" AI CAMERA USING ESP32-CAM "

CONTENTS

INTRODUCTION

🔍 Introduction: AI Camera using ESP32-CAM with 3.5" TFT Display

In recent years, Artificial Intelligence (AI) has revolutionized embedded systems, enabling smart vision applications in compact, low-power devices. This project focuses on building an **AI-powered camera system using the ESP32-CAM module**, integrated with a **3.5-inch ILI9488 TFT display**, to visualize and process real-time video or image data.

The **ESP32-CAM** is a low-cost development board based on the powerful **ESP32 microcontroller**, featuring a built-in OV2640 camera and support for Wi-Fi and Bluetooth. It is capable of running lightweight AI models such as face detection and recognition using onboard resources.

By interfacing the ESP32-CAM with a **3.5" TFT LCD**, we enable direct visual feedback without relying on a smartphone or PC. This makes the project ideal for **standalone security systems, smart doorbells, or IoT edge vision applications**.

✨ *Key Features:*

- Real-time image/video preview on 3.5” TFT display (ILI9488)
- Face detection and face recognition using ESP32-CAM
- Compact and wireless — no external computer required
- Wi-Fi support for remote monitoring or data upload
- Low power consumption and cost-effective

📌 *Applications:*

- Home/Office Surveillance
- Smart Entry Systems (Face-based Access)
- Wildlife Monitoring
- Baby or Pet Monitoring
- DIY IoT Vision Projects

This project showcases how AI and embedded systems can be combined to build an intelligent and interactive camera system using readily available hardware

APPLICATIONS OF AI CAMERA

AI cameras are intelligent vision systems that combine traditional imaging with artificial intelligence to enable real-time analysis, decision-making, and automation. These cameras are widely used across various industries due to their ability to recognize faces, detect objects, track movements, and analyze scenes autonomously. Below are some key applications:

1. Smart Surveillance Systems

AI cameras are extensively used in home and industrial security systems. They can detect human presence, recognize faces, and distinguish between known and unknown individuals. This reduces false alarms and increases response accuracy in real-time monitoring.

2. Face Recognition-Based Access Control

In buildings, offices, schools, and smart homes, AI cameras are used for facial recognition to allow or deny access. This eliminates the need for ID cards or fingerprint systems and enables contactless, secure entry systems.

3. Automated Attendance Systems

AI cameras can automate employee or student attendance by recognizing faces and recording time and date automatically. This reduces manual errors and improves efficiency in institutions and workplaces.

4. Traffic Monitoring and Vehicle Detection

In smart cities, AI cameras are used to monitor traffic flow, recognize license plates, detect violations, and manage traffic signals dynamically based on real-time conditions.

5. Healthcare Monitoring

AI vision systems are used to monitor patients for unusual behavior, fall detection, or unauthorized entry in restricted zones. They also help in ensuring hygiene compliance by detecting masks or PPE kits.

FUTURE OF AI CAMERA

- **Future of AI Camera**

The future of AI cameras is bright and transformative, driven by rapid advancements in artificial intelligence, edge computing, and sensor technology. AI cameras, such as those based on the ESP32-CAM, are evolving from basic image capture tools into intelligent vision systems capable of real-time analysis and decision-making.

Key Trends and Innovations:

1. **Edge AI Processing:**
    - Future AI cameras will increasingly perform processing on the device itself (edge computing), reducing the need to transmit data to external servers. This will improve speed, reduce latency, and enhance privacy.
2. **Improved Object Detection & Recognition:**
    - With advancements in deep learning, AI cameras will be able to detect and classify objects, people, faces, and even emotions more accurately and in more complex environments.
3. **Smart Surveillance:**
    - AI cameras will become essential in security systems, enabling features like motion detection, face recognition, license plate recognition, and anomaly detection, all without human intervention.
4. **Integration with IoT Ecosystems:**
    - Future smart homes and industries will integrate AI cameras into broader IoT systems, allowing devices to react automatically based on visual input (e.g., turn on lights when a person enters a room, alert on unusual activity).
5. **Energy Efficiency & Miniaturization:**
    - AI camera modules will become more energy-efficient and compact, making them suitable for battery-powered applications, including drones, wildlife monitoring, and wearable tech.
6. **Privacy-Focused AI:**
    - Emerging AI camera systems will include features like on-device data encryption, anonymization, and privacy zones to address increasing concerns about surveillance and data protection.

GPT-4O API

GPT-4o API for your **AI Camera project (e.g., ESP32-CAM or other hardware)**, you'll need to connect your device to the OpenAI API to process captured images or text. GPT-4o (the "o" stands for "omni") supports **text, image, and other multimodal inputs**, making it ideal for AI camera projects.

💡 AI Camera Applications with GPT-4o:

- Object recognition
- Surveillance alerts (e.g., detect person, fire, animal, etc.)

🔧 WHAT YOU NEED:

1. **OpenAI API Access**
   - Sign up or log in at https://platform.openai.com
   - Get your **API key** from the dashboard.
2. **Wi-Fi connectivity** on your hardware to send requests to the cloud (OpenAI API).

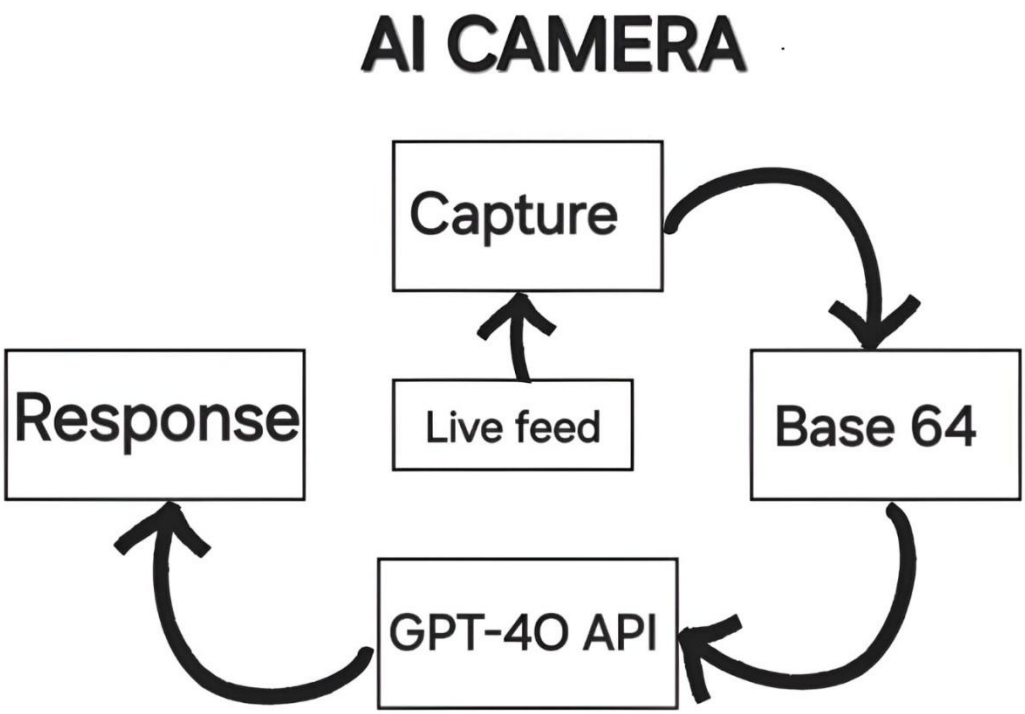
📲 FROM ESP32-CAM TO GPT-4o

Because ESP32-CAM can't directly call APIs like GPT-4o due to limited memory, the general flow is:

