AI POWERED FIRE DETECTION SYSTEM

A PROJECT REPORT

Submitted by

BALAJI. J	**-**	**510121106004**
CHANDIRAGURU. T	**-**	**510121106006**
PRADEEP. D	**-**	**510121106020**
SIDDHARTH. M	**-**	**510121106027**

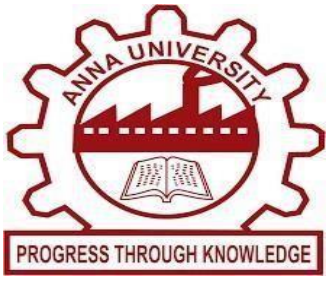
in partial fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

ELECTRONICS AND COMMUNICATION ENGINEERING



ADHIPARASAKTHI COLLEGE OF ENGINEERING

G.B.NAGAR, KALAVAI - 632 506

ANNA UNIVERSITY: CHENNAI 600025

MAY 2025

ANNA UNIVERSITY: CHENNAI 600025

BONAFIDE CERTIFICATE

Certified that this project report **"AI POWERED FIRE DETECTION SYSTEM"** is the bonafide work of

BALAJI. J	-	**510121106004**
CHANDIRAGURU. T	-	**510121106006**
PRADEEP. D	-	**510121106020**
SIDDHARTH. M	-	**510121106027**

who carried out the project work under my supervision.

SIGNATURE

Mrs. S. SHANTHI, M.E.,
Assistant Professor
SUPERVISOR
Department of Electronics and Communication Engineering,
Adhiparasakthi College of Engineering,
G.B. Nagar, Kalavai-632506

SIGNATURE

Dr. G. ASHA, M.E.,
Associate Professor
HEAD OF THE DEPARTMENT
Department to Electronics and Communication Engineering,
Adhiparasakthi College of Engineering,
G.B. Nagar, Kalavai-632506

Submitted for the project evaluation held on _____

at **Adhiparasakthi College of Engineering.**

INTERNAL EXAMINER **EXTERNAL EXAMINER**

ACKNOWLEDGEMENT

With the divine blessings of **Goddess Adhiparaskthi**, we express our deep gratitude to his **Holiness Padma Shri Arul Thiru Bangaru Adigalar**, Founder President and **Thirumathi Lakshmi Bangaru Adigalar**, President for providing an amazing environment for the development and promotion of this under graduate education in our college under ACMEC Trust.

We are very grateful to our respected correspondent **Sakthi Thirumathi Dr. B. Umadevi**, for her encouragement and inspiration. We are very grateful to **Sakthi Thiru R. Karunanidhi**, Secretary for his continuous support. We are highly indebted to our principal **Prof. Dr. S. Mohanamurugan** for his valuable guidance.

We wish to place our sincere gratitude to **Dr. G. Asha**, Head of the Electronics and Communication Engineering for his motivation and permitting us to do this work. We are especially indebted to our project coordinator and supervisor **Mrs. M. Shanthi, Assistant professor** of Electronics and Communication Engineering for her kind guidance, valuable advice and sustained in interest help extended towards us for the completion of this work successfully.

We are thankful to all teaching and non-teaching staffs of our department for their constant cooperation and encouragement in pursuing our project work. We are thankful to our parents and well-wishers for their encouragement and support in pursuing our project work.

ABSTRACT

In recent years, fire accidents have led to severe losses of life, property, and resources. Traditional fire detection systems, such as smoke and temperature sensors, often exhibit delayed response times and limited accuracy, especially in outdoor or industrial environments. To address these limitations, this project proposes an AI-powered fire detection system that integrates the ESP32-CAM module, DeepStack AI, and a mobile application developed using Thunkable.

The system captures real-time video frames via the ESP32-CAM and processes them using a DeepStack AI server equipped with pre-trained fire detection models. Upon detecting fire, the system activates a physical buzzer, and updates the fire status and sensor readings to Firebase.

The mobile application retrieves this real-time data from Firebase, including gas sensor and flame sensor values, AI model confidence, and fire status. It also simulates a virtual buzzer and LED for remote alert visualization, enabling the user to monitor fire conditions effectively from a distance.

This solution is lightweight, low-cost, and scalable, making it ideal for smart homes, warehouses, and agricultural fields. By combining embedded systems, artificial intelligence, and mobile development, the project demonstrates a practical and efficient approach to intelligent fire detection with improved accuracy and real-time awareness.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

SPI – Serial Peripheral Interface

I2C – Inter Integrated Circuit

PCB – Printed Circuit Board

PWM – Pulse Width Modulation

RTC – Real Time Clock

UART – Universal Asynchronous Receiver / Transmitter

RFID – Radio Frequency Identification

UID – Unique Identification

NFC – Near Field Communication

FTDI – Future Technology Devices International

CLI – Command Line Interface

MIDI – Musical Instrument Digital Interface

BAAS – Backend-as-a-Service

HTTP – Hypertext Transfer Protocol

GNU – GNU's Not Unix

CHAPTER I

1.1 INTRODUCTION

In today's world, the safety of lives and property against fire hazards remains a critical concern. Fire-related accidents, both in residential and industrial settings, have continued to cause enormous damage, resulting in injuries, loss of life, and destruction of valuable assets. Despite the availability of traditional fire alarm systems, such as smoke detectors and heat sensors, these technologies often suffer from delayed detection, false alarms, and limited coverage, particularly in open environments or large-scale installations.

The growing advancement in embedded systems, artificial intelligence (AI), and the Internet of Things (IoT) has opened new possibilities for designing smarter, faster, and more efficient fire detection mechanisms. These innovations enable real-time monitoring, remote access, and intelligent decision-making, reducing human dependency and improving response time in case of emergencies.

This project presents a smart fire detection system that bridges embedded hardware, AI-based image processing, and mobile app development to achieve a more reliable and accessible solution. The core of the system is built around the ESP32-CAM — a low-cost microcontroller board with an integrated camera module. It continuously streams video and reads sensor inputs from connected flame and gas (MQ135) sensors.

To enhance detection accuracy, the video stream from the ESP32-CAM is fed into a DeepStack AI server running a custom-trained fire detection model. This AI model processes each frame in real time and identifies the presence of fire based on visual cues. Once fire is detected,

the system not only logs the event but also triggers a buzzer circuit, providing immediate physical alerts on-site.

In addition to local alerts, the system integrates Firebase Realtime Database to record sensor values, fire detection status, and AI confidence scores. These values are retrieved by a custom-built mobile application developed using Thunkable. The app displays a live monitoring dashboard that includes: Flame and gas sensor readings, Fire detection status, AI confidence level, A virtual LED and buzzer simulation

This approach offers a remote and interactive way to monitor fire risks, especially useful in scenarios where physical access is limited or delayed.

Moreover, the solution is designed to be affordable, scalable, and portable, making it suitable for a wide range of applications including smart homes, greenhouses, server rooms, industrial storage areas, and farmlands.

By combining AI-driven image recognition, IoT hardware, and mobile interfaces, this project addresses several limitations of conventional fire detection methods. It introduces a hybrid system that leverages the real-time decision-making capabilities of artificial intelligence with the robustness of embedded control and user-friendly visualization.

METHODOLOGY

The proposed AI-powered fire detection system combines IoT, computer vision, and mobile technology to detect fire accurately and provide real-time alerts. The core of the system is the ESP32-CAM module, which captures live images or video frames and sends them to a locally hosted DeepStack AI server over a Wi-Fi network. The DeepStack

server uses a pre-trained fire detection model to analyze the frames and identify fire or flame patterns. When fire is detected, the server returns the result along with a confidence score. In response, the ESP32-CAM triggers a buzzer to alert people nearby. Simultaneously, the system uploads the fire status to Firebase Realtime Database. This data is instantly accessible through a Thunkable-based mobile application, which displays real-time fire status, sensor readings and model's confidence score. This smart system ensures fast, reliable fire detection, making it a cost-effective and scalable solution for homes, industries, and remote areas prone to fire hazards.

CHAPTER II

2.1BLOCK DIAGRAM OF THE PROJECT

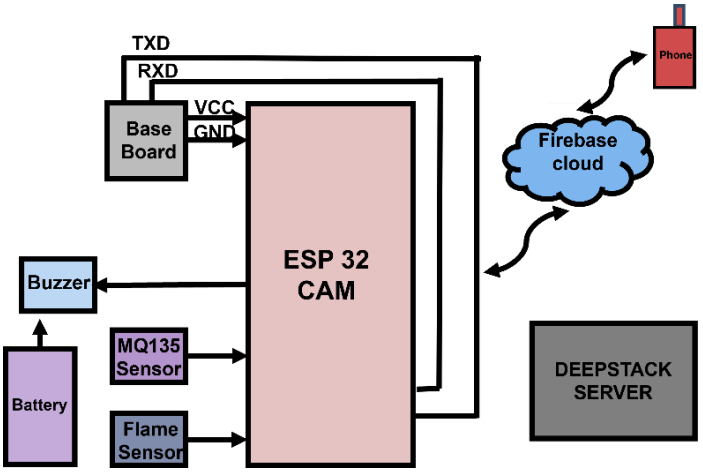


FIG 2.1 BLOCK DIAGRAM

- ESP32-CAM captures live video and sensor data, acting as the core processing unit.
- MQ135 Gas Sensor detects harmful gases or smoke and sends digital readings.
- Flame Sensor detects infrared light from fire and gives digital output.
- Buzzer provide physical alerts when fire is detected.
- Transistor and Battery control buzzer activation with external power.
- Firebase Cloud stores real-time sensor values and fire status for mobile access.
- DeepStack AI Server processes camera feed and returns fire detection confidence.
- Thunkable App displays camera feed, sensor readings, and fire alerts to the user.
- We are using the serial monitor for the display

2.2 CIRCUIT DIAGRAM OF THE PROJECT

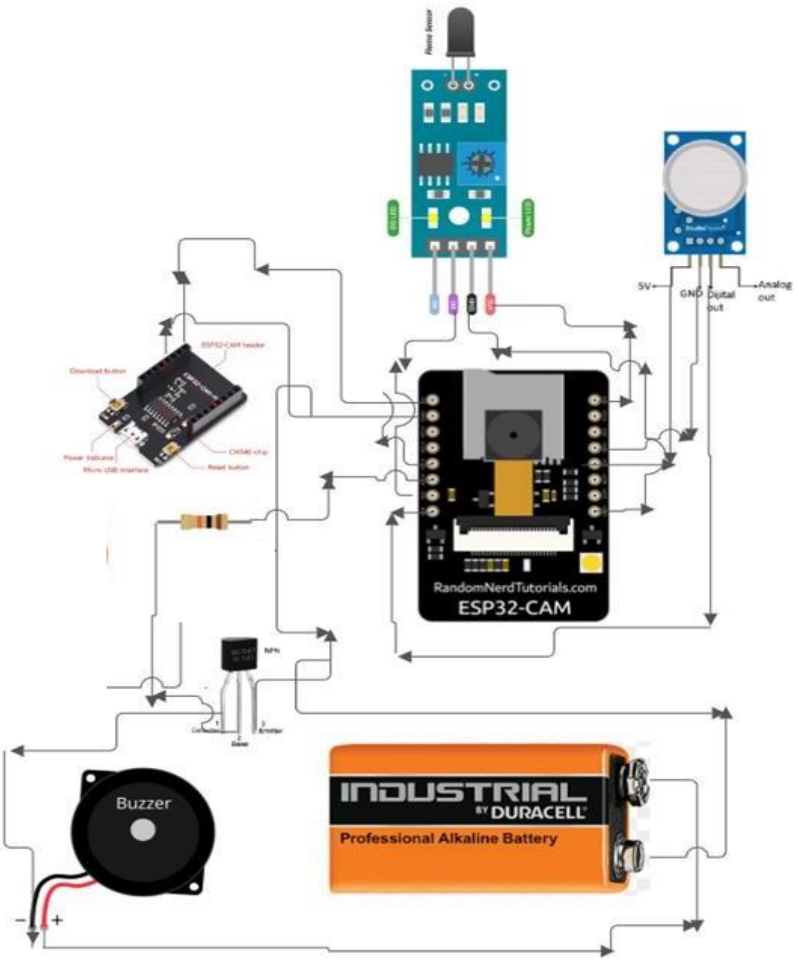


FIG 2.2 CIRCUIT DIAGRAM

CHAPTER III

3. TECHNOLOGY IMPLEMENTED

3.1 ESP32-CAM

ESP32-CAM is a low-cost, compact development board powered by the ESP32 microcontroller with integrated Wi-Fi and Bluetooth, specifically designed for camera-based IoT applications. It combines the power of the dual-core 32-bit Xtensa LX6 CPU with built-in connectivity and multimedia features. One of the key components of the ESP32-CAM is the OV2640 camera module, which allows for real-time image and video capture. This makes it suitable for AI-powered surveillance, facial recognition, fire detection, and other vision-based applications.

The **ESP32-CAM board** also includes a microSD card slot for local data storage, several GPIO pins for sensor integration, and support for UART, SPI, and I2C interfaces. It has 520KB SRAM and 4MB Flash memory, making it capable of running lightweight AI models or acting as a client to send data to cloud servers or Firebase. In our AI-powered fire detection system, the ESP32-CAM captures live images and streams them to an AI model (hosted on a local DeepStack server or cloud) to detect fire. When fire is detected, it triggers hardware responses like activating a buzzer and an LED, and sends alerts to the connected mobile app via Firebase. The ESP32-CAM's combination of processing power, connectivity, and camera capability makes it ideal for intelligent, real-time fire monitoring systems.

