

الجمهورية الجزائرية الديمقراطية الشعبية

REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE

وزارة التعليم العالي والبحث العلمي

MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA  
RECHERCHE SCIENTIFIQUE

جامعة سعيدة - د. مولاي الطاهر

UNIVERSITE DE SAÏDA- DR MOULAY TAHAR



Faculté des Sciences et de la Technologie

Département d'électronique

MEMOIRE DE FIN D'ETUDES EN VUE DE L'OBTENTION

DU DIPLOME DE MASTER EN ELECTRONIQUE

OPTION: ELECTRONIQUE DES SYSTEMES EMBARQUÉS

THEME :

---

SYSTEME DE DETECTION D'OBJETS BASÉ SUR L'ESP32 ET EDGE  
IMPULSE POUR L'AUTOMATISATION DOMOTIQUE VIA HOME  
ASSISTANT

---

Présenté par :

BOUBAKAR Oussama

DRICI Cheikh

Soutenu le 24 Juin 2025, Devant le jury composé de :

CHAMI Nadir	Maître de conférences à l'Université de Saida	Président
HADJI Bagdad	Maître de conférences à l'Université de Saida	Examinateur
MAACHOU Fatima	Maître de conférences à l'Université de Saida	Encadrante

Année Universitaire : 2024 - 2025

الجمهورية الجزائرية الديمقراطية الشعبية

DEMOCRATIC AND POPULAR ALGERIAN REPUBLIC

وزارة التعليم العالي والبحث العلمي

MINISTRY OF HIGHER EDUCATION AND SCIENTIFIC RESEARCH

جامعة سعيدة - د. مولاي الطاهر

UNIVERSITY OF SAIDA-DR MOULAY TAHAR



Faculty of Science and Technology

Department of electronics

**Final thesis for the attainment of a Master's degree in Electronics**

**OPTION: EMBEDDED SYSTEMS ELECTRONICS**

**THEME :**

---

**OBJECT DETECTION SYSTEM BASED ON ESP32 AND EDGE IMPULSE  
FOR HOME AUTOMATION VIA HOME ASSISTANT**

---

**Presented by :**

BOUBAKAR Oussama

DRICI Cheikh

**Defended on June 24, 2025, before a jury composed of:**

CHAMI Nadir	Lecturer at the University of Saida	President
HADJI Bagdad	Lecturer at the University of Saida	Reviewer
MAACHOU Fatima	Lecturer at the University of Saida	Supervisor

**Academic Year : 2024 - 2025**

# Abstract

This project aims to develop an intelligent real-time object detection system using the ESP32-CAM, a microcontroller board with an integrated camera. The Edge Impulse platform is leveraged to train an artificial intelligence (AI) model capable of identifying specific objects from captured images. Once deployed, this autonomous system transmits detection results via the MQTT protocol to Home Assistant, an open-source home automation platform. These data enable automated actions based on the detected objects, making the system suitable for various IoT applications such as security, surveillance, or smart space management.

**Keywords :** ESP32-CAM, MQTT Protocol, Edge Impulse, Home Assistant, IoT, AI.

# Résumé

Ce projet consiste à développer un système intelligent de détection d'objets en temps réel en utilisant l'ESP32-CAM, une carte dotée d'une caméra intégrée. La plateforme Edge Impulse est exploitée pour entraîner un modèle d'intelligence artificielle (IA) capable d'identifier des objets spécifiques à partir des images capturées. Une fois déployé, ce système autonome transmet les résultats des détections via le protocole MQTT à Home Assistant, une plateforme de domotique open-source. Ces données permettent d'automatiser des actions en fonction des objets détectés, rendant le système adapté à des applications IoT variées comme la sécurité, la surveillance ou la gestion intelligente des espaces.

**Mots clés:** ESP32-CAM, Protocole MQTT, Edge Impulse, Home Assistant, IoT, AI.

# Acknowledgements

*We would like to express our deepest thanks to our supervisor Dr. MAACHOU Fatima for her support and her availability throughout this dissertation through the work sessions organized.*

*We thank the members of the jury for their interest in our work. Our thanks also go to all of our teachers who have contributed to our training.*

*We extend my heartfelt thanks to my parents. Your unwavering love, guidance, and encouragement have been the foundation of my success. Your belief in my abilities has always inspired me to strive for excellence. Thank you for your endless sacrifices and for always being there for me.*



# Dedication

*This work is dedicated to my parents, whose unwavering love, support, and guidance have been my greatest source of strength. Your belief in me has been a constant source of inspiration.*

*To my brothers, for always standing by me, encouraging me, and being my greatest allies. Your support and camaraderie have been invaluable.*

*And to my friends, for their steadfast support, understanding, and encouragement. Your presence in my life has been a blessing.*

*Thank you all for being my pillars of strength.*

*Cheikh*

# Dedication

*This work is dedicated to my parents, whose unwavering love, support, and guidance have been my greatest source of strength. Your belief in me has been a constant source of inspiration.*

*To my brothers, for always standing by me, encouraging me, and being my greatest allies. Your support and camaraderie have been invaluable.*

*And to my friends, for their steadfast support, understanding, and encouragement. Your presence in my life has been a blessing.*

*Thank you all for being my pillars of strength.*

*Oussama*

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
	<b>Introduction</b>	<b>1</b>
1.1	Problem Statement . . . . .	1
1.2	Objective . . . . .	1
1.3	Motivation . . . . .	1
1.4	Methodology Overview . . . . .	2
1.4.1	ESP32-CAM (Sensor Node) . . . . .	2
1.4.2	ESP32-WROOM-32 (Actuator Node) . . . . .	2
1.5	Home Assistant . . . . .	2
1.5.1	MQTT Protocol . . . . .	2
1.5.2	Edge Impulse . . . . .	3
1.5.3	Integration Architecture . . . . .	3
1.6	Master of science Organization . . . . .	3
<b>2</b>	<b>Smart Home Technologies and Edge Computing</b>	<b>5</b>
2.1	The Internet of Things (IoT) . . . . .	5
2.1.1	Definition of the Internet of Things . . . . .	5
2.1.2	How IoT Works . . . . .	6
2.1.3	Communicating Objects . . . . .	6
2.1.4	Definition of a Connected Object . . . . .	6
2.1.5	IoT Application Domains . . . . .	6
2.2	Smart Home . . . . .	7
2.2.1	How Smart Homes Work . . . . .	8
2.2.2	Structure of a Smart Home Automation System . . . . .	8
2.3	Home Assistant . . . . .	9
2.3.1	Overview of Devices Supported by Home Assistant . . . . .	9
2.3.2	Home Assistant : Setup and Usage . . . . .	10
2.3.3	Automations in Home Assistant . . . . .	10
2.4	Microcontrollers on the Edge – Is ESP32 with Camera ready for Machine Learning? . . . . .	11
2.4.1	Why Do We Need Edge Computing . . . . .	11
2.4.2	Edge Computing Benefits . . . . .	12

2.5	Edge impulse Platform . . . . .	12
<b>3</b>	<b>Materials and Methods</b>	<b>15</b>
3.1	Introduction . . . . .	15
3.2	Technical Overview of Raspberry pi 5 . . . . .	15
3.2.1	Technical Specifications . . . . .	16
3.3	ESP32 Development Board . . . . .	20
3.3.1	Internal Architecture of ESP32 chip . . . . .	20
3.3.2	ESP32-WROOM-32 Pin layout : . . . . .	23
3.4	ESP32-CAM AI-Thinker Module . . . . .	25
3.4.1	Features and Specifications . . . . .	25
3.4.2	ESP32-CAM Pinout and Functions . . . . .	26
3.4.3	The OV2640 Camera sensor . . . . .	26
3.4.4	Programming the ESP32-CAM . . . . .	27
3.5	Definition and Purpose of MQTT . . . . .	28
3.6	Conclusion . . . . .	32
<b>4</b>	<b>Implementation and Automation of the Object Detection System</b>	<b>33</b>
4.1	Introduction . . . . .	33
4.2	Installing Home Assistant on Raspberry Pi 5 . . . . .	33
4.2.1	Initial Configuration . . . . .	34
4.2.2	Adding Devices and Integrations . . . . .	35
4.3	Edge Impulse-Based Object Detection: Model Training and Deployment . . . . .	36
4.3.1	Data Collection and Dataset Creation . . . . .	38
4.3.2	Labeling Phase . . . . .	39
4.3.3	Designing an Impulse . . . . .	40
4.3.4	Results and discussion : . . . . .	43
4.3.5	Validating Our Model . . . . .	45
4.3.6	Exporting and deploying the trained model to the ESP32-CAM . . . . .	45
4.4	Deploying the Model Using the Arduino IDE . . . . .	46
4.5	Integrating MQTT for Prediction Publishing to Home Assistant . . . . .	47
4.5.1	Overview of MQTT Packet Structure . . . . .	48
4.5.2	Enabling MQTT Communication with Home Assistant . . . . .	48
4.5.3	Inference and MQTT Publishing Code for ESP32-CAM . . . . .	49
4.6	Automation Scenarios Using MQTT Communication . . . . .	51
4.6.1	Programming the ESP32 as an Actuator for MQTT Subscription . . . . .	51
4.6.2	Person Detection Notification Automation via MQTT . . . . .	58
4.7	Conclusion . . . . .	61
	<b>Conclusion</b>	<b>62</b>
	<b>References</b>	<b>63</b>

# List of Figures

1.1	Integration Architecture. . . . .	3
2.1	Structure of a Smart Home. . . . .	8
2.2	Edge computing vs Cloud computing. . . . .	12
2.3	Edge Impulse Development Workflow. . . . .	13
3.1	Raspberry Pi 5. . . . .	16
3.2	Broadcom BCM2712 processor. . . . .	16
3.3	Power Management system using the DA9091 PMIC. . . . .	19
3.4	Raspberry Pi 5 power supply cconnector. . . . .	19
3.5	Functional diagram of ESP32 development board. . . . .	21
3.6	ESP32-WROOM-32development board Pin layout. . . . .	23
3.7	Front (a) and back (b) views of the ESP32-CAM module. . . . .	25
3.8	ESP32-CAM Pinout Diagram. . . . .	26
3.9	OV2640 Camera Sensor for ESP32-CAM. . . . .	27
3.10	ESP32-CAM Connected to FTDI Module on Breadboard. . . . .	27
3.11	ESP32-CAM-MB programmer board. . . . .	28
3.12	MQTT Broker-Client architecture. . . . .	30
3.13	MQTT Publish/Subscribe Architecture. . . . .	31
4.1	Home Assistant interface after startup. . . . .	35
4.2	Integration menu showing discovered devices. . . . .	35
4.3	Installing the Mosquitto broker add-on. . . . .	36
4.4	Mosquitto broker configuration within Home Assistant. . . . .	36
4.5	Edge Impulse Development Workflow. . . . .	37
4.6	Collected dataset showing hand sign classes and person detection. . . . .	38
4.7	Dataset upload and automated train/test split. . . . .	39
4.8	Labeling phase with bounding boxes applied to hand signs. . . . .	40
4.9	Designing an Impulse . . . . .	41
4.10	Configuration the image color depth . . . . .	41
4.11	Feature Explorer . . . . .	42
4.12	Neural Network Settings . . . . .	43
4.13	Training Results with 60 Epochs and Learning Rate of 0.01. . . . .	44
4.14	Model Validation Results. . . . .	45

## List of Figures

---

4.15	Running the Impulse Using a Generated Arduino Library . . . . .	46
4.16	Inference results. . . . .	47
4.17	MQTT configuration within Home Assistant. . . . .	48
4.18	Smart Sensor Program: On-Device Inference and MQTT Publishing on ESP32-CAM. . . . .	50
4.19	ESP32-CAM correctly publishing prediction results to MQTT. . . . .	51
4.20	ESP32 Actuator Subscribing to MQTT Topic. . . . .	52
4.21	Actuator Program Logic Flowchart. . . . .	54
4.22	Serial monitor output of the sensor and actuator response upon detecting "a" . . . . .	55
4.23	Actuator turns on the blue LED in Response to 'a' Hand Sign . . . . .	55
4.24	Serial monitor output of the sensor and actuator response upon detecting "b" . . . . .	56
4.25	Actuator turns OFF the blue LED in Response to 'b' Hand Sign . . . . .	56
4.26	Serial monitor output of the sensor and actuator response upon detecting "L" . . . . .	57
4.27	Actuator controls the Red LED in Response to 'L' Hand Sign . . . . .	57
4.28	Serial monitor output of the sensor and actuator response upon detecting "h" . . . . .	58
4.29	Actuator turns OFF the blue LED in response to 'h' Hand Sign . . . . .	58
4.30	Automation setup in Home Assistant. . . . .	59
4.31	Conditional check for MQTT payload containing "person". . . . .	59
4.32	Notification sent via Home Assistant mobile app. . . . .	60
4.33	ESP32-CAM Serial Monitor Output – Person Detection Notification Trig- gered . . . . .	60

# List of Tables

3.1	Technical Specifications of ESP32-WROOM-32 Module[13]. . . . .	21
3.2	ESP32-WROOM-32 GPIO Pins and Functions [14]. . . . .	24
3.3	ESP32-CAM Pin Descriptions. . . . .	26
3.4	Pin mapping between ESP32-CAM and FTDI adapter. . . . .	28

# Abbreviations

<b>ADC</b>	Analog-to-Digital Converter
<b>AI</b>	Artificial Intelligence
<b>AP</b>	Access Point
<b>CAN</b>	Controller Area Network
<b>CLI</b>	Command Line Interface
<b>DAC</b>	Digital-to-Analog Converter
<b>DMA</b>	Direct Memory Access
<b>DMP</b>	Digital Motion Processor
<b>ETH</b>	Ethernet
<b>ESP</b>	Espressif System's Product
<b>ESP32</b>	Espressif System on Chip 32-bit
<b>ESP32-CAM</b>	ESP32 with Camera Module
<b>ESP32-WROOM-32</b>	ESP32 Wi-Fi Room Module
<b>FTDI</b>	Future Technology Devices International
<b>GPIO</b>	General Purpose Input/Output
<b>I2C</b>	Inter-Integrated Circuit
<b>I2S</b>	Integrated Inter-IC Sound
<b>IIoT</b>	Industrial Internet of Things
<b>IR</b>	InfraRed
<b>IoT</b>	Internet of Things
<b>LED</b>	Light Emitting Diode
<b>MAC</b>	Media Access Control



## Abbreviations

---

<b>MEMS</b>	Micro Electro Mechanical Systems
<b>MLOps</b>	Machine Learning Operations
<b>MPU</b>	Motion Processing Unit
<b>MQTT</b>	Message Queuing Telemetry Transport
<b>OV2640</b>	OmniVision 2640 Camera Sensor
<b>PHY</b>	Physical Layer
<b>PLC</b>	Programmable Logic Controller
<b>PWM</b>	Pulse Width Modulation
<b>RFID</b>	Radio Frequency Identification
<b>ROM</b>	Read Only Memory
<b>SDIO</b>	Secure Digital Input Output
<b>SPI</b>	Serial Peripheral Interface
<b>SRAM</b>	Static Random Access Memory
<b>STA</b>	Station
<b>TCP/IP</b>	Transmission Control Protocol / Internet Protocol
<b>UART</b>	Universal Asynchronous Receiver/Transmitter
<b>Wi-Fi</b>	Wireless Fidelity

# Chapter 1

## Introduction

### 1.1 Problem Statement

The rapid advancement of smart home technologies has created new opportunities to improve daily life, particularly improving safety and convenience within the home environment. However, traditional housing still lacks essential automation features, especially for elderly people who spend most of their time at home. As the global population ages, in 2017, there were an estimated 962 million people aged 60 or over, comprising 13 percent of the global population, and this number is projected to reach 1.4 billion in 2030, this gap is expected to widen [1]. These statistics highlight the urgent need for smart home solutions capable of responding to real-time events. This project addresses the challenge of integrating a real-time object detection system with Home Assistant to automate responses within the context of the smart home. Specifically, it investigates how gesture or object detection can trigger automated actions in a reliable, efficient, and low-latency manner.

### 1.2 Objective

This project aims to develop an intelligent real-time object detection system using the implementation of an intelligent home automation system. The system consists of two ESP32-based microcontrollers: an ESP32-CAM, which acts as a sensor node for real-time object detection, and an ESP32-WROOM-32, which functions as an actuator to respond to detected events. Both devices are integrated with Home Assistant, a popular open-source home automation platform, using the MQTT (Message Queuing Telemetry Transport) protocol for efficient communication.

### 1.3 Motivation

The motivation for this project arises from the growing need for smarter, safer, and more comfortable living environments. With increasing demand for smart home technologies

,particularly for elderly care and security, integrating edge-based real-time object detection becomes essential. The use of edge computing, which processes data locally on devices such as the ESP32-CAM, significantly reduces latency and bandwidth usage, enabling faster and more efficient decision making. This aligns with modern home automation trends that emphasize autonomy, reliability, and adaptability. Furthermore, advances in embedded machine learning have made it feasible to deploy object detection models directly onto microcontrollers, offering cost-effective and scalable solutions.

## 1.4 Methodology Overview

The system architecture is divided into sensor, communication, processing, and actuation layers:

### 1.4.1 ESP32-CAM (Sensor Node)

The ESP32-CAM is a development board with a built-in camera (OV2640) and Wi-Fi/Bluetooth capabilities, based on the ESP32-S chip. It is used in this project to perform real-time object detection tasks using a pre-trained model deployed via Edge Impulse.

### 1.4.2 ESP32-WROOM-32 (Actuator Node)

The ESP32-WROOM-32 is a highly integrated Wi-Fi and Bluetooth module built around the ESP32-D0WDQ6 chip. In this project, it is used to perform actions in response to detected events, acting as an actuator within the smart home automation system.

## 1.5 Home Assistant

Home Assistant is an open source platform designed to manage and automate smart devices. It runs on a Raspberry Pi and is used to:

- Visualize sensor data from ESP32-CAM
- Send control commands to ESP32-WROOM-32
- Automate tasks based on sensor input

### 1.5.1 MQTT Protocol

MQTT is a lightweight, publish-subscribe messaging protocol ideal for low-power devices. It allows communication between the ESP32 devices and Home Assistant. The MQTT broker used is Mosquitto, an open source implementation, which runs on the same Raspberry Pi as Home Assistant.

### 1.5.2 Edge Impulse

Edge Impulse is a machine learning platform specifically designed for embedded and resource-constrained devices. In this project, it was used to train and deploy the object detection model directly onto the ESP32-CAM.

### 1.5.3 Integration Architecture

- ESP32-CAM captures and classifies images locally using an on-board machine learning model deployed via Edge Impulse.
- The inference result is then published via MQTT to Home Assistant.
- Home Assistant processes these data and sends control signals via MQTT.
- ESP32-WROOM-32 receives these signals and triggers the appropriate actuators.
- If the detected and triggered object is a human, a notification is sent to the phone.

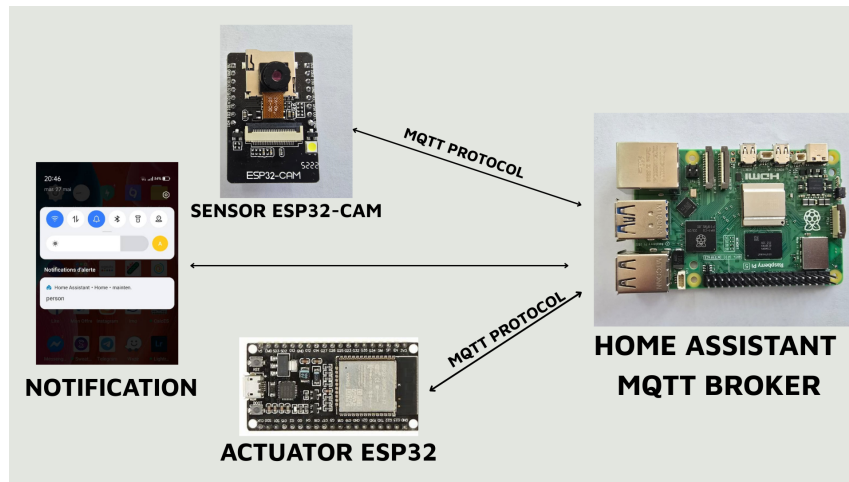


Figure 1.1: Integration Architecture.

## 1.6 Master of science Organization

This master of science is structured to provide a comprehensive overview of the development process :

- **Introduction:** Establishes the project context, objectives, and methodology.
- **Chapter 1:** Introduces the theoretical basis of smart home technologies, IoT systems, edge computing concepts, and the Edge Impulse platform.
- **Chapter 2:** Outlines the components, hardware configurations, and technical approaches used, including Raspberry Pi, ESP32, ESP32-CAM, the MQTT protocol.

