Tunisian Republic

Ministry of Higher Education and Scientific Research

University of Carthage

National Institute of Applied Sciences and Technology

Report

Subject: Smart Devices
Specialty: ICT Engineering
Level of study: 3rd year

Smart Fall Detection Bracelet for Elderly People

Prepared by :

Yosr Attia
Nermine Cheriaa
Hanine Ben Amor
Selim Soussi

Professor:
Mr. Emir Damergi

Academic year:
2024-2025

Acknowledgments

We would like to express our deepest gratitude to everyone who contributed to the successful completion of this project.

First and foremost, we extend our heartfelt thanks to our supervisor, Mr. Emir Damergi, for his invaluable guidance, insightful feedback, and constant encouragement throughout the project. His expertise and support were instrumental in shaping the direction and scope of our work. We deeply appreciate his dedication in meeting with the group on multiple occasions to ensure our progress remained on track, offering practical suggestions and instrumental advice that significantly contributed to the development of our project.

We are also grateful to our university, the National Institute of Applied Sciences and Technology (INSAT), for providing the resources and facilities necessary to bring this project to fruition.

Lastly, we thank our families and friends for their unwavering support and patience during the whole progress of this project.

This report is the culmination of collective effort, and we are deeply appreciative of everyone who played a role in its realization.

List of Figures

Content Table

Abstract

It is of little surprise that falls are often regarded as an inevitable part of aging; however, it is the severe consequences of these incidents, rather than their occurrence, that pose the greatest concern. Elderly individuals, particularly those with conditions such as Alzheimer's disease or mobility impairments, are more susceptible to falls due to frailty, reduced stability, and slower reaction times. Recognizing these challenges, both researchers and industry professionals have proposed various solutions to assist the elderly and their caregivers by detecting falls and triggering timely notifications.

In this project, we present a Smart Fall Detection Bracelet tailored for elderly individuals. This device incorporates an ESP32 microcontroller and motion sensors (accelerometer and gyroscope) to monitor real-time movements and detect falls based on advanced AI algorithms. When a fall is detected, the system sends an immediate alert via MQTT to notify caregivers or family members through a broker server. The bracelet not only publishes fall alerts but also subscribes to remote commands, ensuring seamless communication and swift response.

This innovative IoT solution offers portability, reliability, and real-time connectivity, providing an efficient means to enhance the safety and quality of life for seniors and their caregivers. The report explores the design, implementation, and testing of the device, highlighting its significant potential in addressing the critical challenges associated with elderly care.

Keywords: fall detection, elderly care, IoT, ESP32, accelerometer, MQTT, real-time monitoring, wearable technology, safety, sensor.

I. Theoretical study

1. Problem Statement

The decline of the birth rate and the prolongation of life span lead to the aging of the population, which has become a worldwide problem. According to research, the elderly population will increase dramatically in the future, and the proportion of the elderly in the world population will continue to grow, which is expected to reach 28% in 2050. Aging is accompanied by a decline in human function, which increases the risk of falls. According to statistics, falls among the elderly are increasingly being recognized as an issue of concern. Any individual, especially if an aging person, can be negatively affected in various ways as a result of falling. Even a small fall can profoundly affect the health of elderly people. Yet, in this population, falls continue to be a predominant cause of loss of functioning and death.

The number of elderly people living alone has been continuously growing worldwide. This independence comes with the risk of not receiving prompt attention if an accident occurs. A third of people over 65 years old suffer, on average, one fall per year, and this number grows with age and previous falls. About one-third of those affected develop a fear of falling again. Not receiving attention in the first hour after an accident increases the risk of death and chronic affections.

Falls in the elderly may precipitate adverse physical, psychological, social, financial, medical, governmental, and community consequences:

- Physical consequences: are represented in possible injury i.e. broken bone or soft tissue injury, pain and discomfort, reduced mobility and possible long-term disability, and most of the time lead to increase the loss of independence and the ability to look after himself/herself.

- Psychological consequences: lead to loss of confidence when walking and moving due to the increased fear of repeated falling, fear and anxiety, in addition to distress, and embarrassment.
- Social consequences: are represented in curtailment of possible routine social activities, loss of independence and a reliance on family, friends or possible move into an aging residential or nursing care center.
- Financial and medical consequences: signify the direct costs of medical care associated with injuries represented in increased costs to the statutory services such as physiotherapy rehabilitation, hospital care, and social care. Also, the immediate impact and consequences from the falls seen in the health center were laceration (sometimes resulted in tissue damage) needing cleaning and dressing. For some patients, the wounds resulted in difficult to heal, chronic leg ulcers requiring frequent cleaning and dressing at the health center.
- Governmental and community consequences: indicate the fact that falls can have devastating effects on people's health and that falls greatly contribute to the level of hospital admissions and health insurance costs.

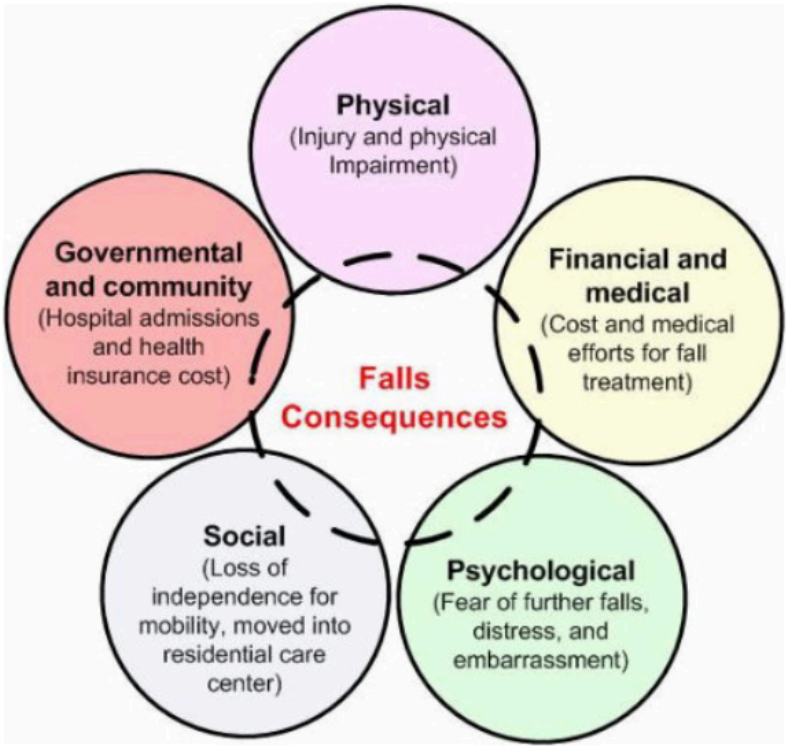


Figure 1 – ***The main consequences related to elderly falling***

2. Market Analysis

2.1. Market Trends and Size

Globally, the elderly care and health tech markets are expanding rapidly:

- The **elderly care market** is valued in the trillions globally, with significant growth driven by aging populations.
- The **wearable health technology market** was valued at $20 billion in 2023 and continues to grow annually.

2.2. Current Solutions and Challenges

Existing products, such as **Life Alert** and **iLife Fall Detection Sensors**, address fall detection but have limitations:

- Dependence on stationary base stations.
- Subscription fees that make them expensive in the long run.
- Limited mobility and features.

2.3. Competitive Landscape

Several existing solutions address fall detection but have limitations:

- **Life Alert Classic**: Requires a stationary base station and subscription fees.
- **iLife Fall Detection Sensor**: Limited mobility, no gyroscope integration, and reliance on a base station.

3. Target Audience

Independent Elderly Users:

- Characteristics: Older adults living alone or in assisted care settings.
- Need: Lightweight, unobtrusive design; reliable detection; simple operation.
- Concerns: Maintaining independence without constant supervision.

Family Caregivers:

- Characteristics: Adult children or professional caregivers responsible for elderly individuals.
- Needs: Instant notifications, remote monitoring capabilities, and reduced false alarms.

Healthcare Institutions:

- Characteristics: Nursing homes, hospitals, and rehabilitation centers.
- Needs: Scalable, reliable solutions to monitor multiple patients effectively.

4. Swot analysis

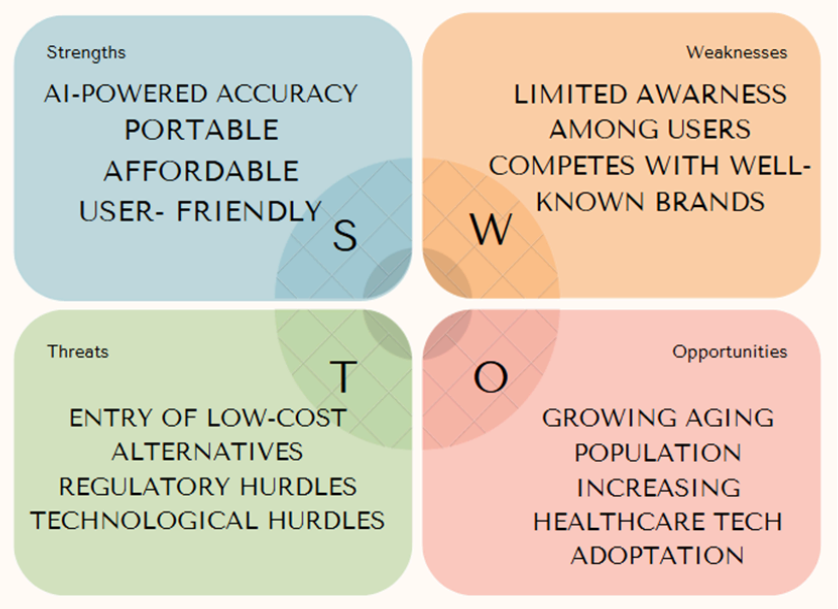


Figure 2 – *SWOT ANALYSIS OF THE SMART BRACELET*

5. Our Smart Fall Detection Bracelet for Elderly People

5.1. Presentation of the solution

The **Fall Detection Bracelet for Elderly People** is a wearable device designed to improve the safety and well-being of older adults who are at risk of falling. Falls are a serious issue among the elderly, leading to injuries that can affect their independence and quality of life. The bracelet uses a combination of AI algorithms trained on the SISFALL dataset, an ESP32 microcontroller, and motion sensors (accelerometer and gyroscope) to monitor movements in real time and detect falls.

When a fall is detected, the system sends an alert via MQTT to notify caregivers or family members. This ensures prompt action and reduces the risk of further complications. Unlike many existing solutions, this device is designed to be affordable, portable, and easy to use, without requiring any subscription fees or fixed base stations. With a user-friendly design, the bracelet ensures safety without interfering with daily activities. This innovative product has the potential to revolutionize elderly care, enhancing both safety and independence.

5.2. User Experience

Our smart system offers a complete user experience through a dedicated mobile application. The application allows the user to be notified in real time. In addition, it offers interactive features such as changing settings, receiving alerts in case of detected falls. Wi-Fi connectivity plays a very important role in the interaction between the application and the smart fall detection system, thus providing full control and continuous and remote monitoring.

5.3. Falling Scenario

The workflow of the system is as follows:

• The elderly people wear the falling detection sensor devices on their waist or wrist. The sensor device runs an algorithm embedded with it to detect and measure the body position of the users. If the sensor devices detect the body position in falling mode, the system will trigger the alarm.

• To prevent false positives from happening, a reset button to cancel the alarm is included. If the alarm was not canceled by the user, the system will send an emergency notification again to the contact person.