3.1.1SPECIFICATIONS & FEATURES:

- Microcontroller : Tensilica Xtensa® 32-bit LX6 dual-core processor
- Operating Voltage : 3.3V
- Digital I/O Pins (DIO): 10
- Analog Input Pins (ADC) : 6 (ADC2 only – not usable when Wi-Fi is active)
- UARTs : 2
- SPIs : 1
- I2Cs : 1
- PWMs : 10
- Touch Pins : 7
- RTC GPIOs : Selected pins usable for deep sleep wake-up
- Deep Sleep Current : < 10 µA
- WiFi : IEEE 802.11 b/g/n 2.4 GHz
- Bluetooth : v4.2 BR/EDR and BLE
- Camera : OV2640 2MP Camera (JPEG/BMP/Grayscale)
- SD Card : MicroSD slot (SPI interface, max 4GB)
- Flash Memory : 4MB SPI flash
- PSRAM : 4 MB external PSRAM
- SRAM : 520 KB on-chip SRAM
- Flash LED : GPIO 4

3.1.2 PIN DIAGRAM

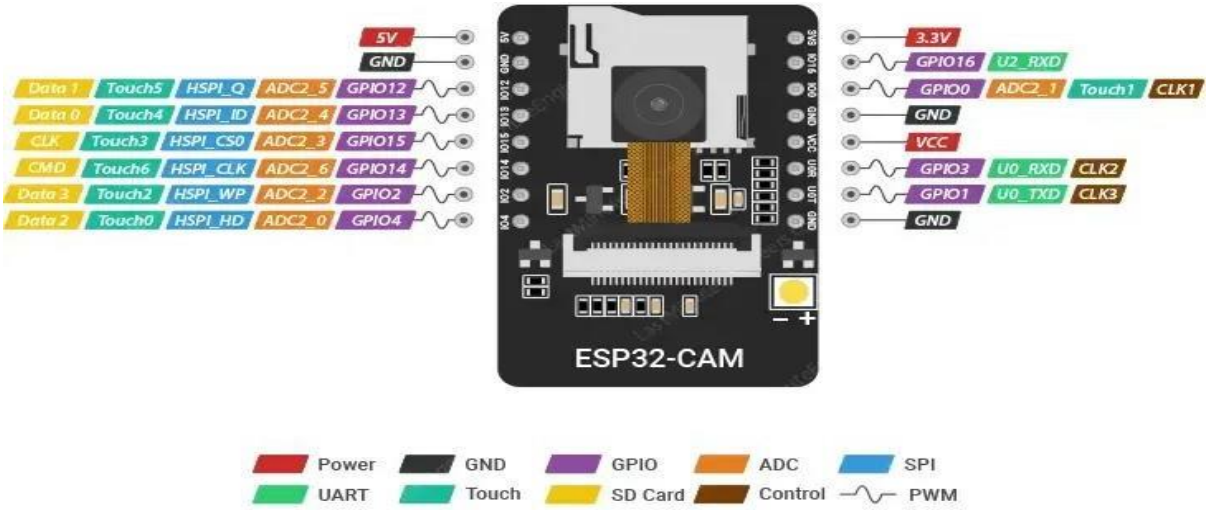


FIG 3.1 ESP32-CAM PIN DIAGRAM

3.1.3 USING ARDUINO IDE

The most basic way to use the ESP32 module is to use serial commands, as the chip is basically a WiFi/Serial transceiver. However, this is not convenient. What we recommend is using the very cool Arduino ESP32 project, which is a modified version of the Arduino IDE that you need to install on your computer. This makes it very convenient to use the ESP32 chip as we will be using the well-known Arduino IDE. Following the below step to install ESP32 library to work in Arduino IDE environment.

3.1.4 INSTALL THE ARDUINO IDE 1.6.4 OR GREATER

Download Arduino IDE from Arduino.cc (1.6.4 or greater) - don't use 1.6.2 or lower version! You can use your existing IDE if you have already installed it.

You can also try downloading the ready-to-go package from the ESP32-Arduino project, if the proxy is giving you problems.

3.1.5 INSTALL THE ESP32 BOARD PACKAGE

Enter https://raw.githubusercontent.com/espressif/arduino-esp32/gh-pages/package_esp32_index.json into Additional Board Manager URLs field in the Arduino v1.6.4+ preferences.

Click ‘File’ -> ‘Preferences’ to access this panel.

Next, use the Board manager to install the ESP32 package.

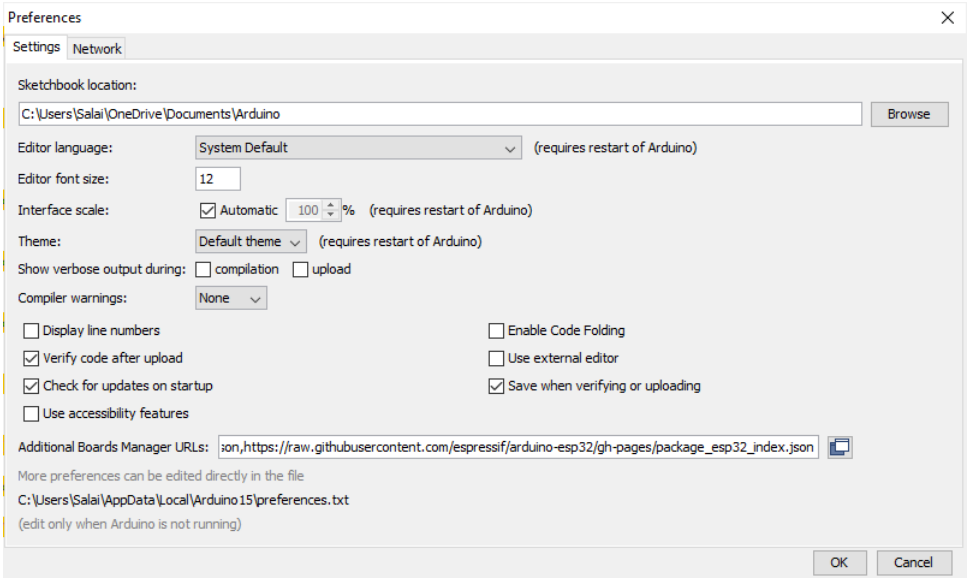


FIG 3.2 ARDUINO PREFERENCE

FIG 3.3 ARDUINO BOARD MANAGER

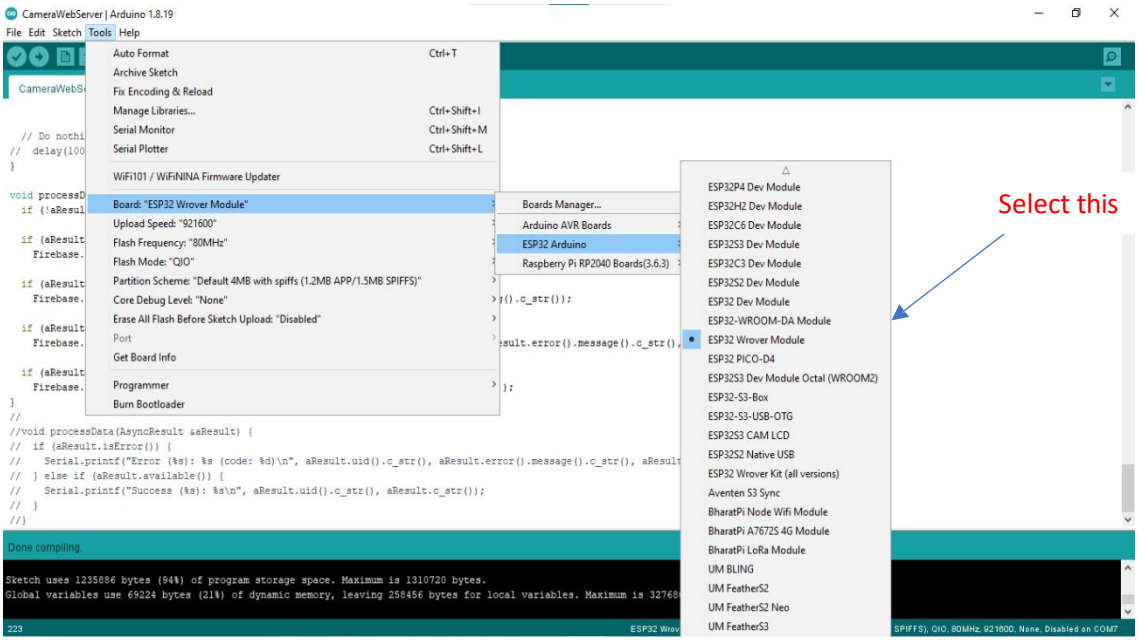
Click ‘Tools’ -> ‘Board:’ -> ‘Board Manager…’ to access this panel.

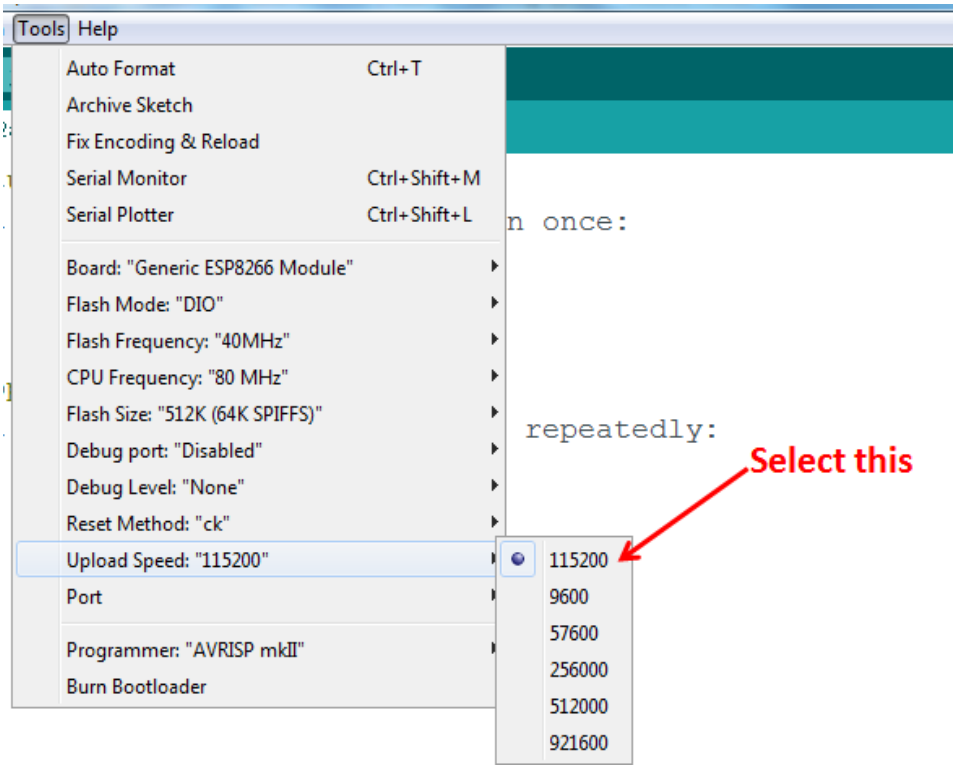
Scroll down to ‘ esp32 by Espressif Systems ’ and click “Install” button to install the ESP32 library package. Once installation completed, close and re-open Arduino IDE for ESP32 library to take effect.

3.1.6 SETUP ESP32 SUPPORT

When you’ve restarted Arduino IDE, select ‘ESP32 Wrover Module’ from the ’Tools’->’ Board’. :Dropdown Menu Select ‘115200’ baud upload speed is a good place to start - later on you can try higher speeds but 115200 is a good safe place to start.

Go to your Windows ‘Device Manager’ to find out which Com Port ‘USB-Serial CH340’ is assigned to. Select the matching COM/serial port for your CH340 USB-Serial interface.





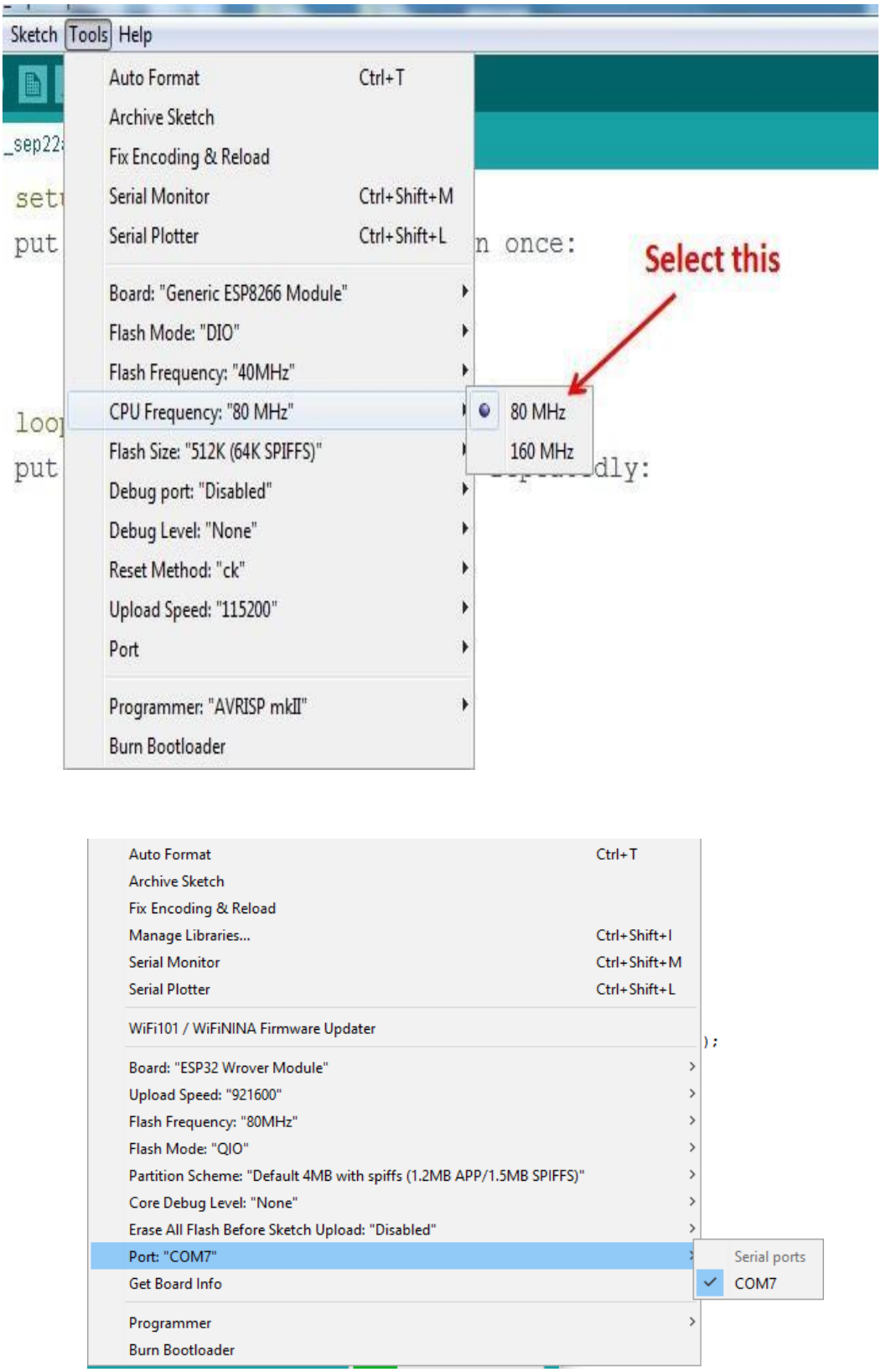


FIG 3.4 ESP32-ARDUINO SETUP

3.1.7 BLINK TEST

We'll begin with the simple blink test. Enter this into the sketch window (and save since you'll have to). Connect a LED as shown in Figure3-5.

```
 void setup()
 {
pinMode(1,OUTPUT);      // GPIO01,
 }

 void loop()
 {
 digitalWrite(1, HIGH); delay(900);
 digitalWrite(1, LOW); delay(500);
 }
```

The ESP32-CAM module generally simplifies firmware uploads; once correctly configured, it usually enters bootload mode automatically, eliminating the need for repeated manual button presses (Flash/RST) during subsequent sketch uploads, which was common with older ESP boards.