- **Chapter 3:** Describes the implementation process, including Home Assistant configuration and Edge Impulse model development.
- **Conclusion:** Summarizes achievements, limitations, and recommendations for future work.

# Chapter 2

## Smart Home Technologies and Edge Computing

### 2.1 The Internet of Things (IoT)

The Internet of Things (IoT) is a global network of physical objects connected to the Internet, based on the idea that everyday items can collect, exchange, and act on data. These objects are equipped with technologies such as sensors, software, and connectivity capabilities, which enable them to transmit information and potentially receive commands. The IoT opens the door to a multitude of scenarios based on the interconnection between the physical and digital worlds.

Kevin Ashton, while working as a brand manager at Procter & Gamble (P &G), is widely credited with coining the term “Internet of Things” in 1999. He used the phrase during a presentation titled “That ‘Internet of Things’ Thing” at MIT, where he advocated for the use of radio-frequency identification (RFID) technology in supply chain management[2]. Ashton proposed attaching RFID and other sensors to physical objects to enable data collection and communication via the Internet, allowing machines to sense and interact with the physical world autonomously, without human intervention.

#### 2.1.1 Definition of the Internet of Things

The “IoT” is defined as a network that links and combines objects with the Internet, following protocols that ensure their communication and information exchange via a variety of devices.

IoT can also be described as a network of networks that enable, through standardized and unified electronic identification systems, as well as wireless mobile devices, the direct and unambiguous identification of digital entities and physical objects, to retrieve, store, transfer, and process data uninterruptedly between the physical and virtual worlds[3].

Several definitions exist, but the most relevant for our work is the one proposed by Weill and Souissi, who define IoT as an extension of the current Internet to any object capable of communicating, directly or indirectly, with electronic equipment themselves connected to

the Internet. This new dimension of the Internet raises significant technological, economic and social issues, notably thanks to the major savings enabled by the standardization of this domain, while ensuring the protection of individual rights and freedoms.

### 2.1.2 How IoT Works

The IoT enables the interconnection of various smart objects through the Internet. To achieve this, several technological systems are required. Here are some examples of these technologies. IoT encompasses various technical solutions such as RFID, TCP/IP, mobile communication technologies, and sensor networks, allowing the identification of objects as well as the capture, storage, processing, and transmission of data. These technologies operate within physical environments and enable seamless interaction between the physical and virtual worlds.

### 2.1.3 Communicating Objects

These are devices capable of communicating with their environment and exchanging data with their peers through various communication media. These objects are present in many fields, from the general public — especially smart home devices — to smart cities in general. They are transforming our habits, behaviors, and society as a whole.

### 2.1.4 Definition of a Connected Object

It is still difficult to provide a precise definition of the generic term "connected object". In general, a connected object is a physical object equipped with sensors or a chip that enables it to transcend its original use and offer new services.

It is an electronic hardware capable of communicating with a computer, smartphone, or tablet via a wireless network (Wi-Fi, Bluetooth, mobile networks, long-range radio networks, etc.), connecting it to the Internet or a local network.

Broadly speaking, there are two categories of connected objects:

- **Passive objects:** with limited storage capacity (about a kilobyte), mainly serving an identification role. Some, such as RFID chips, can integrate a sensor (temperature, humidity) and be rewritable .
- **Active objects:** equipped with several sensors, larger storage capacity, processing ability, and capable of communicating on a network .

### 2.1.5 IoT Application Domains

There has been a rapid increase in the adoption of IoT across various sectors of society and the economy, including intelligent transportation systems, smart healthcare, smart manufacturing, smart homes, smart cities, smart agriculture, and smart energy. IoT technologies are being applied in various sectors of the economy to increase efficiency,

solve technical challenges, and create value to increase company earnings and improve user experience.

The application domains of Internet of Things solutions are vast. Some of the applications of IoT include the following domains:

- Smart homes
- Smart water management
- Internet of Food
- Smart metering
- Smart cities (including logistics, retail, and transportation)
- Industrial IoT (IIoT)
- Precision agriculture and smart farming
- Security and emergency response
- Healthcare and wellness (including wearables)
- Smart environmental monitoring
- Energy management
- Robotics and automation
- Smart grids

## 2.2 Smart Home

A smart home is a residence equipped with internet-connected devices that enable remote monitoring and control of various systems such as lighting, climate, appliances, and security —typically via a mobile device. These systems can be implemented using either hardwired or wireless technologies.

Smart homes aim to improve occupant comfort and energy efficiency by automatically managing heating, lighting, and electronic devices. For example, smart systems can deactivate heating or cooling systems when residents are away, or adjust energy usage based on real-time needs, contributing to optimized resource consumption. In addition to convenience and efficiency, security is a critical aspect of smart homes. Intelligent systems can predict potentially hazardous situations and respond promptly to events that can compromise the safety of residents, offering an additional layer of protection in everyday life[4].



### 2.2.1 How Smart Homes Work

Devices in a smart home are connected to each other and can be accessed through a central control point, such as a mobile phone, tablet, laptop, or even a game console. Components like door locks, televisions, thermostats, home monitors, cameras, lighting systems, and household appliance (e.g., refrigerators) can be managed through unified home automation system.

### 2.2.2 Structure of a Smart Home Automation System

A smart home automation system consists of three main components: a central control unit (the brain), sensors that collect data, and actuators that perform actions. These elements work together to automate tasks and improve comfort, security, and energy efficiency in the home[5].

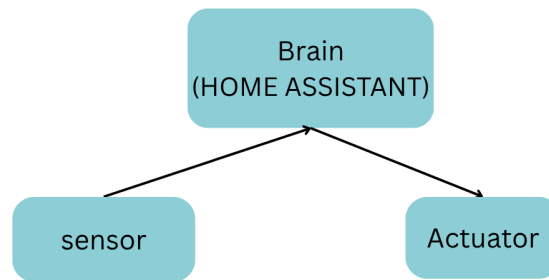


Figure 2.1: Structure of a Smart Home.

#### 1. The brain

The central processing unit, often referred to as the "brain", can be a programmable logic controller (PLC), a computer, or more commonly a Home Assistant platform running on a Raspberry Pi. This unit collects all the data gathered throughout the house and sends commands to various devices. It is considered the brain because it enables intelligent decision-making and coordination within the smart home ecosystem.

#### 2. The Sensors

Sensors are devices that allow the house to perceive its environment. They measure parameters such as temperature, humidity, brightness, CO<sub>2</sub> levels, noise levels, presence. They can also detect smoke, gas leaks, and monitor the electrical consumption of appliances. This data allows the system to assess the home's internal state and respond appropriately.

#### 3. The Actuators

Actuators are peripherals that control household appliances (e.g., heating, television, washing machine) and automation mechanisms such as shutters, garage doors, and blinds. Based on sensors input, the brain processes this information and triggers the appropriate actions to maintain comfort, safety, and energy efficiency.

## 2.3 Home Assistant

Home Assistant is an open-source platform designed to manage home automation by controlling various smart devices. Unlike many commercial automation systems that require all devices to operate within a specific ecosystem, Home Assistant offers a more flexible and privacy-focused solution. It supports integration with a wide range of devices over the network, enabling centralized control and automation. One of its key advantages is the ability to remotely monitor and manage connected devices[5].

### 2.3.1 Overview of Devices Supported by Home Assistant

Home assistant can integrate many devices with multiple communication protocols. Below are some of the most common and important devices that can be integrated with Home Assistant.

#### 1. Smart Lights

Smart lights are among the most commonly integrated devices in Home Assistant. Once integrated, they can be operated remotely. One of their key advantages is the ability to set up automation routines to control when the lights turn on or off, enhancing both convenience and energy efficiency.

#### 2. Smart Switches

Smart switches function similarly to smart lights but are used for appliances that are not inherently smart. They are especially useful when upgrading the appliance itself is not feasible, yet smart functionality is still desired.

#### 3. Thermostats

If an HVAC (Heating, Ventilation, and Air Conditioning) system is present in the home, thermostats can be integrated with Home Assistant. This allows all users to remotely control the thermostat. A key benefit is the ability to implement temperature-based automation, optimizing comfort and energy usage.

#### 4. Smart Sensors

If the space is equipped with smart sensors such as intrusion sensors, fire sensors, or water leak detection sensors, they can be integrated into Home Assistant. These sensors can then trigger specific appliances or functions, improving safety and automatic responsiveness within the smart home system.

### 5. Smart Security Cameras

Smart security cameras that operate over the Internet are also supported by Home Assistant. The area can be monitored through Home Assistant dashboard without the need for physical wiring. In addition, various camera features can be controlled through this interface.

### 2.3.2 Home Assistant : Setup and Usage

Using Home Assistant is generally straightforward, although some configuration steps are required to activate its full functionality. Below is a simplified guide for starting with Home Assistant.

#### 1. Choosing Hardware and Installing Home Assistant

The first step involves selecting suitable hardware (such as a Raspberry Pi or a dedicated server) and installing the Home Assistant operating system. Installation procedures may vary depending on the chosen device.

#### 2. Creating a User Profile and Adding Devices

Once the system is up and running, access the Home Assistant web interface to create a user profile. During this stage, smart devices can be added and configured for integration.

#### 3. Controlling Smart Home Devices via the Dashboard

After adding devices, they can be controlled through the Home Assistant dashboard. The interface can be customized according to specific preferences and use cases.

#### 4. Downloading the Home Assistant Smartphone Application

For added convenience, the Home Assistant mobile app can be installed, enabling remote access and the creation of automation shortcuts through smartphone assistants such as Siri.

### 2.3.3 Automations in Home Assistant

One of the most powerful features offered by Home Assistant is the ability to create Automations, allowing the home to react intelligently to defined events or conditions. These Automations are built around three fundamental elements: triggers, conditions, and actions. A trigger is the initial event that activates the automation. It can be, for example, a change of state in a device (such as a door opening), a time event (such as sunset), or even motion detection in a room. The conditions refine the behavior of automation by adding additional criteria. For instance, the light can be set to turn on only if the sun has set and no one has yet entered. Finally, actions represent what Home Assistant should do once automation is triggered and the conditions are met, such as turning on a light, sending a notification, or launching a more complex scenario.

## 2.4 Microcontrollers on the Edge – Is ESP32 with Camera ready for Machine Learning?

A microcontroller is a cheap and programmable system that generally includes memory and I/O interfaces on a single chip. They have been developed for decades, but the main paradigm has not changed all that time until the first decade of the 21st century. The ubiquity of the Internet has resulted in the appearance of services that offer to send, collect, and analyze data from microcontrollers on cloud services. In most cases, connection between the microcontroller and the Internet has been made through Wi-Fi. Cloud services offer many advantages such as the reliability of cloud services and data visualization, but in the last few years a new paradigm has arrived, edge computing. Some authors declared that "edge computing refers to the enabling technologies that allow computation to be performed at the edge of the network, on downstream data on behalf of cloud services and upstream data on behalf of IoT services" [6].

In recent years, many microcontroller manufacturers have worked on machine learning implementation on microcontrollers. While some have developed special libraries with machine learning functions, others have implemented special hardware with enhanced machine learning capabilities.

According to a study published by Springer Nature Switzerland AG in 2020, the performance of various microcontrollers, such as the ESP32-CAM—was, evaluated with respect to their ability to handle machine learning workloads. The study found that [7]:

- **The ESP32-CAM and M5CAMERA resolution is enough for most machine learning tasks.**
- **The downsampling process could be resource-consuming, while ML models usually work with low resolution images, so it is better to capture images in a lower resolution possible.**
- **The ESP32-CAM is a suitable choice for edge-based machine learning, particularly for real-time object detection and automation applications.**

### 2.4.1 Why Do We Need Edge Computing

Edge computing is becoming essential due to limitations in network bandwidth and the increasing volume of data generated by edge devices. Although cloud computing offers powerful processing capabilities, the speed of data transmission from edge devices to the cloud has become a bottleneck. For example, aircraft and autonomous vehicles generate massive amounts of data every second, requiring real-time analysis that cloud-based systems cannot efficiently handle due to latency and bandwidth constraints. Edge computing addresses these issues by processing data locally, which results in faster response times, improved efficiency, and reduced strain on the network infrastructure.

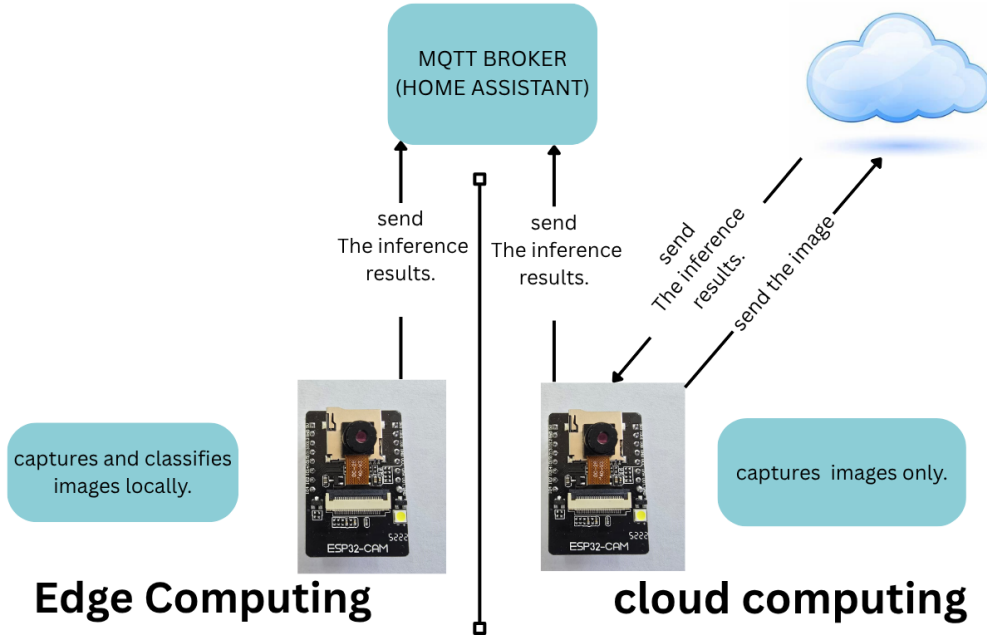


Figure 2.2: Edge computing vs Cloud computing.

### 2.4.2 Edge Computing Benefits

Edge computing brings computation closer to data sources, offering several advantages over traditional cloud-based models. By processing data locally, response times can be significantly reduced. For example, relocating a face recognition application from the cloud to the edge cut its response time from 900 to just 169 milliseconds. Similarly, offloading tasks to nearby computing resources, such as cloudlets has been shown to improve response times by 80 to 200 milliseconds while also lowering energy consumption by 30% to 40%. Advanced approaches that combine task partitioning, migration, and dynamic instantiation between mobile devices and edge nodes have demonstrated dramatic improvements, reducing application runtime and energy use by up to 20 times[6].

## 2.5 Edge impulse Platform

Edge Impulse is a cloud-based machine learning operations (MLOps) platform designed for the development of embedded and edge machine learning (TinyML) systems. We used this platform because of its capability to support deployment across a wide range of hardware targets. Traditional TinyML workflows often suffer from fragmented software ecosystems and diverse hardware environments, which complicate the optimization and portability of machine learning models.

Edge Impulse addresses these limitations by offering a unified MLOps framework that facilitates the scalable development of TinyML applications. It provides integrated support for both software and hardware optimizations, enabling the creation of an extensible and portable software stack suited for various embedded systems.

In October 2022, the platform had supported the development of 118,185 projects by a community of 50,953 developers, highlighting its widespread adoption and utility in the TinyML domain [8].

### Edge Impulse Development Workflow

The Edge Impulse development workflow consists of several key steps:

1. **Data Collection:** Capturing and uploading training data.
2. **Data Preprocessing:** Cleaning and preparing data for training.
3. **Feature Extraction:** Converting raw data into meaningful features.
4. **Model Training:** Training a machine learning model on the extracted features.
5. **Model Testing:** Evaluating the model's performance.
6. **Deployment:** Exporting the model for deployment on the target device.

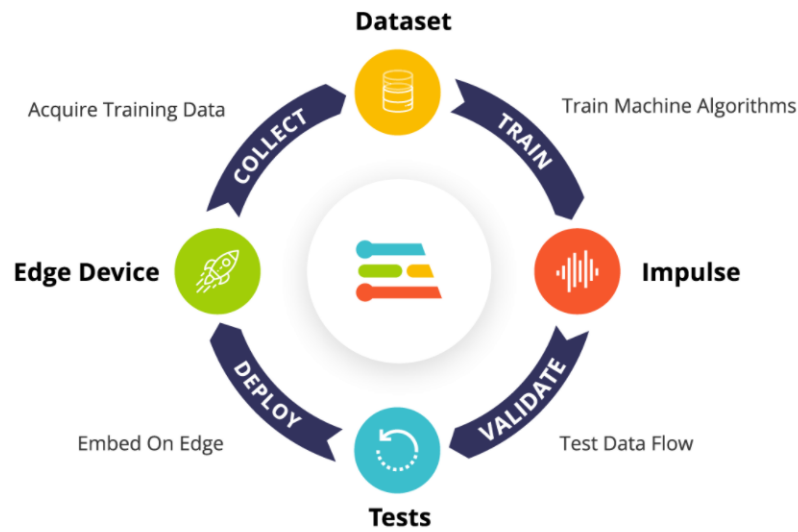


Figure 2.3: Edge Impulse Development Workflow.

### Conclusion

This chapter has explored the foundational concepts of the IoT, smart home technologies, and edge computing, highlighting their convergence in modern automation systems. A particular focus was placed on Home Assistant, a versatile and open-source platform capable of integrating a wide range of smart devices to create efficient, secure, and user-friendly living environments.

In this context, our project aims to take a step further by integrating intelligent edge devices, specifically ESP32-CAM, into the Home Assistant ecosystem. Using Edge Impulse, a powerful TinyML platform, we deploy lightweight, real-time object detection models directly on the ESP32-CAM. This enables local image processing and decision-making at the edge, reducing latency and dependency on cloud services while enhancing privacy and responsiveness.

The integration of object detection into Home Assistant opens up new possibilities for advanced home automation scenarios. For example, the system can recognize specific objects or human presence and trigger appropriate actions, such as sending notifications, turning on lights, or activating security protocols, without relying on external computation resources.

This chapter sets the stage for the practical implementation of our solution, which combines the benefits of edge computing with the versatility of Home Assistant to create a robust, intelligent home automation system.

# Chapter 3

## Materials and Methods

### 3.1 Introduction

In this chapter, we present the materials and methods used to implement our object detection system for smart home automation. We begin by describing the main hardware components, including the Raspberry Pi 5, focusing on its key features, specifications, and power requirements. We then introduce the ESP32-CAM, used for image capture and object detection, and the ESP32 board, which acts as a subscriber that responds to detection events. Finally, we examine the MQTT protocol, detailing its publish/subscribe architecture, the roles of clients, brokers, and topics, and the advantages it offers for real-time, efficient communication in IoT-based smart home environments.

### 3.2 Technical Overview of Raspberry pi 5

Raspberry Pi is a series of compact, affordable single-board computers developed by the Raspberry Pi Foundation. These devices are widely used in various applications, including education, robotics, automation, and IoT. The latest model, Raspberry Pi 5, offers improved performance and connectivity, making it a suitable choice for smart home applications, such as integrating with Home Assistant alongside ESP32-CAM for automation and surveillance purposes[9].



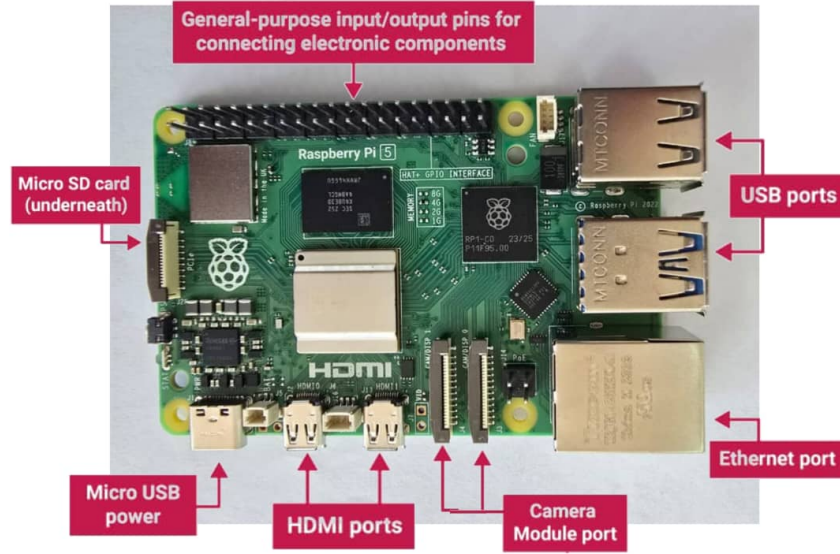


Figure 3.1: Raspberry Pi 5.