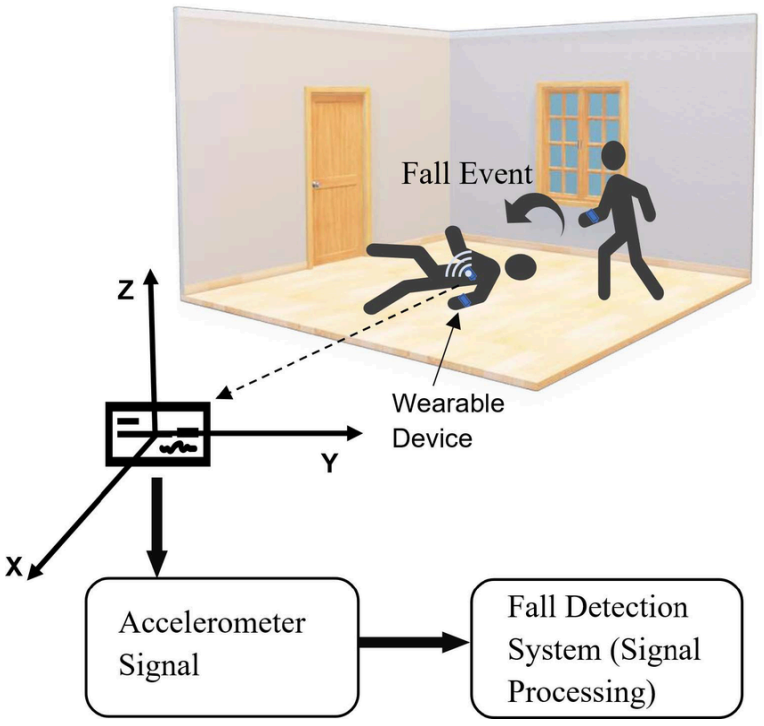


Figure 3 – ***Fall detection system workflow***

The storyboard of falling down until a family member and an ambulance or caregivers are informed is presented in Figure 4.

Figure 4 – ***Falling storyboard***

Here's the complete falling scenario for more clarification:

1. An elderly person is walking down stairs.
2. He misses the steps and falls down from the stairs to the floor.
3. Fall detection sensor system will detect the falling through the algorithm system and then send an alert message. If the alert notification is not reset, the system will send another alert message again and again until assistance to the elderly is given.

5.4. Advantages

- Healthcare Needs: Falls are the leading cause of injury-related fatalities among elderly individuals who often live alone and need reliable systems to ensure their safety in case of emergencies.

- Technological Edge: Integrating IoT and AI provides a highly accurate solution that outperforms traditional systems.

- Market Demand: The growing elderly population and adoption of wearable technology among Baby Boomers increases the demand for such safety-focused technologies.

5.5. Challenges

❖ Educating users about the benefits and usability of the bracelet.

❖ Addressing concerns about comfort, privacy, and false-positive alerts.

❖ Competing with established brands in the market while maintaining affordability

5.6. Estimated Costs of the Bracelet

Electronic Component	**Price (DT)**
ESP32 + Accelerometer MMA7760	55
Battery	50
Total	**105**

→ To make our system simple, functional and affordable to everyone at the same time, we chose not to include the gyroscope in it but instead put it as an optional feature and the final decision returns to the client whether he wants to add it to the system or not as a performance enhancement, so that anyone can benefit from our fall detection system regardless of his financial state.

5.7. Positioning of the Fall Detection Bracelet in the Market

- **AI-Driven Detection**: Combines accelerometer and gyroscope data for superior accuracy.
- **Mobility**: Operates independently of fixed stations, ensuring safety beyond the home.
- **Affordability**: Eliminates subscription fees, lowering the cost of ownership.

6. Introduction to ESP32

The ESP32, developed by Espressif Systems and manufactured by TSMC using a 40 nm process, is a series of low-cost, low-power system-on-chip microcontrollers designed for IoT applications. It integrates Wi-Fi and dual-mode Bluetooth connectivity, making it ideal for embedded devices such as mobile gadgets, wearable technology, and IoT systems. The ESP32 employs various microprocessors, including the Tensilica Xtensa LX6 (dual-core and single-core), Xtensa LX7 (dual-core), or a single-core RISC-V processor. It features built-in components like antenna switches, RF balun, power amplifier, low-noise receive amplifier, filters, and power-management modules. Widely available as standalone chips, modules, or development boards, the ESP32 has become a favorite among hobbyists and developers for its versatility, cost-effectiveness, and robust capabilities, earning a reputation as a go-to solution for both commercial and personal IoT projects.

Figure 3 – *ESP32 Shield*

6.1. ESP32 features and specifications

Here's a high-level summary of the features and specifications of the ESP32:

ESP-32	DESCRIPTION
Core	2
Architecture	32 bits
Clock	Tensilica Xtensa LX106 160-240MHz
WiFi	IEEE802.11 b/g/n
Bluetooth	Yes - classic & BLE
RAM	520KB
Flash	External QSPI - 16MB
GPIO	22
DAC	2
ADC	18
Interfaces	SPI-I2C-UART-I2S-CAN

7. Introduction to Arduino:

7.1. What is Arduino?

Arduino is an open-source electronics platform based on easy-to-use hardware and software. Arduino boards are able to read inputs - light on a sensor, a finger on a button, or a Twitter message - and turn it into an output - activating a motor, turning on an LED, publishing something online. You can tell your board what to do by sending a set of instructions to the microcontroller on the board. To do so you use the Arduino programming language (based on Wiring), and the Arduino Software (IDE), based on Processing.

Figure 4 – *Arduino Logo*

7.2. Why Arduino?

Arduino is a versatile, open-source microcontroller platform widely used for projects in programming, robotics, art, and science due to its simplicity and accessibility. It offers an easy-to-use software environment that runs on Windows, Mac, and Linux, making it beginner-friendly while remaining flexible for advanced users. Arduino is favored by teachers, students, and makers for its affordability (with boards costing under $50), cross-platform compatibility, and straightforward programming interface based on Processing. Its open-source nature extends to both hardware and software, allowing users to modify and expand its capabilities. This accessibility and adaptability have made Arduino a key tool for learning, prototyping, and creative experimentation in countless applications.

7.3. Using the Arduino Software (IDE)

The Arduino Software (IDE) makes it easy to write code and upload it to the board offline. We recommend it for users with poor or no internet connection. This software can be used with any Arduino board.

The editor contains the five main areas:

1. A **toolbar with buttons** for common functions and a series of menus. The toolbar buttons allow you to verify and upload programs, create, open, and save sketches, choose your board and port and open the serial monitor.

2. The **Sidebar** for regularly used tools. It gives you quick access to board managers, libraries, debugging your board as well as a search and replacement tool.

3. The **text editor** for writing your code.

4. **Console controls** give control over the output on the console.

5. The **text console** displays text output by the Arduino Software (IDE), including complete error messages and other information.

The bottom right-hand corner of the window displays the configured board and serial port.

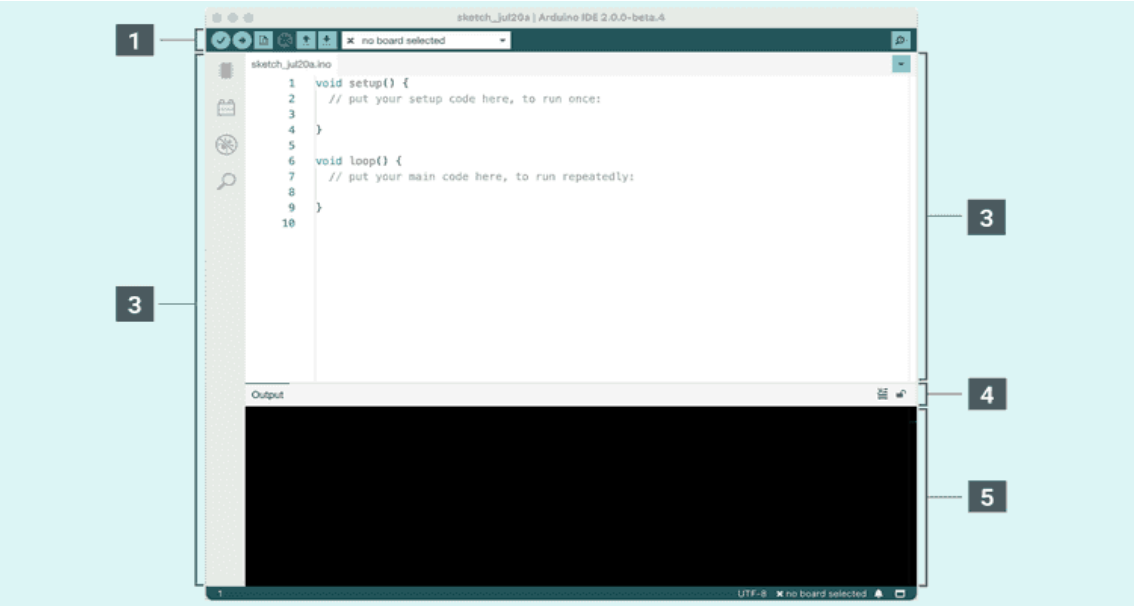


Figure 5 – *The Arduino Software IDE*

7.4. Overview of the Arduino IDE v2.x.x

The Arduino IDE 2 features a new sidebar, making the most commonly used tools more accessible. That's why we opted for this version to work on in our project.

Figure 6 – *Arduino IDE 2*

- **Verify / Upload** - compile and upload your code to your Arduino Board.
- **Select Board & Port** - detected Arduino boards automatically show up here, along with the port number.
- **Sketchbook** - here you will find all of your sketches locally stored on your computer. Additionally, you can sync with the Arduino Cloud, and also obtain your sketches from the online environment.
- **Boards Manager** - browse through Arduino & third party packages that can be installed. For example, using a MKR WiFi 1010 board requires the Arduino SAMD Boards package installed.
- **Library Manager** - browse through thousands of Arduino libraries, made by Arduino & its community
- **Debugger** - test and debug programs in real time.
- **Search** - search for keywords in your code.
- **Open Serial Monitor** - opens the Serial Monitor tool, as a new tab in the console.

8. MQTT: The Standard for IoT Messaging

MQTT is the most commonly used messaging protocol for the Internet of Things (IoT). MQTT stands for MQ Telemetry Transport. The protocol is a set of rules that defines how IoT devices can publish and subscribe to data over the Internet. MQTT is used for messaging and data exchange between IoT and industrial IoT (IIoT) devices, such as embedded devices, sensors, industrial PLCs, etc. The protocol is event driven and connects devices using the publish / subscribe (Pub/Sub) pattern. The sender (Publisher) and the receiver (Subscriber) communicate via Topics and are decoupled from each other. The connection between them is handled by the MQTT broker. The MQTT broker filters all incoming messages and distributes them correctly to the Subscribers.

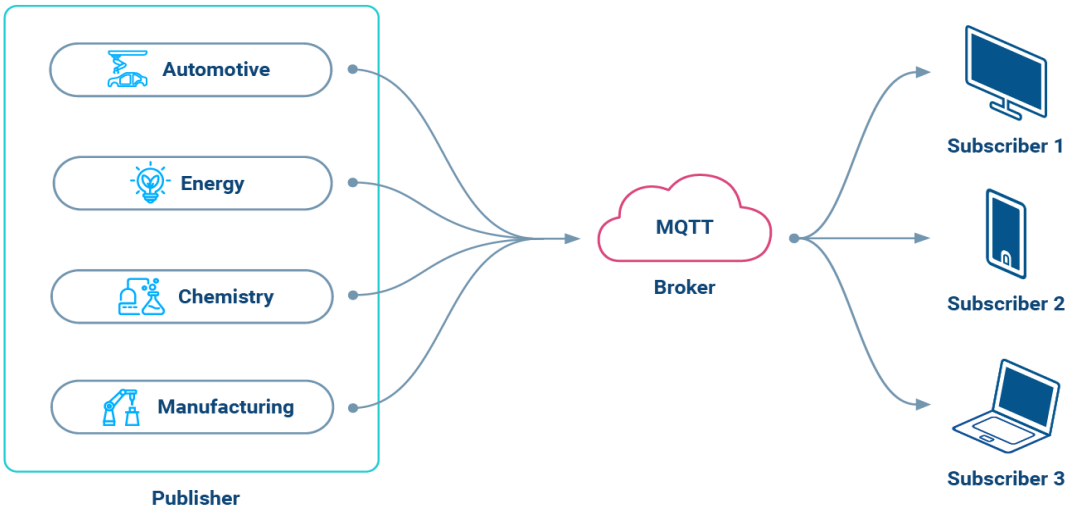


Figure 7 – *MQTT Pattern*

8.1. Why MQTT?

The MQTT protocol has the numerous benefits:

➔ **Lightweight and Efficient:** MQTT clients are very small, require minimal resources so can be used on small microcontrollers. MQTT message headers are small to optimize network bandwidth.