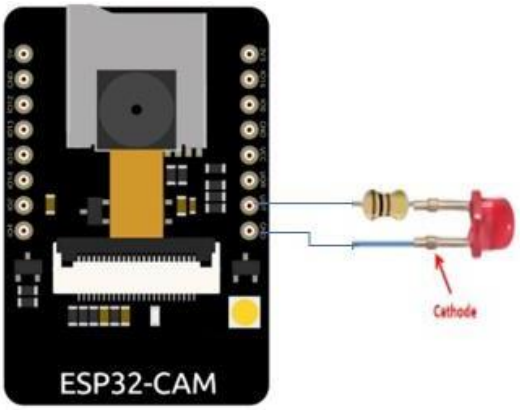


FIG 3.5 CONNECTION DIAGRAM FOR THE BLINK TEST

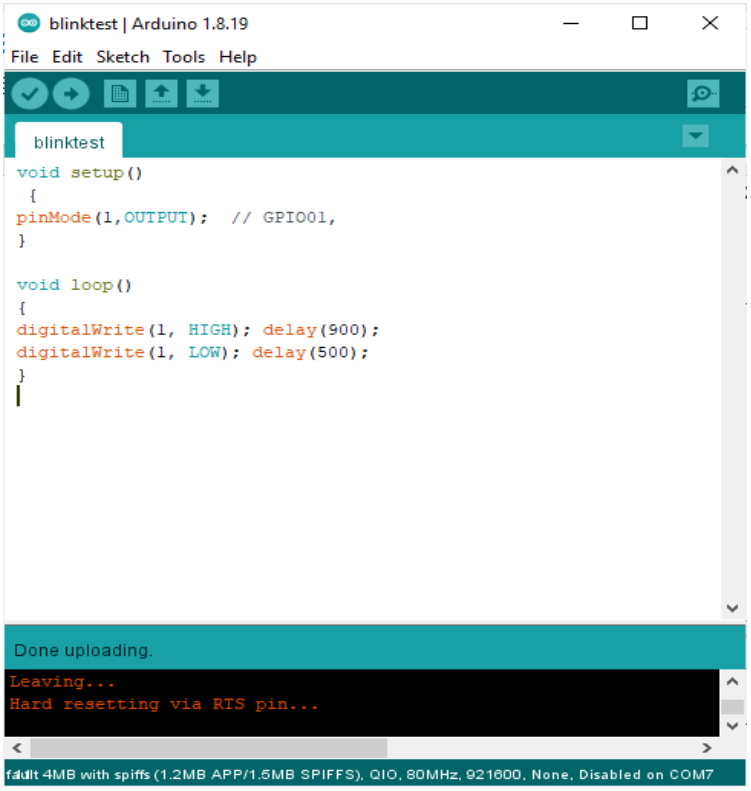


FIG 3.6 UPLOADING SKETCH

Uploading the sketch to ESP32 CAM module.

The sketch will start immediately - you'll see the LED blinking. Hooray!

3.1.8 CONNECTING VIA WIFI

OK once you've got the LED blinking, let’s go straight to the fun part, connecting to a web server. Create a new sketch with this code:

Don’t forget to update:

constchar*ssid ="yourssid";

const char* password = "yourpassword";

to your WiFi access point and password, then upload code via IDE.

3.2 INTERNET OF THINGS

3.2.1 EXPLANATION

The **Internet of things** (**IoT**) is a system of interrelated computing devices, mechanical and digital machines provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

The definition of the Internet of things has evolved due to the convergence of multiple technologies, real-time analytics, machine learning, commodity sensors, and embedded systems. Traditional fields of embedded systems, wireless sensor networks, control systems, automation (including home and building automation), and others all contribute to enabling the Internet of things. In the consumer market, IoT technology is most synonymous with products pertaining to the concept of the "smart home", covering devices and appliances (such as lighting fixtures, thermostats, home security systems and cameras, and other home appliances) that support one or more common ecosystems, and can be controlled via devices associated with that ecosystem, such as smartphones and smart speakers.

There are a number of serious concerns about dangers in the growth of IoT, especially in the areas of privacy and security, and consequently industry and governmental moves to address these concerns have begun.

ARCHITECTURE OF IOT

There is no single consensus on architecture for IoT, which is agreed universally. Different architectures have been proposed by different researchers.

THREE AND FIVE-LAYER ARCHITECTURE

The most basic architecture is a three-layer architecture. It was introduced in the early stages of research in this area. It has three layers, namely, the perception, network, and application layers. The perception layer is the physical layer, which has sensors for sensing and gathering information about the environment. It senses some physical parameters or identifies other smart objects in the environment.

The network layer is responsible for connecting to other smart things, network devices, and servers. Its features areal so used for transmitting and processing sensor data. The application layer is responsible for delivering application specific services to the user. It defines various applications in which the Internet of Things can be deployed, for example, smart homes, smart cities, and smart health.

ARCHITECTURE OF IOT (A: THREE LAYERS) (B: FIVE LAYERS)

The three-layer architecture defines the main idea of the Internet of Things, but it is not sufficient for research on IoT because research often focuses on finer aspects of the Internet of Things. That is why, we have many more layered architectures proposed in the literature. One is the five-layer architecture, which additionally includes the processing and business layers. The five layers are perception, transport, processing, application, and business layers. The role of the perception and application layers is the same as the architecture with three layers. We outline the function of the remaining three layers.

The transport layer transfers the sensor data from the perception layer to the processing layer and vice versa through networks such as

wireless, 3G, LAN, Bluetooth, RFID, and NFC. The processing layer is also known as the middleware layer. It stores, analyses, and processes huge amounts of data that comes from the transport layer. It can manage and provide a diverse set of services to the lower layers.

It employs many technologies such as databases, cloud computing, and big data processing modules. The business layer manages the whole IoT system, including applications, business and profit models, and users' privacy. The business layer is out of the scope of this paper. Hence, we do not discuss it further. Another architecture proposed by Ning and Wang is inspired by the layers of processing in, which is analogous to the distributed network of data processing nodes and smart gateways. Third is the network of nerves, which corresponds to the networking components and sensors.

CLOUD AND FOG BASED ARCHITECTURES

Let us now discuss two kinds of systems architectures: cloud and fog computing. In particular, we have been slightly vague about the nature of data generated by IoT devices, and the nature of data processing. In some system architectures the data processing is done in a large centralized fashion by cloud computers. Such a cloud centric architecture keeps the cloud at the center, applications above it, and the network of smart things below it. Cloud computing is given primacy because it provides great flexibility and scalability. It offers services such as the core infrastructure, platform, software, and storage. Developers can provide their storage tools, software tools, data mining, and machine learning tools, and visualization tools through the cloud.

Lately, there is a move towards another system architecture, namely, fog computing, where the sensors and network gateways do a part

of the data processing and analytics. A fog architecture presents a layered approach, which inserts monitoring, pre-processing, storage, and security layers between the physical and transport layers. The monitoring layer monitors power, resources, responses, and services.

The pre-processing layer performs filtering, processing, and analytics of sensor data. The temporary storage layer provides storage functionalities such as data replication, distribution, and storage. Finally, the security layer performs encryption/decryption and ensures data integrity and privacy. Monitoring and pre-processing are done on the edge of the network before sending data to the cloud.

FOG ARCHITECTURE OF A SMART IOT GATEWAY

Often the terms “fog computing” and “edge computing” are used interchangeably. The latter term predates the former and is construed to be more generic. Fog computing originally termed by Cisco refers to smart gateways and Smart sensors, where as edge computing is slightly more penetrative in nature. This paradigm envisions adding smart data pre-processing capabilities to physical devices such as motors, pumps, or lights. The aim is to do as much of pre-processing of data as possible in these devices, which are termed to be at the edge of the network. In terms of the system architecture, the architectural diagram is not appreciably different. As a result, we do not describe edge computing separately. Finally, the distinction between protocol architectures and system architectures is not very crisp. Often the protocols and the system are codesigned. We shall use the generic 5layer IoT protocol stack.

SOCIAL IOT

Let us now discuss a new paradigm: social IoT (SIoT). Here, we consider social relationships between objects the same way as humans form social relationships. Here are the three main facets of an SIoT system: The SIoT is navigable. We can start with one device and navigate through all the devices that are connected to it. It is easy to discover new devices and services using such a social network of IoT devices. A need of trustworthiness (strength of the relationship) is present between devices (similar to friends on Facebook). We can use models similar to studying human social networks to also study the social networks of IoT devices.

BASIC COMPONENTS

In a typical social IoT setting, we treat the devices and services as bots where they can set up relationships between them and modify the mover time. This will allow us to seamlessly let the devices cooperate among each other and achieve a complex task. To make such a model work, we need to have many interoperating components. Let us look at some of the major components in such a system. An ID: we need a unique method of object identification.

An ID can be assigned to an object based on traditional parameters such as the MAC ID, IPv6 ID, a universal product code, or some other custom method. Metainformation: along with an ID, we need some metainformation about the device that describes its form and operation. This is required to establish appropriate relationships with the device and also appropriately place it in the universe of IoT devices.

Security controls: this is similar to “friend list” settings on Facebook. An owner of a device might place restrictions on the kinds of devices that can connect to it. These are typically referred to as owner controls. Service discovery: such kind of a system is like a service cloud, where we need to have dedicated directories that store details of devices providing certain kinds of services. It becomes very important to keep these directories up to date such that devices can learn about other devices. Relationship management: this module manages relationships with other devices. It also stores the types of devices that a given device should try to connect with based on the type of services provided. For example, it makes sense for a light controller to make a relationship with a light sensor. Service composition: this module takes the social IoT model to a new level. The ultimate goal of having such a system is to provide better integrated services to users. For example, if a person has a power sensor with her air conditioner and this device establishes a relationship with an analytics engine, then it is possible for the ensemble to yield a lot of data about the usage patterns of the air conditioner. If the social model is more expansive, and there are many more devices, then it is possible to compare the data with the usage patterns of other users and come up with even more meaningful data. For example, users can be told that they are the largest energy consumers in their community or among their Face book friends.

REPRESENTATIVE ARCHITECTURE

Most architectures proposed for the SIoT have a server side architecture as well. The server connects to all the interconnected components, aggregates(composes) the services, and acts as a single point of service for users. The server-side architecture typically has three layers.

The first is the base layer that contains a data base that stores details of all the devices, their attributes, meta information, and their relationships. The second layer (Component layer) contains code to interact with the devices, query their status, and use a subset of them to effect a service. The topmost layer is the application layer, which provides services to the users. On the device (object) side, we broadly have two layers. The first is the object layer, which allows a device to connect to other devices, talk to them (via standardized protocols), and exchange information. The object layer passes information to the social layer. The social layer manages the execution of users' applications, executes queries, and interacts with the application layer on the server.

3.2.2 IOT CHARACTERISTICS:

1. INTELLIGENCE

IoT comes with the combination of algorithms and computation, software & hardware that makes it smart. Ambient intelligence in IoT enhances its capabilities which facilitate the things to respond in an intelligent way to a particular situation and supports them in carrying out specific tasks. In spite of all the popularity of smart technologies, intelligence in IoT is only concerned as means of interaction between devices, while user and device interaction is achieved by standard input methods and graphical user interface.

2. CONNECTIVITY

Connectivity empowers Internet of Things by bringing together everyday objects. Connectivity of these objects is pivotal because simple

object level interactions contribute towards collective intelligence in IoT network. It enables network accessibility and compatibility in the things. With this connectivity, new market opportunities for Internet of things can be created by the networking of smart things and applications.

3. DYNAMIC NATURE

The primary activity of Internet of Things is to collect data from its environment, this is achieved with the dynamic changes that take place around the devices. The state of these devices change dynamically, example sleeping and waking up, connected and/or disconnected as well as the context of devices including temperature, location and speed. In addition to the state of the device, the number of devices also changes dynamically with a person, place and time.

4. ENORMOUS SCALE

The number of devices that need to be managed and that communicate with each other will be much larger than the devices connected to the current Internet. The management of data generated from these devices and their interpretation for application purposes becomes more critical. Gartner (2015) confirms the enormous scale of IoT in the estimated report where it stated that 5.5 million new things will get connected every day and 6.4 billion connected things will be in use worldwide in 2016, which is up by 30 percent from 2015. The report also forecasts that the number of connected devices will reach 20.8 billion by 2020.

5. SENSING

IoT wouldn’t be possible without sensors which will detect or measure any changes in the environment to generate data that can report on their status or even interact with the environment. Sensing technologies provide the means to create capabilities that reflect a true awareness of the physical world and the people in it. The sensing information is simply the analogue input from the physical world, but it can provide the rich understanding of our complex world.

6. HETEROGENEITY

Heterogeneity in Internet of Things as one of the key characteristics. Devices in IoT are based on different hardware platforms and networks and can interact with other devices or service platforms through different networks. IoT architecture should support direct network connectivity between heterogeneous networks. The key design requirements for heterogeneous things and their environments in IoT are scalabilities, modularity, extensibility and interoperability.

7. SECURITY

IoT devices are naturally vulnerable to security threats. As we gain efficiencies, novel experiences, and other benefits from the IoT, it would be a mistake to forget about security concerns associated with it. There is a high level of transparency and privacy issues with IoT. It is important to secure the endpoints, the networks, and the data that is transferred across all of it means creating a security paradigm.

There are a wide variety of technologies that are associated with Internet of Things that facilitate in its successful functioning. IoT technologies possess the above-mentioned characteristics which create value and support human activities; they further enhance the capabilities of the IoT network by mutual cooperation and becoming the part of the total system

3.2.3 REASON FOR USING IOT

1. EVERYTHING WILL BE MEASURED.

IoT means that everything from household appliances, to construction equipment, to vehicles and buildings will transmit data and communicate with other objects or people.