### 3.2.1 Technical Specifications

Raspberry Pi 5 is the most advanced model in the Raspberry Pi family, featuring significant improvements over its predecessors. Among the most notable upgrades is the introduction of the Broadcom BCM2712 system-on-chip (SoC), which forms the core of the Raspberry Pi 5's processing power.



Figure 3.2: Broadcom BCM2712 processor.

#### Processor and Architecture

The BCM2712 integrates a high-performance CPU, GPU, and I/O subsystems into a compact, energy-efficient chip designed specifically for embedded computing. The key architectural features are as follows:

- **Processor:** Quad-core ARM Cortex-A76, clocked up to 2.4 GHz.

- **SoC:** Broadcom BCM2712 — a 5th generation chip from the Broadcom BCM series, released in Q3 2023[10].
- **Architecture:** Based on a quad-core Arm Cortex-A76 cluster with 512 KB L2 cache per core and a shared 2 MB L3 cache. Includes a 12-core VideoCore VII GPU, a hardware video scaler, and an HDMI controller for dual 4Kp60 display outputs. Also integrates a Raspberry Pi-designed HEVC decoder and image signal processor.
- **Memory Interface:** A 32-bit LPDDR4X interface offering up to 17 GB/s of memory bandwidth.
- **PCI Express:** One  $\times 1$  and one  $\times 4$  PCI Express interfaces for high-bandwidth peripherals. The  $\times 4$  interface connects to the RP1 southbridge, responsible for most I/O functionality[11].

**Key Features:** Building on its robust architecture, the Raspberry Pi 5 offers several headline specifications that position it as a capable platform for demanding applications.

- Quad-core Arm Cortex-A76 @ 2.4 GHz
  - ARMv8-A instruction set architecture (ISA)
  - 64 KB L1 instruction and data caches
  - 512 KB L2 cache per core
  - 2 MB shared L3 cache

### GPU and Memory

- **GPU:** The system features a 12-core VideoCore VII GPU, delivering high-performance graphics capabilities, including smooth 3D rendering and support for dual 4Kp60 video playback.
- **RAM:** It is available in configurations 4 GB or 8 GB LPDDR4X SDRAM, providing up to 17 GB/s of bandwidth, which improves multitasking, streaming, and peripheral interaction.

### Storage and Boot

The Raspberry Pi 5 provides flexible storage and boot options, enhancing usability for both hobbyist and professional environments.

- **Storage:** A microSD card slot serves as the default boot and storage medium, while USB boot functionality allows users to run operating systems directly from high-speed external drives.
- **Boot EEPROM:** Continuing from the Pi 4B, the Pi 5 incorporates an EEPROM that enables firmware upgrades, improved reliability, and more robust boot configurations[11].

### Networking

Networking capabilities on the Raspberry Pi 5 have been significantly modernized to meet current connectivity standards.

- **Ethernet:** The device supports Gigabit Ethernet with Power over Ethernet Plus (PoE+) via an optional PoE+ HAT, enabling network connectivity and power delivery over a single cable.
- **Wireless:** Integrated Wi-Fi 6 (802.11ax) and Bluetooth 5.0 support ensure high-speed wireless performance and compatibility with a wide range of modern peripherals.
- **Network Configuration:** Users can easily configure wireless settings through the graphical desktop interface. Additionally, the Pi 5 can operate in access point mode, allowing it to share a wired internet connection wirelessly[11].

### Connectivity and Display

The Raspberry Pi 5 provides versatile connectivity options to support a wide range of peripherals and display configurations.

- **USB Ports:** It includes two USB 3.0 ports capable of 5 Gbps transfer speeds and two USB 2.0 ports. This combination ensures compatibility with both high-speed storage devices and legacy peripherals such as keyboards and mice.
- **Display Output:** Dual Micro-HDMI ports allow simultaneous output to two 4K displays at 60 Hz, making the device ideal for multimedia, digital signage, or multi-monitor productivity setups.

### Power Supply

The Raspberry Pi 5 is powered via a USB-C connector, requiring a recommended 5V/5A power adapter to ensure reliable operation under high load conditions. The power system is managed by the Renesas DA9091 “Gilmour” Power Management IC (PMIC), working in conjunction with the BCM2712 and the RP1 I/O controller.

The DA9091 integrates eight switch-mode power supplies, delivering the multiple voltage levels required by the board. Notably, it includes a quad-phase core supply capable of delivering up to 20A to the Cortex-A76 CPU cores and other digital components of the SoC[11].

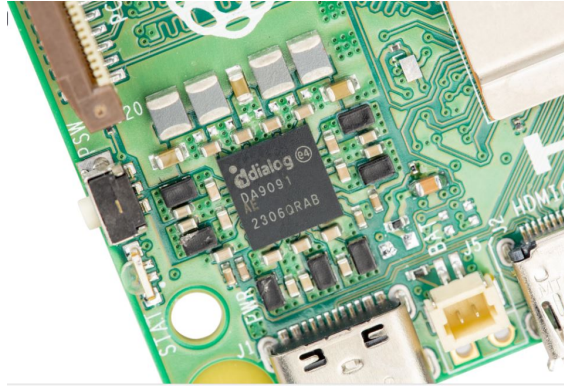


Figure 3.3: Power Management system using the DA9091 PMIC.

### Power States

The Raspberry Pi 5 supports several power states that enable efficient energy management depending on system activity. These modes allow the device to reduce power consumption when idle or not in active use:

- **OFF:** The board is completely powered down, with no 5V input applied.
- **WARM-STANDBY:** The system is halted via software (e.g., `sudo halt`) or a soft power button press. All power rails remain enabled, allowing a quick restart.
- **STANDBY:** Only the +5V power rail is active. Other internal regulators are disabled. This state can be configured via EEPROM to replace warm-standby behavior.
- **SLEEP:** Selected power domains such as CPU cores are turned off while system RAM is preserved (suspend-to-RAM). The system resumes to full operation upon power button activation.
- **ACTIVE:** All power rails are enabled, and the device is fully operational—running a desktop or headless Linux environment.



Figure 3.4: Raspberry Pi 5 power supply connector.

## GPIO and Operating Systems

The Raspberry Pi 5 features a 40-pin General-Purpose Input/Output (GPIO) header, which provides flexible digital interfaces for connecting a wide range of components such as sensors, actuators, displays, and HATs. This makes it an ideal platform for IoT applications, prototyping, and DIY electronics projects.

In terms of software, the Raspberry Pi 5 is compatible with various operating systems including Raspberry Pi OS, Ubuntu, and Windows IoT Core. It also supports other Linux-based distributions tailored for embedded systems.

In this study, the device runs **Home Assistant**, a widely adopted open-source home automation platform. Home Assistant takes full advantage of the Raspberry Pi 5's GPIO capabilities and improved processing power to manage and automate smart home devices in a scalable and reliable environment[12].

## 3.3 ESP32 Development Board

ESP32-WROOM-32 is a powerful, generic Wi-Fi + Bluetooth<sup>®</sup> + Bluetooth Low Energy (BLE) microcontroller unit (MCU) module. It is designed for a wide range of applications, from low-power sensor networks to high-performance tasks such as voice encoding, music streaming, and MP3 decoding.

At the core of this module lies the ESP32-D0WDQ6 chip [13], a scalable and adaptive processor. The chip features two individually controllable CPU cores, with a clock frequency adjustable between 80 MHz and 240 MHz. To support ultra-low-power applications, the chip allows the main CPU to be powered down while a dedicated low-power co-processor continues to monitor peripherals for signal changes or threshold crossings.

### 3.3.1 Internal Architecture of ESP32 chip

Figure 3.5 provides a detailed overview of the internal architecture of the ESP32 System-on-Chip (SoC). The diagram highlights the integration of dual-core Xtensa<sup>®</sup> LX6 microprocessors, operating up to 240 MHz, alongside a dedicated Ultra Low Power (ULP) co-processor for background tasks during deep sleep modes. The system incorporates a complete wireless communication stack, including Wi-Fi (802.11 b/g/n) MAC/baseband and Bluetooth v4.2 (Classic + BLE) controller, allowing robust and flexible wireless connectivity.

On-chip hardware accelerators for cryptographic functions—such as AES, SHA, RSA, and random number generation—enhance security performance without burdening the main CPUs. In addition, the ESP32 integrates a wide range of peripheral interfaces including SPI, I2C, UART, SDIO, ADC, DAC, and PWM, enabling it to interface seamlessly with sensors, actuators, and external memory.

This highly integrated SoC architecture makes the ESP32 an ideal choice for real-time, energy-constrained IoT applications, where processing power, connectivity, and peripheral

flexibility must coexist efficiently within a compact footprint.

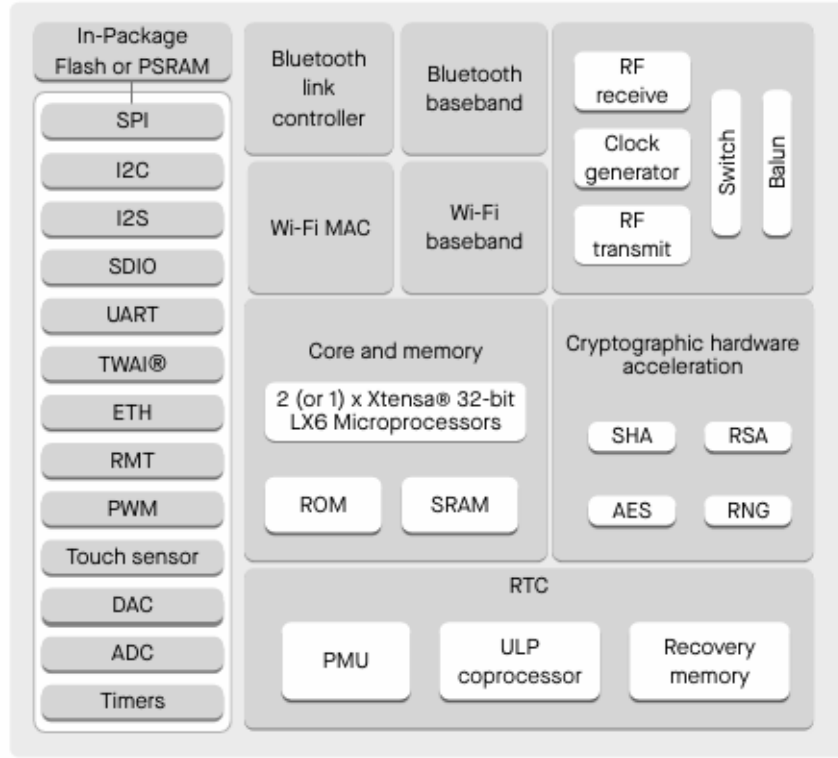


Figure 3.5: Functional diagram of ESP32 development board.

To complement this architectural overview, Table 3.1 presents the detailed technical specifications of the ESP32-WROOM-32 development board.

Table 3.1: Technical Specifications of ESP32-WROOM-32 Module[13].

Category	Details
<b>CPU and Memory</b>	<ul style="list-style-type: none"> <li>• Xtensa dual-core 32-bit LX6 microprocessor (up to 240 MHz)</li> <li>• 448 KB ROM, 520 KB SRAM</li> <li>• 16 KB SRAM in RTC</li> <li>• Embedded: ESP32-D0WD-V3 or ESP32-D0WDR2-V3</li> </ul>

*Continued on next page*

Category	Details
<b>Wi-Fi</b>	<ul style="list-style-type: none"> <li>• IEEE 802.11 b/g/n</li> <li>• Bit rate up to 150 Mbps (802.11n)</li> <li>• Supports A-MPDU and A-MSDU aggregation</li> <li>• 0.4 <math>\mu</math>s guard interval</li> <li>• Frequency: 2412–2484 MHz</li> </ul>
<b>Bluetooth®</b>	<ul style="list-style-type: none"> <li>• Bluetooth v4.2 BR/EDR and BLE</li> <li>• Supports Class 1, 2, 3 transmitter</li> <li>• Adaptive Frequency Hopping (AFH)</li> <li>• Audio codecs: CVSD and SBC</li> </ul>
<b>Peripherals</b>	<ul style="list-style-type: none"> <li>• Up to 26 GPIOs (including 5 strapping)</li> <li>• Interfaces: UART, SPI, I2C, I2S, SDIO, PWM, IR, ADC, DAC, RTC, touch sensor</li> <li>• TWAI® (CAN 2.0 compatible)</li> </ul>
<b>Integrated Components</b>	<ul style="list-style-type: none"> <li>• 40 MHz crystal oscillator</li> <li>• SPI flash: 4 / 8 / 16 MB</li> <li>• Optional 2 MB PSRAM (ESP32-D0WDR2-V3)</li> </ul>
<b>Antenna Options</b>	<ul style="list-style-type: none"> <li>• ESP32-WROOM-32E: On-board PCB antenna</li> <li>• ESP32-WROOM-32UE: External antenna via u.FL connector</li> </ul>
<b>Operating Conditions</b>	<ul style="list-style-type: none"> <li>• Voltage: 3.0–3.6 V</li> <li>• Temperature: <ul style="list-style-type: none"> <li>– <b>–85°C version:</b> –40°C to 85°C</li> <li>– <b>–105°C version:</b> –40°C to 105°C (only with 4/8 MB flash)</li> </ul> </li> </ul>
<b>Certifications</b>	<ul style="list-style-type: none"> <li>• Bluetooth BQB</li> <li>• RF: ESP32-WROOM-32E/UE certified</li> <li>• Environmental: RoHS / REACH</li> </ul>

*Continued on next page*



Category	Details
Testing Standards	HTOL, HTSL, uHAST, TCT, ESD

Among the various integrated components of the ESP32-WROOM-32 module, the embedded flash memory is particularly noteworthy. It supports over 100,000 program/erase cycles and ensures data retention for more than 20 years. These characteristics make it suitable for applications requiring both durability and long-term reliability.

### 3.3.2 ESP32-WROOM-32 Pin layout :

Figure 3.6 illustrates the physical pinout of the ESP32-WROOM-32 module, while Table 3.2 details the functional roles of each GPIO pin. They provide a clear overview of the module's interfacing capabilities, including digital and analog I/O, communication protocols (UART, SPI, I<sup>2</sup>C), and specialized features such as capacitive touch and DAC outputs.

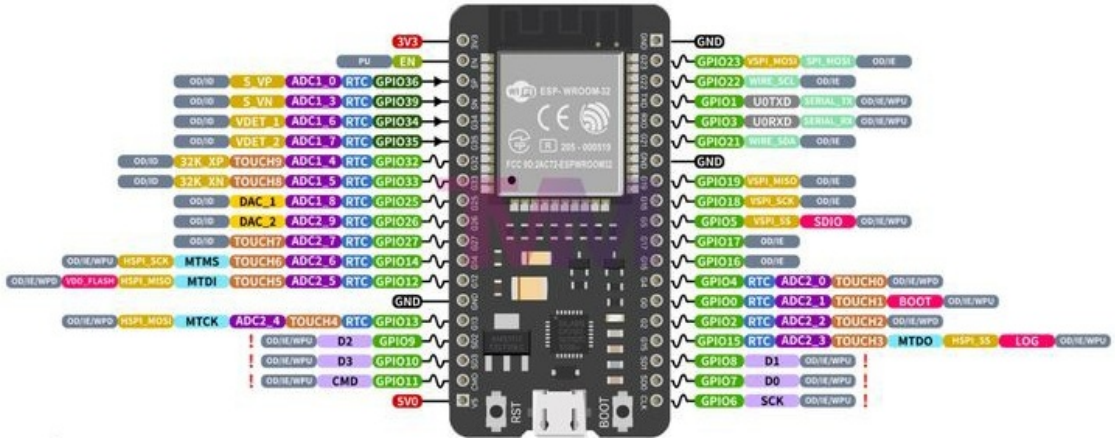


Figure 3.6: ESP32-WROOM-32 development board Pin layout.



Table 3.2: ESP32-WROOM-32 GPIO Pins and Functions [14].

Pin Name	GPIO	Type	Functions & Notes
IO34	GPIO34	Input only	ADC1.CH6, RTC_GPIO4
IO35	GPIO35	Input only	ADC1.CH7, RTC_GPIO5
IO32	GPIO32	I/O	ADC1.CH4, TOUCH9, RTC_GPIO9, XTAL_32K_P
IO33	GPIO33	I/O	ADC1.CH5, TOUCH8, RTC_GPIO8, XTAL_32K_N
IO25	GPIO25	I/O	DAC1, ADC2.CH8
IO26	GPIO26	I/O	DAC2, ADC2.CH9
IO27	GPIO27	I/O	ADC2.CH7
IO14	GPIO14	I/O	HSPI.CLK
IO12	GPIO12	I/O	HSPI.MISO
IO13	GPIO13	I/O	HSPI.MOSI
IO15	GPIO15	I/O	HSPI.CS
IO2	GPIO2	I/O	Default boot pin, general purpose
IO0	GPIO0	I/O	Boot mode selection
IO4	GPIO4	I/O	Touch sensor, general GPIO
IO16	GPIO16	I/O	RTC GPIO
IO17	GPIO17	I/O	UART signals
IO5	GPIO5	I/O	General GPIO
IO18	GPIO18	I/O	VSP1.CLK
IO19	GPIO19	I/O	VSP1.MISO
IO21	GPIO21	I/O	I2C SDA
IO22	GPIO22	I/O	I2C SCL
IO23	GPIO23	I/O	VSP1.MOSI
TXD0	GPIO1	Output	UART0 TX
RXD0	GPIO3	Input	UART0 RX

The ESP32 microcontroller is compatible with various programming environments, which enhances its accessibility and flexibility across a wide range of applications. It can be programmed using the Arduino IDE, PlatformIO (within Visual Studio Code), Lua, MicroPython, JavaScript (via third-party tools like Espruino), and Espressif’s official IoT Development Framework (ESP-IDF). This diversity allows developers to select tools that best match their expertise and project requirements. For the purposes of this project, the Arduino IDE was chosen due to its ease of use, extensive library support, and widespread community adoption.

### 3.4 ESP32-CAM AI-Thinker Module

The ESP32-CAM AI-Thinker is a low-cost, compact development board based on the ESP32 microcontroller with an onboard camera module. It is an excellent solution for Internet of Things (IoT) applications, prototype development, and DIY projects. ESP32 is a single 2.4 GHz Wi-Fi-and-Bluetooth combo chip designed with the TSMC low-power 40 nm technology [15]. It is designed to achieve the best power and RF performance, showing robustness, versatility, and reliability in a wide variety of applications and power scenarios, making it ideal for smart applications such as home automation, remote surveillance, and object detection[16].

The ESP32-CAM functions as an edge device that performs object detection locally. It communicates the detection results to Home Assistant, which runs on a Raspberry Pi acting as a local gateway. This configuration enables real-time automation and enhances data privacy by eliminating the need for cloud-based processing.

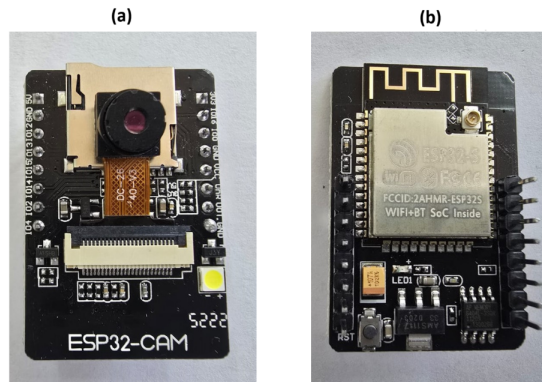


Figure 3.7: Front (a) and back (b) views of the ESP32-CAM module.

#### 3.4.1 Features and Specifications

The ESP32-CAM module integrates a wide range of hardware features that enable wireless communication, image processing, and peripheral interfacing. The main specifications are listed below:

- Dual-core Xtensa LX6 processor, up to 240 MHz
- 520 KB SRAM with support for external 4MB PSRAM
- Built-in Wi-Fi (802.11 b/g/n) and Bluetooth 4.2 (BLE)
- Multiple GPIOs, ADCs, DACs, and PWM controllers
- Supports UART, SPI, and I2C communication interfaces
- Image output formats: JPEG, BMP, and Grayscale
- Integrated OV2640 camera module
- MicroSD card support (up to 4GB)
- Low power consumption modes
- Supports multiple networking modes: STA, AP, STA+AP [16]

### 3.4.2 ESP32-CAM Pinout and Functions

The ESP32-CAM module features 16 physical pins, as shown in Figure 3.8 that serve multiple functions, including power supply, communication, and peripheral interfacing. Due to internal resource sharing with the camera and PSRAM, only a subset of GPIOs is accessible. Table 3.3 provides an overview of the main pin functions of the ESP32-CAM module.