➔ **Bi-directional Communications:** MQTT allows for messaging between device to cloud and cloud to device. This makes for easy broadcasting messages to groups of things.

➔ **Scale to Millions of Things:** MQTT can scale to connect with millions of IoT devices.

➔ **Reliable Message Delivery:** Reliability of message delivery is important for many IoT use cases. This is why MQTT has 3 defined quality of service levels: 0 - at most once, 1- at least once, 2 - exactly once

➔ **Support for Unreliable Networks:** Many IoT devices connect over unreliable cellular networks. MQTT's support for persistent sessions reduces the time to reconnect the client with the broker.

➔ **Security Enabled:** MQTT makes it easy to encrypt messages using TLS and authenticate clients using modern authentication protocols, such as OAuth.

8.2. MQTT's Key Features

➔ MQTT Brokers: An MQTT Broker is the central server that receives messages from clients and routes them to the appropriate subscribers. There are Open Source MQTT Brokers as well as Commercial MQTT Brokers.

➔ MQTT Clients: An MQTT Client is any device or application that connects to the broker to publish (publisher) or subscribe (subscriber) to messages.

➔ MQTT Message Types: Fourteen standardized message types are used to connect and disconnect clients from brokers, broadcast data, acknowledge data receipts, and maintain the client-server connection. For data transfer, MQTT uses the TCP protocol. However, these three message types are the most commonly used:

- **CONNECT:** Initiates a connection between the client and the broker.
- **PUBLISH:** Sends data from the client to the broker.
- **SUBSCRIBE:** Registers a client to receive messages on specific topics

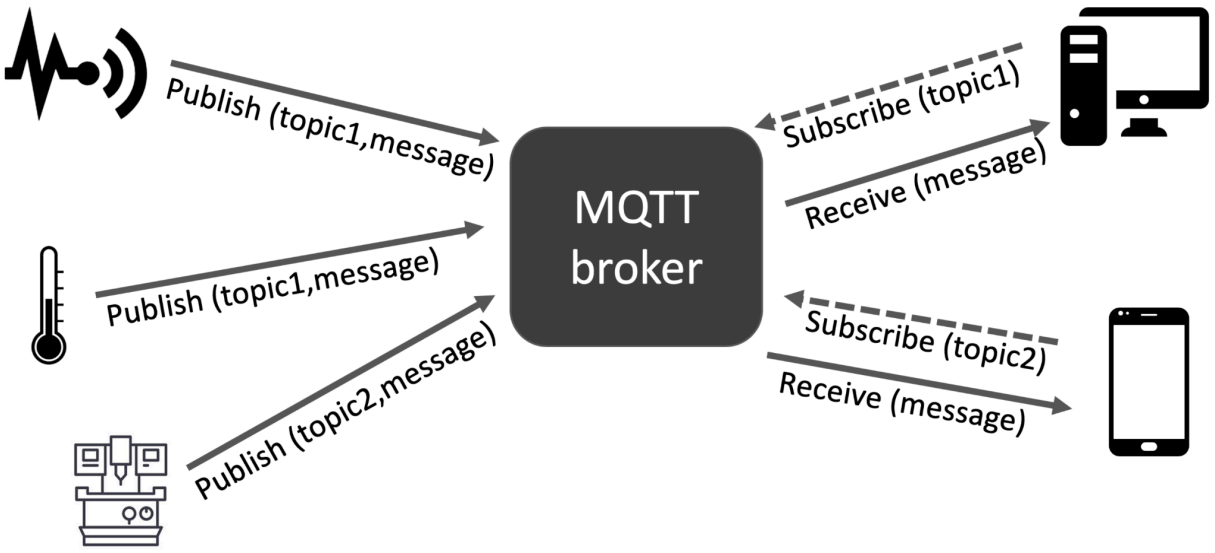


figure 8 – ***Key Features of MQTT***

8.3. How does MQTT protocol work?

The MQTT protocol begins with clients, which can be either publishers or subscribers, connecting to a broker. This connection is initiated by the client sending a CONNECT message, to which the broker responds with a CONNACK message, establishing the connection.

Once connected, a publisher client can send a PUBLISH message containing a topic and the data payload to the broker. The broker receives this message and checks which clients have subscribed to the specified topic. Clients interested in this topic would have previously sent a SUBSCRIBE message to the broker, which acknowledges these subscriptions with a SUBACK message.

When the broker identifies subscribers for the published topic, it forwards the message to these subscriber clients. Depending on the Quality of Service (QoS) level set, the broker and clients exchange acknowledgment messages to ensure reliable delivery. The three QoS levels include: QoS 0 (at most once), QoS 1 (at least once), and QoS 2 (exactly once).

To maintain the connection, clients periodically send PINGREQ messages if no data is being transmitted, and the broker responds with PINGRESP messages to confirm the connection is still active. When a client wants to disconnect, it sends a DISCONNECT message, and the broker acknowledges and closes the connection.

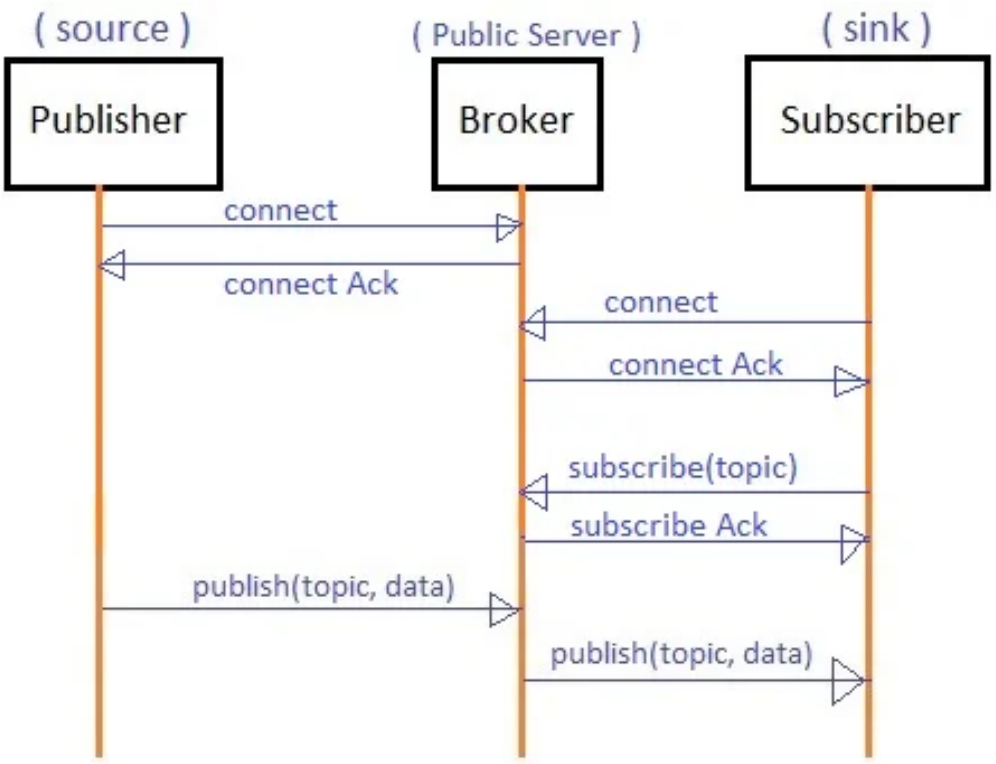


Figure 9 – *MQTT Protocol*

9. Introduction to AI and Machine Learning

9.1. Definition of Artificial intelligence

Artificial Intelligence (AI) refers to systems capable of performing tasks that usually require human intelligence.

Examples of AI Applications:

- ➢ Self-driving cars.
- ➢ Google Search engine.
- ➢ Facial recognition systems.
- ➢ Voice assistants like Siri or Alexa.

9.2. Definition of Machine Learning

Machine Learning (ML) is a subset of AI that focuses on using algorithms to analyze data and make predictions or decisions without being explicitly programmed. Machine Learning enables computers to learn and make predictions based on data.

Goal: To use algorithms to identify patterns and make decisions or predictions from data.

Example Applications:

- ➢ Google's self-driving cars.
- ➢ Spam detection in email systems.
- ➢ Fraud detection in banking.
- ➢ Facial recognition in security systems.

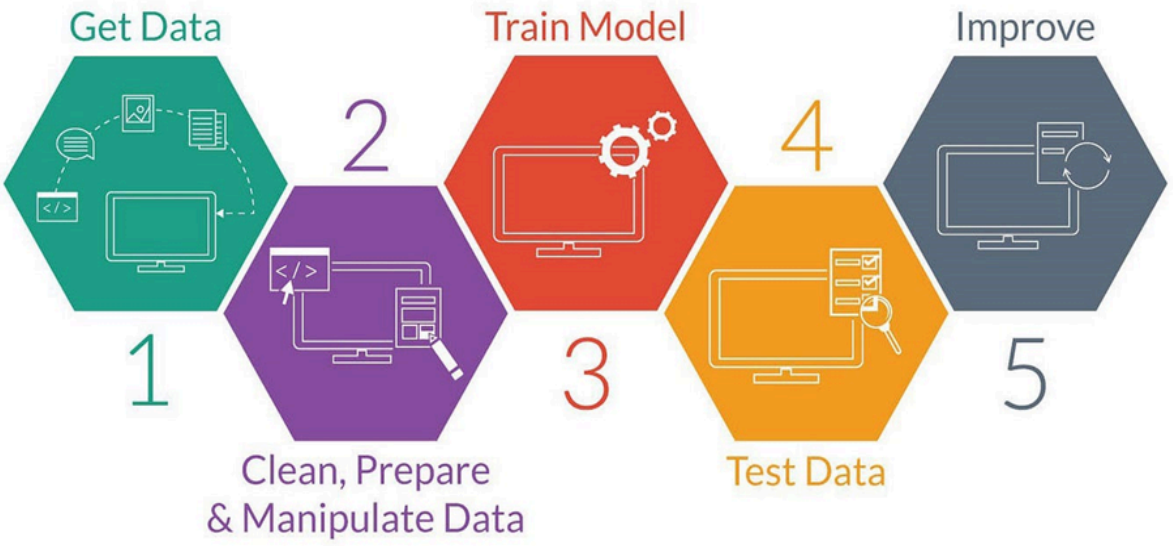


Figure 10 – ***ML workflow diagram***

9.3. Types of Machine Learning

9.3.1. Supervised Machine Learning

The algorithm learns from labeled data (inputs with corresponding outputs (e.g., detecting whether a fall occurred based on sensor readings).

• Common task: **Binary Classification**: Predicts one of two possible outcomes (e.g., fall detected: yes or no).

- Examples:
  - Spam detection in emails.
  - Fraud detection in banking transactions.

9.3.2. Unsupervised Machine Learning

This type of ML uses unlabeled data to find hidden patterns.
Example: Grouping activities from sensor data for behavioral insights.

9.3.3. Reinforcement Machine Learning

The system learns by interacting with its environment and receiving feedback.

9.4. Role of AI and Machine Learning in Fall Detection systems

9.4.1. Traditional fall detection systems

Traditional fall detection systems rely on simple sensors such as accelerometers and gyroscopes, often integrated into basic wearable devices with limited sensor capabilities. These systems use rule-based algorithms with fixed thresholds to detect falls. However, their methodology lacks advanced pattern recognition and relies on static decision-making. This approach results in moderate accuracy, ranging from 70-85%, and a high rate of false positives due to rigid detection criteria. On the plus side, traditional systems have a lower initial cost, use simple hardware, and are easy to deploy, though they are limited in scalability and adaptability.