That means everything will be able to be measured and tracked all the time. Cloud-based apps and tools will be able to analyze and translate that data into useful information. All this data can fuel better decisions and help develop better outcomes.

Big data has already made waves in nearly every industry, showing the value of information and analytics. Imagine the possibilities if nearly every object used in a day transmitted data that could then be smartly analyzed in real time.

2. METRICS WILL BE USED IN REALTIME.

IoT creates massive amounts of data that can be analyzed and used to make better decisions. That’s great -- but it’s even more exciting than that. This data can be analyzed and used in real time.

That means data is collected and instantly put to use to make improvements.

With IoT, information is turned into action at an unprecedented speed. Not only will technology respond to data and changes instantly, but it will be able to be used to predict problems and take actions to prevent them.
Constant monitoring can detect major issues and mitigate them before they happen.

3. ACTIONABLE DATA WILL BE SHARED.

All the data that IoT delivers won't exist in a vacuum -- it will be shared among co-workers, stakeholders and other parties. For example, think about how wearable tech allows individuals to collect health data and share it with doctors and providers, to improve care.
When this type of technology is applied in other industries, the impact will be huge.

CHAPTER IV

4. HARDWARE DESCRIPTION

This project consists of ESP32-CAM, ESP32-CAM-MB Base Board, Flame Sensor, MQ135 (smoke) Sensor, Battery, Resistor, Buzzer, Breadboard, and Jumper Wires (Male to Male).

4.1 ESP32 CAM-MB (BASE BOARD)

4.1.1 DESCRIPTION

The ESP32-CAM-MB is a specially designed base board (also called a programmer shield) for the ESP32-CAM module. It features a built-in CH340G USB to Serial converter chip, which allows direct programming of the ESP32-CAM via a USB cable without needing an external FTDI module.

4.1.2 DIAGRAM

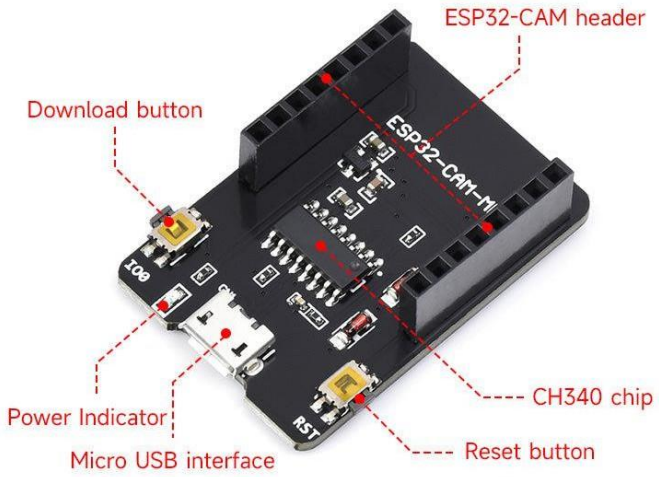


FIG 4.1 ESP32 CAM BASEBOARD

4.1.3 SPECIFICATIONS & FEATURES

- Working voltage: +5V
- Model: ESP32-CAM-MB
- USB to Serial Chip: CH340G
- Micro USB Port: Simplifies connection to PC for programming and power supply.
- RESET Button: Handy for restarting the ESP32-CAM or entering flashing mode.
- Plug-and-Play Interface: The ESP32-CAM board slots directly into the MB module for stable and compact development.

4.1.4 WORKING

In this project, the **ESP32-CAM-MB** module is used to:

- Upload the program code to the ESP32-CAM through a USB connection.
- Provide a stable power supply (via USB 5V).
- Easily reset the ESP32-CAM or switch it to programming mode using the onboard reset button.
- Maintain a clean and compact setup by securely holding the ESP32-CAM module.

4.2 MQ135 (SMOKE) SENSOR

4.2.1 DESCRIPTION

An MQ135 air quality sensor is one type of MQ gas sensor used to detect, measure, and monitor a wide range of gases present in air like ammonia, alcohol, benzene, smoke, carbon dioxide, etc. It operates at a 5V

supply with 150mA consumption. Preheating of 20 seconds is required before the operation, to obtain the accurate output.

It is a semiconductor air quality check sensor suitable for monitoring applications of air quality. It is highly sensitive to NH3, NOx, CO2, benzene, smoke, and other dangerous gases in the atmosphere. It is available at a low cost for harmful gas detection and monitoring applications.

If the concentration of gases exceeds the threshold limit in the air, then the digital output pin goes high. The threshold value can be varied by using the potentiometer of the sensor. The analog output voltage is obtained from the analog pin of the sensor, which gives the approximate value of the gas level present in the air.

4.2.2 PIN DIAGRAM

FIG 4.2 MQ135 (SMOKE) SENSOR

4.2.3 PIN IDENTIFICATION AND CONFIGURATION

Pin Number	Pin Name	Description
1	VCC	The VCC pin powers the sensor, typically with +5V
2	GND	This pin is connected to the Ground of the system

3	A0	This is an analog output pin.
4	D0	This is an digital output pin.

TAB 4.1 MQ135 PIN CONFIGURATION

4.2.4 SPECIFICATIONS & FEATURES

- Working voltage: +5V
- Measures and detects NH3, alcohol, NOx, Benzene, CO2, smoke etc.
- Range of analog output voltage: 0V-5V
- Range of digital output voltage: 0V-5V (TTL logic).
- Potentiometer adjust the sensitivity.
- High sensitivity and faster response.
- Small board PCB size: 3.2cm x 1.4cm

4.2.5 WORKING

To measure or detect the gases, use analog pins or digital pins. Just apply 5V to the module and you can observe that the module's power LED turns ON (glows) and the output LED turns OFF when no gas is detected by the module. This means that the output of the digital pin is 0V. Note that the sensor must be kept for preheating time for 20seconds (as mentioned in the specifications) before the actual operation.

Now, once when the MQ135 sensor is operated to detect, then the LED output goes high along with the digital output pin. Otherwise, use the potentiometer until the output increases. Whenever the sensor detects a certain gas concentration, the digital pin goes high (5V), otherwise it stays low (0V).

We can also use analog pins to get the same result. The output analog values (0-5V) are read from the microcontroller. This value is directly

proportional to the gas concentration detected by the sensor. By the experimental values, we can observe the working and reaction of the MQ135 sensor with different gas concentrations and the programming developed accordingly.

4.2.6 APPLICATIONS

- Used as a domestic air pollution detector.
- Used as air quality monitors.
- Used in the detection of excess or leakage of gases like nitrogen oxide, ammonia, alcohol, aromatic compounds, smoke, and sulfide.

4.3 FLAME SENSOR

4.3.1 DESCRIPTION

A flame-sensor is one kind of detector which is mainly designed for detecting as well as responding to the occurrence of a fire or flame. The flame detection response can depend on its fitting. It includes an alarm system, a natural gas line, propane & a fire suppression system. This sensor is used in industrial boilers. The main function of this is to give authentication whether the boiler is properly working or not. The response of these sensors is faster as well as more accurate compare with a heat/smoke detector because of its mechanism while detecting the flame.

4.3.2 PIN DIAGRAM

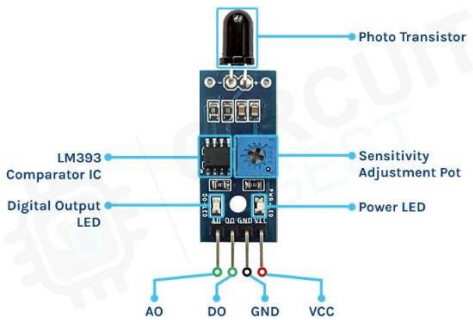


FIG 4.3 FLAME SENSOR PIN DIAGRAM

4.3.3 PIN IDENTIFICATION & CONFIGURATION

Pin Number	Pin Name	Description
1	VCC	The VCC pin powers the sensor, typically with +5V
2	GND	This pin is connected to the Ground of the system
3	A0	This is an analog output pin.
4	D0	This is an digital output pin.

TAB 4.2 FLAMSE SENSOR PIN CONFIGURATION

4.3.4 SPECIFICATIONS & FEATURES

- Operating voltage: +5V
- Detection angle is 600
- Photosensitivity is high
- Response time is fast
- Accuracy can be adjustable
- Power indicator & digital switch o/p indicator
- The PCB size is 3cm X 1.6cm

4.3.5 WORKING

This sensor/detector can be built with an electronic circuit using a receiver like electromagnetic radiation. This sensor uses the infrared flame flash method, which allows the sensor to work through a coating of oil, dust, water vapor, otherwise ice.

4.3.6 APPLICATIONS

- Hydrogen stations
- Industrial heating
- Fire detection
- Fire alarm
- Fire fighting robot

4.4 BUZZER

4.4.1 DESCRIPTION

An audio signaling device like a beeper or buzzer may be electromechanical or piezoelectric or mechanical type. The main function of this is to convert the signal from audio to sound. Generally, it is powered through DC voltage and used in timers, alarm devices, printers, alarms, computers, etc. Based on the various designs, it can generate different sounds like alarm, music, bell & siren.

4.4.2 PIN DIAGRAM

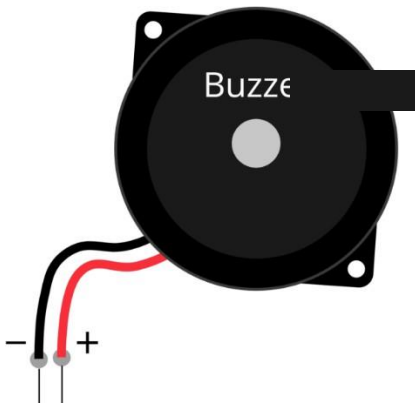


FIG 4.4 BUZZER

4.4.3 PIN IDENTIFICATION & CONFIGURATION

Pin Name	Description
+	The VCC pin powers the sensor, typically with +6V - +12V
-	This pin is connected to the Ground of the system

TAB 4.3 BUZZER PIN CONFIGURATION

4.4.4 SPECIFICATIONS & FEATURES

- Color is black
- The frequency range is 3,300Hz
- Operating Temperature ranges from – 20° C to +60°C
- Operating voltage ranges from 3V to 24V DC
- The sound pressure level is 85dBA or 10cm
- The supply current is below 15mA

4.4.5 WORKING

The working principle of a buzzer depends on the theory that, once the voltage is given across a piezoelectric material, then a pressure difference is produced. A piezo type includes piezo crystals among two conductors.

Once a potential disparity is given across these crystals, then they thrust one conductor & drag the additional conductor through their internal property. So this continuous action will produce a sharp sound signal.

4.4.6 APPLICATIONS

- Alarm Circuits

- Portable Devices
- Security Systems
- Timers

4.5 BC547 TRANSISTOR

4.5.1 DESCRIPTION

The BC547 transistor is an NPN transistor. A transistor is nothing but the transfer of resistance which is used for amplifying the current. A small current of the base terminal of this transistor will control the large current of emitter and base terminals. The main function of this transistor is to amplify as well as switching purposes. The maximum gain current of this transistor is 800A.

4.5.2 PIN DIAGRAM

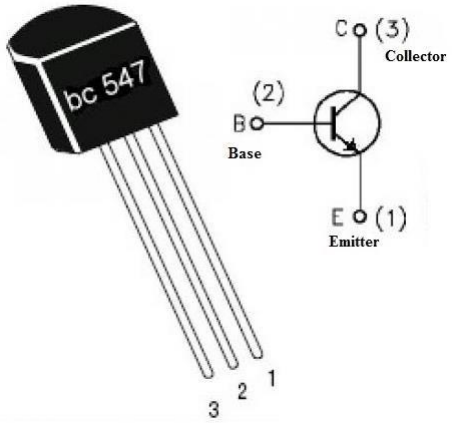


FIG 4.5 BC547 TRANSISTOR

4.5.3 PIN IDENTIFICATION & CONFIGURATION

Pin Name	Description
1 – COLLECTOR	This pin is denoted with symbol ‘C’ and the flow of current will be through the collector terminal.
2 – BASE	This pin controls the transistor biasing.
3 – EMITTER	The current supplies out through emitter terminal

TAB 4.4 BC547 TRANSISTOR PIN CONFIGURATION

4.5.4 SPECIFICATIONS & FEATURES

- The gain of DC current (hFE) = 800 A
- Continuous Ic (collector current) = 100mA
- VBE (emitter-base voltage) = 6V
- IB (base current) = 5mA
- The polarity of the transistor is NPN
- The transition frequency is 300MHz
- It is obtainable in semiconductor package like-92
- Power dissipation is 625mW

4.5.5 WORKING

The working states of BC547 transistor include the following.

- Forward Bias.
- Reverse Bias.

In a forward bias mode, the two terminals like emitter & collector are connected to allow the flow of current through it. Whereas in a reverse bias

mode, it doesn’t allow the flow of current through it because it works as an open switch.

4.5.6 APPLICATIONS

- Amplification of current
- Audio Amplifiers
- Switching Loads < 100mA
- Transistor Darlington Pairs
- Drivers like an LED driver, Relay Driver, etc.
- Amplifiers like Audio, signal, etc..

4.6 BATTERY

4.6.1 DESCRIPTION

The battery is a compact, portable DC power source commonly used in electronic circuits and embedded systems. It provides a stable DC voltage output and comes with two wires: Red (positive) and Black (negative), which can be easily connected to breadboards or soldered circuits.

4.6.2 PIN DIAGRAM

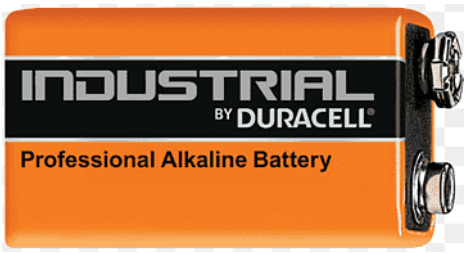


FIG 4.6 BATTERY

4.6.3 PIN IDENTIFICATION & CONFIGURATION

Pin Name	Description
+	This pin powers the component
-	This pin is connected to the Ground of the system

TAB 4.5 BATTERY PIN CONFIGURATION

4.6.4 APPLICATIONS

- Power supply for embedded systems and IoT projects
- Used in toys, alarm systems, and sensor circuits
- Drives external actuators like buzzers, motors, and high-power LEDs

4.7 RESISTOR

4.7.1 DESCRIPTION

The definition of the resistor is, it is a basic two-terminal electrical and electronic component used to restrict the current flow in a circuit. The resistance toward the flow of current will result in the voltage drop. These devices may provide a permanent, adjustable resistance value. The value of resistors can be expressed in Ohms.