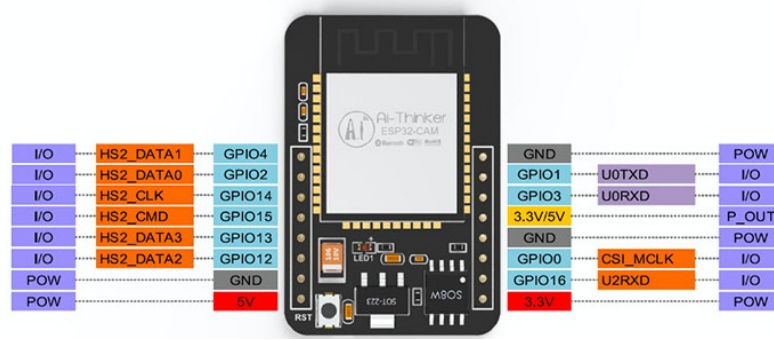


Figure 3.8: ESP32-CAM Pinout Diagram.

Table 3.3: ESP32-CAM Pin Descriptions.

Pin Type	Description
Power Pins	5V and 3.3V power supply options.
GPIO Pins	Only 10 GPIOs are available due to internal use by the camera and PSRAM.
UART Pins	Required for programming and external communication.
MicroSD Card Pins	Used for additional storage.
ADC Pins	Available but cannot be used when Wi-Fi is enabled.
Touch Pins	Capacitive touch-sensitive GPIOs.
SPI, PWM Pins	Used for peripheral device control.

### 3.4.3 The OV2640 Camera sensor

The OV2640 camera sensor on the ESP32-CAM is what sets it apart from other ESP32 development boards and makes it ideal for use in video projects like a video doorbell or nanny cam. The OV2640 camera has a resolution of 2 megapixels, which translates to a maximum of 1600×1200 pixels, sufficient for many surveillance applications [16].

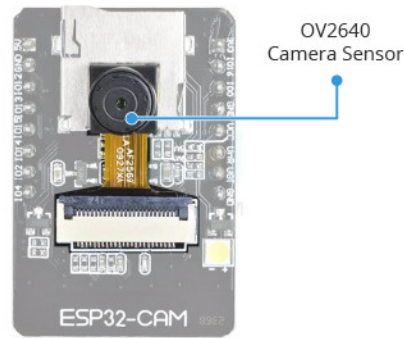


Figure 3.9: OV2640 Camera Sensor for ESP32-CAM.

### 3.4.4 Programming the ESP32-CAM

The ESP32-CAM does not have a built-in USB port for programming. To upload code, an external USB-to-serial adapter like an FTDI module is needed. This adapter allows communication between the computer and the ESP32-CAM through serial connection. It is also possible to use an ESP32-CAM-MB programmer board for easier setup.

#### 1. Using a FTDI

Figure 3.10 shows the FTDI adapter connected to the ESP32-CAM according to the pin mapping provided in Table 3.4. The main component of the FTDI board is the CH340G USB-to-Serial converter, which enables communication between the computer and the ESP32-CAM. Additionally, GPIO0 must be connected to GND during uploading to place the ESP32-CAM into flashing mode.

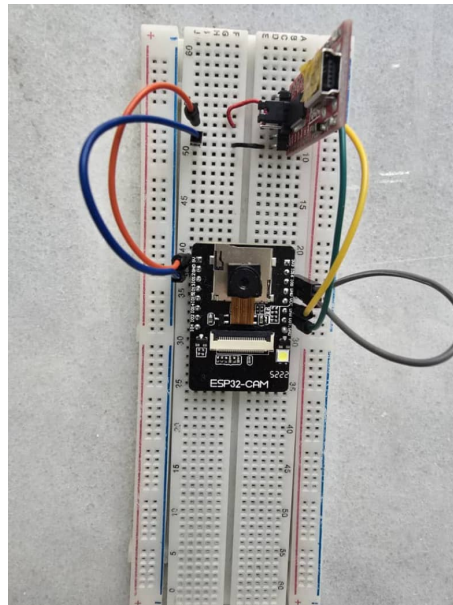


Figure 3.10: ESP32-CAM Connected to FTDI Module on Breadboard.

FTDI	ESP32-CAM
TX	RX (GPIO3)
SDA	GPIO21
GND	GND
V <sub>CC</sub>	V <sub>CC</sub>

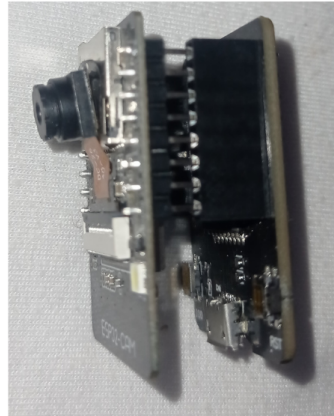
Table 3.4: Pin mapping between ESP32-CAM and FTDI adapter.

## 2. ESP32-CAM-MB programmer board

The ESP32-CAM AI-Thinker MB programmer is a shield that connects directly to the ESP32-CAM's GPIO header, allowing easy programming via a USB interface. It also includes RESET and BOOT (IO0) buttons, which make it easy to reset the ESP32-CAM or put it into flashing mode.



ESP32-CAM-MB programmer



ESP32-CAM and the programmer

Figure 3.11: ESP32-CAM-MB programmer board.

## 3.5 Definition and Purpose of MQTT

Message Queuing Telemetry Transport (MQTT) is a lightweight machine-to-machine (M2M) communication protocol designed specifically for resource-constrained devices. With a fixed header of only 2 bytes, MQTT is particularly well-suited for IoT applications that operate under limitations such as low bandwidth, limited computational power, restricted memory, or battery constraints. The protocol operates over a TCP/IP connection and can be deployed across various network types, including wired Ethernet and wireless local area networks (WLANs). Originally developed in 1999 by Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, now Eurotech), MQTT has since been adopted as an open standard by the Organization for the Advancement of Structured Information Standards (OASIS) [3].

The MQTT protocol consists of several key components that work together to enable efficient and scalable machine-to-machine communication. These include the broker, clients (publishers and subscribers), topics, the publish/subscribe architecture, and quality of service (QoS) mechanisms, each of which is detailed below.

### 1. MQTT broker

The MQTT broker plays a central role in the MQTT architecture by facilitating communication between MQTT clients, both publishers and subscribers. Here are some of the main reasons why MQTT brokers are important:

- **Scalability:** MQTT brokers can handle a large number of simultaneous connections, which is essential for IoT and M2M environments, where thousands or even millions of devices may be connected. Their ability to efficiently handle high message throughput ensures the protocol scales well.
- **Security:** MQTT brokers can provide security measures like authentication and encryption to ensure that the data transmitted between IoT devices and applications is secure.
- **Session management:** The MQTT broker is responsible for managing client sessions, including maintaining information about the client's subscriptions and handling messages that are retained for delivery to clients when they come online. This session management feature ensures that messages are not lost when clients disconnect and later reconnect to the broker.

### 2. MQTT Clients

In IoT, an MQTT client usually refers to publishers and subscribers. A publisher is a client that sends messages, while a subscriber is a client that receives messages. However, an MQTT client can also be both a publisher and a subscriber.

An MQTT client can be any device, ranging from a tiny microcontroller to a large server, that runs an MQTT library and connects to an MQTT broker over a network. An MQTT client library is a software module or package that implements the MQTT protocol and provides an interface for devices or applications to communicate using MQTT. These libraries make it easier to add MQTT support to applications or devices without implementing the protocol from scratch[17].

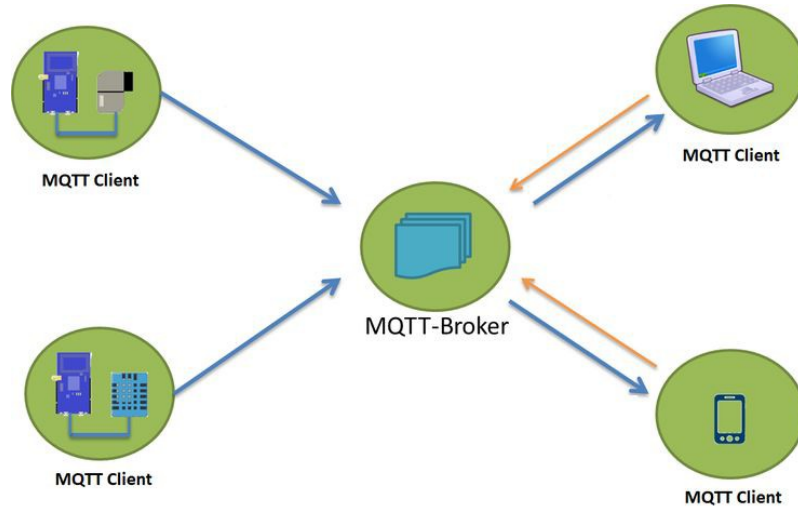


Figure 3.12: MQTT Broker-Client architecture.

### 3. Topics

Topics are hierarchical strings that define the subject or category of a message. When publishers send messages to the broker, they associate them with a specific topic. Subscribers express their interest in receiving messages by subscribing to one or more MQTT topics. The broker then routes messages to the appropriate subscribers based on their topic subscriptions.

### 4. MQTT Publish/Subscribe Architecture (Pub/Sub)

In the Pub/Sub architecture, there are publishers that generate messages and subscribers that receive those messages. Publish-subscribe is a broader concept that can be implemented using various protocols or technologies. The Pub/Sub architecture offers a unique alternative to traditional client-server (request-response) models. In the request-response approach, the client directly communicates with the server endpoint, creating a bottleneck that slows down performance. On the other hand, the Pub/Sub model decouples the publisher of the message from the subscribers. The publisher and subscriber are unaware that the other exists. A broker handles the connection between them. This decoupling produces a faster and more efficient communication process[17].

By eliminating the need for direct communication between publishers and subscribers, Pub/Sub architecture removes the exchange of IP addresses and ports. It also provides decoupling, allowing operations on both components to continue communication uninterrupted during publishing or receiving. The Pub/Sub features three dimensions of decoupling for optimal efficiency:

- **Space decoupling:** Publisher and subscriber do not need to know each other (for example, no exchange of IP address and port).



- **Time decoupling:** Publisher and subscriber do not need to run at the same time.
- **Synchronization decoupling:** Operations on both components do not need to be interrupted during publishing or receiving.

One of the most significant advantages of the Pub/Sub software architecture is its ability to filter all incoming messages and distribute them to subscribers correctly, eliminating the need for the publisher and subscriber to know one another's existence.

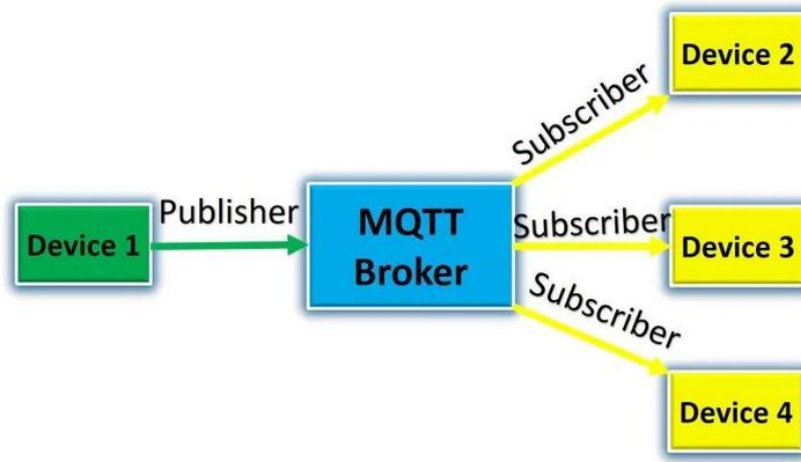


Figure 3.13: MQTT Publish/Subscribe Architecture.

### 5. MQTT QoS

The term "Quality of Service" in MQTT defines the reliability guarantee level for message delivery between devices. QoS represents a formal agreement between the sender and receiver about how message delivery will be handled and acknowledged. In IoT environments where connectivity can be unpredictable, QoS provides critical mechanisms to ensure that messages reach their destination appropriately, based on their importance. This may involve sacrificing some reliability for speed or ensuring that critical commands are delivered exactly once without duplication[17]. MQTT's Quality of Service framework offers three distinct reliability levels :

- **At most once (QoS 0):** QoS 0 offers "fire and forget" messaging with no acknowledgment from the receiver.
- **At least once (QoS 1):** QoS 1 ensures that messages are delivered at least once by requiring a PUBACK acknowledgment.
- **Exactly once (QoS 2):** QoS 2 guarantees that each message is delivered exactly once by using a four-step handshake (PUBLISH, PUBREC, PUBREL, PUBCOMP).



## 3.6 Conclusion

This chapter has introduced the main technologies that serve as the foundation of our smart object detection system. We explored the ESP32-CAM, a compact microcontroller with a built-in camera ideal for image capture in IoT environments, and the ESP32 board, which acts as an MQTT subscriber to process detection results. The Raspberry Pi 5 was presented as the central controller running Home Assistant to orchestrate automation tasks. Finally, we examined the MQTT protocol, which ensures lightweight, scalable, and reliable communication between distributed devices in the system.

With a solid understanding of the hardware and communication technologies involved, the next chapter focuses on the practical implementation. We will detail the setup, configuration, and integration of the system components, and demonstrate how they interact to perform real-time object detection and trigger automation workflows.

# Chapter 4

## Implementation and Automation of the Object Detection System

### 4.1 Introduction

This chapter presents the practical implementation of the object detection system for smart home automation. It begins with the installation and configuration of Home Assistant on a Raspberry Pi 5, which serves as the central hub for monitoring and automation tasks. The MQTT communication protocol is then set up using the Mosquitto broker to enable efficient and reliable message exchange between components.

Next, we describe the development of the object detection model using Edge Impulse, covering data acquisition, model training, and deployment to the ESP32-CAM. We also explain how the ESP32 microcontroller is configured as an MQTT subscriber to react to detection events.

Finally, we define and implement real-world automation scenarios within Home Assistant, demonstrating how the system reacts to object detection results, triggering notifications or controlling actuators, in a smart home context.

### 4.2 Installing Home Assistant on Raspberry Pi 5

The installation of Home Assistant was carried out using a Raspberry Pi 5 as the primary hardware platform. The setup required additional components, including a microSD card, a stable power supply, and a reliable internet connection.

We started by downloading the Home Assistant Operating System image specifically designed for the Raspberry Pi architecture. This image was flashed onto the microSD card using a utility such as Balena Etcher.

Once the microSD card was prepared, it was inserted into the Raspberry Pi. After powering on the device, an initialization period of approximately 5 to 10 minutes was necessary.

Upon completion, the Home Assistant interface became accessible via a web browser at:

- <http://homeassistant.local:8123>
- <http://192.168.243.14:8123> — where 192.168.243.14 is the static IP address assigned to the Raspberry Pi on the local network. To ensure consistent network accessibility, a DHCP reservation was configured in the router to assign a fixed IP to the Raspberry Pi based on its MAC address.

### 4.2.1 Initial Configuration

When accessing <http://homeassistant.local:8123/> for the first time, the Home Assistant onboarding interface appears, guiding the user through a few essential configuration steps.

These steps include:

- Creating a user account.
- Setting up basic preferences such as the home name, location, and time zone.
- Enabling network discovery.

Once the initial setup is complete and Home Assistant has started, an intuitive web-based dashboard is displayed, as shown in Figure 4.1. The main interface features a left-hand sidebar that provides access to key sections such as **Overview**, **Map**, **Logbook**, **History**, **Settings**, and **Developer Tools**. The central area presents a customizable dashboard made up of cards representing different devices, sensors, or automation states. These cards offer real-time data and interactive controls for elements such as lights, cameras, switches, and other integrated components. Users can personalize the layout by adding, removing, or rearranging cards and views according to their preferences.

Once the setup is complete, Home Assistant begins scanning the local network to automatically detect and suggest integration with compatible smart devices, as shown in Figure 4.2.

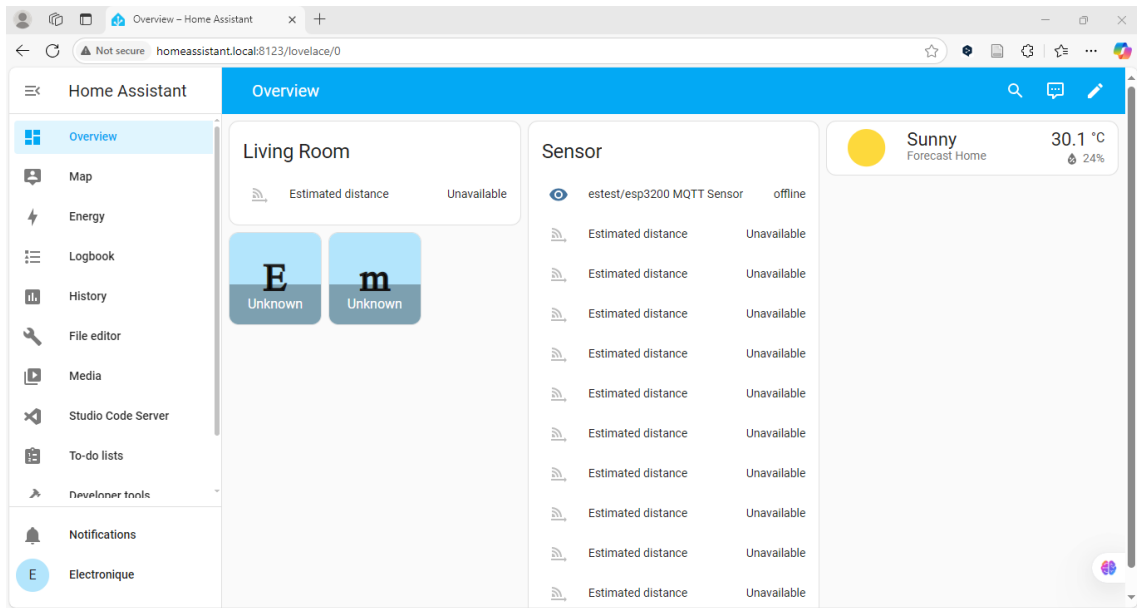


Figure 4.1: Home Assistant interface after startup.

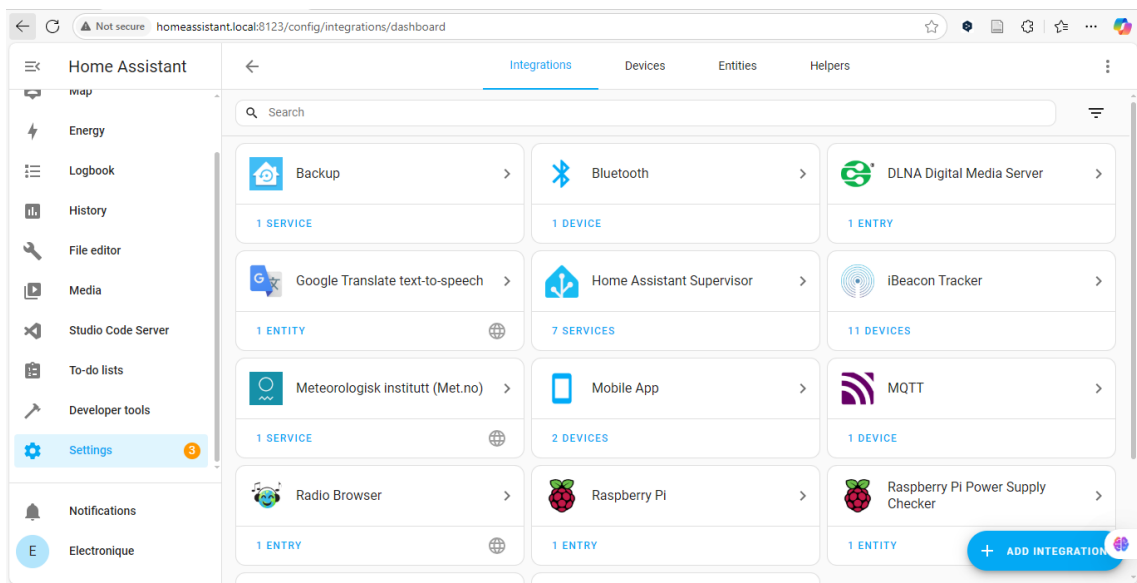


Figure 4.2: Integration menu showing discovered devices.