9.4.2. ML-Based fall detection systems

In contrast, AI/ML-based systems leverage multi-modal sensors, advanced wearables, computer vision, and IoT device integration for comprehensive monitoring. These systems utilize machine learning models and deep learning neural networks to enable advanced pattern recognition and adaptive decision-making.

They achieve superior accuracy levels of 90-98% and maintain low false positive rates by incorporating advanced contextual awareness and adaptive learning capabilities. While the initial investment for AI/ML-based systems is higher and requires complex infrastructure with data training, they offer significant scalability and improved long-term efficiency, making them ideal for dynamic and large-scale implementations.

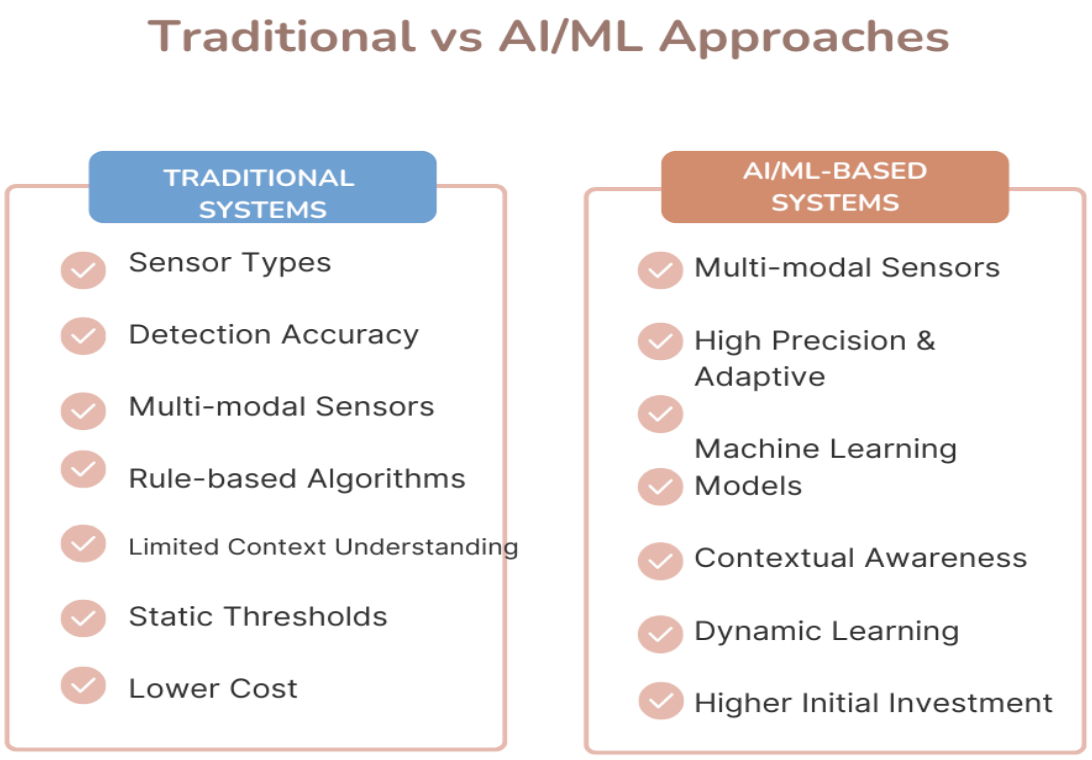


Figure 11 – *Traditional Vs. AI/ML-based fall detection systems*

9.5. Benefits and challenges of and Machine Learning in IOT

Benefits:

- **Automation**: Reduces manual monitoring efforts.
- **Accuracy**: Continuously improves through learning.
- **Scalability**: Handles large datasets and adapts to new data.

Challenges:

- **Data Dependency**: Requires extensive, high-quality datasets for training.
- **Complexity**: Algorithms can be computationally intensive.
- **Ethical Concerns**: Privacy and security in data collection and usage.

II. Practical Study

1. Installation Steps

This section outlines the tools, libraries, and steps used to implement the fall detection system. The project required two main development environments: Arduino IDE for microcontroller programming and Visual Studio Code (VS Code) for AI model development. The system was implemented on the ESP32 microcontroller and integrated with various libraries for both sensor data processing and machine learning inference.

1.1. Development Environments

1.1.1. Arduino IDE for ESP32 Programming

The Arduino IDE was used to write and upload code to the ESP32 microcontroller.

Steps to Set Up Arduino IDE for ESP32:

1. Download and Install Arduino IDE:
    - Available at Software | Arduino

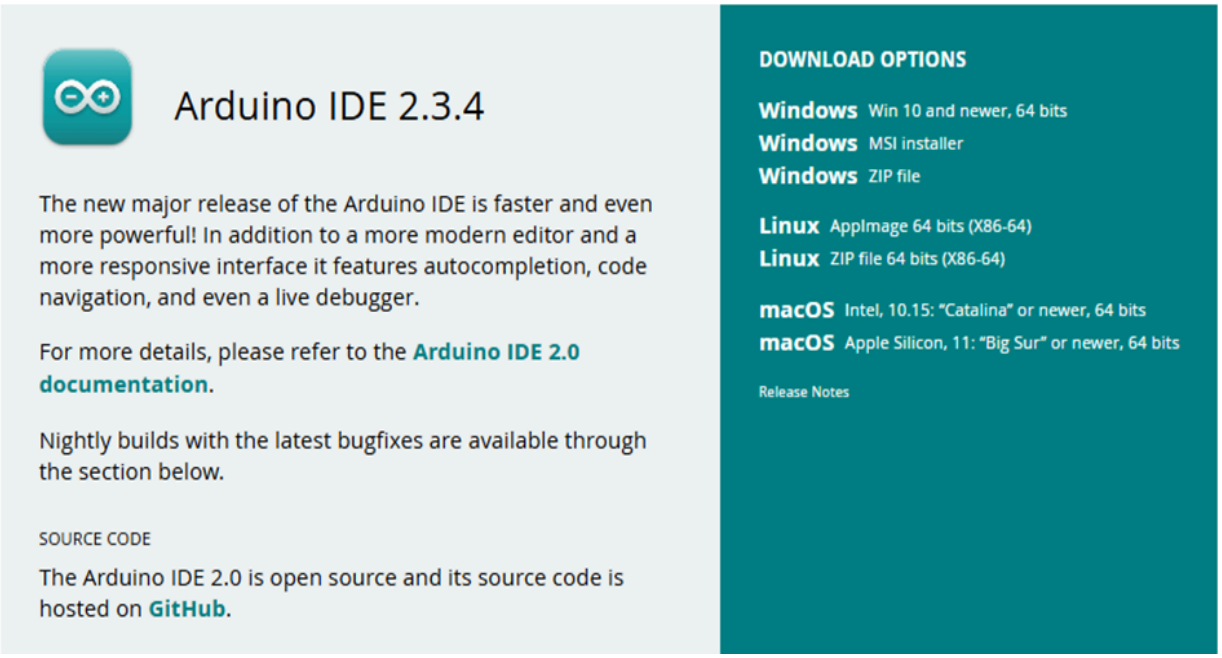


Figure 12 – *Arduino IDE 2.3.4*

2. Add ESP32 Board Support:

In the Arduino IDE: Open **Tools > Board > Boards Manager** and search for **ESP32**. Click **Install**.

3. Connect ESP32:
    - Connect the ESP32 to the computer via USB.
    - Select the appropriate **board** (e.g., ESP32 Dev Module) and **port** from the **Tools** menu.

1.1.2. Visual Studio Code for AI Model Development

VS Code was used for developing and training the machine learning model using TensorFlow and Keras.

Steps to Set Up VS Code for AI Development:

1. Install Visual Studio Code:
    - Download from Visual Studio Code - Code Editing. Redefined

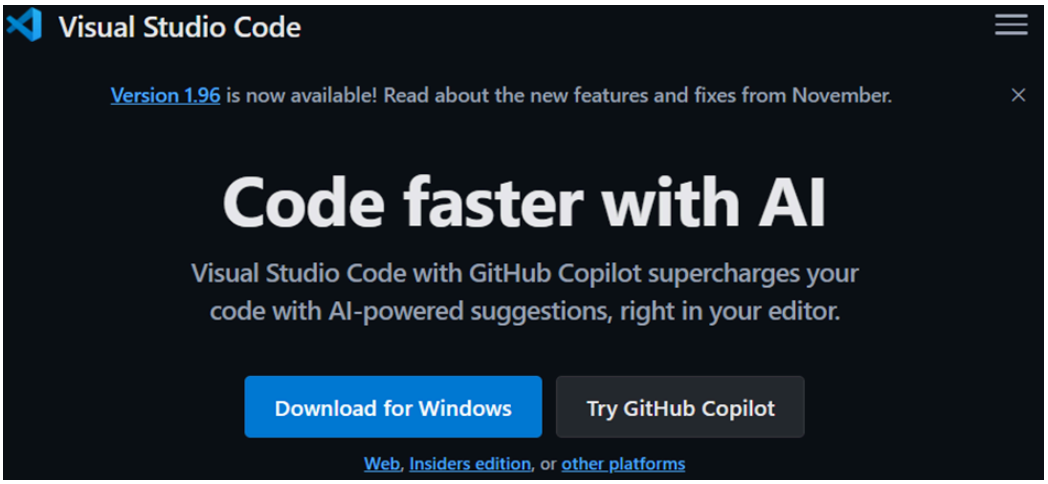


Figure 13 – *Visual Studio Code v1.96*

2. Install Python and Required Extensions:
    - Download Python from Download Python | Python.org

Figure 14 – *Python v 3.13.1 Download Page*

- Install extensions in VS Code like **Python** and **Pylance** for Python development.

3. Create a Virtual Environment:
    - For a better experience and in order to eliminate independencies interfering, create a virtual environment by running the following commands:

      python -m env
      source env/bin/activate

1.2. Required Libraries and Installation

1.2.1. Arduino Required Libraries

Open Tools > Manage Libraries.

- Search for each library among the following and click **Install:**
    - **WiFi**: For ESP32's Wi-Fi connectivity.
    - **PubSubClient**: For MQTT communication Management.
    - **Wire**: For I2C communication with peripherals like gyroscope or other peripherals.
    - **TensorFlow Lite for Microcontrollers**: For running the TensorFlow Lite model on the ESP32, enabling AI interference directly on the microcontroller.

1.2.2. Python Libraries

- **TensorFlow/Keras**: Used to train the machine learning model and export it as a TensorFlow Lite format for ESP32.
- **pandas and numpy**: For data manipulation and preparation.
- **scikit-learn**: For preprocessing and splitting data into training and testing sets.

To **install** the Python Libraries:

- Use pip commands in the virtual environment:
  pip install tensorflow keras pandas numpy sklearn

2. Model Training

2.1. Dataset Overview

Description:

The SisFall Dataset is a comprehensive dataset developed and published on 20 January 2017 to address the challenges of fall detection in elderly individuals. It includes recordings of both falls and activities of daily living (ADLs) using wearable devices. This dataset stands out for incorporating data from both young adults and elderly participants, ensuring a broader representation of fall and movement patterns.

Features:

- **Participants:**

This database was generated with collaboration of 38 volunteers divided into two groups: elderly people and young adults. Elderly people group was formed by 15 participants (8 male and 7 female), a group formed by retired employees of the Universidad de Antioquia and parents of current employees. The young adults group was formed by 23 participants (11 male and 12 female). The table below shows their ages, heights and weights.

	Sex	Age	Height (m)	Weight (kg)
Elderly	Female	62–75	1.50–1.69	50–72
	Male	60–71	1.63–1.71	56–102
Adult	Female	19–30	1.49–1.69	42–63
	Male	19–30	1.65–1.83	58–81

Figure 15 – ***Age, height and weight of the participants***

- **Types of activities:**

The SisFall dataset includes a diverse set of **ADLs** and **FALLS** that represent routine movements performed by individuals during their everyday lives. These activities are critical for distinguishing between normal behaviors and falls in the dataset.