ESP32-CAM --> Send Image to Base64 format
|
v
Server Calls GPT-4o API
|
v
Returns Answer to ESP32

FLOW CHART OF IMPLEMENT

- **Flow Chart of Implementation – AI Camera**



1. **Live Feed** is captured by the ESP32-CAM or camera module.
2. The image frame is **captured** periodically.
3. The image is encoded to **Base64 format**.
4. The encoded data is sent to the **GPT-4o API** (or relevant AI processor).
5. A **response** is generated based on image interpretation, which could trigger a command or display feedback.

COMPONENTS FOR AI CAMERA

The proposed system consists of different components, the brief introduction about components was given below.

- ESP32-CAM.
- 3.5 TFT DISPLAY.
- Boost converter.
- Buzzer.
- Battery.
- Type-C TP4056 Battery Charger Module.
- 4x4 inch Double Sided PCB / Veroboard 4x4 inch.
- Slide Switch/ SPDT Switch.
- Push Button.

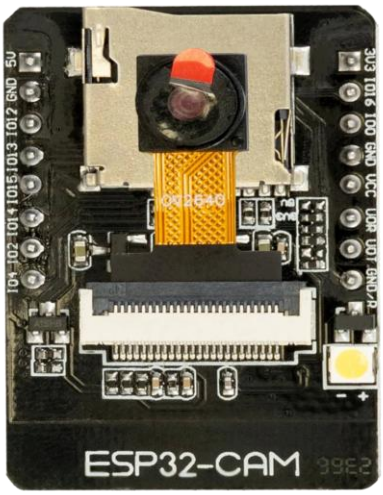
- **ESP32-CAM Development Board**

ESP32-CAM is a low-cost ESP32-based development board with onboard camera, small in size. It is an ideal solution for IoT application, prototypes constructions and DIY projects.

The board integrates WiFFi, traditional Bluetooth and low power BLE , with 2 high performance 32-bit LX6 CPUs. It adopts 7-stage pipeline architecture, on-chip sensor, Hall sensor, temperature sensor and so on, and its main frequency adjustment ranges from 80MHz to 240MHz.

Fully compliant with WiFi 802.11b/g/n/e/i and Bluetooth 4.2 standards, it can be used as a master mode to build an independent network controller, or as a slave to other host MCUs to add networking capabilities to existing devices

ESP32-CAM can be widely used in various IoT applications. It is suitable for home smart devices, industrial wireless control, wireless monitoring, QR wireless identification, wireless positioning system signals and other IoT applications. It is an ideal solution for IoT applications.



Notes:

1.Please be sure that the power supply for the module should be at least 5V 2A, otherwise maybe there would be water ripple appearing on the image

2.ESP32 GPIO32 pin is used to control the power of the camera, so when the camera is in working, pull GPIO32 pin low.

3.Since IO pin is connected to camera XCLK, it should be left floating in using, and do not connect it to high/low level.

4.The product has been equipped with default firmware before leaving the factory, and we do not provide additional ones for you to download. So, please be cautious when you choose to burn other firmwares.

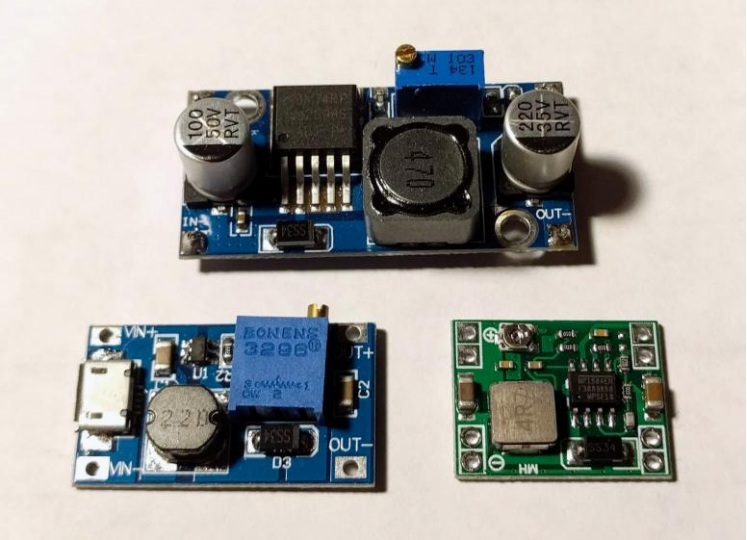
SPECIFICATION

- SPI Flash: default 32Mbit
- Up to 160MHz clock speed，Summary computing power up to 600 DMIPS
- Built-in 520 KB SRAM, external 4MPSRAM
- Bluetooth: Bluetooth 4.2 BR/EDR and BLE standards
- Wi-Fi: 802.11b/g/n/e/i
- Spectrum Range: 2412 ~2484MHz
- Antenna: onboard PCB antenna, gain 2dBi
- Power consumption: Turn off the flash: 180mA@5V Turn on the flash and adjust the brightness to the maximum: 310mA@5V Deep-sleep: the lowest power consumption can reach 6mA@5V Moderm-sleep: up to 20mA@5V Light-sleep: up to 6.7mA@5V

- Supports UART/SPI/I2C/PWM/ADC/DAC
- Support OV2640 and OV7670 cameras, Built-in Flash lamp
- Image Output Format: JPEG(OV2640 support only), BMP, GRAYSCALE
- Support for serial port local and remote firmware upgrades (FOT
- Serial Port Baud-rate: Default 115200 bps
- Power supply range: 5V

- **Boost converter**

A boost converter or step-up converter is a DC-to-DC converter that increases voltage, while decreasing current, from its input (supply) to its output (load)



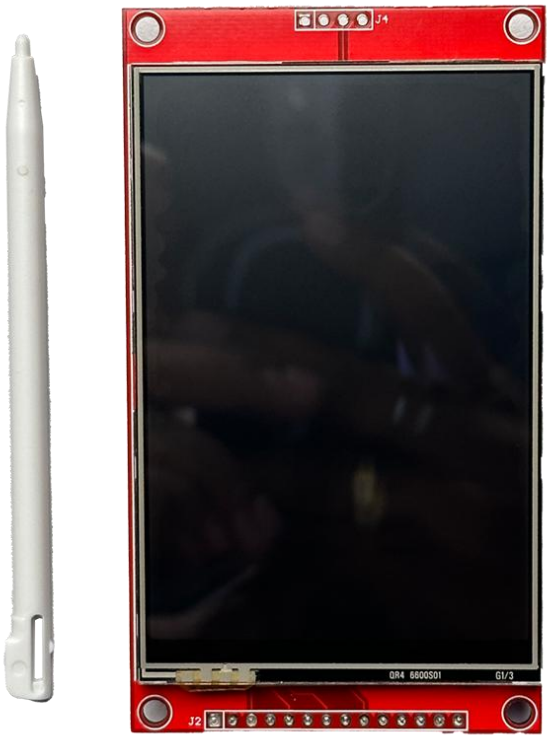
Low-cost converter modules: two buck and one boost