## 4.2.2 Adding Devices and Integrations

To integrate MQTT functionality into the Home Assistant setup, the Mosquitto broker add-on was used. This process involved:

1. Navigating to the **Settings** menu.
2. Accessing the **Add-ons** section.
3. Searching for the **Mosquitto broker** add-on and initiating the installation.
4. Once installed, the service was activated by clicking **Start**.

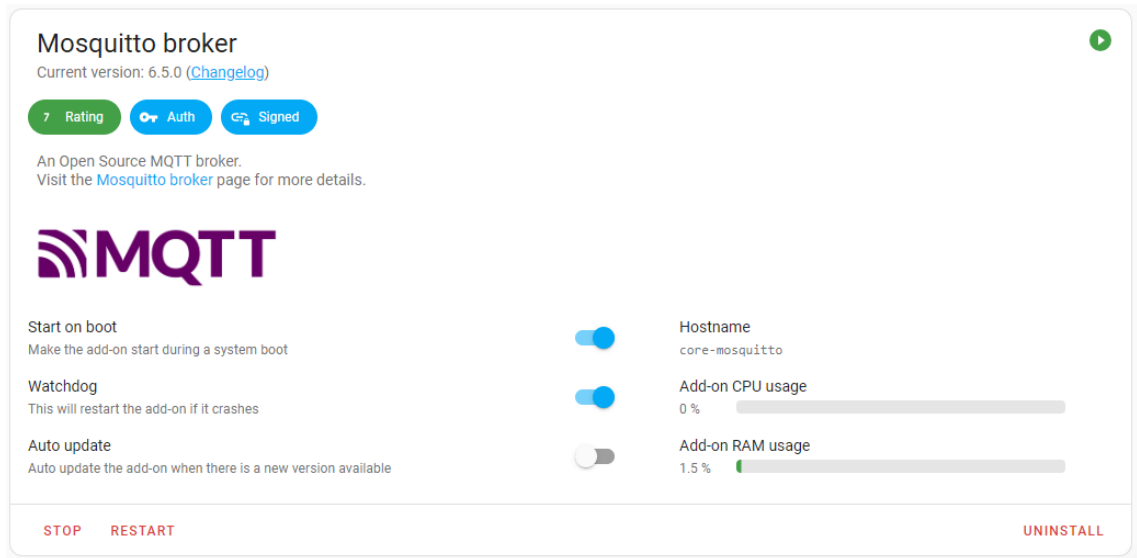


Figure 4.3: Installing the Mosquitto broker add-on.

This integration enables MQTT communication within Home Assistant, allowing seamless interaction with devices such as the ESP32 and ESP32-CAM.

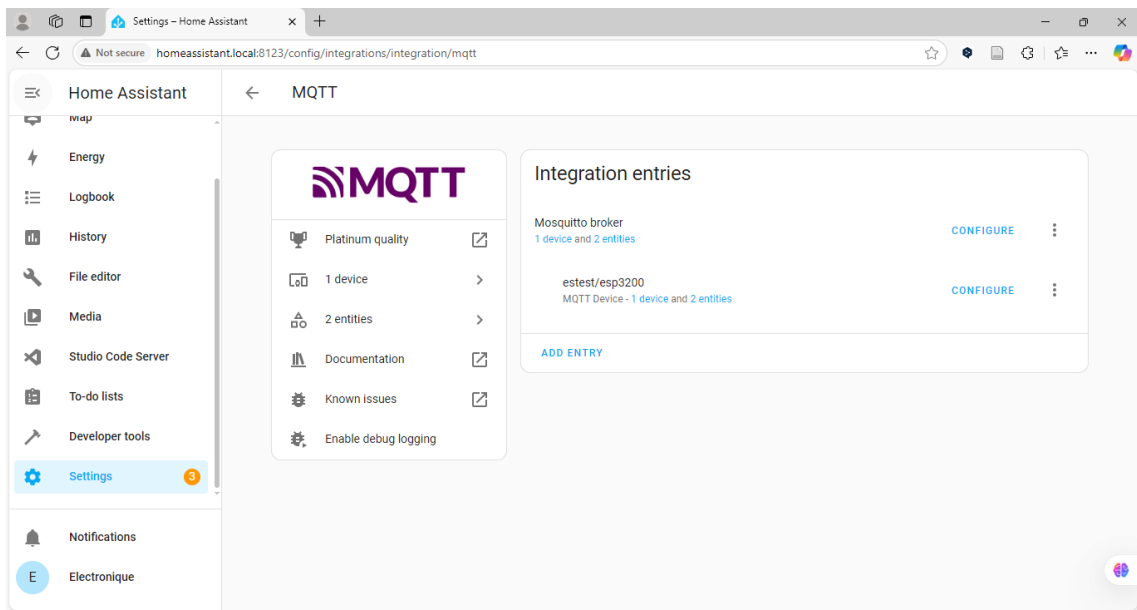


Figure 4.4: Mosquitto broker configuration within Home Assistant.

### 4.3 Edge Impulse-Based Object Detection: Model Training and Deployment

Edge Impulse offers a comprehensive platform for developing machine learning models optimized for edge devices such as the ESP32-CAM. This section details the process of integrating Edge Impulse with the ESP32-CAM.

The workflow in Figure 2.3 illustrates the key steps involved in building the model, from data collection to deployment on the ESP32-CAM.

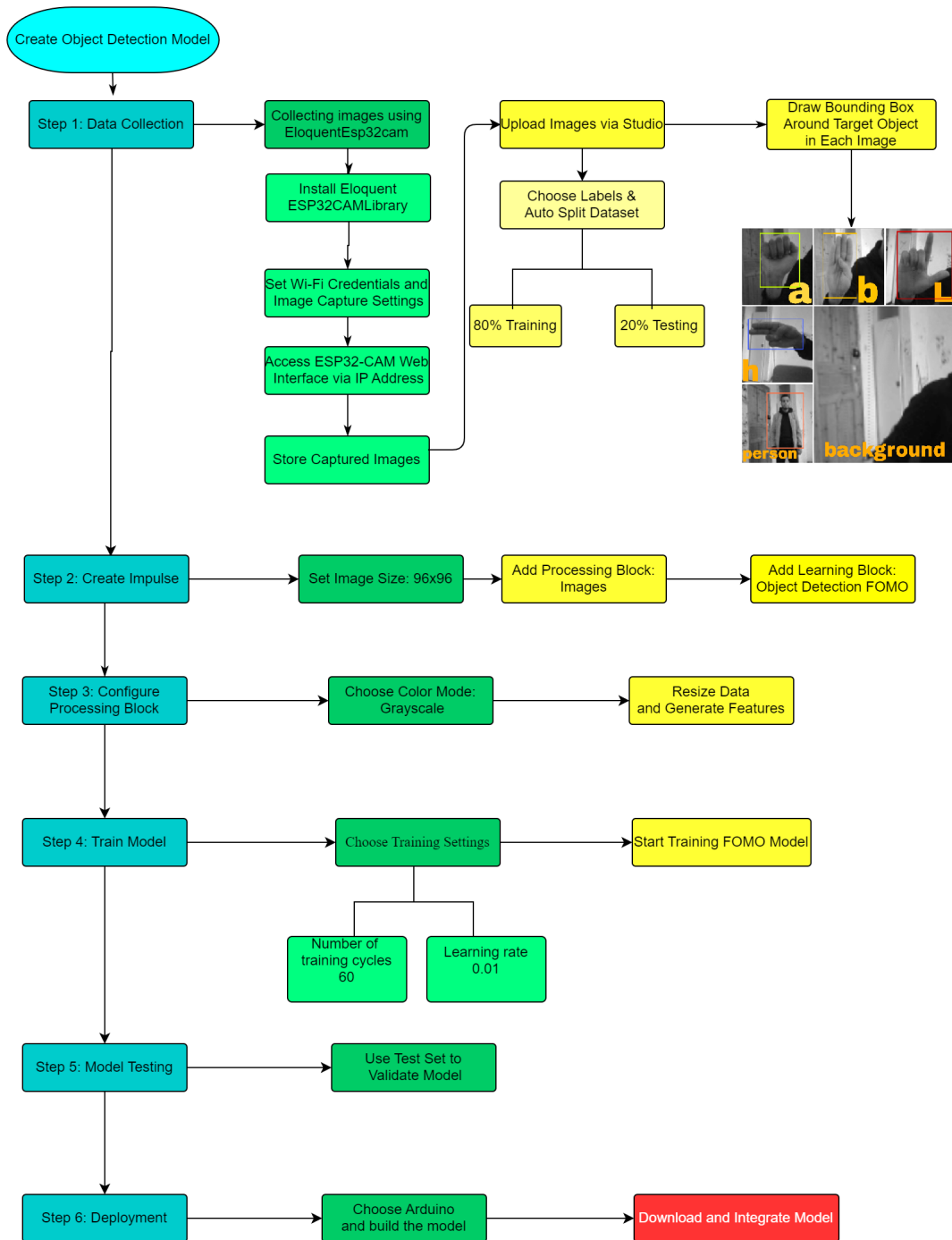


Figure 4.5: Edge Impulse Development Workflow.

### 4.3.1 Data Collection and Dataset Creation

The first step in developing an object detection system with the ESP32-CAM is to build a well-structured dataset. This process consists on :

- Capturing raw image data using the ESP32-CAM or other compatible devices.
- Labeling the collected images to create an annotated dataset suitable for training.

In our case, we initially experimented with capturing images using both a mobile phone and the ESP32-CAM. For the ESP32-CAM, it was necessary to flash the firmware with the `EloquentEsp32CAM` library to enable image streaming and capture. However, we quickly realized that using the ESP32-CAM was more suitable, as it allowed us to collect data in conditions closer to the system's real-world deployment environment. Captured images were saved locally and organized into labeled folders corresponding to different object classes, ensuring a structured dataset ready for upload and annotation in Edge Impulse.

In this project, we collected images for five categories: four different hand signs labeled "a", "b", "h", and "L", as well as the "Person" class. The Figure 4.6 shows the five categories.

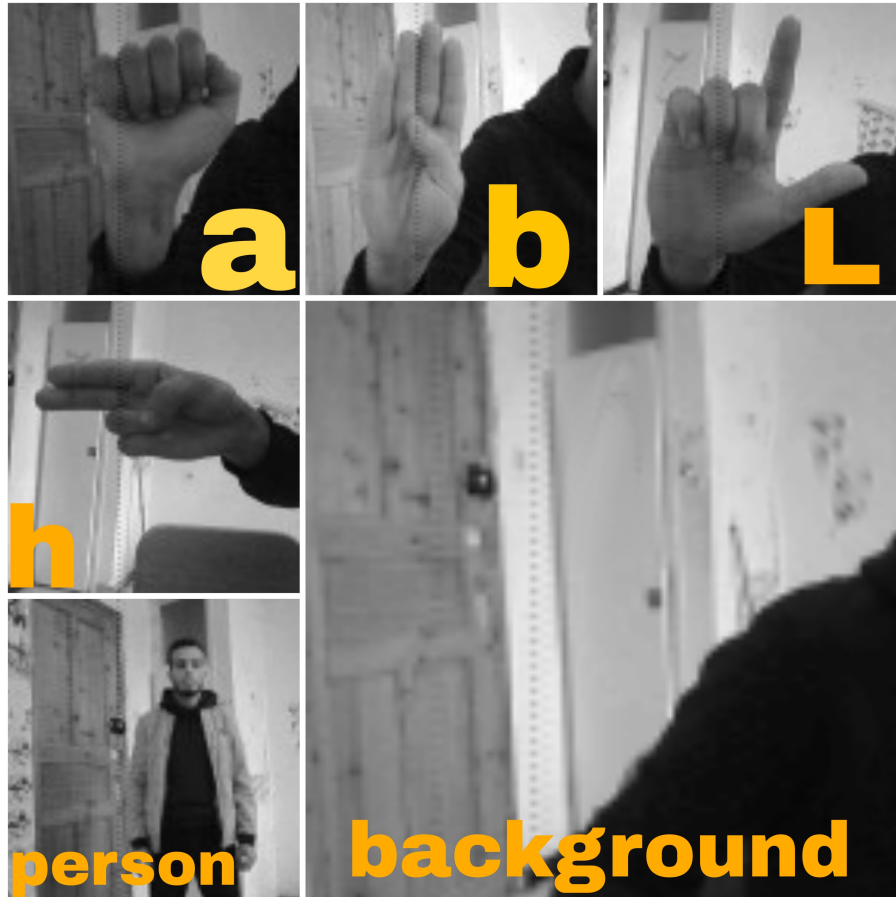


Figure 4.6: Collected dataset showing hand sign classes and person detection.

High-quality and diverse data are essential for training a robust machine learning model. Generally, model accuracy improves with both the volume and variability of the training dataset. Therefore, it is crucial to include images of the target object taken from multiple angles, under different lighting conditions, and against varied backgrounds. The background images, in particular, play a significant role. A dataset enriched with diverse background conditions, such as changes in lighting, textures, and colors, helps the model generalize better and recognize the target object more accurately in real-world scenarios.

This diversity not only improves the precision of object detection but also minimizes the risk of overfitting. By training the model with heterogeneous data, we increase its adaptability to unseen inputs and challenging environments.

### 4.3.2 Labeling Phase

After uploading the images from folders to Edge Impulse, we chose the option to automatically split the dataset into :

- **78%** of the images being used for training the model,
- **22%** reserved for testing.

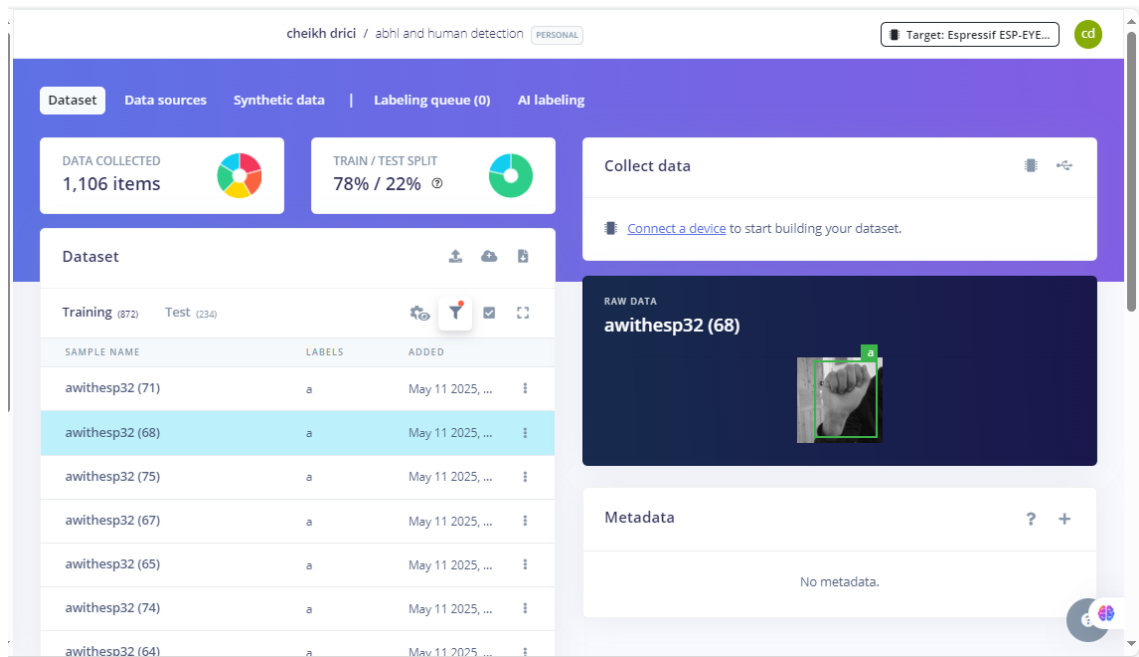


Figure 4.7: Dataset upload and automated train/test split.

Then, we proceeded to the labeling phase, an essential step that significantly impacts model performance. It consists to :

- Proceed to the Labeling Queue to access the images.
- Draw bounding boxes around the target object.
- Assign the appropriate label to each box ("a", "b", "h", "L", "Person"), except for background regions, which should not be labeled.



Accuracy and consistency during annotation are essential to ensure the model focuses on relevant features while minimizing noise. They directly improve the quality of the training data and significantly enhance the model's accuracy.

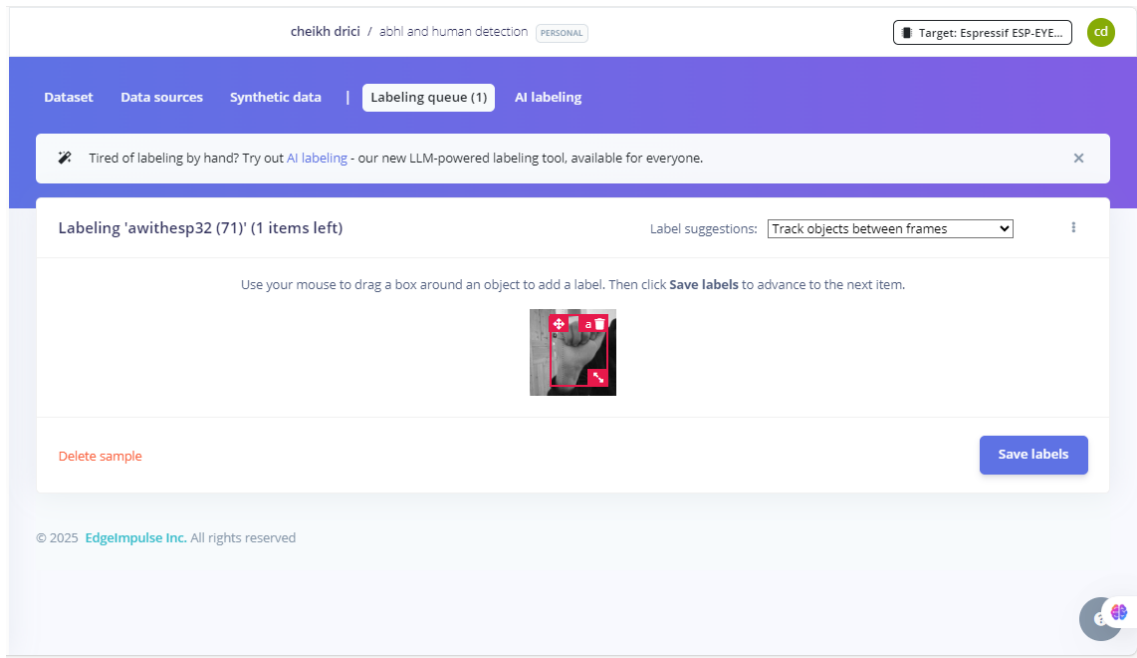


Figure 4.8: Labeling phase with bounding boxes applied to hand signs.

### 4.3.3 Designing an Impulse

In this step, we constructed an Impulse, , as shown in Figure 4.9, which is a core pipeline in Edge Impulse. It defines how input data is processed, features are extracted, and classification is performed. An Impulse consists of three main components:

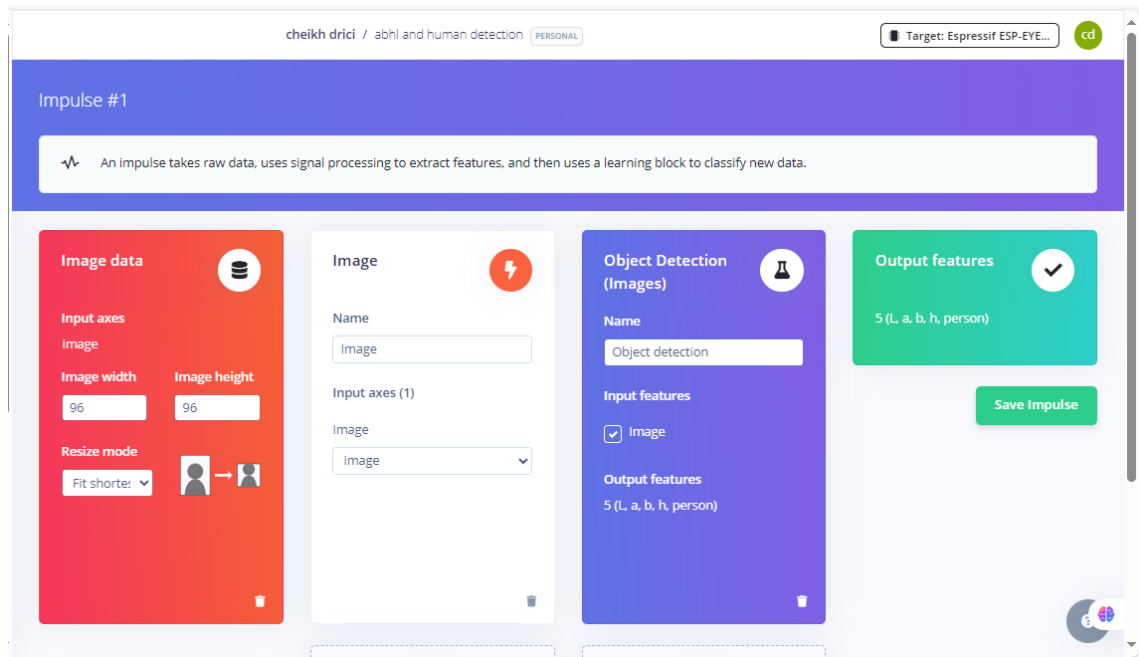


Figure 4.9: Designing an Impulse

## 1. Input Block

This block defines the data type for training. In our case, it is **Image data**. Edge Impulse automatically recognizes the input block when the dataset is uploaded. We adjusted the image dimensions to be square (96x96) for compatibility with Edge Impulse's pre-trained models.