4.7.2 DIAGRAM

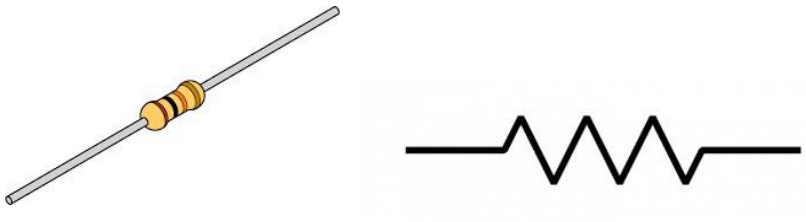


FIG 4.7 RESISTOR

4.7.3 APPLICATIONS

- DC Power Supplies
- Filter Circuit Networks
- Oscillators
- Voltage Regulators

4.8 BREAD BOARD & JUMPER WIRES

4.8.1 DESCRIPTION

In electronics, prototyping is essential, but traditional soldering methods are not economical, reusable, or component-safe. To overcome these limitations, the solderless breadboard was developed in the 1970s, offering an efficient, reusable platform for circuit design without soldering. Earlier, in the 1960s, techniques like wire-wrap and large wooden boards (similar to bread slicing boards) were used for circuit prototyping, which led to the term "breadboard." Modern solderless breadboards consist of interconnected holes for placing component leads. The top and bottom two rows are typically used for power rails - one row for positive and another for negative supply - internally connected horizontally for easy power distribution.

Jumper wires are insulated wires with connector pins at both ends. These wires carry electrical signals or power between modules, sensors, and microcontrollers.

4.8.2 DIAGRAM

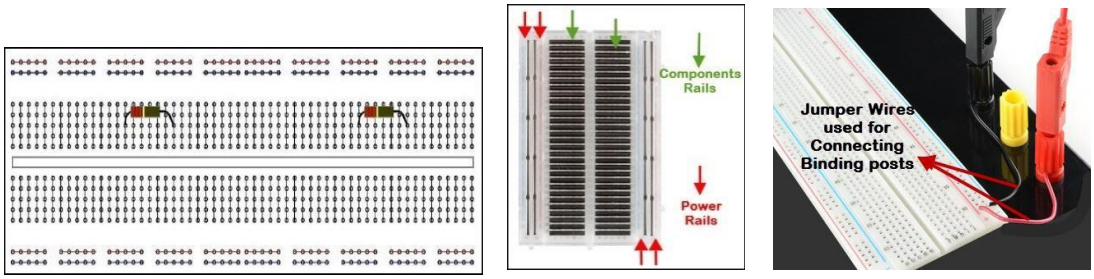


FIG 4.8 BREAD BOARD & JUMPER WIRES

4.8.3 APPLICATIONS

- Rapid prototyping
- Temporary circuit assembly

CHAPTER V

5. SOFTWARE DESCRIPTION

This project consists of software components include Arduino IDE, Firebase Console, Thunkable, DeepStack AI Server, Python Script, VS Code.

5.1 ARDUINO IDE

The Arduino integrated development environment (IDE) is a cross-platform application (for Microsoft Windows, macOS, and Linux) that is written in the Java programming language. It originated from the IDE for the languages Processing and Wiring. It includes a code editor with features such as text cutting and pasting, searching and replacing text, automatic indenting, brace matching, and syntax highlighting, and provides simple one- click mechanisms to compile and upload programs to an Arduino board. It also contains a message area, a text console, a toolbar with buttons for common functions and a hierarchy of operation menus. The source code for the IDE is released under the GNU General Public License, version 2.

The Arduino IDE supports the languages C and C++ using special rules of code structuring. The Arduino IDE supplies a software library from the Wiring project, which provides many common input and output procedures. User-written code only requires two basic functions, for starting the sketch and the main program loop, that are compiled and linked with a program stub main() into an executable cyclic executive program with the GNU toolchain, also included with the IDE distribution. The Wiring platform consisted of a printed circuit board (PCB) with an ATmega128 microcontroller, an IDE based on Processing. In 2005,

Massimo Banzi, with David Mellis, another IDII student, and David Cuartielles, extended Wiring by adding support for the cheaper ATmega8 microcontroller. The new project, forked from Wiring, was called Arduino.

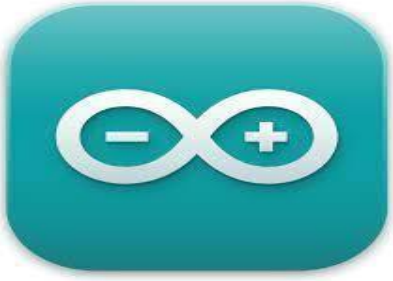


FIG 5.1 ARDUINO IDE

5.1.1 SKETCH

A sketch is a program written with the Arduino IDE. Sketches are saved on the development computer as text files with the file extension .ino. Arduino Software (IDE) pre-1.0 saved sketches with the extension .pde.

A minimal Arduino C/C++ program consists of only two functions:

- **setup( ):** This function is called once when a sketch starts after power-up or reset. It is used to initialize variables, input and output pin modes, and other libraries needed in the sketch. It is analogous to function main( ).
- **loop( ):** After setup( ) function exits (ends), the loop( ) function is executed repeatedly in the main program. It controls the board until the board is powered off or is reset. It is analogous to the function while(1).

FIG 5.2 ARDUINO IDE 1.8.19 WORKSPACE

5.1.2 FEATURES OF ARDUINO IDE

On October 18, 2019, Arduino Pro IDE (alpha preview) was released. Later, on March 1, 2021, the beta preview was released, renamed IDE 2.0. The system still uses Arduino CLI (Command Line Interface), but improvements include a more professional development environment, autocompletion support, and Git integration. The application frontend is based on the Eclipse Theia Open Source IDE. The main features available in the new release are:

- Modern, fully featured development environment
- Dual Mode, Classic Mode (identical to the Classic Arduino IDE) and Pro Mode (File System view)
- New Board Manager
- New Library Manager
- Board List
- Basic Auto-Completion (Arm targets only)

- Git Integration
- Serial Monitor
- Dark Mode

5.1.3 APPLICATIONS OF IDE

- Arduboy, a handheld game console based on Arduino
- Arduinome, a MIDI controller device that mimics the Monome
- Ardupilot, drone software and hardware
- ArduSat, a cubesat based on Arduino.
- C-STEM Studio, a platform for hands-on integrated learning of computing, science, technology, engineering, and mathematics (C- STEM) with robotics.
- Data loggers for scientific research

5.2 FIREBASE CONSOLE

Firebase is a cloud-based Backend-as-a-Service (BaaS) platform developed by Google. It provides a comprehensive suite of tools and services that help developers build high-quality web and mobile applications. Key features include real-time databases, user authentication, cloud storage, analytics, and serverless computing via Firebase Functions.

Firebase is categorized as a NoSQL cloud database, storing data in a flexible, JSON-like format. The Firebase Realtime Database allows data to be synchronized across all connected clients in real time. It supports offline usage, ensuring data integrity even when devices temporarily lose internet connectivity.

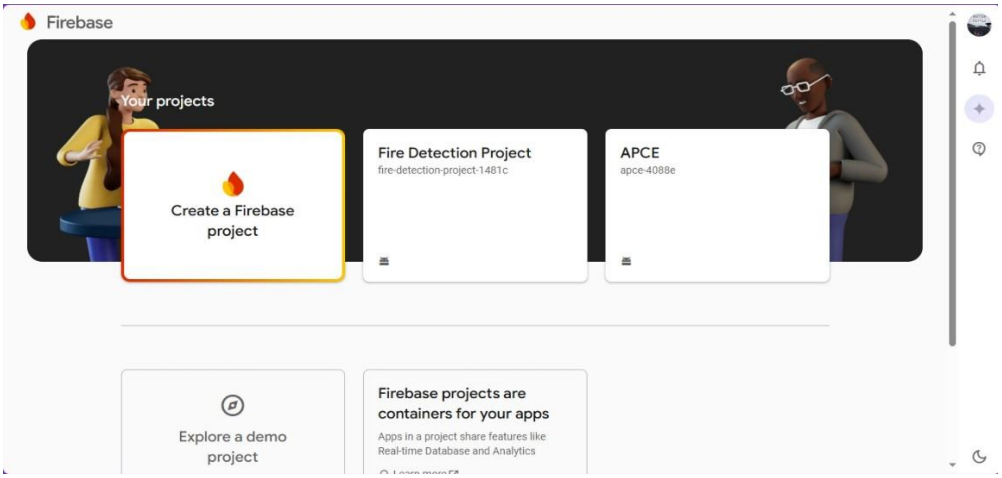


FIG 5.3 FIREBASE CONSOLE

5.2.1 APPLICATIONS IN IOT

Firebase plays a significant role in Internet of Things (IoT) applications:

- It acts as a cloud backend for connected devices to upload sensor data.
- It enables real-time communication between devices and user interfaces such as mobile apps or dashboards.
- Firebase Authentication and Security Rules ensure secure access to data.
- Integration with platforms like Flutter, Python, or embedded systems allows smooth development of full-stack IoT solutions.

Due to its scalability, ease of integration, and real-time capabilities, Firebase is widely used in IoT systems for data monitoring, device control, alerting, and cloud storage.

5.3 THUNKABLE

Thunkable is a modern, intuitive drag-and-drop platform that enables users to build cross-platform mobile applications without writing traditional code. With a visual blocks-based interface, Thunkable simplifies the app development process for beginners, hobbyists, educators, and professionals alike. It supports real-time testing on both Android and iOS devices, allowing creators to rapidly prototype and deploy applications with native functionality.

Thunkable empowers anyone to build functional apps with rich user interfaces, Firebase integration, device sensors, and advanced logic – all within a unified interface. The platform is especially useful in educational settings and rapid prototyping environments, and it aligns with the global push toward democratizing software development.

With over a million active users globally, Thunkable has become a go-to platform for students, startups, and makers who want to transform ideas into real-world applications. Its ability to connect with external APIs, integrate AI services, and deploy apps on multiple platforms with a single build makes it a powerful choice for modern app development.

FIG 5.4 THUNKABLE

5.3.1 HOW TO DEVELOP APP USING THUNKABLE?

Here is a step-by-step guide to get started with Thunkable for building Android and iOS applications:

- Open a web browser like Chrome, Firefox, or Safari.
- Go to the official website: https://thunkable.com
- Click on the “Get Started” button.
- Sign up using your Google account or other supported login methods.
- Once logged in, you'll be redirected to the Thunkable dashboard.
- Click on “Create New App” and give your project a name.
- Use the drag-and-drop components from the left panel to design your app interface.
- Switch to the Blocks tab to add logic using visual code blocks.
- You can test your app in real-time using the Thunkable Live app on your phone.
- When finished, click “Download” or “Publish” to get the app for Android or iOS.

We are using a single screen in Thunkable. This screen displays the live ESP32-CAM video feed, real-time sensor values (flame and MQ135), AI model confidence, and fire status. It also changes background color based on fire detection and shows virtual LED and buzzer alerts.

5.4 DEEPSTACK AI SERVER

DeepStack is an AI server that empowers every developer in the world to easily build state-of-the-art AI systems both on premise and in the cloud. The promises of Artificial Intelligence are huge but becoming a machine learning engineer is hard. DeepStack is device and language agnostic. You can run it on Windows, Mac OS, Linux, Raspberry PI and use it with any programming language. DeepStack is developed and maintained by DeepQuest AI .

FIG 5.5 DEEPSTACK AI SERVER

5.4.1 RUN DEEPSTACK

Run the command below as it applies to the version you have installed

deepstack --MODELSTORE-DETECTION "C:/path-to-custom-models-folder" --PORT 80

Basic Parameters

-v /path-to/custom-models folder:/modelstore/detection

This specifies the local directory where you stored your custom models

-p 80:5000 This makes DeepStack accessible via port 80 of the machine.

5.4.2 DEEPSTACK FIRENET MODEL

The DeepStack_FireNET model is a custom-trained YOLOv5x object detection model designed specifically to detect fire in indoor, outdoor, and aerial imagery. It uses the **FireNET dataset** with YOLO annotations and has been trained for **60 epochs**, achieving high accuracy:

- mAP@0.50: 0.993
- mAP@0.95: 0.829

This model integrates seamlessly with the **DeepStack Object Detection API**, allowing real-time fire detection from images and videos via a simple HTTP POST request. Once the server is set up with the model, fire detection can be performed programmatically, with bounding boxes and confidence values returned for detected fire regions.

5.5 PYTHON SCRIPT

Python scripts play a crucial role in integrating AI models into practical applications. When used with DeepStack, Python enables communication with the DeepStack AI server by sending image data via REST API requests and processing the server’s detection results.

Typically, a Python script performs the following key tasks:

- Captures or loads input images or video frames
- **Sends POST requests** to the DeepStack server's object detection endpoint
- **Receives detection results** such as confidence score.

- **Performs actions** such as uploading data to a firebase database

The script often utilizes modules such as:

- requests for HTTP communication
- json for parsing responses
- opencv-python (cv2) for image handling and visualization
- os and time for system-level and timing operations

Python's simplicity and powerful libraries make it ideal for automating tasks, processing images, and deploying AI models efficiently in real-world AI and IoT applications.

5.6 VS CODE

Visual Studio Code (VS Code) is a lightweight but powerful source-code editor developed by Microsoft. It is widely used by developers for writing, editing, debugging, and executing code in a wide range of programming languages and technologies. VS Code is a free, open-source, and cross-platform tool available for Windows, macOS, and Linux.

VS Code provides support for a wide variety of programming languages such as Python, JavaScript, Dart, Java, C++, and many others through built-in features and an extensive library of extensions available via the VS Code Marketplace.

It includes powerful developer tools such as:

- Integrated Terminal: Run commands and scripts directly from the editor.

- IntelliSense: Offers smart code completion, syntax highlighting, and real-time error detection.
- Debugging Tools: Built-in debugger to set breakpoints and inspect variables for supported languages.
- Version Control Integration: Git and GitHub integration for source control management.

In modern software development, VS Code plays a central role as a unified development environment. It is especially useful when working on full-stack projects, scripting in Python, or developing cross-platform apps using frameworks like Flutter. Its rich ecosystem of extensions and its ability to customize the development workflow make it a preferred choice among individual developers and teams alike.