- 19 types of **ADLs**, including walking, jogging, sitting, bending, and jumping.
- 15 types of **falls**, such as forward, backward, and lateral falls caused by slipping, tripping, or fainting.

Code	Activity	Trials	Duration
F01	Fall forward while walking caused by a slip	5	15 s
F02	Fall backward while walking caused by a slip	5	15 s
F03	Lateral fall while walking caused by a slip	5	15 s
F04	Fall forward while walking caused by a trip	5	15 s
F05	Fall forward while jogging caused by a trip	5	15 s
F06	Vertical fall while walking caused by fainting	5	15 s
F07	Fall while walking, with use of hands in a table to dampen fall, caused by fainting	5	15 s
F08	Fall forward when trying to get up	5	15 s
F09	Lateral fall when trying to get up	5	15 s
F10	Fall forward when trying to sit down	5	15 s
F11	Fall backward when trying to sit down	5	15 s
F12	Lateral fall when trying to sit down	5	15 s
F13	Fall forward while sitting, caused by fainting or falling asleep	5	15 s
F14	Fall backward while sitting, caused by fainting or falling asleep	5	15 s
F15	Lateral fall while sitting, caused by fainting or falling asleep	5	15 s

Figure 16 – ***Types of falls selected***

Code	Activity	Trials	Duration
D01	Walking slowly	1	100 s
D02	Walking quickly	1	100 s
D03	Jogging slowly	1	100 s
D04	Jogging quickly	1	100 s
D05	Walking upstairs and downstairs slowly	5	25 s
D06	Walking upstairs and downstairs quickly	5	25 s
D07	Slowly sit in a half height chair, wait a moment, and up slowly	5	12 s
D08	Quickly sit in a half height chair, wait a moment, and up quickly	5	12 s
D09	Slowly sit in a low height chair, wait a moment, and up slowly	5	12 s
D10	Quickly sit in a low height chair, wait a moment, and up quickly	5	12 s
D11	Sitting a moment, trying to get up, and collapse into a chair	5	12 s
D12	Sitting a moment, lying slowly, wait a moment, and sit again	5	12 s
D13	Sitting a moment, lying quickly, wait a moment, and sit again	5	12 s
D14	Being on one's back change to lateral position, wait a moment, and change to one's back	5	12 s
D15	Standing, slowly bending at knees, and getting up	5	12 s
D16	Standing, slowly bending without bending knees, and getting up	5	12 s
D17	Standing, get into a car, remain seated and get out of the car	5	25 s
D18	Stumble while walking	5	12 s
D19	Gently jump without falling (trying to reach a high object)	5	12 s

Figure 17 – ***Types of activities of Daily living selected***

- **Sensor Configuration:**

Accelerometer and gyroscope data were recorded at a frequency of 200 Hz. The device was mounted on the waist to closely track the body's center of mass.

- **Recorded Data:**

Includes acceleration and rotation data across three axes: X, Y, Z. Data is labeled as either fall or ADL for supervised learning.

- **Data preprocessing:**

Preprocessing is critical in the performance of the classification algorithms and their computational burden. The process involves multiple stages to clean, transform, and structure the dataset, ensuring optimal performance and accuracy. The objective of this stage is to maximize the separation between ADL and Falls.

1. **Cleaning:**

Raw sensor data often contains noise, outliers, or irrelevant readings, especially during simulations or experiments.

- Noise Removal: Applied filters (e.g., low-pass) to smooth abrupt variations in sensor readings.
- Outlier Detection: Removed extreme values outside expected ranges to prevent skewed results.
- Handling Missing Data: Removed or interpolated incomplete records for consistency.

2. **Scaling**

Scaling normalizes the range of feature values, ensuring all accelerometer and gyroscope readings are on a comparable scale using StandardScaler to standardize data to a mean of 0 and standard deviation of 1, ensuring comparability across features.

3. **Encoding**

Converted categorical labels into numerical values:

- ➔ Fall: 1
- ➔ ADL: 0

4. **Splitting**

Divide data into:

- Training Set (70%): Model training.
- Validation Set (20%): Hyperparameter tuning.
- Test Set (10%): Final performance evaluation.

2.2. Model Design and Training

2.2.1 Architecture

The model architecture was designed using a 1D Convolutional Neural Network (Conv1D) for fall detection. This architecture is well-suited for **time-series data** such as accelerometer and gyroscope readings, as it efficiently captures patterns over sequences.

Architecture Details:

Input Layer:

Accepts normalized accelerometer data with a shape of (sequence_length, 1).

Convolutional Layers:

- First layer: 64 filters, kernel size of 3, ReLU activation, and same padding.
- Second layer: 32 filters, kernel size of 3, ReLU activation, and same padding.
- Flatten Layer: Converts the 2D data into a 1D feature vector.

Dense Layers:

- Fully connected layer with 128 units and ReLU activation.
- Output layer with 1 unit and sigmoid activation for binary classification (fall vs. no fall).

Optimizer and Loss Function:

- Optimizer: Adam, for efficient gradient descent.
- Loss Function: Binary Cross-Entropy, suitable for binary classification tasks.

```
def create_snn(input_shape):
    model = keras.Sequential([
        layers.Conv1D(filters=64, kernel_size=3, activation='relu', input_shape=input_shape, padding='same'),
        layers.Conv1D(filters=32, kernel_size=3, activation='relu', padding='same'),
        layers.Flatten(),
        layers.Dense(128, activation='relu'),
        layers.Dense(1, activation='sigmoid')  # Adjust output layer based on your task
    ])
    model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
    return model
```

Figure 18 – *Model architecture*

2.2.2 Training process

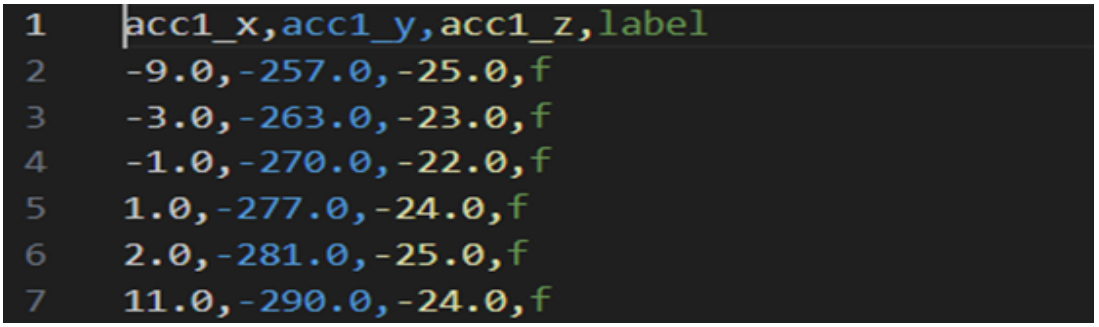
The training process involved loading, preprocessing, and normalizing the data, followed by training the Conv1D model.

Steps in Training

1. Data Loading:

Preprocessed data was loaded from a .pkl pre prepared file containing training, validation, and test datasets.

This is the way data is recleaned and organized:



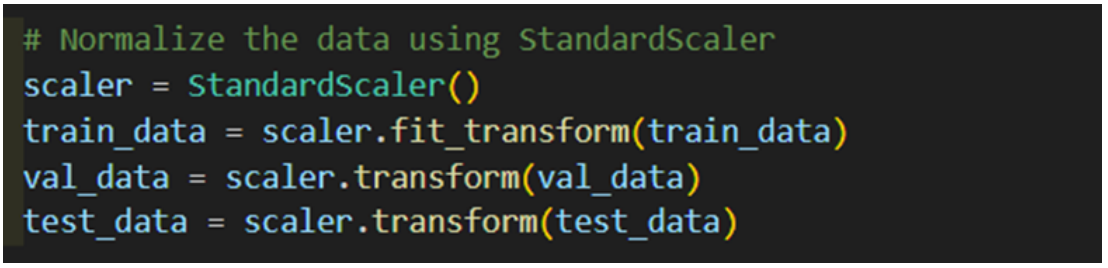
```
acc1_x,acc1_y,acc1_z,label
-9.0,-257.0,-25.0,f
-3.0,-263.0,-23.0,f
-1.0,-270.0,-22.0,f
1.0,-277.0,-24.0,f
2.0,-281.0,-25.0,f
11.0,-290.0,-24.0,f
```

Figure 19 – ***Dataset features after cleaning***

→ After that, labels were binary encoded to represent falls (1) and non-falls (0).

2. Data Normalization:

The features were scaled using StandardScaler to ensure consistent input ranges.



```
# Normalize the data using StandardScaler
scaler = StandardScaler()
train_data = scaler.fit_transform(train_data)
val_data = scaler.transform(val_data)
test_data = scaler.transform(test_data)
```

Figure 20 – ***Normalizing with standardScaler***

3. Data Reshaping:

The training, validation, and test data were reshaped to add a feature dimension, making the shape compatible with Conv1D ((samples, sequence_length, 1)).

4. Training Parameters and execution:

- **Epochs**: 10: the model will go through the entire training dataset **10 times** during the training process
- **Batch Size**: 32: The training data is divided into smaller subsets (batches) of **32 samples** each. Instead of processing the entire dataset at once, the model processes one batch at a time.

- **Validation**: Performed on the validation set to monitor the model's performance during training.
- **Execution**: The model was trained using the **fit** function, which tracked accuracy and loss over each epoch

```
# Train the model
model.fit(
    train_data,
    train_labels,
    epochs=10,
    batch_size=32,
    validation_data=(val_data, val_labels)
)       You, last week • Fall detection model for elderly read
```

Figure 21 – *Code for model training*

5. Model Saving:

- The trained model was saved as **snn_model.h5** for future inference.
- The scaler used for normalization was saved as **scaler.pkl** to ensure consistent preprocessing during testing.

2.2.3. Validation and test

Testing Procedure

- **Model Evaluation**: The trained model (snn_model.h5) was loaded for testing.
- **Data Preparation**: Test data (X_test) and labels (y_test) were loaded from the preprocessed dataset (preprocessed_data.pkl).
- **Prediction**: The model predicted labels using a threshold of 0.4 to classify falls (1) and non-falls (0).
- **Metrics**: The classification report was generated, detailing precision, recall, F1-score, and accuracy and visualizing true/false positives and negatives for detailed performance analysis.

Results Summary :

The classification report provided the following key metrics:

- **Precision**: 0.65 (fall), 0.46 (non-fall)
- **Recall**: 0.97 (fall), 0.05 (non-fall)
- **F1-Score**: 0.78 (fall), 0.08 (non-fall)
- **Accuracy**: 0.56
- **Macro Average**:
    - Precision: 0.58
    - Recall: 0.51
    - F1-Score: 0.43
- **Weighted Average**:
    - Precision: 0.65
    - Recall: 0.54
    - F1-Score: 0.54

95833/95833 ━━━━━━━━━━━━━━━━━━━━ 68s 706us/step
Classification Report:

	precision	recall	f1-score	support
0	0.65	0.97	0.78	1986757
1	0.46	0.05	0.08	1079870
accuracy			0.65	3066627
macro avg	0.56	0.51	0.43	3066627
weighted avg	0.58	0.65	0.54	3066627

Figure 22 – ***Classification Report Table***

The confusion matrix also provides a clear visual representation of the model's performance, especially for understanding true positives, false positives, true negatives, and false negatives.

It complements the classification report by offering a tangible perspective on where the model excels or struggles.