## 2. Processing Block

This block extracts features from the input data. The operations vary depending on the data type. In the Image tab, we set grayscale for the color depth of image (Figure 4.10) to reduce memory usage, resulting in 9,216 features ( $96 \times 96 \times 1$ ), compared to 27,648 features for RGB ( $96 \times 96 \times 3$ ).

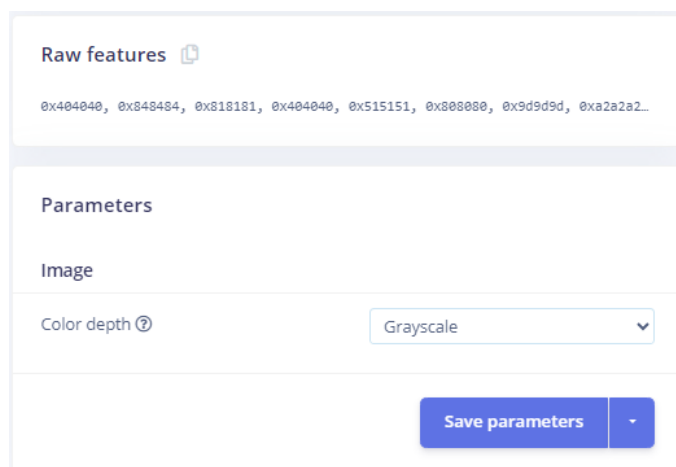


Figure 4.10: Configuration the image color depth .

Subsequently, we proceeded to the Generate Features step, where the dataset was processed and visualized within the Feature Explorer (Figure 4.11). This step provides a graphical representation of the extracted features, allowing for an initial assessment of data quality and class separability.

As observed, the Feature Explorer reveals that the samples corresponding to the labels “a”, “b”, “h”, “L”, and “person” are not entirely well separated. They appear somewhat scattered and do not form perfectly distinct clusters, which can potentially affect the model’s classification accuracy. This dispersion is largely attributed to the limitations of the ESP32-CAM in capturing consistent image features. Therefore, collecting and adding more training images would be beneficial, as it may help improve the separation and clustering of the different labels.

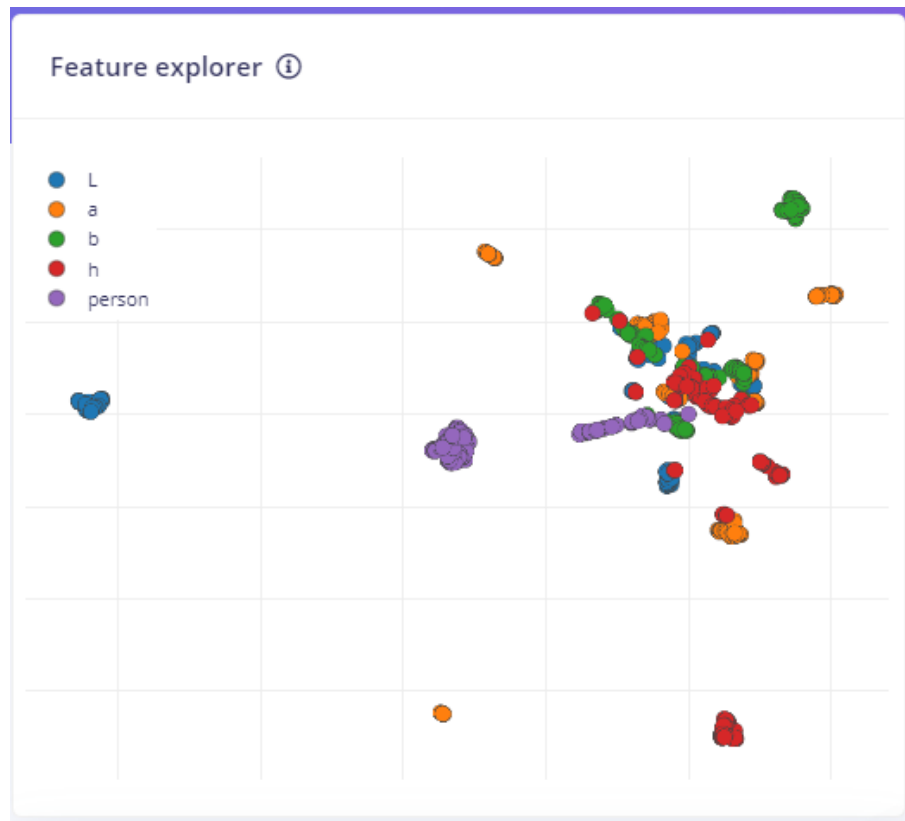


Figure 4.11: Feature Explorer

### 3. Learning Block

This block contains the neural network used to train the model. We selected the Object Detection (Images) learning block, which is well-suited for our use case. The training phase involves selecting neural network settings and tuning the parameters:

- Input Layer: 9,216 features (from the preprocessing step).

- Feature Extractor: FOMO (Faster Objects, More Objects) with MobileNetV2 0.35, which internally consists of multiple convolutional layers. It is the model which works with ESP32.
- Output Layer: 5 classes ("a", "b", "h", "L", and "Person").

The screenshot displays the 'Neural Network settings' window. Under 'Training settings', the 'Number of training cycles' is set to 60, 'Use learned optimizer' is unchecked, 'Learning rate' is 0.01, 'Training processor' is set to CPU, and 'Data augmentation' is checked. The 'Neural network architecture' section shows a flow from an 'Input layer (9,216 features)' to the 'FOMO (Faster Objects, More Objects) MobileNetV2 0.35' model, with a 'Choose a different model' button below it, and finally to an 'Output layer (5 classes)'.

Figure 4.12: Neural Network Settings

### 4.3.4 Results and discussion :

We experimented with several combinations of epochs and learning rates to optimize model performance. The best results were obtained with 60 epochs and a learning rate of 0.01, which led to an overall F1 score of 93.3% on the validation set.

Choosing the right number of epochs is essential : Too few can epochs result in under-fitting, while too many may cause overfitting. We should choose 60 epochs for a balance.

A high learning rate speeds up training but may cause overshooting. A low rate ensures precision but slows convergence. We used 0.01 for stable and efficient learning.

- **The background** class being perfectly classified indicates the model successfully ignored irrelevant regions.

- The **hand signs** classes :

- "L" achieved a 93.9% F1 score, with minimal misclassification (6.7%) as background. This is called a confusion matrix.
- "A" was also well-identified, with a 92.9% F1 score. This indicates strong performance for class "A", with an F1 showing both good precision and recall. However, the 7.1% misclassification as background implies that a few instances of "A" were not detected at all.
- "B" reached 100% precision and recall, indicating excellent separability.
- "H" showed a slight weakness, with 20% of its instances misclassified as background. This indicates the model struggled to consistently detect this class, potentially due to insufficient training examples, visual similarity with the background. Nevertheless, an F1 score of 0.86 suggests reasonably good performance overall.

- The **Person** class was detected with high precision and recall, achieving an F1 score of 0.98. Only a small percentage (3.4%) was incorrectly classified as background.

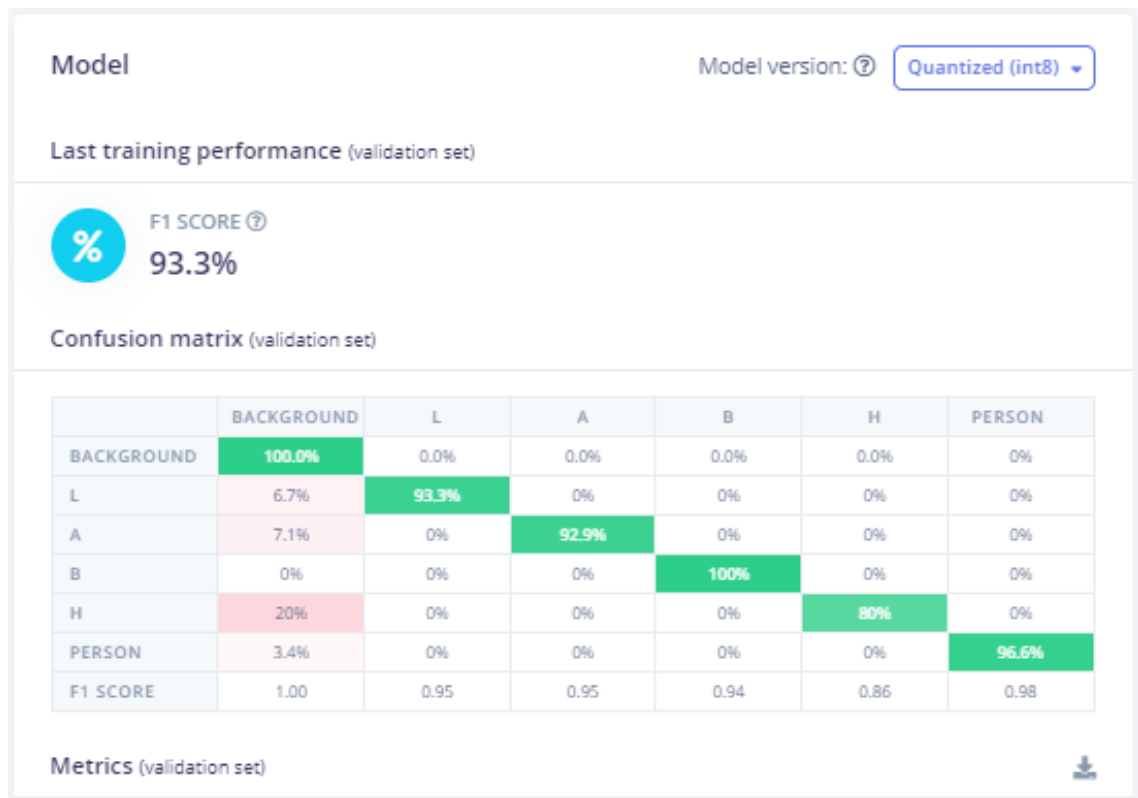


Figure 4.13: Training Results with 60 Epochs and Learning Rate of 0.01.

### 4.3.5 Validating Our Model

After training, we validated the model using the test dataset to ensure that it generalized well to unseen data and had not simply memorized the training samples. In the Model Testing section, we used the Classify All option to evaluate the entire test set. The model achieved a test accuracy of 93.16%, as shown in Figure 4.14, demonstrating strong performance and reliable object detection capabilities.

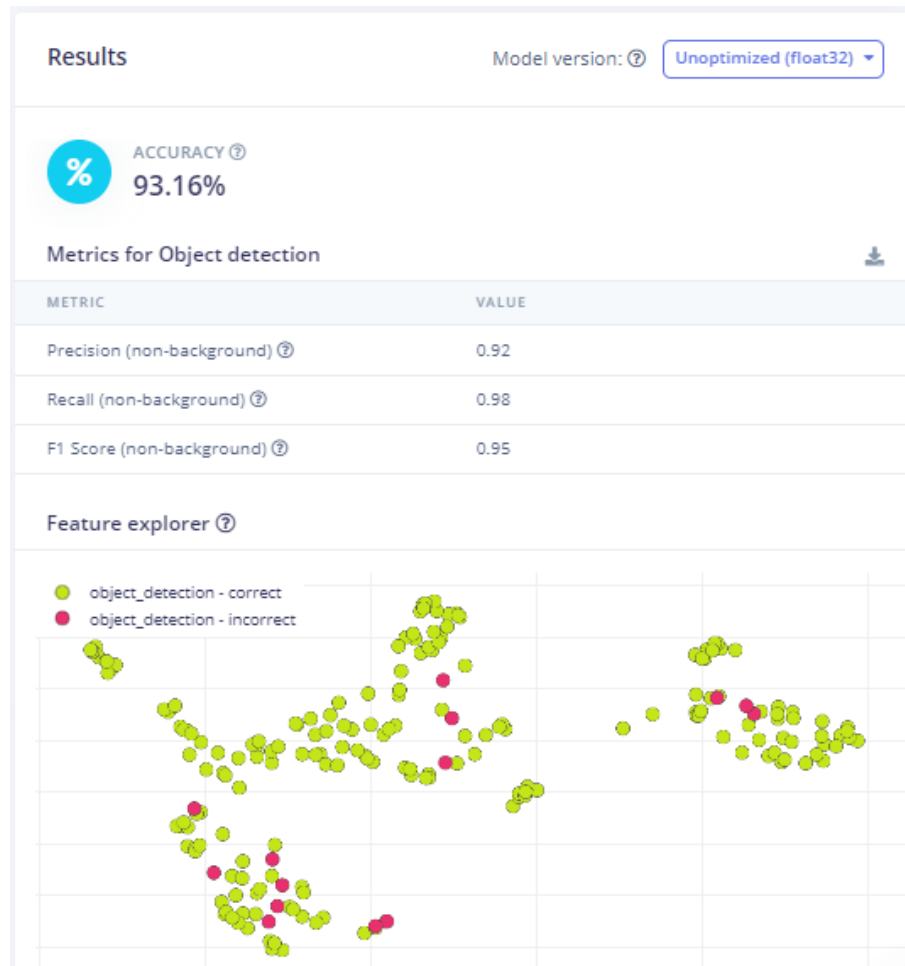


Figure 4.14: Model Validation Results.

### 4.3.6 Exporting and deploying the trained model to the ESP32-CAM

The next step is to generate an Arduino Library to deploy our model in Arduino IDE. To do this, we navigate to the Deployment tab in the edge impulse and select the Arduino Library as the export format. We choose the Quantized (int8) option along with the TFLite compiler, which ensures low memory usage and enables offline inference. Once the build is complete, we download the generated .zip file containing an Arduino-compatible library for ESP32-CAM.

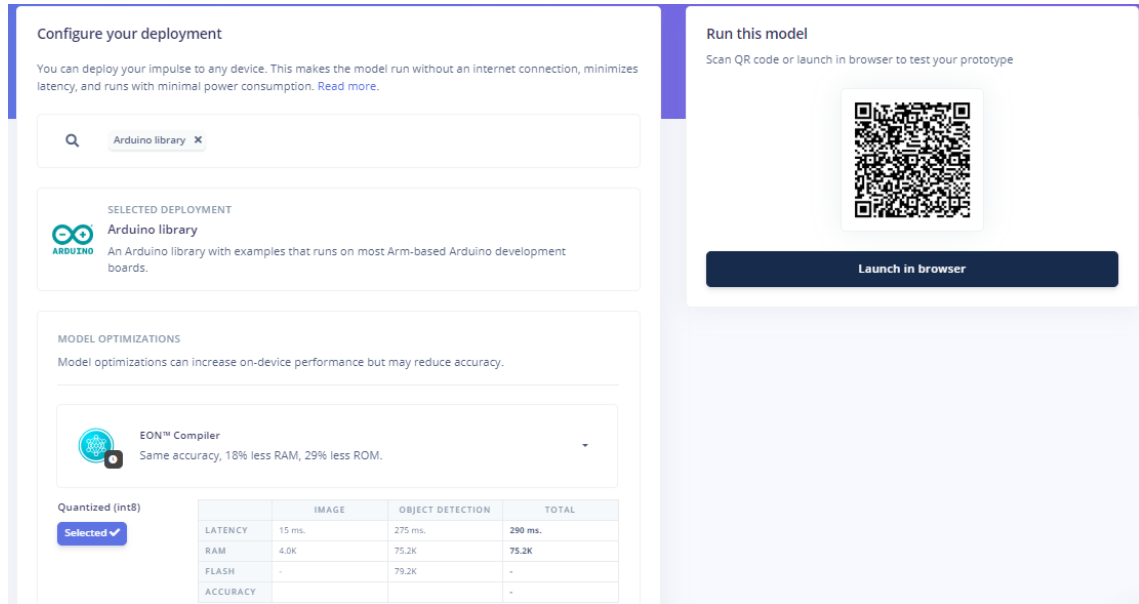


Figure 4.15: Running the Impulse Using a Generated Arduino Library

## 4.4 Deploying the Model Using the Arduino IDE

Before deploying our model to the ESP32-CAM development board, we first need to install the appropriate board support package in the Arduino IDE. This is done by adding the following URL to the Additional Board Manager URLs field under the Preferences menu:

[https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package\\_esp32\\_dev\\_index.json](https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_dev_index.json)

Then, we install the package named esp32 by Espressif Systems. This enables support for ESP32 boards, including the ESP32-CAM, within the Arduino environment.

Next, we proceed to upload our trained model by importing the downloaded Arduino library (.zip file) into the Arduino IDE. This library contains the model and the necessary files to run inference directly on the AI Thinker ESP32-CAM.

Once running, the camera captures a live image, and the model compares it with the patterns it learned during training. Based on this comparison, it detects and classifies the object within the image, such as recognizing a specific hand sign or identifying a person.

Inference results are printed on the serial monitor, showing the detected class and confidence score, along with the position and size of the bounded box in the image as shown in Figure 4.16.

For example : found a "b" (score: 0.98) at (x: 16, y: 16, width: 16, height: 16)

```

Output  Serial Monitor X
Message (Enter to send message to 'AI Thinker ESP32-CAM' on 'COM8')

Object detection bounding boxes:
  h (0.550781) [ x: 16, y: 24, width: 8, height: 8 ]
Predictions (DSP: 1 ms., Classification: 178 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  L (0.890625) [ x: 24, y: 16, width: 8, height: 8 ]
  b (0.550781) [ x: 24, y: 24, width: 8, height: 8 ]
Predictions (DSP: 1 ms., Classification: 177 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  b (0.671875) [ x: 16, y: 24, width: 8, height: 8 ]
Predictions (DSP: 1 ms., Classification: 178 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  b (0.980469) [ x: 16, y: 16, width: 16, height: 16 ]

Object detection bounding boxes:
  a (0.917969) [ x: 24, y: 16, width: 8, height: 8 ]
  L (0.656250) [ x: 16, y: 24, width: 8, height: 8 ]
Predictions (DSP: 1 ms., Classification: 177 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  person (0.753906) [ x: 16, y: 24, width: 8, height: 8 ]
Predictions (DSP: 1 ms., Classification: 175 ms., Anomaly: 0 ms.):

```

Figure 4.16: Inference results.

## 4.5 Integrating MQTT for Prediction Publishing to Home Assistant

To enable communication between the ESP32-CAM and Home Assistant, we integrated MQTT (Message Queuing Telemetry Transport). In this setup, the ESP32-CAM acts as a publisher, sending prediction labels to a specific MQTT topic. Home Assistant, or any MQTT-compatible client, can then subscribe to this topic to receive and react to the predictions in real time.

To ensure a successful connection between the ESP32-CAM and Home Assistant via MQTT, it is essential to properly configure the MQTT broker within Home Assistant, as shown in Figure 4.17.



The screenshot shows the MQTT configuration interface in Home Assistant. The form is titled "MQTT" and contains the following fields and options:

- Broker\***: A text input field containing "core-mosquitto". Below it, a small text label reads: "The hostname or IP address of your MQTT broker."
- Port\***: A text input field containing "1883". Below it, a small text label reads: "The port your MQTT broker listens to. For example 1883."
- Username**: A text input field containing "mqtt\_test". Below it, a small text label reads: "The username to log in to your MQTT broker."
- Password**: A password input field containing "pfe2025". Below it, a small text label reads: "The password to log in to your MQTT broker."
- Advanced options**: A section with a toggle switch and the text "Enable and select **Next** to set advanced options."
- SUBMIT**: A blue button at the bottom right of the form.

Figure 4.17: MQTT configuration within Home Assistant.