It is a class of switched-mode power supply (SMPS) containing at least two semiconductors, a diode and a transistor, and at least one energy storage element: a capacitor, inductor, or the two in combination. To reduce voltage ripple, filters made of capacitors (sometimes in combination with inductors) are norma ly added to such a converter's output (load-side filter) and input (supply-side filter)

Overview

Power for the boost converter can come from any suitable DC source, such as batteries, solar panels, rectifiers, and DC generators. A process that changes one DC voltage to a different DC voltage is ca led DC to DC conversion. A boost converter is a DC to DC converter with an output voltage greater than the source voltage. A boost converter is sometimes ca led a step-up converter since it "steps up" the source voltage. Since power ( ) must be conserved, the output current is lower than the source current.

TFT display (ILI9488)



About 3.5in TFT Touchscreen display

3.5 inch TFT Touchscreen SPI Interface 480×320 TFT Touch Screen Display for Arduino is big (3.5” diagonal) bright and colorful! 480×320 pixels with individual RGB pixel control, this has way more resolution than a black and white 128×64 display, and double our 2.8” TFT.

As a bonus, this display has a resistive touchscreen sensor allowing you to control your application simply by touching any area of the screen. This display has a controller built into it with RAM buffering so that almost no work is done by the microcontroller.

Features :

- 3.5-inch color screen,support 65K color display,display rich colors
- 480X320 resolution, optional touch function
- Easy to expand the experiment with SD card slot
- Provide a rich sample program
- Military-grade process standards, long-term stable work

- Provide underlying driver technical support
- Using the SPI serial bus, it only takes a few IOs to illuminate the display

Specifications

Name	Parameter
Display Color	RGB 65K color
Screen Size	3.5(inch)
Type	TFT
Driver IC	ILI9488
Resolution	480*320 (Pixel)
Module Interface	4 Wire SPI Interface
Active Area (AA area)	48.96×73.44(mm)
Module PCB Size	56.34×98(mm)
Operating Temperature	-20℃~60℃
Storage Temperature	-30℃~70℃
VCC power voltage	3.3V~5V
Logic IO port voltage	3.3V(TTL)
Rough Weight(Package containing)	With touch: 57 (g)

- **BUZZER**

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.
The **pin configuration of the buzzer** is shown below. It includes two pins namely positive and negative. The positive terminal of this is represented with the '+' symbol or a longer terminal. This terminal is powered through 6Volts whereas

the negative terminal is represented with the '-'symbol or short terminal and it is connected to the GND terminal.

Working Principle

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.

Advantages

The **advantages of a buzzer** include the following.

- Battery label indication
- Feedback image capture
- Wifi connect/disconnect feedback
- Battery charging indication

Disadvantages

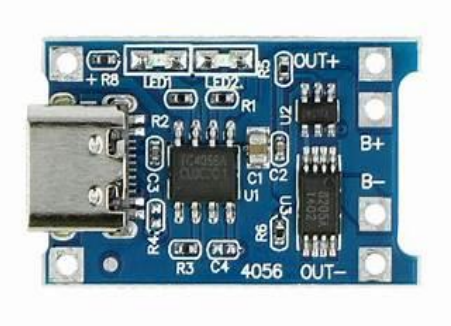
The **disadvantages of the buzzer** is noise.

Applications

The **applications of the buzzer** include the following.

- Electronics used in Automobiles
- Alarm Circuits
- Portable Devices
- Security Systems

- **TYPE C CHARGING MODULE**



About TP4056 Type C Battery Charging Module

This TP4056 Type C 1A Battery Charging Board with Current Protection is a tiny module, perfect for charging single cell 3.7V 1 Ah or higher cells such as 16550s that don't have their own protection circuit. Based on the TP4056 charger IC and DW01 battery protection IC this module will offer 1A charge current and then cut off when finished.
Furthermore, when the battery voltage drops below 2.4V the protection IC will switch the load off to protect the cell from running at too low of a voltage – and also protects against over-voltage and reverse polarity connection (it will usually destroy itself instead of the battery) however please check you have it connected correctly the first time.
Manufactured under full machinery processing. Each module is tested OK before shipment from **techiesms.com** so it has high reliability for the best performance.
In conclusion, it is a special design for a single lithium battery charging. TP4056
Other features include the current monitor, under-voltage lockout, automatic recharge, and two status pins to indicate charge termination and the presence of an input voltage.

Using the module:

1. Connect Type C cable for power, or 5V DC to pads marked IN+ and IN- on the left-hand side of the module
2. Connect the cell to charge to B+/B- pads on the right-hand side of the module.
3. A load (something for the battery to power) can be connected to the OUT+/OUT- pads on the right-hand side
4. Important! Disconnect load when charging
5. The red LED indicates charging in progress, the green LED indicates charging has finished.
6. Never charge your battery at a rate greater than 1C.

Features :

1. LED indicator: Red is charging Blue is fully charged.
2. Current Protection: Yes
3. Use mature charging chip TP4056 for simple peripheral circuits, good protection performance, and high charging accuracy.

- **SWITCH:-**

USE

- It use for on/of the AI Camara
- Save the battery power

- **Button:-**

- Trigger Button it use for Capture image in live feed

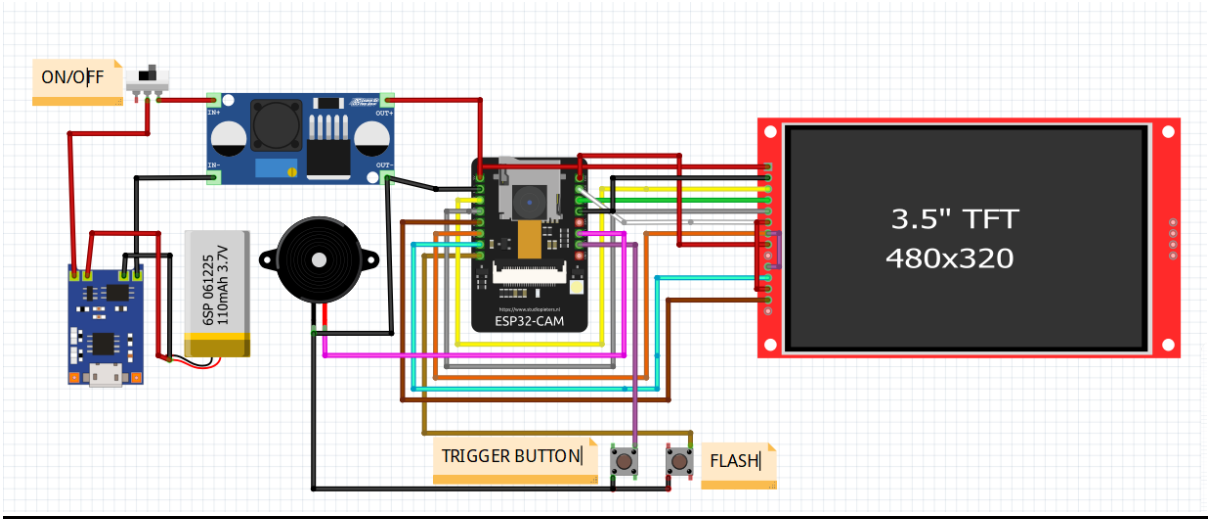
Lithium Ion Polymer Battery - 3.7v 300mAh

