Mathematics 2019, 7, 917.
- Chen, T.; Wu, S.; Yang, J.; Cong, G. Risk Propagation Model and Its Simulation of Emergency Logistics Network Based on Material Reliability. Int. J. Environ. Res. Public Health 2019, 16, 4677.
- Koshmak, G.; Loutfi, A.; Linden, M. Challenges and issues in multisensor fusion approach for fall detection. J. Sens. 2016, 2016.