## 4.5.1 Overview of MQTT Packet Structure

An MQTT Publish packet includes:

- **topicName:** The topic under which the prediction is published (e.g., `estest/esp3200/state`), allowing subscribers to receive relevant data.
- **QoS:** Quality of Service level (0 = at most once, 1 = at least once, 2 = exactly once).
- **Payload:** The content of the message, which in this case is the prediction label (e.g., "Person" or "a").

## 4.5.2 Enabling MQTT Communication with Home Assistant

To enable real-time communication with Home Assistant, we developed additional code within the inference sketch (generated by Edge Impulse) to handle MQTT communication. This allows ESP32-CAM to publish prediction results on a specific MQTT topic, which can then be received by an MQTT client integrated in Home Assistant.

This involved including the Wi-Fi and MQTT libraries required for communication with Home Assistant, setting up the necessary Wi-Fi and MQTT parameters, and implementing a minimum confidence threshold to ensure that only reliable predictions are published.

► **Include the required libraries:**

```
// Add WiFi library
#include <WiFi.h>
// Add MQTT libraries for Home Assistant communication
#include <PubSubClient.h>
#include <ArduinoJson.h>
```

► **Wi-Fi and MQTT Configuration:**

```
// WiFi and MQTT credentials
#define WIFI_SSID "Redmi Note 8 (2021)"
#define WIFI_PASSWORD "0016111200"

// MQTT Configuration : Home Assistant MQTT broker details
#define MQTT_BROKER "192.168.247.14" // Replace with your Home Assistant IP
#define MQTT_PORT 1883
#define MQTT_USERNAME "mqtt_test"
#define MQTT_PASSWORD "pfe2025"
#define MQTT_CLIENT_ID "esp32cam_handsign"

// MQTT topics
#define MQTT_TOPIC_STATE "etest/esp3200/state"
#define MQTT_TOPIC_AVAILABILITY "etest/esp3200/availability"
#define MQTT_TOPIC_CONFIG "homeassistant/sensor/esp32cam/human_detector/config" // Discovery topic
```

► **Minimum Confidence Threshold:**

```
// Set minimum confidence threshold for MQTT messages
#define MIN_CONFIDENCE 0.85 // Only send detections with 85% or higher confidence
```

### 4.5.3 Inference and MQTT Publishing Code for ESP32-CAM

The Figure 4.18 shows the flowchart that illustrates the Inference and MQTT Publishing Code for ESP32-CAM. The code essentially consists of:

1. **Setup and Initialization:**

- Initializes serial communication.
- Connects the ESP32-CAM to a Wi-Fi network.
- Configures and establishes the MQTT connection.
- Initializes the camera AI Thinker ESP32-CAM Module.

2. **Main Detection Loop:**

- Ensures MQTT is connected.
- captures a live image from the camera.
- Processes the image using the embedded machine learning model.
- If the predicted label has a confidence score greater than or equal to 0.85, it is published via MQTT.
- Releases used memory and repeats the process.

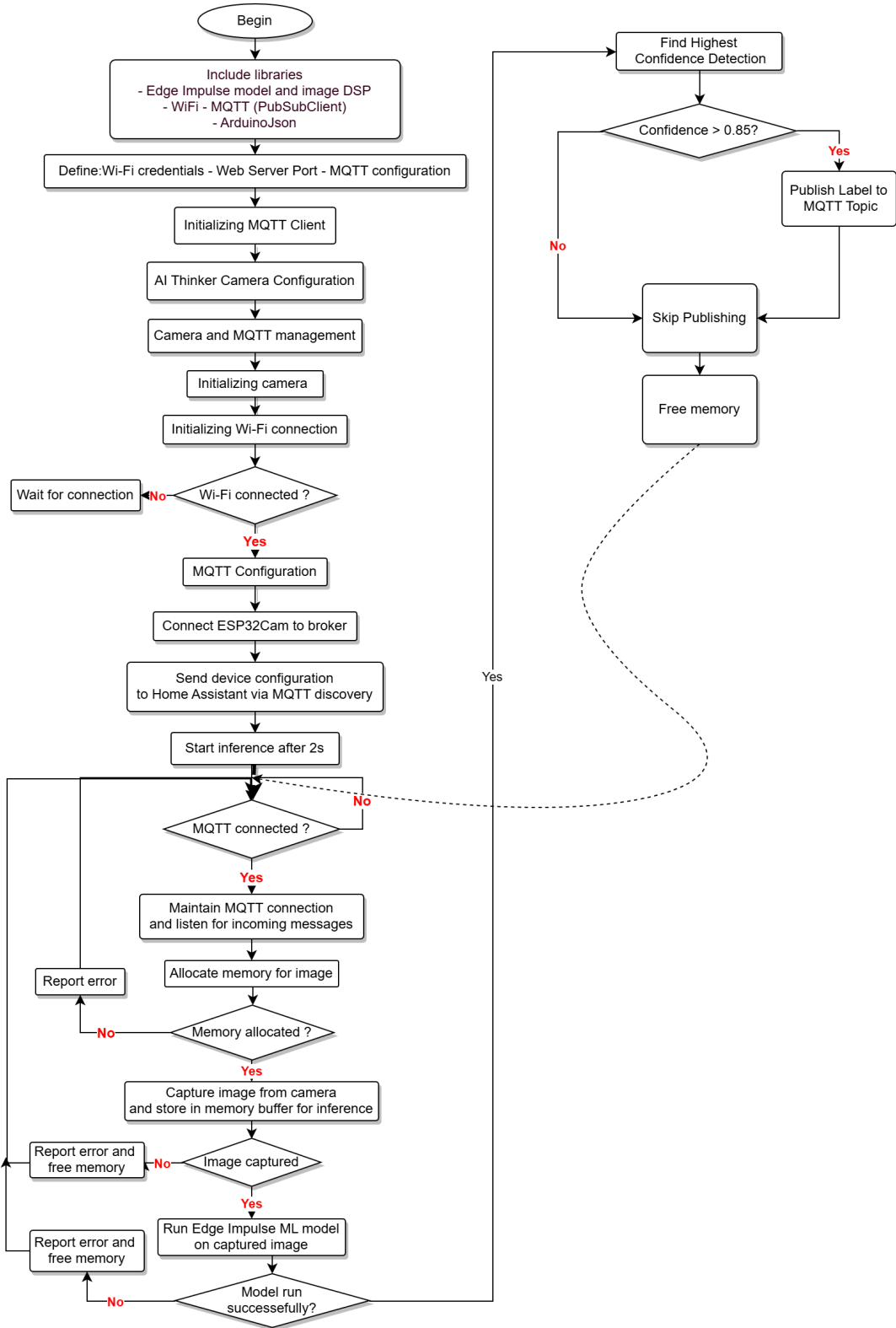
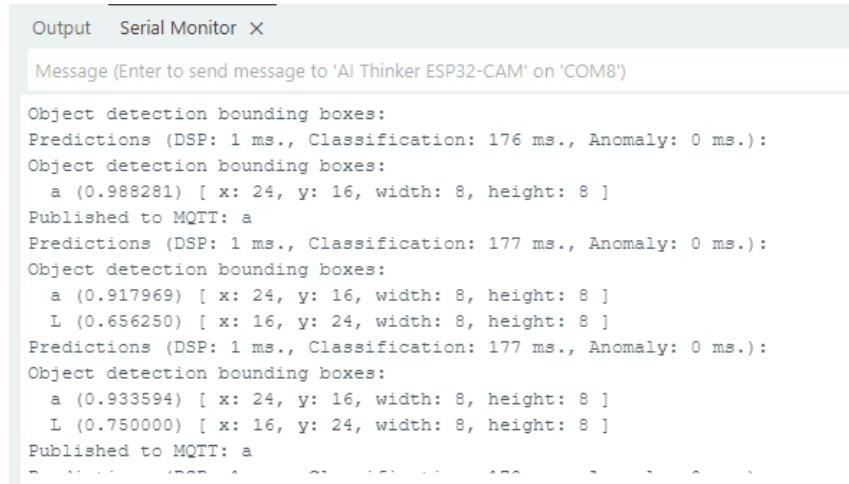


Figure 4.18: Smart Sensor Program: On-Device Inference and MQTT Publishing on ESP32-CAM.

### Results and discussion :

As observed in the Serial Monitor in Figure 4.19, the ESP32-CAM successfully publishes real-time prediction messages to the MQTT topic `estest/esp3200/state`. This shows that the device effectively operates as a smart vision sensor, capable of performing real-time object detection and transmitting results over MQTT to be integrated into Home Assistant.



```
Output Serial Monitor X
Message (Enter to send message to 'AI Thinker ESP32-CAM' on 'COM8')

Object detection bounding boxes:
Predictions (DSP: 1 ms., Classification: 176 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  a (0.988281) [ x: 24, y: 16, width: 8, height: 8 ]
Published to MQTT: a
Predictions (DSP: 1 ms., Classification: 177 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  a (0.917969) [ x: 24, y: 16, width: 8, height: 8 ]
  L (0.656250) [ x: 16, y: 24, width: 8, height: 8 ]
Predictions (DSP: 1 ms., Classification: 177 ms., Anomaly: 0 ms.):
Object detection bounding boxes:
  a (0.933594) [ x: 24, y: 16, width: 8, height: 8 ]
  L (0.750000) [ x: 16, y: 24, width: 8, height: 8 ]
Published to MQTT: a
```

Figure 4.19: ESP32-CAM correctly publishing prediction results to MQTT.

## 4.6 Automation Scenarios Using MQTT Communication

In this project, we implemented two types of automation based on MQTT communication. The first type involves programming the ESP32 as an actuator, where it subscribes on the MQTT topic and reacts to the prediction labels sent by the ESP32-CAM by performing predefined actions. The second type is a notification-based automation using Home Assistant, where the detection of the presence of a person, triggers an instant notification to the user.

### 4.6.1 Programming the ESP32 as an Actuator for MQTT Subscription

In this section, we developed a program in which the ESP32 acts as an actuator, subscribing to the topic `"estest/esp3200/state"`, and controlling the LEDs based on the message received, 'a', 'b', 'h' or 'L', from the MQTT broker.

However, in a real-world application, this ESP32 can be connected to a relay module, allowing it to control a wide range of household appliances. For example, it can switch lights, fans, door locks, or even smart plugs. In addition to relays, servomotors can also be used to control mechanical elements.

For example, a servo motor can be used to open or close window blinds, adjust a water valve, or even unlock a drawer. In a practical scenario, receiving the label "a" through MQTT could rotate the servo by 90° to open a flap or door, while receiving "b" could return it to the closed position.

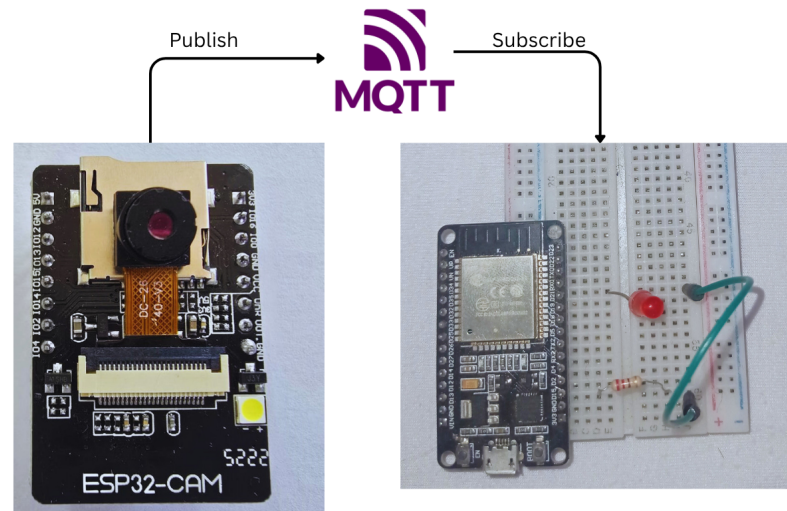


Figure 4.20: ESP32 Actuator Subscribing to MQTT Topic.

The Figure 4.21 shows the flowchart that illustrates the code, which essentially consists of:

### ► Wi-Fi and MQTT Configuration:

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

// WiFi credentials
const char* ssid = "Redmi Note 8 (2021)";
const char* password = "0016111200";

// MQTT Broker settings
const char* mqttServer = "192.168.182.14";
const int mqttPort = 1883;
const char* mqttUser = "mqtt_test"; // Optional
const char* mqttPassword = "pfe2025"; // Optional
```

### ► MQTT Subscription Configuration:

```
// MQTT Topic
const char* mqttTopic = "etest/esp3200/state";
```

### ► A callback function

A callback function processes incoming messages and controls the actuator accordingly. This means that whenever the ESP32 receives a new message from the MQTT broker on the subscribed topic, the predefined callback function is automatically triggered. It analyzes the message content, such as a detected label like 'a', 'b', 'h' or 'L', and executes the corresponding action. In our case, it activates the corresponding LED to indicate the detection result. This mechanism ensures real-time response, enabling the system to immediately react to object detection results from the ESP32-CAM.

```
void callback(char* topic, byte* payload, unsigned int length) {  
  // Process the incoming message  
  Serial.print("Message received on topic: ");  
  Serial.println(topic);  
  
  // Convert the message to a string  
  char message[length + 1];  
  for (int i = 0; i < length; i++) {  
    message[i] = (char)payload[i];  
  }  
  message[length] = '\0';  
  