Description

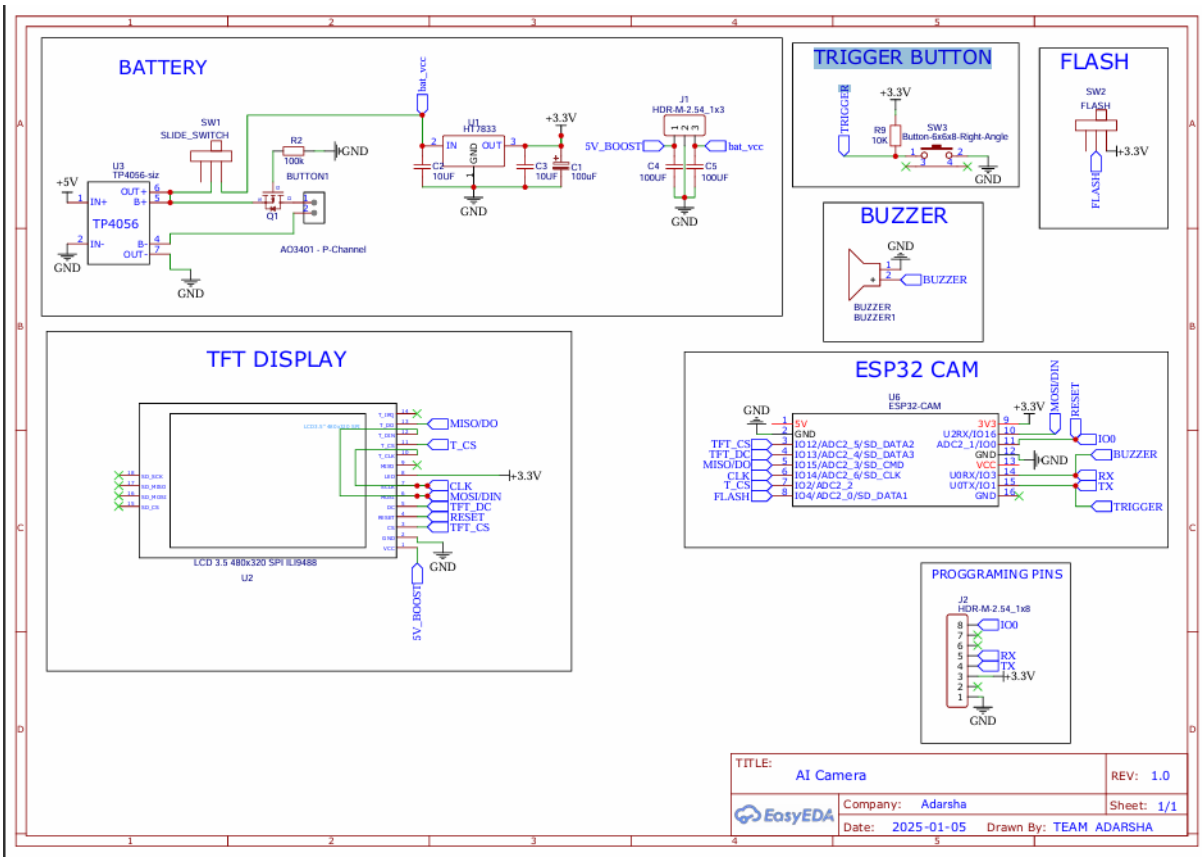
Lithium-ion polymer (also known as 'lipo' or 'lipoly') batteries are thin, light, and powerful. The output ranges from 4.2V when completely charged to 3.7V. This battery has a capacity of 500mAh for a total of about 1.9 Wh. If you need a larger (or smaller!) battery, we have a full range of LiPoly batteries.

The batteries come pre-attached with a 2-pin JST-PH connector as shown and include the necessary protection circuitry. Because they have a high quality JST-or-compatible connector, the cable won't snag or get stuck in a matching JST jack, they click in and out smoothly.

CONNECTION OF MODULE



Schematic



SOURCE CODE

The Code was test with

Arduino 1.8.13
ESP32 BOARD PACKAGE 1.0.6
TJpg_Decoder By Bodmer 0.0.3
TFT_SPI By Bodmer 2.3.4
lvgl By kisvegabor 7.8.1

```
*/
#include <WiFi.h>
#include <HTTPClient.h>
#include <TFT_eSPI.h>
#include <TJpg_Decoder.h>
#include <Base64.h>
#include <lvgl.h>
#include "esp_camera.h"
#include <ArduinoJson.h>

const char* ssid = "SSID";
const char* password = "PASS";

const String apiKey = "YOUR GPT-4o API KEY";

#define PWDN_GPIO_NUM 32
#define RESET_GPIO_NUM -1
#define XCLK_GPIO_NUM 0
#define SIOD_GPIO_NUM 26
#define SIOC_GPIO_NUM 27
#define Y9_GPIO_NUM 35
#define Y8_GPIO_NUM 34
#define Y7_GPIO_NUM 39
#define Y6_GPIO_NUM 36
```

```
#define Y5_GPIO_NUM 21
#define Y4_GPIO_NUM 19
#define Y3_GPIO_NUM 18
#define Y2_GPIO_NUM 5
#define VSYNC_GPIO_NUM 25
#define HREF_GPIO_NUM 23
#define PCLK_GPIO_NUM 22
#define FLASH_GPIO_NUM 4  // Flash pin (if using external flash)
#define BUZZER_PIN 3      // Buzzer connected to GPIO3 (for sound indication)
#define BUTTON_PIN 1      // Button pin for triggering capture (GPIO 1)


TFT_eSPI tft = TFT_eSPI();
static lv_disp_buf_t disp_buf;              // Buffer to hold pixel data for LVGL
static lv_color_t buf[LV_HOR_RES_MAX * 10];  // Buffer to store pixel data for LVGL

// Declare pointers for LVGL keyboard and text area objects
static lv_obj_t* kb;
static lv_obj_t* ta;

String userInputText = "";       // Holds the text entered by the user
String capturedImageBase64 = "";  // Holds the Base64-encoded image data

bool flashOn = false;             // Track flash state
bool isCaptureMode = false;       // Indicates if the system is in capture mode
bool isResponseDisplayed = false;  // Indicates if a response has been displayed

camera_fb_t* capturedFrameBuffer = nullptr;  // Holds the captured frame buffer


// Function to encode image to Base64
String encodeImageToBase64(uint8_t* imageData, size_t imageSize) {
  return base64::encode((const uint8_t*)imageData, imageSize);
}
```