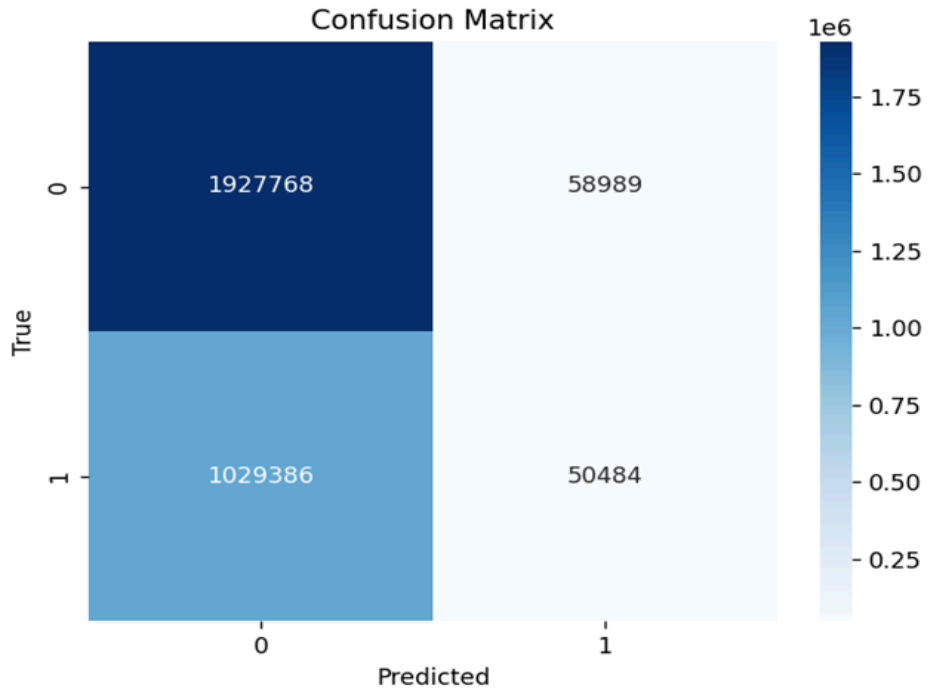


Figure 23 – *Confusion Matrix*

2.3. Model Deployment

The deployment phase involved converting the trained model to the TensorFlow Lite (TFLite) format, performing optimizations for edge devices, validating its performance, and preparing it for integration into embedded systems.

2.3.1. Model Conversion to TensorFlow Lite

The trained HDF5 model (snn_model.h5) was converted to TFLite with integer quantization for efficient inference. This step included optimization techniques and the use of a representative dataset to enable full integer quantization.

- **Model Conversion:**
  The trained model was loaded using TensorFlow's TFLite converter and converted to the TFLite format.

- **Optimizations:**
  Integer quantization was applied to reduce model size and improve inference efficiency. This optimization ensures the model operates efficiently on resource-constrained devices while maintaining acceptable accuracy.

```
# Load the HDF5 model
model = tf.keras.models.load_model(model_path)
# Convert to TensorFlow Lite
converter = tf.lite.TFLiteConverter.from_keras_model(model)
converter.optimizations = [tf.lite.Optimize.DEFAULT]
```

Figure 24 – *Converter initialization*

```
converter.representative_dataset = representative_dataset_gen
converter.target_spec.supported_ops = [tf.lite.OpsSet.TFLITE_BUILTINS_INT8]
converter.inference_input_type = tf.int8  # Set input to int8
converter.inference_output_type = tf.int8  # Set output to int8
# Convert the model
tflite_model = converter.convert()
```

Figure 25 – *Model conversion and optimization*

2.3.2. Model Testing

The converted TFLite model was rigorously tested to ensure compatibility and correctness with both dummy data and real sensor inputs.

- Dummy Data Testing: Initial tests were conducted using random inputs to validate the model's execution pipeline.
- Real Sensor Data Testing: Inputs were derived from accelerometer values (e.g., accx, accy, accz) to simulate actual use cases.

```
actual_input_quantized = np.round(actual_input / input_scale + input_zero_point).astype(np.int8)
# Set the tensor
interpreter.set_tensor(input_details[0]['index'], actual_input_quantized)
# Run the model
interpreter.invoke()
# Get the output
output_data = interpreter.get_tensor(output_details[0]['index'])
print(output_data)
```

Figure 26 – *Real Sensor Data Testing*

2.3.3. C Header File Generation

To enable integration into embedded systems, the optimized TFLite model was converted into a C header file. This file includes the model as a byte array, ready to be loaded directly onto microcontrollers.

Process:

- The TFLite model file was read and converted into a C array.
- A .h file was created containing the array and its length.

```
# Load the TFLite model file
with open("snn_model_int8.tflite", "rb") as f:
    model_data = f.read()
# Convert to a C array
c_array = ', '.join(f'0x{byte:02x}' for byte in model_data)
# Write to a .h file
with open("model_snn_fall_prediction.h", "w") as f:
    f.write("#ifndef MODEL_DATA_H\n#define MODEL_DATA_H\n\n")
    f.write(f"unsigned char model_data[] = {{ {c_array} }};\n")
    f.write(f"unsigned int model_data_len = {len(model_data)};\n\n")
    f.write("#endif // MODEL_DATA_H\n")
```

Figure 27 – *C Header File Generation*

3. Arduino Code and Model Implementation

3.1. System Workflow

Setup:

- Initializes the ESP32, connects to Wi-Fi, and establishes MQTT communication.
- Configures the MMA7660 accelerometer.

Continuous Monitoring:

- Reads acceleration data in the loop() function.
- Processes data using the TensorFlow Lite model for inference.
- Publishes alerts if a fall is detected.

Error Handling:

- Reconnects to Wi-Fi or MQTT broker if the connection is lost.
- Handles accelerometer or inference errors gracefully, ensuring robust functionality.

3.2. Wi-Fi and MQTT Communication

- The ESP32 connects to a Wi-Fi network using the SSID and password provided in the code.
- MQTT is utilized for communication between the ESP32 and the central server.
- MQTT Topics:
  - ➔ elderly/fall_alert: Used to publish alerts when a fall is detected.
  - ➔ elderly/control: Subscribed to receive control messages.

```
const char* mqtt_topic_pub = "elderly/fall_alert";
const char* mqtt_topic_sub = "elderly/control";
```

Figure 28 – *MQTT Topics*

The PubSubClient library handles MQTT functionality, ensuring connection persistence and reestablishment if interrupted:

```
// Connexion au serveur MQTT
PubSubClient client(espClient);
client.setServer(mqtt_server, mqtt_port);
client.setCallback(mqttCallback);
connectMQTT();
}
```

Figure 29 – *MQTT connection*

3.3. Accelerometer Integration

The MMA7660 accelerometer is used to capture real-time acceleration data along the x, y, and z axes.

The readAcceleration() function reads raw data from the accelerometer, converts it to human-readable values, and calculates the net acceleration using the formula:

$$Acceleration = \sqrt{x^2 + y^2 + z^2}$$

If the model using the calculated acceleration predicts a fall, an alert is published:

```
float ax, ay, az;
if (readAcceleration(&ax, &ay, &az)) {
  float acceleration = sqrt(ax * ax + ay * ay + az * az);
  Serial.printf("Acceleration: %.2f g\n", acceleration);
  if(predict_fall(&ax, &ay, &az)){
      Serial.println("Fall detected!");
      client.publish(mqtt_topic_pub, "Fall detected! Please check.");
      Serial.print("sent to sub");
  }
```

Figure 30 – *Fall prediction Logic using the calculated acceleration*

3.4. Integration of TensorFlow Lite Model

The TensorFlow Lite model, model_snn_fall_prediction.h, was imported to the ESP32 using the TensorFlow Lite Micro framework.

The model operates on accelerometer data (x, y, z axes) to infer whether a fall has occurred.

The predict_fall() function processes the data using the interpreter and outputs the probability of a fall or no-fall scenario.

If the probability of a fall exceeds that of no fall or if the confidence in "no fall" is below 20%, an alert is triggered.

```
bool predict_fall(float* ax, float* ay, float* az){
    if (interpreter->Invoke() != kTfLiteOk) {
    Serial.println("Error during inference!");
    return;
  }
   // Process the output
  float prob_not_fall = output->data.f[0];  // Probability of "Not Fall"
  float prob_fall = output->data.f[1];      // Probability of "Fall"
  if ((prob_fall > prob_not_fall) ||(proba_fall>0.2)) {
    return true;
    } else {
    return false;
  }
```

Figure 31 – *Fall prediction function*

4. Mobile Application Set Up

4.1. Installation

In Play Store or App store, research for **“ MQTT Alert for IOT ”** and install it

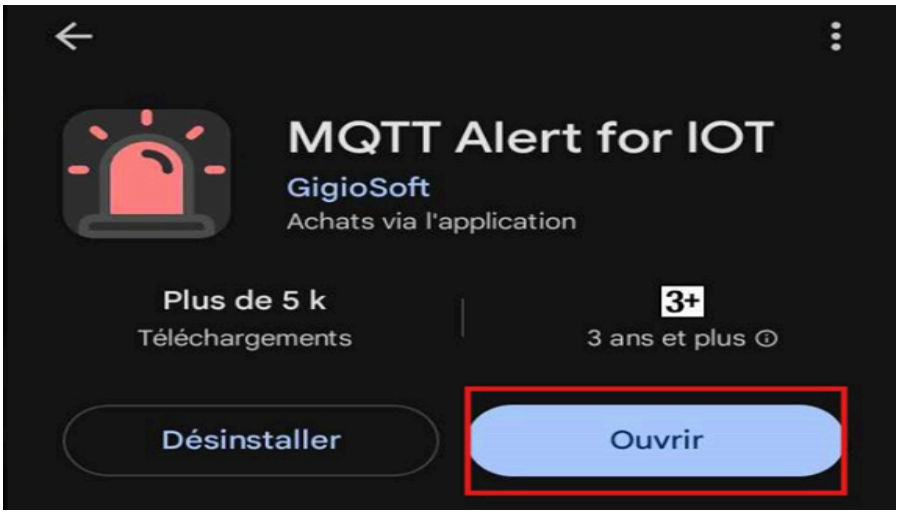


Figure 32 – ***Application in Playstore***

4.2. Configuration:

After installation, open the application and choose **Broker** as we will configure the MQTT broker in order to establish the connection between our two edge devices.

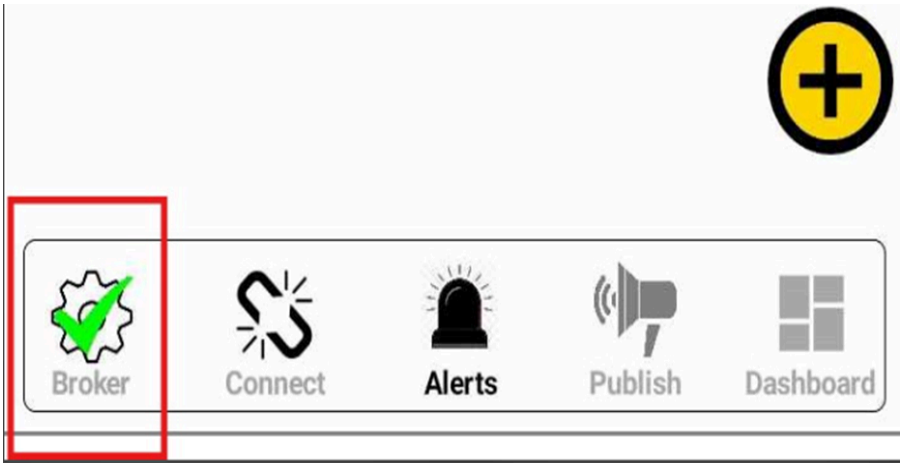


Figure 33 – ***Application configuration (Step 1)***

The configuration consists of naming the connection, setting up the broker server and port, setting up the client username and password, without forgetting configs such as clean session, autoconnect, keep Alive etc.