FIG 5.6 VS CODE

CHAPTER VI
6. CODING

6.1 ESP32 CAM CODING

```
#include "esp_camera.h"

#include <WiFi.h>

#include <Arduino.h>

#include <WiFiClientSecure.h>

#include <FirebaseClient.h>

#define CAMERA_MODEL_AI_THINKER // Has PSRAM

#include "camera_pins.h"

// ===========================

// Enter your WiFi credentials

// ===========================

#define WIFI_SSID "Chandru"

#define WIFI_PASSWORD "12345678"

#define Web_API_KEY
"AIzaSyDws_fJwn7DUBjMZ1GXYCBxSNBrtn_PNxw"

#define DATABASE_URL "https://fire-detection-project-1481c-default-
rtdb.firebaseio.com/"

#define USER_EMAIL "Salaichandru007@gmail.com"

#define USER_PASS "FireDetection"
```

```
void startCameraServer();

void setupLedFlash(int pin);

const int flameSensorPin = 14;

const int mq135SensorPin = 13;

const int buzzerPin= 12;

static bool alertSent = false;

// Firebase setup

void processData(AsyncResult &aResult);

UserAuth user_auth(Web_API_KEY, USER_EMAIL, USER_PASS);

FirebaseApp app;

WiFiClientSecure ssl_client;

using AsyncClient = AsyncClientClass;

AsyncClient aClient(ssl_client);

RealtimeDatabase Database;

// Timing

unsigned long lastSendTime = 0;

const unsigned long sendInterval = 10000; // 10 seconds

// Data variables

int flameSensorValue = 1;
```

```
int mq135SensorValue = 1;
String fireStatus = "No Fire";
void setup() {
Serial.begin(115200);
  pinMode(flameSensorPin,INPUT);
  pinMode(mq135SensorPin,INPUT);
  pinMode(buzzerPin,OUTPUT);
  Serial.setDebugOutput(true);
  Serial.println();
  // Start with buzzer off
  digitalWrite(buzzerPin, LOW);

  camera_config_t config;
  config.ledc_channel = LEDC_CHANNEL_0;
  config.ledc_timer = LEDC_TIMER_0;
  config.pin_d0 = Y2_GPIO_NUM;
  config.pin_d1 = Y3_GPIO_NUM;
  config.pin_d2 = Y4_GPIO_NUM;
  config.pin_d3 = Y5_GPIO_NUM;
  config.pin_d4 = Y6_GPIO_NUM;
```

```
  config.pin_d5 = Y7_GPIO_NUM;
  config.pin_d6 = Y8_GPIO_NUM;
  config.pin_d7 = Y9_GPIO_NUM;
  config.pin_xclk = XCLK_GPIO_NUM;
  config.pin_pclk = PCLK_GPIO_NUM;
  config.pin_vsync = VSYNC_GPIO_NUM;
  config.pin_href = HREF_GPIO_NUM;
  config.pin_sccb_sda = SIOD_GPIO_NUM;
  config.pin_sccb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 20000000;
  config.frame_size = FRAMESIZE_UXGA;
  config.pixel_format = PIXFORMAT_JPEG;  // for streaming
  //config.pixel_format = PIXFORMAT_RGB565; // for face detection/recognition
  config.grab_mode = CAMERA_GRAB_WHEN_EMPTY;
  config.fb_location = CAMERA_FB_IN_PSRAM;
  config.jpeg_quality = 12;
  config.fb_count = 1;
```

```
  // if PSRAM IC present, init with UXGA resolution and higher JPEG quality for larger pre-allocated frame buffer.
  if (config.pixel_format == PIXFORMAT_JPEG) {
    if (psramFound()) {
      config.jpeg_quality = 10;
      config.fb_count = 2;
      config.grab_mode = CAMERA_GRAB_LATEST;
    } else {
      // Limit the frame size when PSRAM is not available
      config.frame_size = FRAMESIZE_SVGA;
      config.fb_location = CAMERA_FB_IN_DRAM;
    }
  } else {
    // Best option for face detection/recognition
    config.frame_size = FRAMESIZE_240X240;
#if CONFIG_IDF_TARGET_ESP32S3
    config.fb_count = 2;
#endif
  }


#if defined(CAMERA_MODEL_ESP_EYE)
```

```
  pinMode(13, INPUT_PULLUP);
  pinMode(14, INPUT_PULLUP);
#endif
  // camera init
  esp_err_t err = esp_camera_init(&config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x", err);
    return;
  }
  sensor_t *s = esp_camera_sensor_get();
  // initial sensors are flipped vertically and colors are a bit saturated
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);        // flip it back
    s->set_brightness(s, 1);   // up the brightness just a bit
    s->set_saturation(s, -2);  // lower the saturation
  }
  // drop down frame size for higher initial frame rate
  if (config.pixel_format == PIXFORMAT_JPEG) {
  s->set_framesize(s, FRAMESIZE_QVGA);
  }
```

```
#if defined(CAMERA_MODEL_M5STACK_WIDE) ||
defined(CAMERA_MODEL_M5STACK_ESP32CAM)
  s->set_vflip(s, 1);
  s->set_hmirror(s, 1);
#endif
#if defined(CAMERA_MODEL_ESP32S3_EYE)
  s->set_vflip(s, 1);
#endif
// Setup LED FLash if LED pin is defined in camera_pins.h
#if defined(LED_GPIO_NUM)
setupLedFlash(LED_GPIO_NUM);
#endif
  WiFi.begin(WIFI_SSID, WIFI_PASSWORD);
  WiFi.setSleep(false);
  Serial.print("WiFi connecting");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
```

```
  startCameraServer();


  Serial.print("Camera Ready! Use 'http://");

  Serial.print(WiFi.localIP());

  Serial.println("' to connect");


    // Firebase

  ssl_client.setInsecure();

  ssl_client.setConnectionTimeout(1000);

  ssl_client.setHandshakeTimeout(5);

  initializeApp(aClient, app, getAuth(user_auth), processData, "
authTask");

  app.getApp<RealtimeDatabase>(Database);

  Database.url(DATABASE_URL);

}

void loop() {

  app.loop();


  if (app.ready()) {

    unsigned long currentTime = millis();
```

```
  if (currentTime - lastSendTime >= sendInterval) {

    lastSendTime = currentTime;



    // Read sensor

    flameSensorValue = digitalRead(flameSensorPin);  // 0 = flame
detected, 1 = no flame

    mq135SensorValue = digitalRead(mq135SensorPin);  // 0 = smoke
detected, 1 = no smoke



     // --- Fire Detection Logic ---

    if (flameSensorValue == 0 || mq135SensorValue == 0) {

      fireStatus = "Fire Detected";

      if(!alertSent){

        for(int i=0;i<5;i++){

          digitalWrite(buzzerPin, HIGH); // Activate buzzer

          delay(500);

          digitalWrite(buzzerPin, LOW); // Deactivate buzzer

          delay(500);

        }

        alertSent=true;

      }

    } else {
```

```
      fireStatus = "No Fire";
      alertSent = false;
    }
    // --- END Fire Detection Logic ---
    // Log to Serial
    Serial.print("Flame Sensor Value: ");
    Serial.println(flameSensorValue);
    Serial.print("MQ135 Sensor Value: ");
    Serial.println(mq135SensorValue);
    Serial.print("Fire Status: ");
    Serial.println(fireStatus);

    // Send to Firebase
    Database.set<int>(aClient, "/sensor/flame", flameSensorValue, processData, "Send_Flame_Value");
    Database.set<int>(aClient, "/sensor/mq135", mq135SensorValue, processData, "Send_MQ135_Value");
    Database.set<String>(aClient, "/sensor/status", fireStatus, processData, "Send_Fire_Status");

  }
}
```

```
}


void processData(AsyncResult &aResult) {

  if (!aResult.isResult()) return;

  if (aResult.isEvent())

    Firebase.printf("Event task: %s, msg: %s, code: %d\n",
aResult.uid().c_str(), aResult.eventLog().message().c_str(),
aResult.eventLog().code());

  if (aResult.isDebug())

    Firebase.printf("Debug task: %s, msg: %s\n", aResult.uid().c_str(),
aResult.debug().c_str());

  if (aResult.isError())

    Firebase.printf("Error task: %s, msg: %s, code: %d\n",
aResult.uid().c_str(), aResult.error().message().c_str(),
aResult.error().code());

  if (aResult.available())

    Firebase.printf("task: %s, payload: %s\n", aResult.uid().c_str(),
aResult.c_str());

}
```

6.2 DEEPSTACK AI CODING

```
import cv2

import requests
```

```
import numpy as np

import datetime

import json

import firebase_admin

from firebase_admin import credentials, db

import os

import time



# --- DeepStack Configuration ---

DEEPSTACK_API_URL =
"http://192.168.43.8/v1/vision/custom/firenetv1"

# --- ESP32-CAM Stream URL ---

ESP32_CAM_URL = "http://192.168.43.205:81/stream"

FIREBASE_SERVICE_ACCOUNT_KEY = "fire-detection-project-
1481c-firebase-adminsdk-fbsvc-4b8240bb9e.json"

FIREBASE_DATABASE_URL = "https://fire-detection-project-1481c-
default-rtdb.firebaseio.com/"



# Initialize Firebase Admin SDK

try:

    cred =
credentials.Certificate(FIREBASE_SERVICE_ACCOUNT_KEY)

    firebase_admin.initialize_app(cred,  {
```

```
        'databaseURL': FIREBASE_DATABASE_URL
    })
    # Firebase path where AI confidence will be sent (e.g., sensor/ai_confidence)
    ref_ai_confidence = db.reference('sensor/ai_confidence')
    print("Firebase Admin SDK initialized successfully.")
except Exception as e:
    print(f"ERROR: Could not initialize Firebase Admin SDK. Check your service account key path and Firebase URL: {e}")
    exit() # Exit if Firebase initialization fails, as it's critical

# --- AI Detection Parameters ---
# The label IN our firenet model outputs for fire (e.g., "fire", "flame", "Fire").
# This is used to filter which objects DeepStack detects to consider as 'fire'.
FIRE_LABEL = "fire"
RESIZE_WIDTH = None # e.g., 640
RESIZE_HEIGHT = None # e.g., 480

# --- Firebase Update Frequency Control ---
# Variable to keep track of the last confidence sent to Firebase
# Initialize with an impossible value to force the first update.
```

```
last_reported_confidence_value = -1.0

CONFIDENCE_UPDATE_THRESHOLD = 0.02 # e.g., update if confidence changes by 2%



# --- Video Capture Setup ---

cap = cv2.VideoCapture(ESP32_CAM_URL)



if not cap.isOpened():

    print(f"ERROR: Could not open ESP32-CAM stream at {ESP32_CAM_URL}. Check URL and camera power.")

    exit()



print(f"Successfully opened stream from {ESP32_CAM_URL}. Starting AI detection loop...")



while True:

    ret, frame = cap.read()

    if not ret:

        print("WARNING: Could not receive frame from ESP32-CAM. Attempting to reconnect...")

        cap = cv2.VideoCapture(ESP32_CAM_URL) # Try to re-initialize camera capture

        if not cap.isOpened():
```

```
            print("ERROR: Failed to reconnect to camera. Exiting.")
            break # Exit the loop if reconnection fails
        time.sleep(1)
        continue
    if RESIZE_WIDTH and RESIZE_HEIGHT:
        frame_to_process = cv2.resize(frame, (RESIZE_WIDTH,
RESIZE_HEIGHT))
    else:
        frame_to_process = frame

    # Encode frame as JPEG for sending to DeepStack API
    _, img_encoded = cv2.imencode('.jpg', frame_to_process)
    files = {'image': ('frame.jpg', img_encoded.tobytes(), 'image/jpeg')}

    # Default confidence if no fire detected or an error occurs
    current_ai_confidence = 0.0

    try:
        # Send frame to DeepStack API with a timeout
        response = requests.post(DEEPSTACK_API_URL, files=files,
timeout=10) # 10-second timeout
```

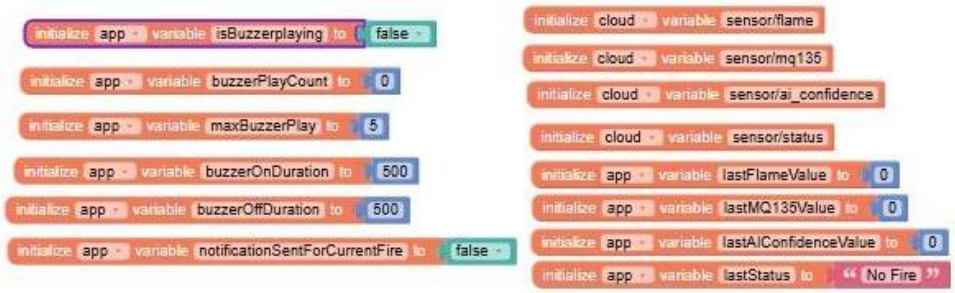
```
        response.raise_for_status() # Raise HTTPError for bad responses (4xx or 5xx)

        data = response.json()



        if data.get("success") and "predictions" in data:

            max_fire_confidence_in_frame = 0.0

            for obj in data["predictions"]:

                if obj.get("label", "").lower() == FIRE_LABEL.lower():

                    # Take the highest confidence value for 'fire' found in this frame

                    max_fire_confidence_in_frame = max(max_fire_confidence_in_frame, obj.get("confidence", 0.0))



            current_ai_confidence = max_fire_confidence_in_frame



        # Ensure confidence is within a valid 0.0 to 1.0 range

        current_ai_confidence = max(0.0, min(1.0, current_ai_confidence))



    except requests.exceptions.Timeout:

        print(f"WARNING: DeepStack API request timed out for {ESP32_CAM_URL}.")

        current_ai_confidence = -0.01 # Signify timeout error. App Inventor can check for negative values.
```