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

  displayInit();
  drawBorders();

  pinMode(BUZZER_PIN, OUTPUT);
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  pinMode(FLASH_GPIO_NUM, OUTPUT);
  digitalWrite(FLASH_GPIO_NUM, LOW);  // Ensure flash is off initially

  displayLargeCenteredMessage("AI CAMERA BY ADARSHA");
  delay(1000);

  WiFi.begin(ssid, password);

  displayLargeCenteredMessage("Connecting to WiFi...");
  delay(500);
  Serial.println("Connecting to WiFi...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.println("...");
  }
  Serial.println("WiFi Connected!");
  Serial.println("IP Address: " + WiFi.localIP().toString());
  displayLargeCenteredMessage("WiFi Connected!");
  delay(500);

  cameraInit();  // Initialize camera
  lvglInit();    // Initialize LVGL (LittlevGL) graphics library
  lv_layout();   // Initialize LVGL layout and interface
}

// Initializes the TFT display and sets up touch calibration and image decoder
void displayInit() {
  tft.begin();
  tft.setRotation(1);  // Set display rotation (1 is usually landscape mode)
```

```
  tft.fillScreen(TFT_BLACK);

  uint16_t calData[5] = { 292, 3562, 332, 3384, 7 };  // Calibration data for the touch screen
  tft.setTouch(calData);                              // Set the touch calibration data for the screen

  drawBorders();

  TJpgDec.setJpgScale(1);           // Set the scale of the JPEG image (1 means no scaling)
  TJpgDec.setSwapBytes(true);       // Set byte swapping to handle different color formats correctly
  TJpgDec.setCallback(tft_output);  // Set the callback function to draw the image on TFT display
}

// LVGL display flush callback function for rendering the display buffer on the screen
void my_disp_flush(lv_disp_drv_t* disp, const lv_area_t* area, lv_color_t* color_p) {
  uint32_t w = (area->x2 - area->x1 + 1);
  uint32_t h = (area->y2 - area->y1 + 1);

  tft.startWrite();
  tft.setAddrWindow(area->x1, area->y1, w, h);
  tft.pushColors(&color_p->full, w * h, true);
  tft.endWrite();

  lv_disp_flush_ready(disp);
}

// Callback function used by the JPEG decoder to draw an image on the TFT screen
bool tft_output(int16_t x, int16_t y, uint16_t w, uint16_t h, uint16_t* bitmap) {
  if (y >= tft.height()) return 0;
  tft.pushImage(x, y, w, h, bitmap);
```

```
  return 1;
}

// Captures and displays a camera feed on the TFT display
void displayCameraFeed() {
  camera_fb_t* fb = esp_camera_fb_get();  // Capture a frame from the camera
  if (!fb) {
    Serial.println("Camera Capture Failed: Frame buffer is NULL!");
    return;
  }

  // Check if the captured frame is in the correct format (JPEG)
  if (fb->format != PIXFORMAT_JPEG) {
    Serial.println("Camera Capture Failed: Incorrect format!");
    esp_camera_fb_return(fb);
    return;
  }

  // Draw the captured JPEG image on the screen using the JPEG decoder
  TJpgDec.drawJpg(80, 5, (const uint8_t*)fb->buf, fb->len);
  esp_camera_fb_return(fb);
  drawBorders();
}

void cameraInit() {
  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_sscb_sda = SIOD_GPIO_NUM;
  config.pin_sscb_scl = SIOC_GPIO_NUM;
  config.pin_pwdn = PWDN_GPIO_NUM;
  config.pin_reset = RESET_GPIO_NUM;
  config.xclk_freq_hz = 9000000;
  config.pixel_format = PIXFORMAT_JPEG;
  config.frame_size = FRAMESIZE_QVGA;
  config.jpeg_quality = 12;
  config.fb_count = 1;

  if (esp_camera_init(&config) != ESP_OK) {
    displayMessage("Camera Initialization Failed!");
    return;
  }
  displayLargeCenteredMessage("Camera Initialized!");
  delay(500);
  drawBorders();
}

void lvglInit() {
  lv_init();  // Initialize LVGL

  lv_disp_buf_init(&disp_buf, buf, NULL, LV_HOR_RES_MAX * 10);   // Initialize
the display buffer and driver

  lv_disp_drv_t disp_drv;
  lv_disp_drv_init(&disp_drv);
  disp_drv.hor_res = 480;
  disp_drv.ver_res = 320;
  disp_drv.flush_cb = my_disp_flush;  // Custom flush function to update screen
  disp_drv.buffer = &disp_buf;
  lv_disp_drv_register(&disp_drv);

  // Initialize touch input device driver
```

```
  lv_indev_drv_t indev_drv;
  lv_indev_drv_init(&indev_drv);
  indev_drv.type = LV_INDEV_TYPE_POINTER;
  indev_drv.read_cb = my_touchpad_read;  // Custom touchpad read function
  lv_indev_drv_register(&indev_drv);

  // Initialize layout
  lv_layout();
}

void lv_layout() {
  // Create a material theme for consistency
  lv_theme_t*                                th                                =
lv_theme_material_init(LV_THEME_DEFAULT_COLOR_PRIMARY,
LV_THEME_DEFAULT_COLOR_SECONDARY, LV_THEME_MATERIAL_FLAG_DARK,
LV_THEME_DEFAULT_FONT_SMALL,       LV_THEME_DEFAULT_FONT_NORMAL,
LV_THEME_DEFAULT_FONT_SUBTITLE, LV_THEME_DEFAULT_FONT_TITLE);

  lv_obj_t* scr = lv_obj_create(NULL, NULL);
  lv_scr_load(scr);

  // Flash button
  lv_obj_t* flashBtn = lv_btn_create(scr, NULL);
  lv_obj_set_pos(flashBtn, 412, 55);  // Adjust as needed
  lv_obj_set_size(flashBtn, 50, 50);
  lv_obj_set_event_cb(flashBtn, flash_btn_event_cb);
  lv_obj_t* flashLabel = lv_label_create(flashBtn, NULL);
  lv_label_set_text(flashLabel, "Flash");

  // Capture button
  lv_obj_t* captureBtn = lv_btn_create(scr, NULL);
  lv_obj_set_pos(captureBtn, 408, 125);  // Adjust as needed
  lv_obj_set_size(captureBtn, 70, 65);
  lv_obj_set_event_cb(captureBtn, capture_btn_event_cb);
  lv_obj_t* captureLabel = lv_label_create(captureBtn, NULL);
  lv_label_set_text(captureLabel, "Capture");
```