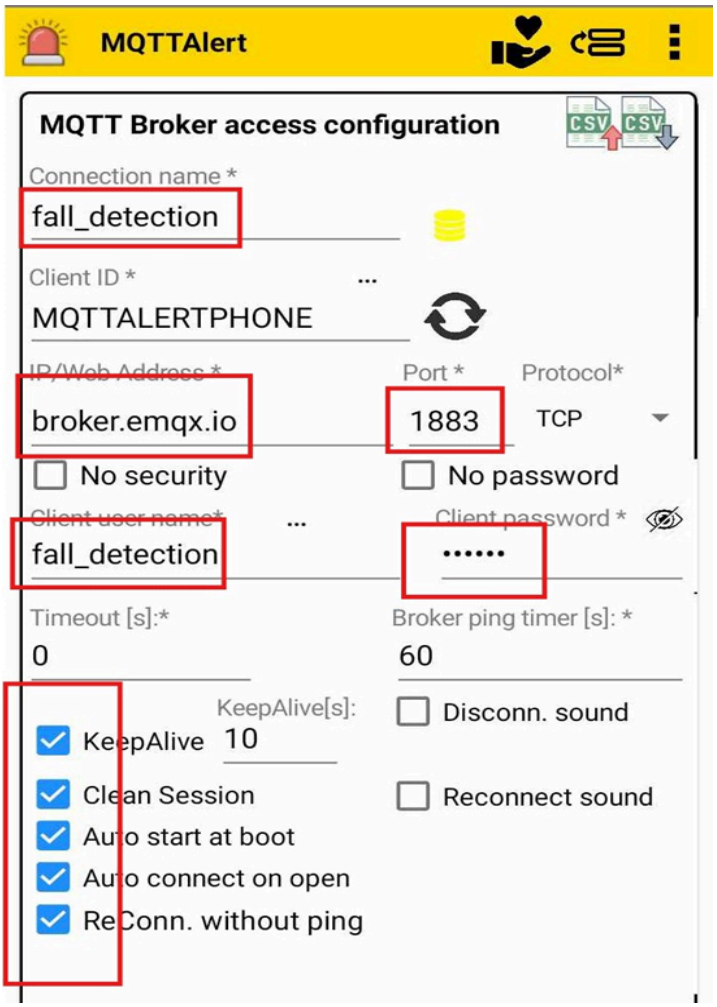


Figure 34 – *MQTT Broker access configuration*

After that, we verify our connection has been established. Then, we move to the next step: the MQTT alert configuration

Figure 35 – Application configuration (Step 2)

Next step consists of setting up the MQTT Alert configurations such as the MQTT event name and topic, the Monitor option indicates that the alarm will repop-up for the user 10 times until he opens the message.

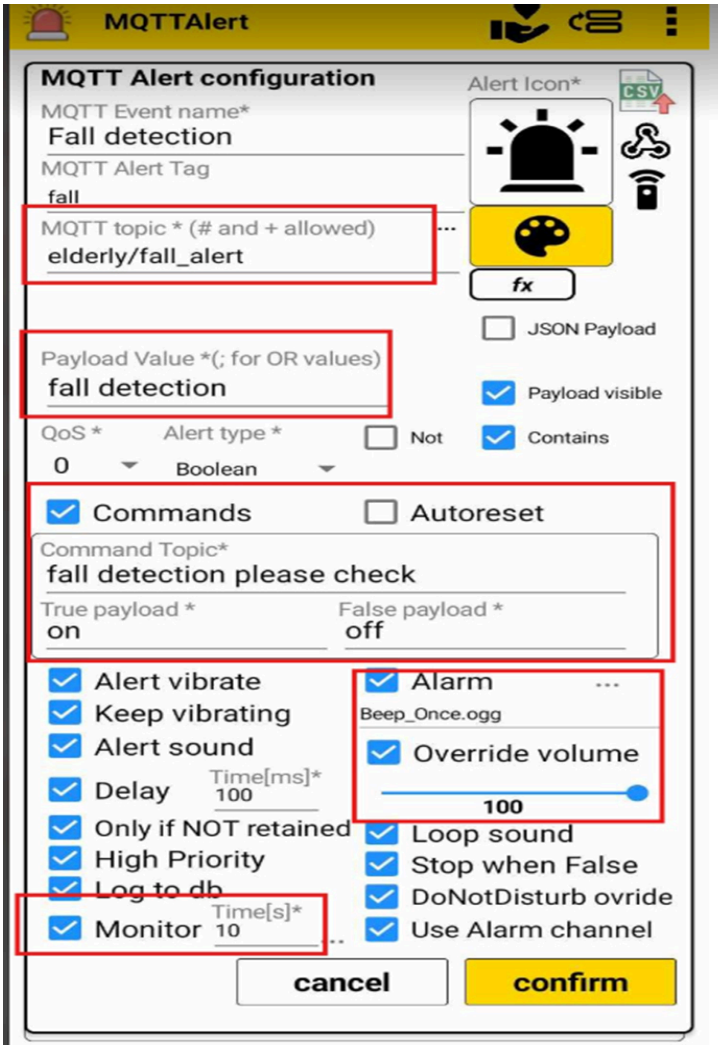


Figure 36 – *MQTT Alert configuration*

After that, we need to enable the connection

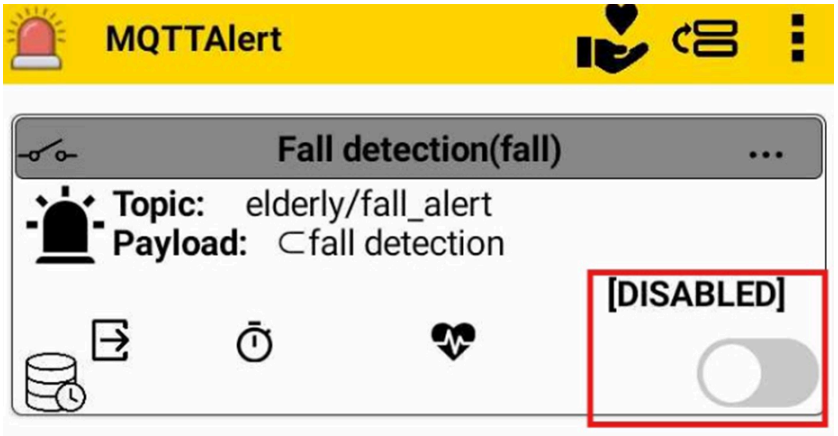


Figure 37 – Enable Alert

This ready green flag should appear once it's ready as shown below:

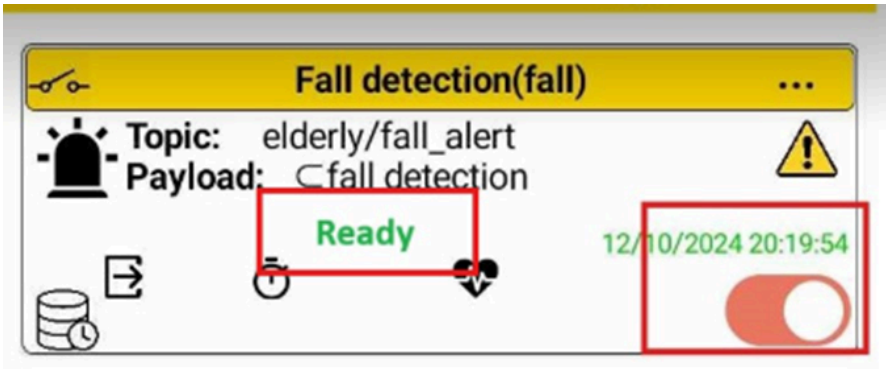


Figure 38 – *Established configurations' verification*

All that remains is going to our mobile settings and allow the MQTT Alert application to send notifications

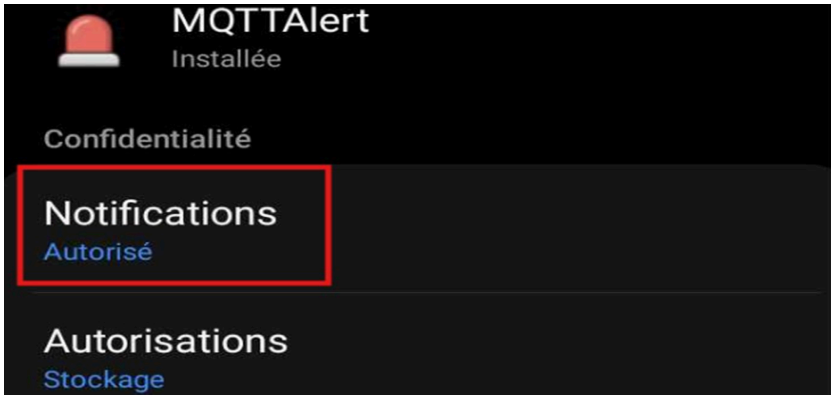


Figure 39 – Enable notifications parameter in mobile phone

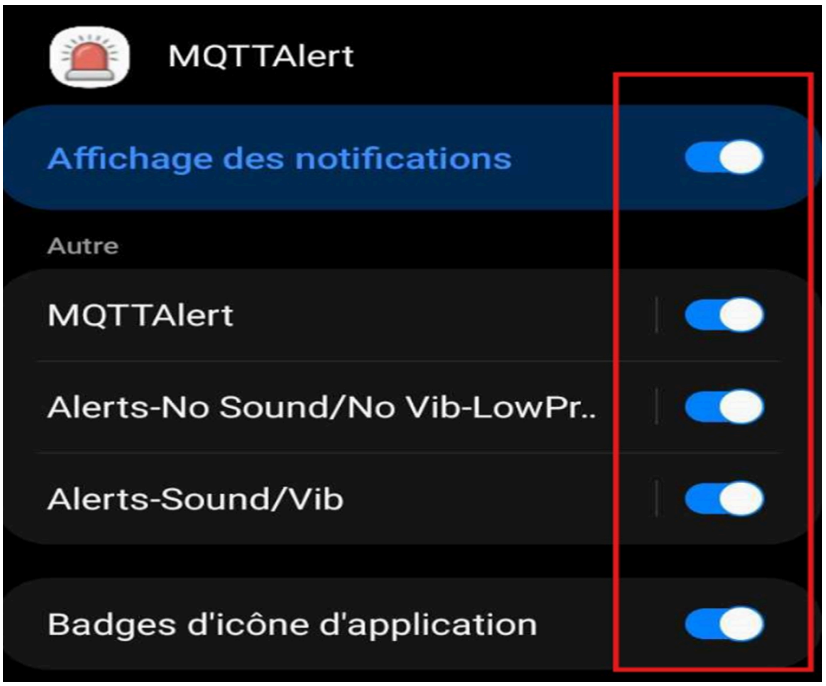


Figure 40 – *Setting up notifications for MQTT Alert app*

5. Execution and Tests

Once all the prerequisite configuration is done correctly, it's time to simulate a fall and test our fall detection system reactivity.

The first step was to test that the connection to the MQTT broker is established. The following output confirmed it was successfully done:

```
16:49:13.666 -> bscribed to topic: elderly/control
16:49:13.666 -> Acceleration: 1.04 g
16:49:14.153 -> MQTT disconnected! Reconnecting...
16:49:14.153 -> Connecting to MQTT broker...
16:49:30.366 -> .....
16:49:32.374 -> Wi-Fi connected successfully.
16:49:32.374 -> IP Address: 192.168.145.33
```

Figure 41 – MQTT connection test output

The next step was to simulate a real fall scenario and see the results instantly.

Firstly, when the app is already open in the smartphone:

As it is shown in the top screen of the mobile application, a new MQTT Alert event as well as a message "Fall detected! Please check" with all the fall details such as date and time of its occurrence are sent in the meantime the fall is detected.

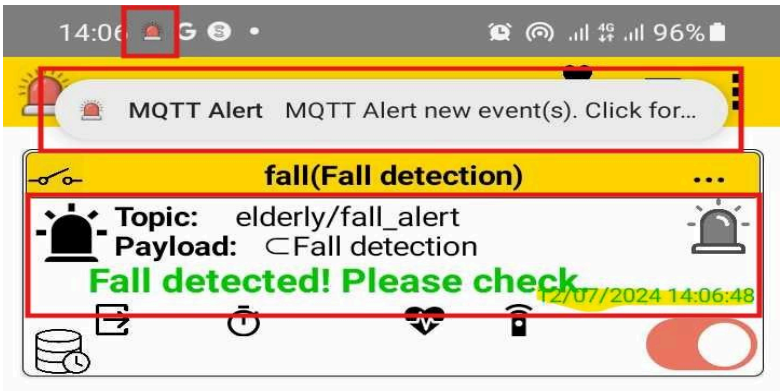


Figure 42 – MQTT Alert message "Fall detected!"