  Serial.print("Message content: ");  
  Serial.println(message);  
}
```



Figure 4.21: Actuator Program Logic Flowchart.

## Results and discussion

1. **Detecting 'a'** As shown in the Figure 4.22, the model selects the prediction with the highest confidence,"a", which is the correct result. The actuator then receives this message from MQTT and turns on the LED.

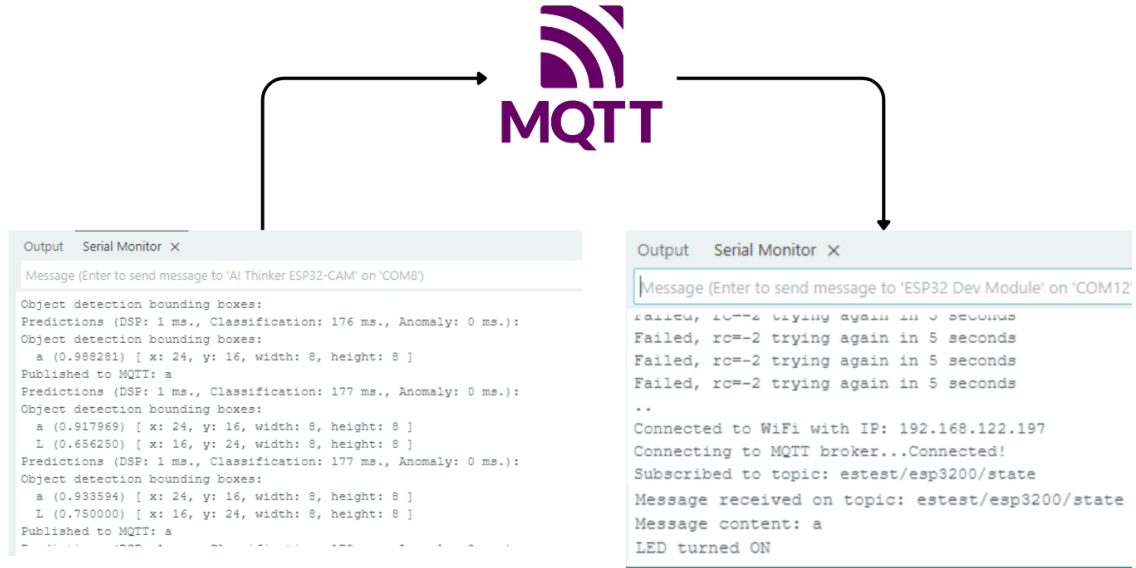


Figure 4.22: Serial monitor output of the sensor and actuator response upon detecting "a" .

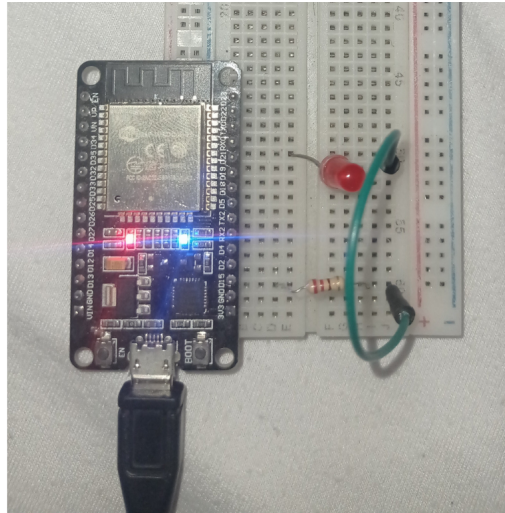


Figure 4.23: Actuator turns on the blue LED in Response to 'a' Hand Sign .

2. **Detecting 'b'** As shown in the Figure 4.24, the model confuses the "L" and "b" hand signs. However, it ultimately selects the prediction with the highest confidence—"b"—which is the correct result. The actuator then receives this message and turns off the LED.



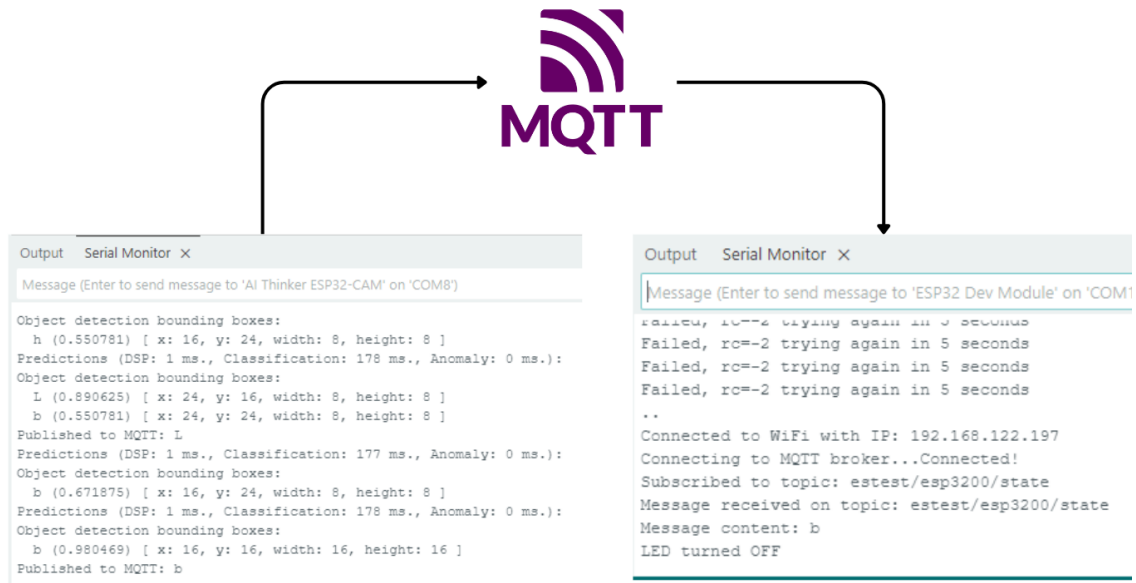


Figure 4.24: Serial monitor output of the sensor and actuator response upon detecting "b" .

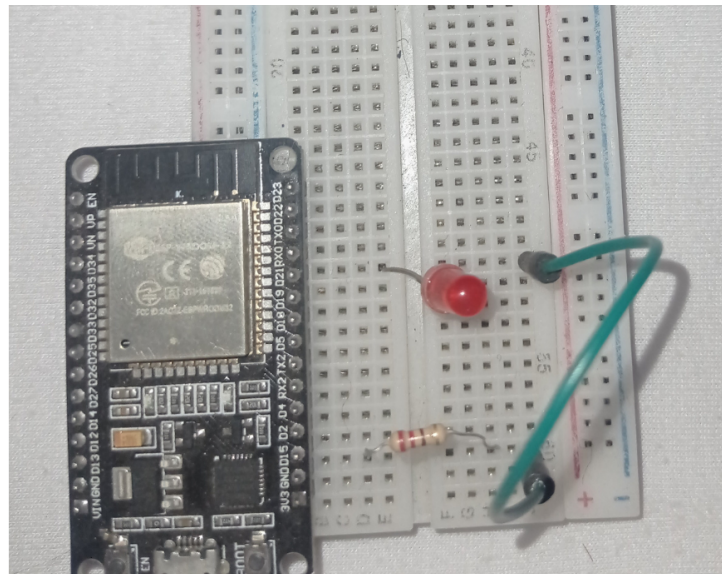


Figure 4.25: Actuator turns OFF the blue LED in Response to 'b' Hand Sign .

### 3. Detecting 'L'

The serial monitor output from the sensor, in Figure 4.26, shows that the "L" hand sign was detected, but with a confidence level below 0.85, so no message was sent. However, when the confidence exceeded 0.85, the message was successfully transmitted, and the actuator received it and turned on the LED.

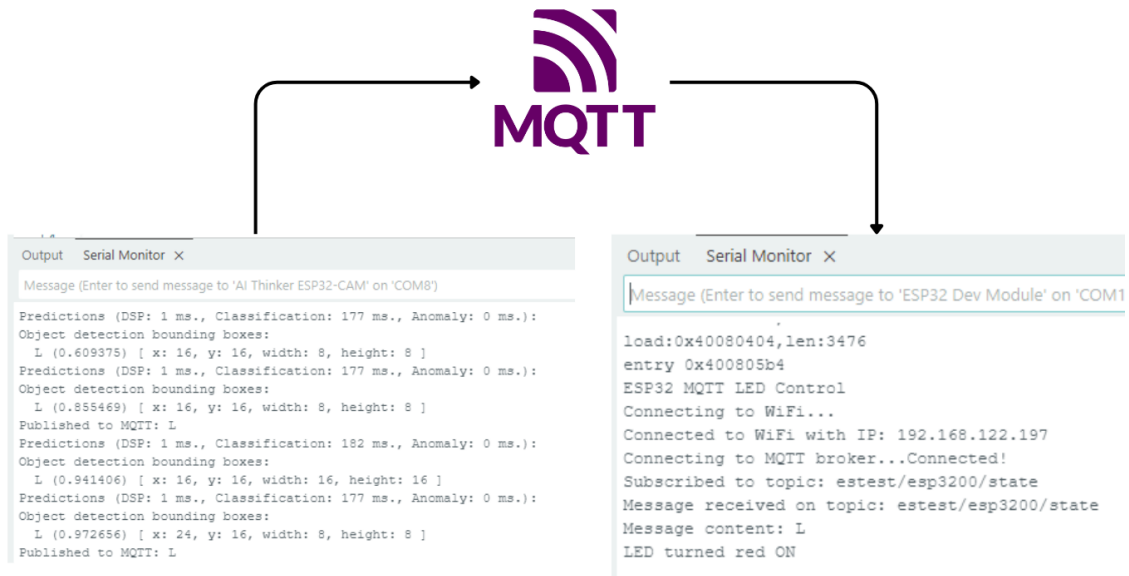


Figure 4.26: Serial monitor output of the sensor and actuator response upon detecting "L" .

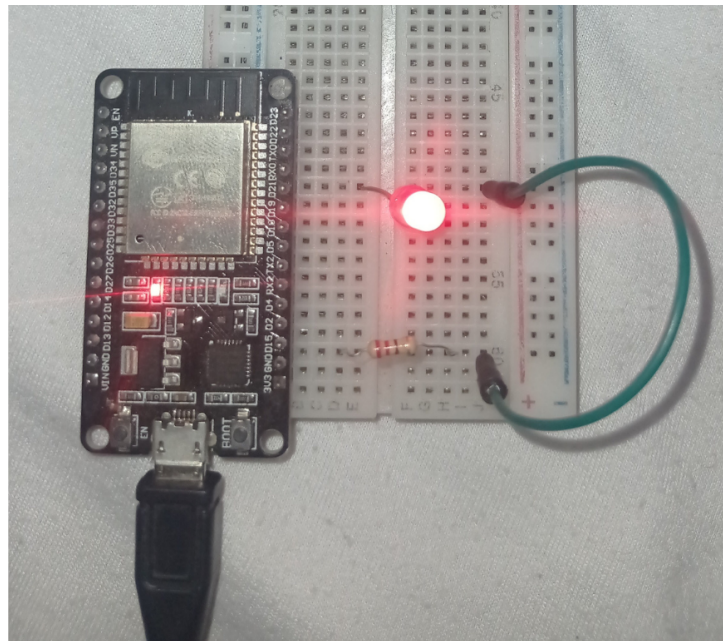


Figure 4.27: Actuator controls the Red LED in Response to 'L' Hand Sign .

4. **Detecting 'h'** The serial monitor output from the sensor , in Figure 4.26, shows that the "h" hand sign was detected perfectly immediately, with a confidence of 0.93, the message was successfully transmitted, and the actuator received it and turned off the LED.

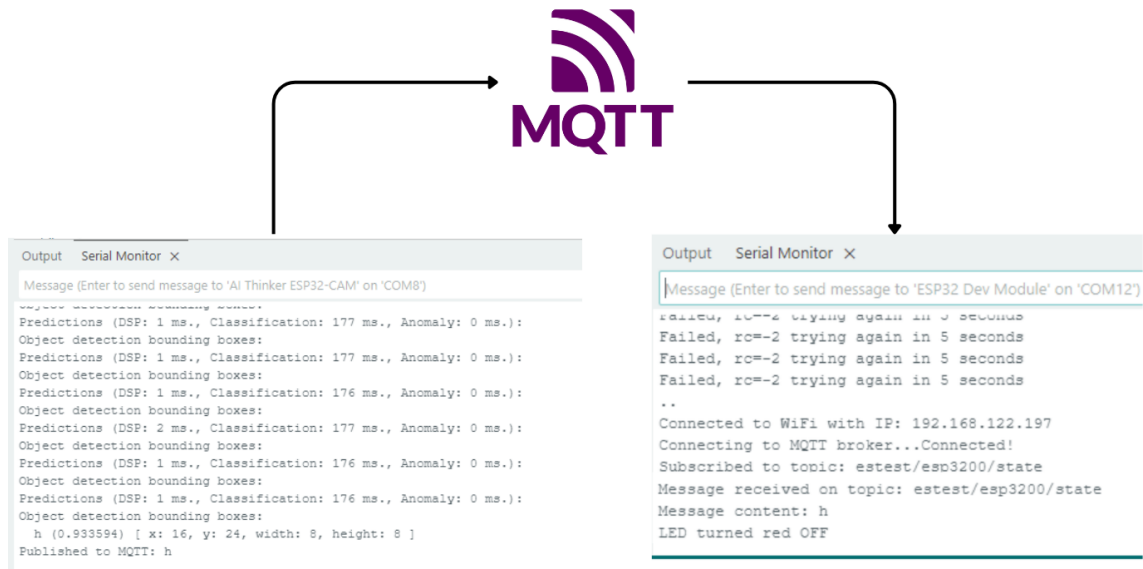


Figure 4.28: Serial monitor output of the sensor and actuator response upon detecting "h" .

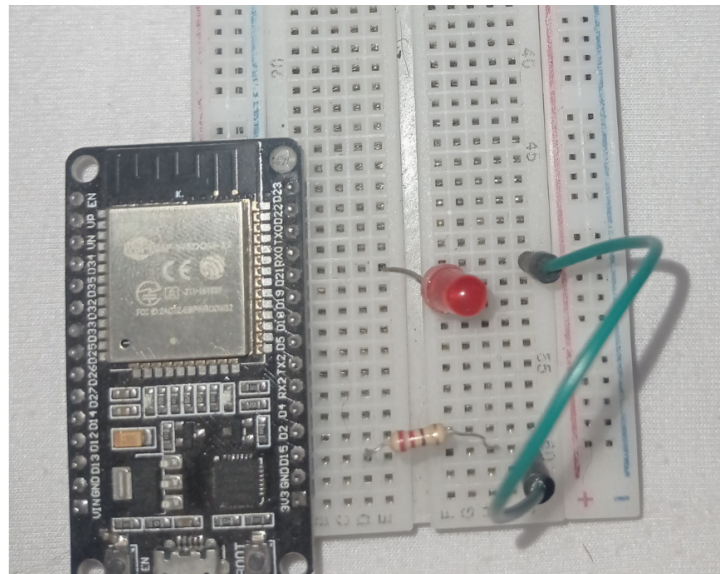


Figure 4.29: Actuator turns OFF the blue LED in response to 'h' Hand Sign .

## 4.6.2 Person Detection Notification Automation via MQTT

As part of the system's smart automation capabilities, we implemented a workflow that triggers a mobile notification when a person is detected by the ESP32-CAM. This automation was configured in Home Assistant, as shown in Figure 4.30, and the setup involves the following components:

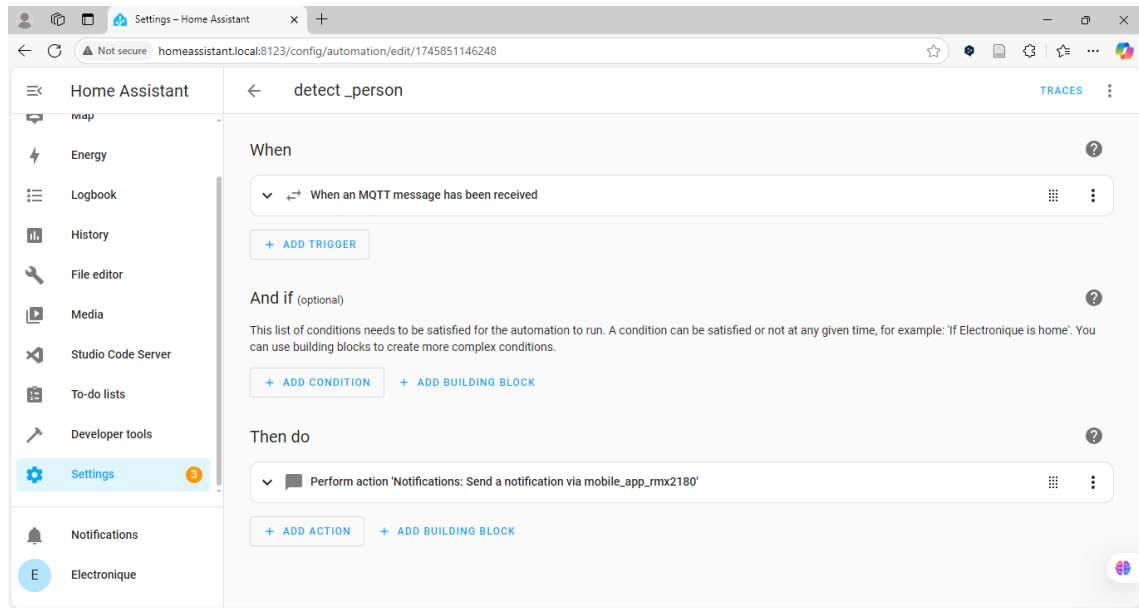


Figure 4.30: Automation setup in Home Assistant.

## ► Trigger

The automation is initiated when an MQTT message is published from the ESP32-cam device on `estest/esp3200/state` topic. This message is transmitted in real-time when the object detection model, deployed on the ESP32-CAM using Edge Impulse, detects a person in the camera feed.

## ► Condition

The automation proceeds only if the payload of the MQTT message contains the exact label **"person"**. This condition ensures that the system responds exclusively to person-detecting events, filtering out any unrelated classifications or noise from the object-detection model.



Figure 4.31: Conditional check for MQTT payload containing "person".

## ► Action

Once the condition is satisfied, the system sends a push notification to the smartphone of the user via its mobile app. This enables real-time alerts for person detection events, enhancing the effectiveness of the system in smart surveillance applications.

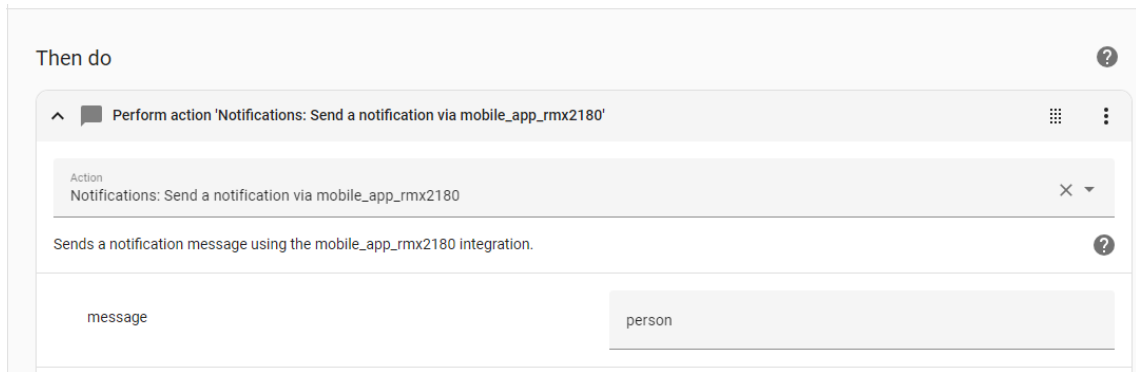


Figure 4.32: Notification sent via Home Assistant mobile app.

The full process can be summarized as follows:

- ESP32-CAM detects "person" with confidence  $\geq 0.85$ .
- The model publishes the label "person" on the MQTT topic `estest/esp3200/state`.
- Home Assistant receives the message.
- An automation rule is triggered.
- A mobile notification alerts the user.

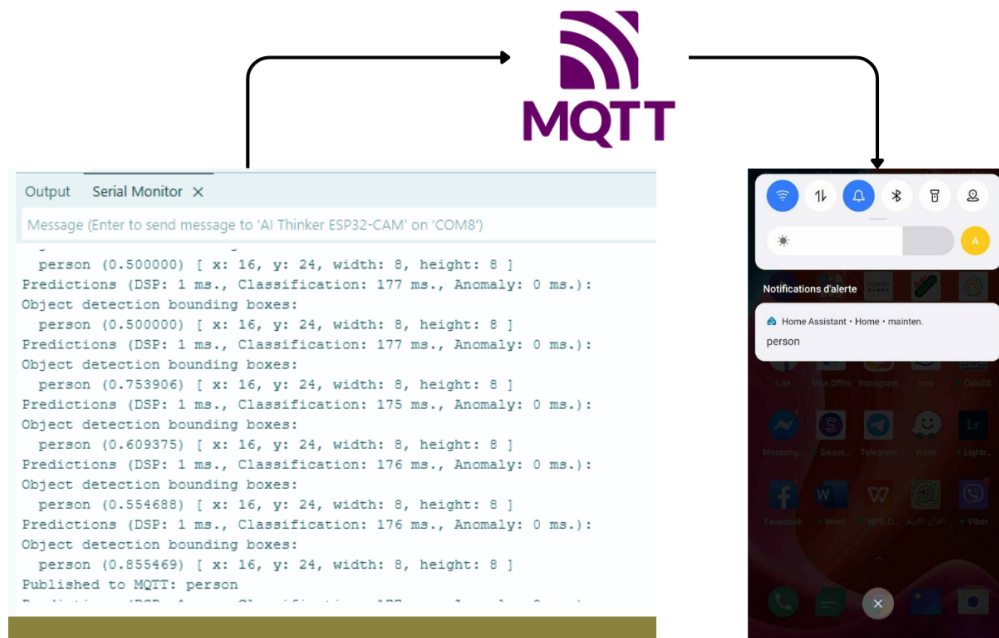


Figure 4.33: ESP32-CAM Serial Monitor Output – Person Detection Notification Triggered .

### 4.7 Conclusion

The results demonstrate that our system operates effectively across all key components. The machine learning model successfully identifies the target classes—including hand signs and person detection—with high accuracy. The integration with the ESP32-CAM module was smooth and reliable throughout the testing phase.

Additionally, MQTT communication was successfully implemented, allowing real-time data transmission from the ESP32-CAM to Home Assistant. This enabled seamless automation, such as turning the LED on or off based on detected labels. The actuator module correctly responded to subscribed MQTT messages, confirming effective communication between devices.

Overall, the combination of Edge Impulse, Arduino IDE, ESP32-CAM, and MQTT integration with Home Assistant offers a robust and scalable solution for intelligent detection and automation in IoT environments.

# Conclusion

The architecture of our object detection system functions smoothly and effectively. The ESP32-CAM module successfully runs a lightweight machine learning model with good precision, accurately detecting hand signs and other target classes. Integration with Home Assistant via the MQTT protocol worked flawlessly. Upon detection, the ESP32-CAM publishes messages to the MQTT broker, which then triggers appropriate responses from the actuators. Notifications are delivered instantly, demonstrating the reliability and responsiveness of the entire system.

This project confirms that the ESP32-CAM is capable of running embedded machine learning models efficiently—provided the models are lightweight and tailored to the device’s hardware limitations.

## Future Improvements

To improve the system’s performance, scalability, and range of functionalities, the following enhancements are suggested:

### 1. Hardware Enhancements

- Upgrade to a higher-resolution camera ( 5MP or more) for better image quality.
- Use a more powerful microcontroller or separate processing unit to support more demanding models and faster inference times.

### 2. Model and Dataset Optimization

- Expand the hand sign dataset by collecting more diverse and simplified examples to improve accuracy and generalization.
- Include all possible hand signs to enable control over a wider range of smart devices.
- Add fall detection as a new feature to increase the system’s usefulness, especially for elderly care and home safety applications.

These upgrades would make the system more robust, versatile, and capable of handling more advanced tasks in real-time, ultimately making it more suitable for broader smart home automation scenarios.



# References

1. Nations, U. *DESA-EN* United Nations. Publisher: United Nations. <https://www.un.org/en/desa> (2025).
2. *That ‘Internet of Things’ Thing* SciSpace - Paper. Volume: 22. <https://scispace.com/papers/that-internet-of-things-thing-49qce0gtwy> (2025).
3. O. Aghenta, L., Tariq Iqbal, M. & Department of Electrical and Computer Engineering, Faculty of Engineering and Applied Science, Memorial University of Newfoundland (MUN), St. John’s, NL A1B 3X5, Canada. Design and implementation of a low-cost, open source IoT-based SCADA system using ESP32 with OLED, ThingsBoard and MQTT protocol. *AIMS Electronics and Electrical Engineering* **4**, 57–86. ISSN: 2578-1588. <http://www.aimspress.com/article/10.3934/ElectrEng.2020.1.57> (2025) (2020).
4. *Smart Home: Definition, How It Works, Pros and Cons* Investopedia. <https://www.investopedia.com/terms/s/smart-home.asp> (2025).
5. Assistant, H. *Documentation* Home Assistant. <https://www.home-assistant.io/docs/> (2025).
6. Shi, W., Cao, J., Zhang, Q., Li, Y. & Xu, L. Edge Computing: Vision and Challenges. *IEEE Internet of Things Journal* **3**. Conference Name: IEEE Internet of Things Journal, 637–646. ISSN: 2327-4662. <https://ieeexplore.ieee.org/abstract/document/7488250> (2025) (Oct. 2016).
7. Dokic, K. *Microcontrollers on the Edge – Is ESP32 with Camera Ready for Machine Learning?* in *Image and Signal Processing* (eds El Moataz, A., Mammass, D., Mansouri, A. & Nouboud, F.) (Springer International Publishing, Cham, 2020), 213–220. ISBN: 978-3-030-51935-3.
8. Hymel, S. *et al. Edge Impulse: An MLOps Platform for Tiny Machine Learning* Apr. 28, 2023. arXiv: [2212.03332\[cs\]](https://arxiv.org/abs/2212.03332). <http://arxiv.org/abs/2212.03332> (2025).
9. *Raspberry Pi Documentation* <https://www.raspberrypi.com/documentation/> (2025).



- 
10. *Raspberry Pi 5 B (Broadcom BCM2712) Benchmark, Test and specs* [https://www.cpu-monkey.com/en/cpu-raspberry\\_pi\\_5\\_b\\_broadcom\\_bcm2712](https://www.cpu-monkey.com/en/cpu-raspberry_pi_5_b_broadcom_bcm2712) (2025).
  11. Upton, E. *Introducing: Raspberry Pi 5!* Raspberry Pi. <https://www.raspberrypi.com/news/introducing-raspberry-pi-5/> (2025).
  12. Assistant, H. *Raspberry Pi Home Assistant*. <https://www.home-assistant.io/installation/raspberrypi/> (2025).
  13. *Getting Started - - — Arduino ESP32 latest documentation* [https://docs.espressif.com/projects/arduino-esp32/en/latest/getting\\_started.html](https://docs.espressif.com/projects/arduino-esp32/en/latest/getting_started.html) (2025).
  14. Systems, E. *ESP32 Series Datasheet* version 4.2. Accessed: 2025-04-02. 2024. [https://www.espressif.com/sites/default/files/documentation/esp32\\_datasheet\\_en.pdf](https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf).
  15. daniel. *ESP32-Cam - Complete Guide* DIY Engineers. <https://www.diyengineers.com/2023/04/13/esp32-cam-complete-guide/> (2025).
  16. *ESP32-CAM camera development board* — <https://docs.ai-thinker.com/en/esp32-cam> (2025).
  17. *MQTT Essentials: Your 2025 Learning Hub for IoT & IIoT Data Streaming* Section: website. <https://www.hivemq.com/mqtt/> (2025).