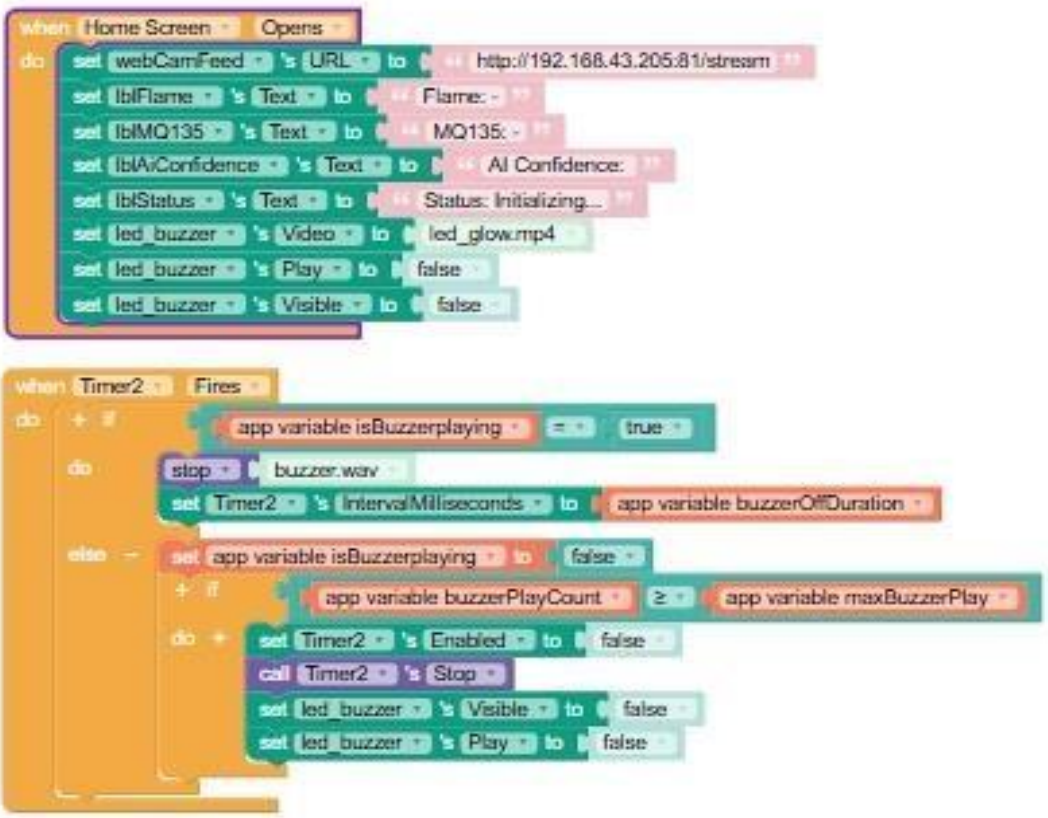
```
    except requests.exceptions.ConnectionError:
        print(f"ERROR: Could not connect to DeepStack API at
{DEEPSTACK_API_URL}. Is the server running and IP correct?")
        current_ai_confidence = -0.02 # Signify connection error
    except requests.exceptions.RequestException as req_err:
        print(f"ERROR: DeepStack API request failed: {req_err}")
        current_ai_confidence = -0.03 # Signify general API error
    except json.JSONDecodeError as json_err:
        print(f"ERROR: DeepStack response was not valid JSON:
{json_err}")
        current_ai_confidence = -0.04 # Signify invalid JSON response
    except Exception as e:
        print(f"An unexpected error occurred during DeepStack processing:
{e}")
        current_ai_confidence = -0.05 # Signify unknown error
    formatted_current_confidence_for_check =
float(f"{current_ai_confidence:.2f}")


    if abs(formatted_current_confidence_for_check -
last_reported_confidence_value) >
CONFIDENCE_UPDATE_THRESHOLD:
        print(f"[{datetime.datetime.now().strftime('%H:%M:%S')}] Sending
to Firebase:
AI_Confidence={formatted_current_confidence_for_check}")
        try:
```

```
            # Send the confidence as a string formatted to two decimal places

            ref_ai_confidence.set(f"{formatted_current_confidence_for_chec
k:.2f}")

            last_reported_confidence_value =
formatted_current_confidence_for_check

        except Exception as firebase_e:

            print(f"ERROR: Could not send AI confidence to Firebase:
{firebase_e}")

cap.release()

print("Stream processing and DeepStack detection stopped.")
```

6.3 THUNKABLE BLOCKS

SCREEN 1 BLOCK






when Timer1 Fires
do
set lblStatus 's Text to + join " Status : " cloud variable sensor/status
set app variable " lastStatus " to cloud variable sensor/status
set lblAiConfidence 's Text to + join " AI Confidence: " cloud variable sensor/ai_confidence
set app variable " lastAiConfidenceValue " to cloud variable sensor/ai_confidence
set lblMQ135 's Text to + join " MQ135: " cloud variable sensor/mq135
set app variable " lastMQ135Value " to cloud variable sensor/mq135
set lblFlame 's Text to + join " Flame: " cloud variable sensor/flame
set app variable " lastFlameValue " to cloud variable sensor/flame
+ if cloud variable sensor/status = " Fire Detected "
do + if app variable isBuzzerplaying = false and app variable buzzerPlayCount < app variable maxBuzzerPlay
do + set app variable buzzerPlayCount to app variable buzzerPlayCount + 1
set led_buzzer 's Visible to true
set led_buzzer 's Play to true
set lblStatus 's Background Color to
set app variable " isBuzzerPlaying " to true
set Timer2 's IntervalMilliseconds to app variable buzzerOnDuration
call Timer2 's Start
play buzzer.wav
set Timer2 's Enabled to true
else - set led_buzzer 's Visible to false
set led_buzzer 's Play to false
set lblStatus 's Background Color to
call Timer2 's Stop
set Timer2 's Enabled to false
set app variable " isBuzzerPlaying " to false
stop buzzer.wav
set app variable buzzerPlayCount to 0