Once we regain control over the situation, we can manually reset the alert:

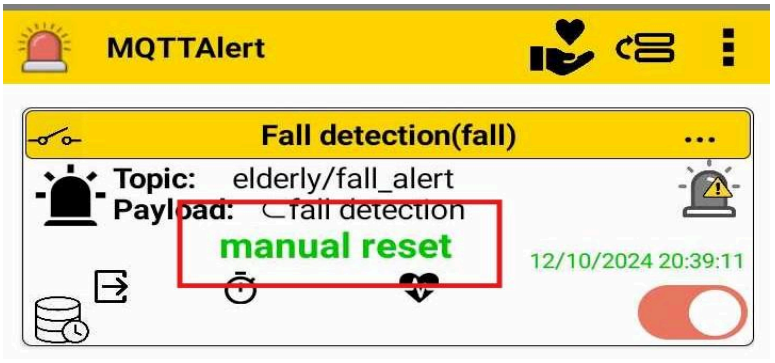


Figure 43 – MQTT Alert Manual reset

Secondly, we even simulated a fall from outside the MQTT Alert Application (when the mobile application and/or the smartphone are close). This didn't prevent the message of the MQTT Alert from being sent as the phone suddenly opened and a popped-up notification appeared indicating a new event was detected:

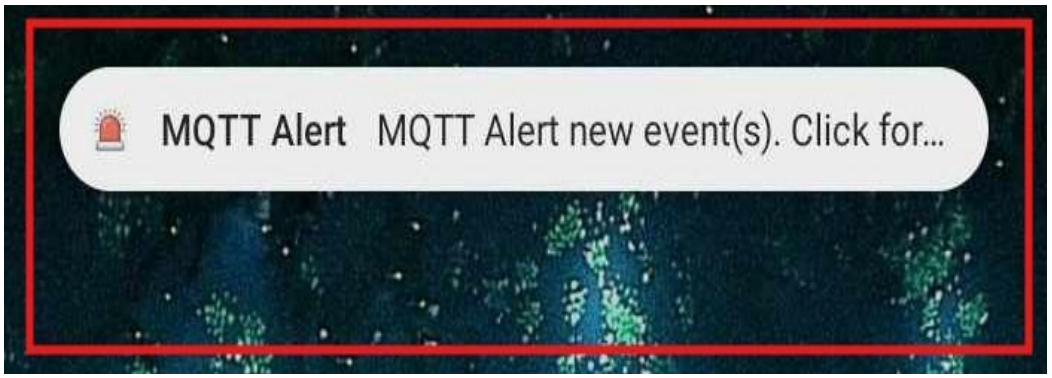


Figure 44 – *MQTT Alert popped-up notification*

We only need to develop this notification by simply clicking on it to be redirected to the fall detected message and all the other details of the event.

6. Problems Encountered and Solutions

6.1 Absence of Gyroscope Data in the ESP32 Sensor

Problem:

The dataset used for training included gyroscope data alongside accelerometer data. However, the ESP32 sensor hardware only provides accelerometer data. This mismatch created inconsistencies between the data used during training and the data available during real-time inference.

- Gyroscope data provides additional motion insights (angular velocity), which improves the accuracy of fall detection models.
- Training the model with features unavailable at inference time could result in unreliable predictions.

Solution:

To address this issue, we recleaned the dataset by removing gyroscope data and retaining only the accelerometer readings (X, Y, and Z axes). This ensured the model was trained on data that aligned with the real-world sensor inputs, making sure its performance was not reliant on unavailable data, avoiding unnecessary bias, overfitting, or reliance on unavailable features.

Impact:

- While the cleaning process ensured compatibility with the ESP32 sensor, it also reduced the amount of data available for training, which may have slightly decreased the model's accuracy.
- Gyroscope data loss limited the ability to detect complex motion patterns, such as rotational falls.

6.2 Accuracy of the Model and Hardware Constraints

Problem:

The initial model accuracy was around 80%, which was deemed insufficient.
The accuracy constraints can be explained by ***two factors:***

- **Hardware Constraints**: The ESP32 microcontroller has limited **processing power, memory**, and **energy efficiency**, making it unsuitable for heavy machine learning models.
- We could not implement a complex (heavy) model architecture due to the limited computational power and memory of the ESP32 microcontroller, which inherently limited its capacity to learn intricate patterns from the data.
- When converting the model to TensorFlow Lite format with integer quantization (from 32-bit floating-point to 8-bit integers), the accuracy dropped significantly due to reduced precision.

Solution:

- Model Architecture: A lightweight model was designed and optimized to balance resource constraints and performance. Techniques such as reducing the number of layers, using simpler activation functions, and minimizing parameters were applied.
- Quantization: Despite the quantization-induced accuracy loss, the trade-off was necessary to meet the hardware limitations of the ESP32.

Impact:

The simplified architecture ensured the model could run on the ESP32 efficiently but compromised detection precision, because despite the optimizations, the final accuracy of the quantized model was still not sufficient for real-time fall detection.

6.3 Issues with TensorFlow Lite Library Importation

Problem:

During the final stages of deployment, we encountered technical difficulties in importing the TensorFlow Lite library onto the ESP32 microcontroller.

As a result, the trained and quantized TFLite model could not be deployed on the hardware.

Solution:

To overcome this challenge, we pivoted to an alternative solution: Instead of deploying the TFLite model, we implemented a *threshold-based approach* for fall detection.

This method calculates the resultant acceleration (a) from the accelerometer's X, Y and Z axis values using the formula:

$$Acceleration = \sqrt{x^2 + y^2 + z^2}$$

If the resultant acceleration exceeds a predefined threshold, it is considered indicative of a fall.

Impact:

While this approach does not leverage machine learning, it ensures basic fall detection functionality. The threshold-based method is *computationally efficient* and well-suited for the ESP32's constraints, though it lacks adaptability and precision compared to a trained model.

Conclusion

The Smart Fall Detection Bracelet for Elderly People marks a significant achievement in utilizing IoT and AI technologies to address critical challenges in elderly care. By integrating advanced motion sensors, machine learning algorithms, and MQTT-based real-time notifications, this innovative device ensures timely detection and response to falls, significantly enhancing the safety and independence of older adults.

This project was not only a testament to the application of theoretical knowledge gained through coursework but also an invaluable opportunity to broaden our horizons. It allowed us to deepen our understanding of key concepts in IoT, embedded systems, and machine learning while exploring new technologies and methodologies to create a functional and impactful solution.

Despite certain challenges, such as hardware constraints and the need for further optimization of model accuracy, the project successfully demonstrated the feasibility and potential of wearable technology to improve the quality of life for its users. Moving forward, future improvements could include enhancing model precision, integrating additional sensors, and scaling the system for broader applications.

Ultimately, this work highlights the transformative role of technology in fostering safer and more inclusive environments for vulnerable populations while underscoring the importance of hands-on projects in bridging academic learning with real-world innovation.

Bibliography

[1] Arduino IDE https://www.arduino.cc/en/software

[2] Download Python https://www.python.org/downloads/

[3] ESP32 - Wikipedia https://en.wikipedia.org/wiki/ESP32

[4] ESP32 for IoT: A Complete Guide https://www.nabto.com/guide-to-iot-esp-32/

[5] Getting Started with Arduino IDE 2
https://docs.arduino.cc/software/ide-v2/tutorials/getting-started-ide-v2/

[6] MQTT https://mqtt.org/

[7] MQTT Essentials - All Core Concepts Explained https://www.hivemq.com/mqtt/

[8] SisFall: A Fall and Movement Dataset
https://pmc.ncbi.nlm.nih.gov/articles/PMC5298771/

[9] Using the Arduino Software (IDE)
https://docs.arduino.cc/learn/starting-guide/the-arduino-software-ide/

[10] Visual Studio Code https://code.visualstudio.com/

[11] What is MQTT? https://www.haproxy.com/glossary/what-is-mqtt

[12] What it is, How MQTT Protocol Works
https://macautoinc.com/industrial-communication-protocols/mqtt/

Appendix : Arduino Code

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <Wire.h>

// Informations Wi-Fi
const char* ssid = "Galaxy A22D7B6";
const char* password = "nwqr5916";

// Informations du serveur MQTT
const char* mqtt_server = "broker.emqx.io";
const int mqtt_port = 1883;
const char* mqttUser = "fall_detection";
const char* mqttPassword = "azerty";
const char* mqtt_topic_pub = "elderly/fall_alert";
const char* mqtt_topic_sub = "elderly/control";


WiFiClient espClient;
PubSubClient client(espClient);

// Paramètres de seuil pour détecter une chute
const float FALL_THRESHOLD = 2.5;  // Seuil d'accélération (en g)


void mqttCallback(char* topic, byte* payload, unsigned int length) {
  Serial.printf("Message received on topic %s: ", topic);
  for (unsigned int i = 0; i < length; i++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();
}

void setup() {
  Serial.begin(115200);
  Wire.begin();

  // Initialize MMA7660
  if (!mmaInit()) {
    Serial.println("MMA7660 initialization failed!");
    while (1);
  }
```

```
  Serial.println("MMA7660 initialized successfully!");

  // Connexion à Wi-Fi
  connectWiFi();

  // Connexion au serveur MQTT
  client.setServer(mqtt_server, mqtt_port);
  client.setCallback(mqttCallback);
  connectMQTT();
}

void loop() {
  client.loop();

  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("Wi-Fi disconnected! Reconnecting...");
    connectWiFi();
  }

  if (!client.connected()) {
    Serial.println("MQTT disconnected! Reconnecting...");
    connectMQTT();
  }


  float ax, ay, az;
  if (readAcceleration(&ax, &ay, &az)) {
    float acceleration = sqrt(ax * ax + ay * ay + az * az);
    Serial.printf("Acceleration: %.2f g\n", acceleration);

    if (acceleration > FALL_THRESHOLD) {
      Serial.println("Fall detected!");
      client.publish(mqtt_topic_pub, "Fall detected! Please check.");
      Serial.print("sent to sub");
    }
  }
  delay(500);
}

void connectWiFi() {
  Serial.println("Connecting to Wi-Fi...");
  WiFi.begin(ssid, password);
  unsigned long startAttemptTime = millis();
  while (WiFi.status() != WL_CONNECTED && millis() - startAttemptTime <
10000) {
```

```
    delay(500);
    Serial.print(".");
  }

  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nWi-Fi connected successfully.");
    Serial.print("IP Address: ");
    Serial.println(WiFi.localIP());
  } else {
    Serial.println("\nWi-Fi connection failed. Restarting...");
    ESP.restart();
  }
}

void connectMQTT() {
  while (!client.connected()) {
    Serial.println("Connecting to MQTT broker...");
    if (client.connect("ESP32Client")) {
      Serial.println("MQTT connected successfully.");
      client.subscribe(mqtt_topic_sub);  // Subscribe to topic
      Serial.printf("Subscribed to topic: %s\n", mqtt_topic_sub);
    } else {
      Serial.print("MQTT connection failed. State: ");
      Serial.println(client.state());
      delay(2000);
    }
  }
}

bool mmaInit() {
  Wire.beginTransmission(0x4C);
  Wire.write(0x07);
  Wire.write(0x01);
  return (Wire.endTransmission() == 0);
}

bool readAcceleration(float* ax, float* ay, float* az) {
  Wire.beginTransmission(0x4C);
  Wire.write(0x00);
  if (Wire.endTransmission() != 0) {
    return false;
  }

  Wire.requestFrom(0x4C, 3);
  if (Wire.available() < 3) {
```

```
    return false;
  }

  int8_t rawX = Wire.read();
  int8_t rawY = Wire.read();
  int8_t rawZ = Wire.read();

  *ax = rawX / 21.33;
  *ay = rawY / 21.33;
  *az = rawZ / 21.33;

  return true;
}
```