```
  // Bottom message label
  lv_obj_t* messageLabel = lv_label_create(scr, NULL);
  lv_label_set_text(messageLabel, "Press the button to capture a new image");
  // Position the label at the bottom
  lv_obj_align(messageLabel, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, -30);
}

void flash_btn_event_cb(lv_obj_t* btn, lv_event_t event) {
  if (event == LV_EVENT_CLICKED) {
    flashOn = !flashOn;
    digitalWrite(FLASH_GPIO_NUM, flashOn ? HIGH : LOW);
    Serial.println(flashOn ? "Flash ON" : "Flash OFF");
  }
}

void capture_btn_event_cb(lv_obj_t* btn, lv_event_t event) {
  if (event == LV_EVENT_CLICKED) {
    Serial.println("Button clicked, capturing image...");
    displayLargeCenteredMessage("Capturing Image...");
    delay(100);
    captureAndProcessImage();
  }
}

void createKeyboard() {

  // Create a new screen for the keyboard and text area
  lv_obj_t* keyboard_screen = lv_obj_create(NULL, NULL);
  lv_scr_load(keyboard_screen);  // Load the new screen
  lv_obj_set_style_local_bg_opa(keyboard_screen,            LV_OBJ_PART_MAIN,
LV_STATE_DEFAULT, LV_OPA_COVER);
  lv_obj_set_style_local_bg_color(keyboard_screen,          LV_OBJ_PART_MAIN,
LV_STATE_DEFAULT, LV_COLOR_BLACK);

  // Create the text area
  lv_obj_t* ta_question = lv_textarea_create(keyboard_screen, NULL);
  lv_obj_set_size(ta_question, 400, 100);
```

```
  lv_obj_align(ta_question, NULL, LV_ALIGN_IN_TOP_MID, 0, 20);
  lv_textarea_set_placeholder_text(ta_question, "Enter your question:");
  lv_textarea_set_text(ta_question, "");

  // Create the keyboard
  lv_obj_t* kb = lv_keyboard_create(keyboard_screen, NULL);
  lv_obj_set_size(kb, LV_HOR_RES, LV_VER_RES / 2);
  lv_obj_align(kb, NULL, LV_ALIGN_IN_BOTTOM_MID, 0, 0);
  lv_keyboard_set_cursor_manage(kb, true);

  // Apply keyboard styles
  static lv_style_t kb_style;
  lv_style_init(&kb_style);
  lv_style_set_bg_color(&kb_style, LV_STATE_DEFAULT, LV_COLOR_GRAY);
  lv_style_set_bg_opa(&kb_style, LV_STATE_DEFAULT, LV_OPA_COVER);
  lv_style_set_border_width(&kb_style, LV_STATE_DEFAULT, 0);

  lv_obj_add_style(kb, LV_OBJ_PART_MAIN, &kb_style);

  lv_keyboard_set_textarea(kb, ta_question);
  lv_obj_set_event_cb(kb, keyboard_event_cb);  // Attach event callback
}

void keyboard_event_cb(lv_obj_t* kb, lv_event_t event) {
  lv_keyboard_def_event_cb(kb, event);  // Handle default keyboard behavior

  if (event == LV_EVENT_APPLY) {
    // Retrieve the text from the text area linked to the keyboard
    lv_obj_t* ta_question = lv_keyboard_get_textarea(kb);
    const char* text = lv_textarea_get_text(ta_question);

    userInputText = String(text);  // Store user input
    Serial.println("User Input: " + userInputText);

    // Process the input with the captured image
    if (!capturedImageBase64.isEmpty()) {
      exampleVisionQuestionWithImage(userInputText, capturedImageBase64);
```

```
    } else {
      displayMessage("Error: Failed to encode image.");
    }
  }

  if (event == LV_EVENT_CANCEL) {

    // Retrieve the text area linked to the keyboard
    lv_obj_t* ta_question = lv_keyboard_get_textarea(kb);

    if (ta_question) {
      // Clear the text area content (you can set an empty string)
      lv_textarea_set_text(ta_question, "");
    }
  }
}

void captureAndProcessImage() {

  // Capture the image from the camera
  capturedFrameBuffer = esp_camera_fb_get();

  if (!capturedFrameBuffer) {
    Serial.println("Camera capture failed");
    displayMessage("Error: Camera capture failed");

  // Encode the captured image to Base64
  capturedImageBase64 = encodeImageToBase64(capturedFrameBuffer->buf,
capturedFrameBuffer->len);

  // Immediately turn off the flash when the button is pressed
  if (flashOn) {
    digitalWrite(FLASH_GPIO_NUM, LOW);  // Turn off flash (GPIO 4)
    flashOn = false;
    Serial.println("Flash OFF");
  }
```

```
  beep();

  if (capturedImageBase64.isEmpty()) {
    Serial.println("Failed to encode the image!");
    displayMessage("Error: Image encoding failed");

    esp_camera_fb_return(capturedFrameBuffer);  // Return the frame buffer
    capturedFrameBuffer = nullptr;
  }

  Serial.println("Image encoded to Base64");

  // Stop the live feed by setting isCaptureMode to true
  isCaptureMode = true;

  // Display the captured image
  tft.fillScreen(TFT_BLACK);  // Clear the screen
  drawBorders();              // Draw borders around the screen

  if (capturedFrameBuffer) {
    TJpgDec.drawJpg(80,  5,  capturedFrameBuffer->buf,  capturedFrameBuffer-
>len);
  }
  displayLargeCenteredMessage("Opening the keyboard...");

  delay(3000);  // Wait for 3 seconds

  // Show the keyboard for user input
  createKeyboard();

  drawBorders();
}

void  exampleVisionQuestionWithImage(const  String&  userInputText,  const
String& base64Image) {
  String url = "" + base64Image;
```

```
  DynamicJsonDocument doc(4096);
  doc["model"] = "gpt-4o";
  JsonArray messages = doc.createNestedArray("messages");
  JsonObject message = messages.createNestedObject();
  message["role"] = "user";
  JsonArray content = message.createNestedArray("content");

  JsonObject textContent = content.createNestedObject();
  textContent["type"] = "text";
  textContent["text"] = userInputText;  // User question

  JsonObject imageContent = content.createNestedObject();
  imageContent["type"] = "image_url";
  JsonObject imageUrlObject = imageContent.createNestedObject("image_url");
  imageUrlObject["url"] = url;
  imageContent["image_url"]["detail"] = "auto";

  doc["max_tokens"] = 400;
  String payload;
  serializeJson(doc, payload);

  Serial.println("Sending request to ChatGPT...");
  Serial.println("User Input: " + userInputText);
  Serial.println("Base64 Image Size: " + String(base64Image.length()));

  // Clear the screen
  tft.fillScreen(TFT_BLACK);
  drawBorders();

  // Display the captured image
  if (capturedFrameBuffer) {
    TJpgDec.drawJpg(80,  5,  capturedFrameBuffer->buf,  capturedFrameBuffer-
>len);
  }

  // Display the "Sending request" message
  displayLargeCenteredMessage("Sending request to ChatGPT...");
```