CHAPTER VII

7. RESULT

7.1 HARDWARE SETUP

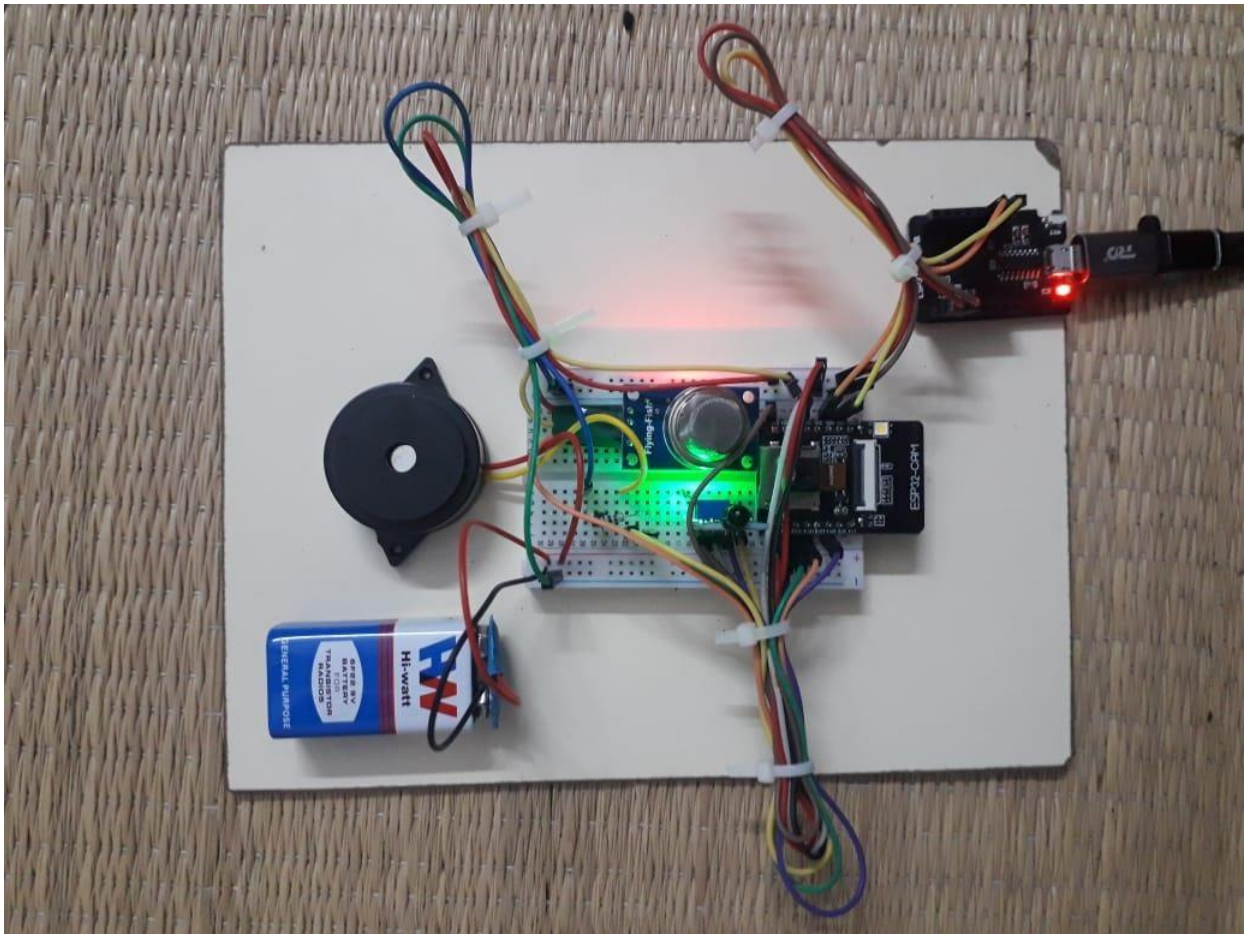


FIG 7.1 HARDWARE SETUP

7.2 THUNKABLE APP

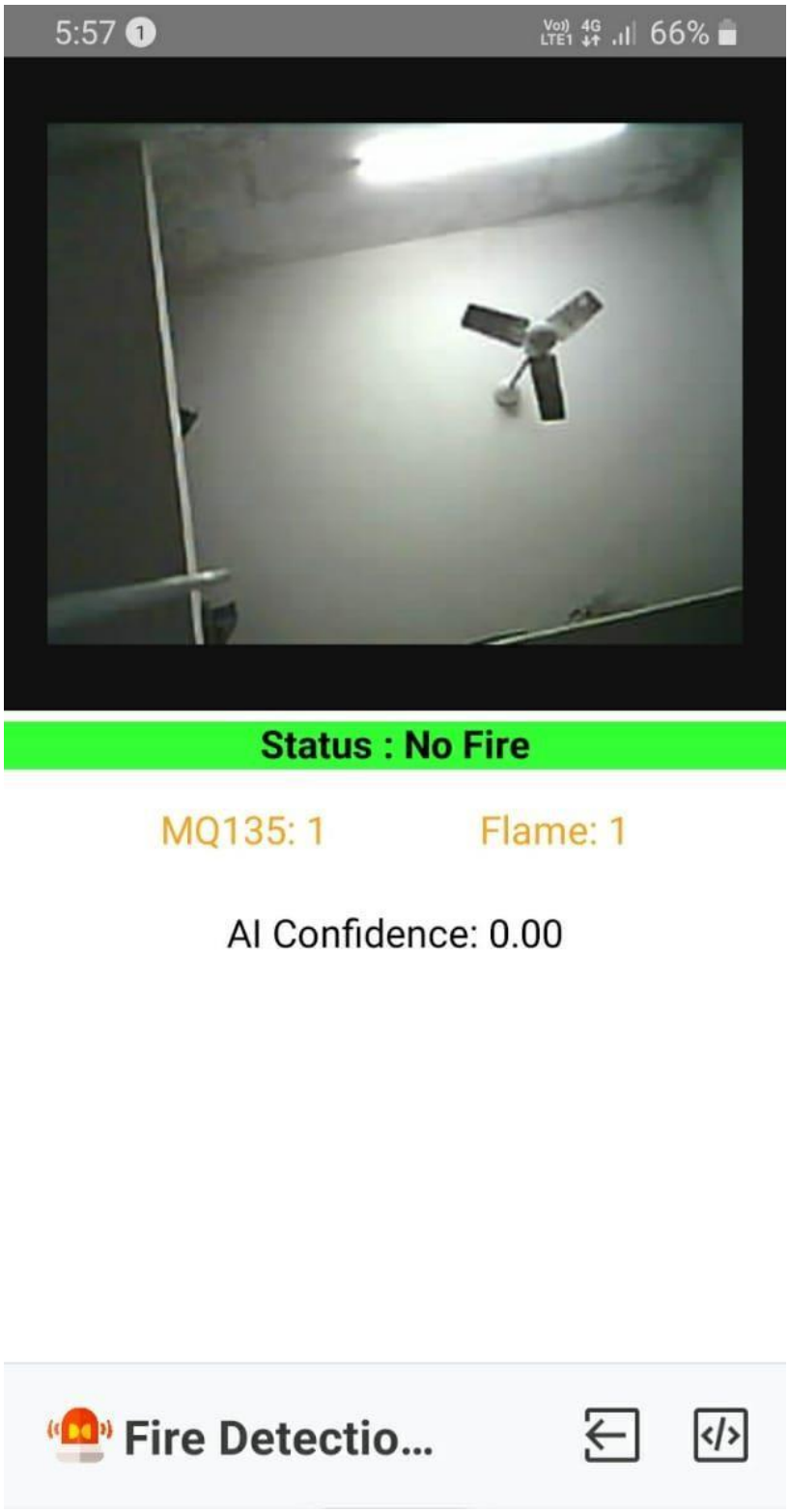


FIG 7.2 NO FIRE OUTPUT

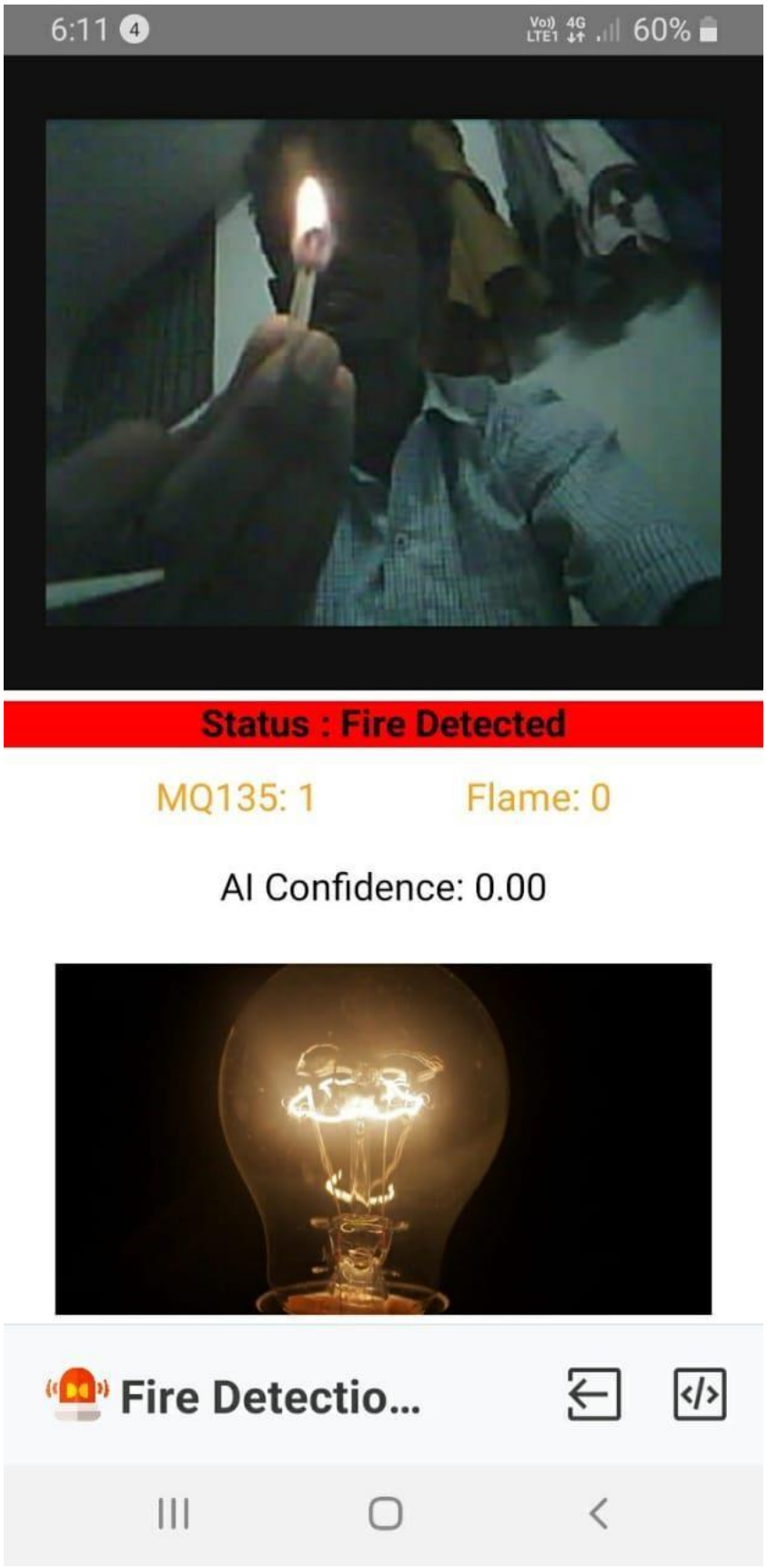


FIG 7.3 FIRE DETECTED OUTPUT

7.3 FIREBASE DATABASE

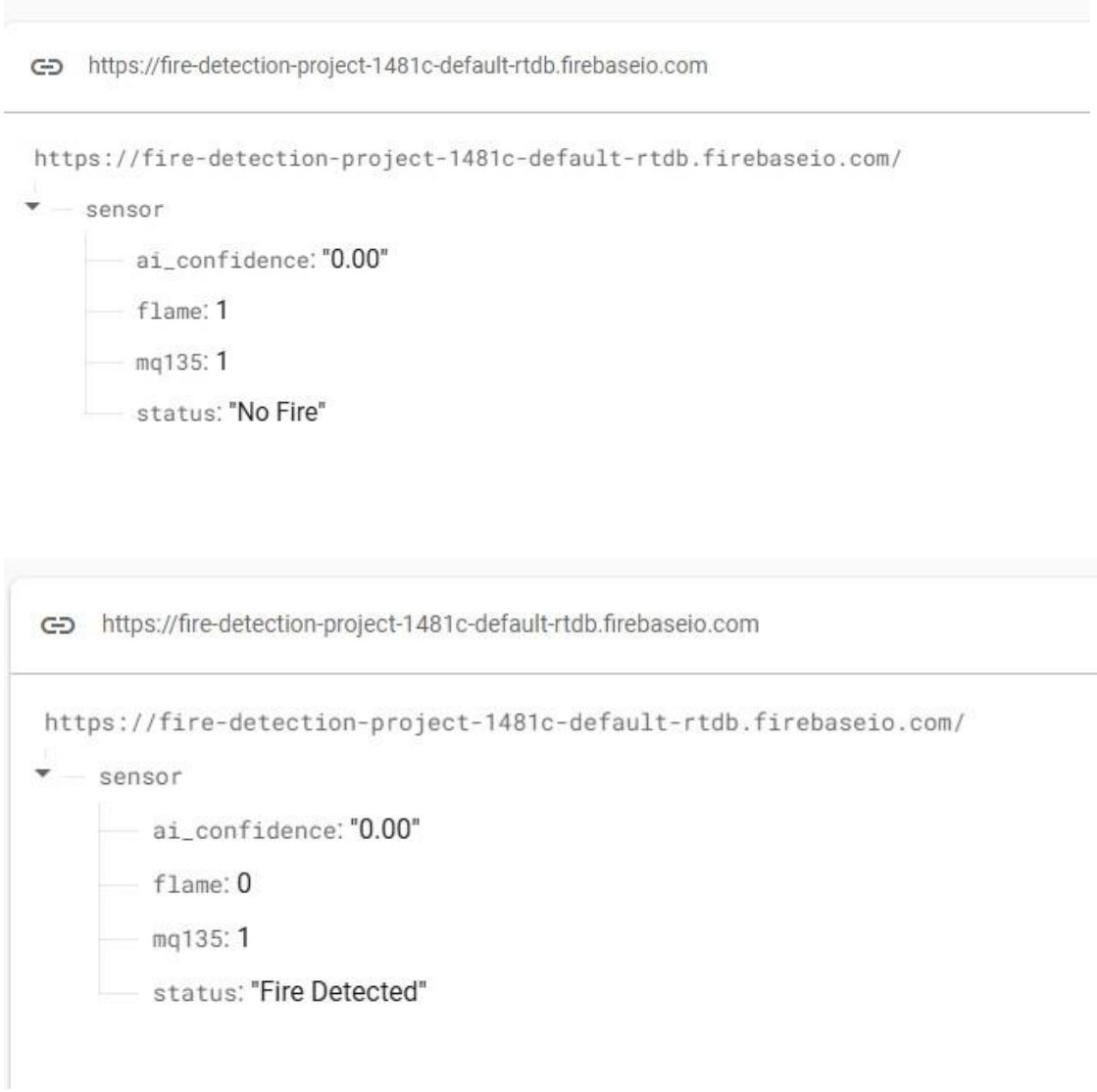
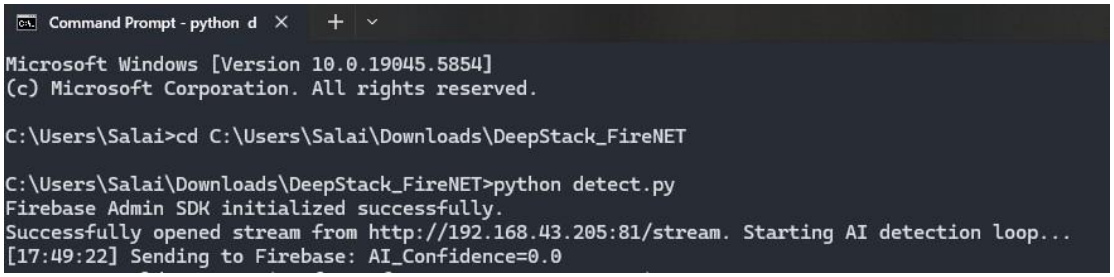


FIG 7.4 FIBEBASE DATABASE OUTPUT

7.4 DEEPSTACK SERVER



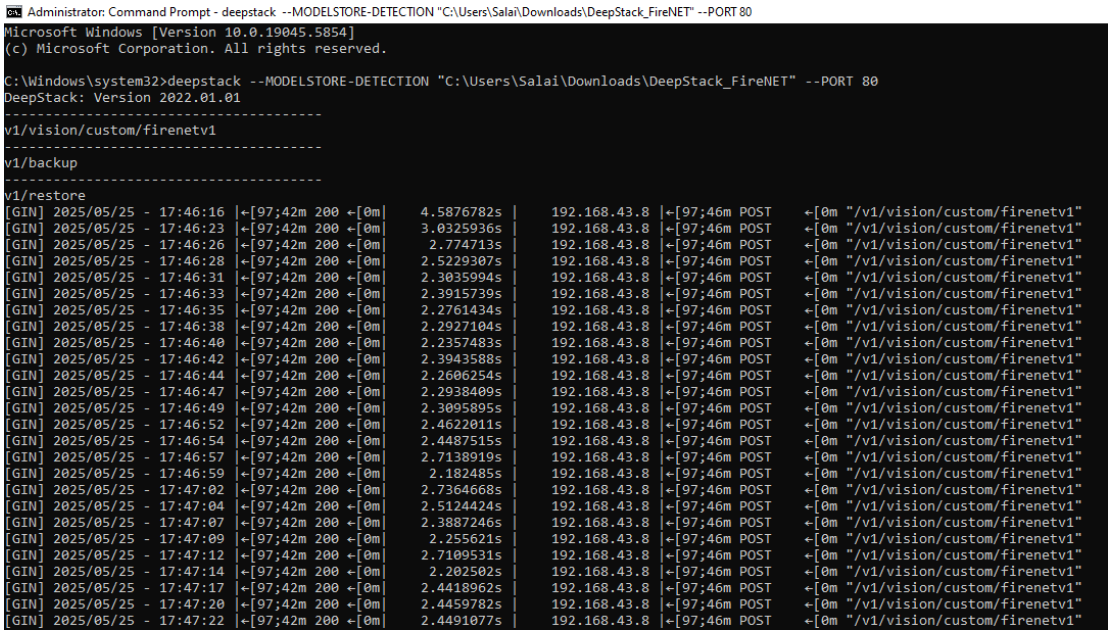


FIG 7.5 DEEPSTACK SERVER OUTPUT

7.5 ARDUINO SERIAL MONITOR

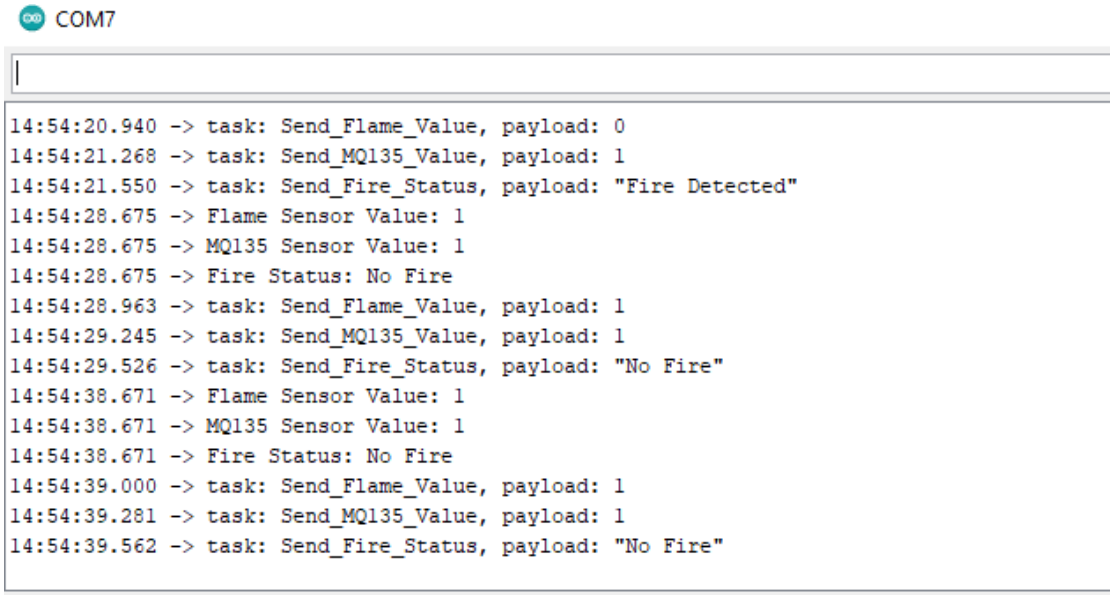


FIG 7.6 ARDUINO SERIAL MONITOR

CHAPTER VIII

8. CONCLUSION

The developed Smart Fire Detection System provides an efficient and real-time solution for early fire detection using a combination of sensor-based hardware and AI-powered visual monitoring. The integration of the ESP32-CAM, flame and MQ135 sensors, and DeepStack AI enables dual-mode detection—both physical and intelligent—enhancing the accuracy and reliability of the system.

The mobile application, built using Thunkable, acts as a central monitoring interface that receives real-time sensor data and AI-based fire status from Firebase. It visually alerts the user through dynamic UI changes and simulates virtual indicators (LED and buzzer) for remote awareness, even when the user is not on-site.

This system is cost-effective, scalable, and easily deployable in smart homes, farms, or industrial areas. It lays the foundation for further improvements, such as integrating temperature sensors, SMS/email alerts, or cloud storage for historical data—making it a powerful tool in advancing modern fire safety technology.

REFERENCES

[1] Jiawei Lan, Ye Tao, Zhibiao Wang, Haoyang Yu, Wenhua Cui, "Light-YOLOv8-Flame: A Lightweight High-Performance Flame Detection Algorithm," arXiv preprint, arXiv:2504.08389, 2025.

[2] H. Xu, B. Li, and F. Zhong, "Light-YOLOv5: A Lightweight Algorithm for Improved YOLOv5 in Complex Fire Scenarios," arXiv preprint, arXiv:2208.13422, 2022.

[3] W. Thomson, N. Bhowmik, and T. P. Breckon, "Efficient and Compact Convolutional Neural Network Architectures for Non-temporal Real-time Fire Detection," arXiv preprint, arXiv:2010.08833, 2020.

[4] M. Hasan, M. M. A. H. Prince, M. S. Ansari, S. Jahan, A. S. M. Miah, and J. Shin, "FireLite: Leveraging Transfer Learning for Efficient Fire Detection in Resource-Constrained Environments," arXiv preprint, arXiv:2409.20384, 2024.

[5] A. Sharma, K. Yadav, "Design and Implementation of an ESP32-Based Intelligent Industrial Surveillance System for Real-Time Fire Detection," ResearchGate, 2025.

[6] Scylla AI, "Smoke and Fire Detection System," Scylla.ai, 2025.

[7] NoemaTech, "AI Fire and Smoke Detection App," Noema.tech, 2025.

[8] Dragonfruit AI, "Fire & Smoke Alarm App," Dragonfruit.ai, 2025.

[9] F. Oliveira, L. Nascimento, M. Gomes, "Developing a Fire Monitoring System Based on MQTT, ESP-NOW, and ESP32-CAM," Applied Sciences, vol. 15, no. 2, p.s 500, 2025. DOI: 10.3390/app15020500

[10] DeepQuestAI, DeepStack FireNET: AI-Powered Fire Detection Using DeepStack, GitHub Repository.