```
  String result;
  if (sendPostRequest(payload, result)) {
    displayCapturedImageAndResponse(result);
    Serial.println("Response received from ChatGPT:");
    Serial.println(result);
  } else {
    Serial.println("Error: Unable to get a response from ChatGPT");
    displayCapturedImageAndResponse("Error: Unable to get response from ChatGPT");
  }
}

bool sendPostRequest(const String& payload, String& result) {
  HTTPClient http;
  http.begin("https://api.openai.com/v1/chat/completions");
  http.addHeader("Content-Type", "application/json");
  http.addHeader("Authorization", "Bearer " + apiKey);

  int httpCode = http.POST(payload);
  if (httpCode > 0) {
    if (httpCode == HTTP_CODE_OK) {
      result = http.getString();
      http.end();
      return true;
    } else {
      Serial.println("HTTP Error Code: " + String(httpCode));
    }
  } else {
    Serial.println("POST request failed.");
  }
  http.end();
  return false;
}

void displayCapturedImageAndResponse(const String& result) {
  // Parse the ChatGPT response
```

```
  StaticJsonDocument<2000> doc;
  DeserializationError error = deserializeJson(doc, result);
  const char* response = "Error: Parsing failed";  // Default message if parsing fails
  if (!error) {
    response = doc["choices"][0]["message"]["content"];
  }

  // Display the captured image
  if (capturedFrameBuffer) {
    TJpgDec.drawJpg(80,  5,  capturedFrameBuffer->buf,  capturedFrameBuffer->len);
  }

  // Display the response message
  displayMessage(response);  // Custom function for displaying message

  while (!isResponseDisplayed) {
    uint16_t touchX = 0, touchY = 0;
    if (tft.getTouch(&touchX, &touchY, 600)) {  // If touch detected
      Serial.println("Touch detected, reinitializing display...");
      isResponseDisplayed = 1;
    }
    delay(10);
  }

  if (isResponseDisplayed) {
    isResponseDisplayed = 0;
    reinitializeDisplay();  // Reinitialize display on touch
  }
}

void reinitializeDisplay() {
  // Clear the screen and reset the display
  tft.fillScreen(TFT_WHITE);  // Clear the screen
  drawBorders();            // Draw borders around the screen
```

```
  // Reinitialize any required settings or layouts
  capturedFrameBuffer = nullptr;
  capturedImageBase64 = "";
  isCaptureMode = false;

  Serial.println("Display reinitialized successfully");

  // Load the initial layout for the interface (if you are using a layout function)
  lv_layout();  // This should load the layout you want to display
}

void loop() {
  static unsigned long lastPressTime = 0;  // Debounce button press
  unsigned long currentMillis = millis();

  // Handle button press for toggling flash and capturing image
  if (digitalRead(BUTTON_PIN) == LOW) {
    // Check debounce time (ignores multiple button presses in quick succession)
    if (currentMillis - lastPressTime > 200) {
      lastPressTime = currentMillis;


      // Proceed with capturing the image
      displayLargeCenteredMessage("Capturing Image...");
      captureAndProcessImage();
    }

    // Wait until the button is released to avoid multiple triggers
    while (digitalRead(BUTTON_PIN) == LOW) {
      delay(10);  // Debounce and wait for release
    }
  }

  // Show live camera feed if not in capture mode and not displaying the response
  if (!isCaptureMode && !isResponseDisplayed) {
    displayCameraFeed();
  }
```

```
  // Check for touch event only after the image and response have been displayed
  if (isResponseDisplayed) {
    uint16_t touchX = 0, touchY = 0;
    if (tft.getTouch(&touchX, &touchY, 600)) {  // If touch detected
      Serial.println("Touch detected, reinitializing display...");
      reinitializeDisplay();       // Reinitialize display on touch
      isResponseDisplayed = false;  // Reset the flag after reinitialization
    }
  }

  lv_task_handler();  // Handle LVGL tasks
  delay(5);          // Small delay to balance responsiveness and CPU usage
}

// Function to handle touchpad input for LVGL
bool my_touchpad_read(lv_indev_drv_t* indev_driver, lv_indev_data_t* data)
{
  uint16_t touchX, touchY;

  bool touched = tft.getTouch(&touchX, &touchY, 600);  // Get touch input from
the screen, with a threshold of 600 (sensitivity)

  if (!touched) {
    data->state = LV_INDEV_STATE_REL;
  } else {
    data->state = LV_INDEV_STATE_PR;   // Set the state as "pressed" (touch
detected)
    data->point.x = touchX;
    data->point.y = touchY;
  }

  return false;
}

void drawBorders() {
  tft.drawRect(78, 2, 324, 246, TFT_WHITE);
```

```
  tft.drawRect(3, tft.height() - 70, 472, 68, TFT_WHITE);
}

void displayMessage(const String& message) {
  tft.fillRect(5, tft.height() - 65, 464, 60, TFT_BLACK);  // Reserve bottom section
for messages with black background
  tft.setTextColor(TFT_WHITE, TFT_BLACK);                  // White text on black
background
  tft.setTextSize(1);
  tft.setCursor(5, tft.height() - 65);

  int cursorX = 5, cursorY = tft.height() - 65;
  String currentWord;

  for (char c : message) {
    if (c == ' ' || c == '\n') {
      if (cursorX + tft.textWidth(currentWord) > tft.width() - 5) {
        cursorX = 5;
        cursorY += 10;
      }
      if (cursorY >= tft.height() - 5) break;
      tft.setCursor(cursorX, cursorY);
      tft.print(currentWord);
      cursorX += tft.textWidth(currentWord) + tft.textWidth(" ");
      currentWord = "";
    } else {
      currentWord += c;
    }
  }

  if (!currentWord.isEmpty()) {
    if (cursorX + tft.textWidth(currentWord) > tft.width() - 5) {
      cursorX = 5;
      cursorY += 10;
    }
    tft.setCursor(cursorX, cursorY);
    tft.print(currentWord);
```

```
  }
}

void displayLargeCenteredMessage(const String& message) {
  tft.fillRect(0, tft.height() - 70, tft.width(), 70, TFT_BLACK);  // Clear the message
area with black background
  tft.setTextSize(2);
  tft.setTextColor(TFT_WHITE, TFT_BLACK);  // White text on black background

  int x = (tft.width() - tft.textWidth(message)) / 2;
  int y = tft.height() - 50;  // Ensure this doesn't overlap with buttons

  tft.setCursor(x, y);
  tft.print(message);
  drawBorders();
}

void beep() {
  digitalWrite(3, HIGH);
  delay(300);
  digitalWrite(3, LOW);
}
```

WORKING

The AI Camera system is designed to capture, process, and display live video feed using an **ESP32-CAM** module integrated with a **3.5” TFT display**. The setup is powered by a rechargeable **3.7V Li-Po battery**, regulated through a **TP4056 charging module** and a **DC-DC booster** to supply 5V to the system. Below is a step-by-step explanation of how the system operates:

🟢 Power Initialization

- When the **ON/OFF switch** is turned on, power from the Li-Po battery passes through the **DC-DC booster**, which steps it up to 5V.
- This 5V is supplied to the **ESP32-CAM** and the **TFT display**, initializing the system.

🎥 Camera Activation

- The **ESP32-CAM** boots up and starts capturing a **live video feed**.
- The video stream is processed and sent to the **3.5” TFT display (480x320 resolution)** for real-time viewing.

📸 Image Capture via Trigger

- A **Trigger Button** connected to the ESP32’s GPIO pin allows the user to **manually capture** an image.
- When pressed, the ESP32 processes the image frame and can save it or send it (via Wi-Fi/Bluetooth, if programmed).

💡 Flash Function

- The **Flash Button** is used to activate an external LED (flash), useful for low-light photography.
- When pressed, it powers the LED through a GPIO pin momentarily or as long as held.

📢 Buzzer Feedback

- A **buzzer** is connected to provide audio feedback whenever a photo is captured or when motion is detected (if implemented).

- It’s controlled by the ESP32 via a digital output pin.

📏 **Optional Motion Sensing**

- The **Ultrasonic Sensor** detects motion or distance and can automatically trigger image capture or alerts.
- This makes the system suitable for surveillance or automated monitoring.

ADVANTAGES

Advantages of AI Camera

1. **Real-Time Monitoring**

   AI cameras can provide live video feeds, allowing users to monitor their homes or environments in real-time using smartphones, computers, or displays like a TFT screen.

2. **Enhanced Security**

   With features like facial recognition, motion detection, and anomaly alerts, AI cameras significantly improve home and workplace security by identifying intrusions or suspicious activity.

3. **Automated Decision Making**

   AI cameras can analyze images and respond intelligently without human intervention—for example, turning on lights when someone enters a room or sending an alert if an unknown person is detected.

4. **Remote Accessibility**

   Paired with IoT or Bluetooth modules, users can access the camera feed and control systems remotely via mobile apps, increasing convenience and control.

5. **Energy Efficiency**

   By detecting occupancy and movement, AI cameras can help automate lights, fans, and appliances, reducing unnecessary energy use and saving electricity.

6. **Integration with Smart Systems**

   AI cameras work well with other smart devices (like relays, sensors, and alarms), making them a vital part of home automation systems.

DISADVANTAGES

High Initial Cost

One of the primary disadvantages of implementing an AI camera-based smart home automation system is the **high initial cost**. This includes the cost of hardware components such as:

- **ESP32-CAM module**
- **3.5” TFT display**
- **Wiring and connectors**
- **Installation materials**

Additionally, if the system involves **cloud services, data storage, or premium AI APIs (e.g., GPT-4o)**, operational expenses may also increase.

While these costs can be justified by long-term savings in energy and security benefits, the **upfront investment may be a barrier** for many users, especially in budget-sensitive regions. The expense of troubleshooting, technical support, or future upgrades adds to the overall cost, making affordability a key challenge for widespread adoption.

System Complexity

The setup typically requires:

- Interfacing multiple modules (ESP32-CAM, “3.5” TFT display, Boost converter).
- Writing and uploading firmware code.
- Configuring GPT-4o.
- Handling real-time data (like video streaming and AI responses).
- Managing wireless communication protocols.

This level of integration demands a good understanding of **electronics, programming, and networking**. Even a small error in wiring or code logic can result in system failure or performance issues.

CONCLUSION

The implementation of an AI camera-based smart home automation system marks a significant step toward intelligent and secure living environments. By integrating the **ESP32-CAM**, **Arduino**, **Bluetooth module**, and **3.5" TFT display**, the system provides both **remote control** and **real-time video monitoring** of home appliances.

This project demonstrates how automation and artificial intelligence can work together to enhance the **comfort, safety, and energy efficiency** of modern homes. The live feed feature allows for visual feedback, while the Android app ensures convenient user control through wireless communication. Additionally, the use of AI in image capture and processing opens up new possibilities for smart surveillance and gesture-based control.

With growing demand for smarter homes in India and across the globe, this system lays the foundation for more **scalable**, **affordable**, and **intelligent automation solutions**. In the future, enhancements such as cloud integration, voice control, and advanced AI analytics can further increase its capabilities.

🚧 Limitations of AI Camera Using ESP32-CAM

⚙️ 1. Limited Processing Power

- **CPU:** 240 MHz (Tensilica LX6)
- **RAM:** ~512 KB SRAM + 4MB PSRAM (optional)
- Not suitable for:
    - Deep learning models (e.g. YOLO, CNNs)
    - Real-time object classification or segmentation

☑ ESP32-CAM is ideal for basic models like person detection or face detection, **not** full AI inference.

📦 2. Limited Memory

- Tiny SRAM limits complex code and large image buffers

- PSRAM (if present) improves this, but is still very constrained
- Memory overflows can cause crashes or unstable streaming

📶 3. Wi-Fi Performance Issues

- Streaming video over Wi-Fi can be:
  - **Laggy**
  - **Unstable**
  - Susceptible to packet loss
- Live MJPEG streaming consumes significant bandwidth

🔋 4. Power Consumption

- ESP32-CAM can consume **160–250 mA** when active
- Not ideal for long-term battery use unless deep sleep and sensors (like PIR) are added

📽️ 5. Camera Quality & Performance

- **OV2640 camera** is limited to:
  - Max 2MP (1600×1200)
  - Fixed focus
  - No autofocus, zoom, or stabilization
- Poor low-light performance
- Frame drops at higher resolutions

🧠 6. Limited On-Device AI Support

- No GPU or dedicated AI accelerator
- TinyML inference support is limited to **simple classification tasks**
  - E.g. "person detected" / "no person"
- Cannot run models like YOLOv5 or object tracking

🛠️ 7. Development Complexity

- Requires:
  - Arduino IDE / PlatformIO + ESP32 libraries
  - Careful memory and power management
  - Low-level debugging for Wi-Fi + streaming + camera interface
- Not beginner-friendly for large-scale vision projects

SCOPE FOR FURTHER IMPROVEMENT.

☑ Current Capabilities of a Basic ESP32-CAM AI Camera

A basic AI Camera using ESP32-CAM can typically:

- Stream live video over Wi-Fi (via IP address)
- Detect faces and eyes using on-device Haar cascades
- Perform motion detection or photo capture
- Trigger events via GPIO or HTTP
- Integrate with Firebase / Blynk / Telegram / Home Assistant

🚀 Scope for Further Improvement (Advanced Ideas)

1. Cloud-Based AI Processing (Hybrid Edge + Cloud)

- Use ESP32-CAM to **capture images** and send to:
  - OpenAI Vision API
  - Google Vision API
  - AWS Rekognition
  - Your own Python Flask server with YOLOv8
- Enables:
  - Face recognition
  - License plate recognition
  - Object tagging
  - Emotion analysis

2. Edge AI Models (Offline, On-Device)

- Use **TinyML** with:
  - TensorFlow Lite for Microcontrollers (TFLM)
  - Edge Impulse
- Deploy:
  - Person detection
  - Mask/no-mask detection
  - Pose detection
- ☑ Advantage: Works **without internet**, in real time

3. AI + IoT Integration

Combine with smart home/IoT actions:

Use Case	How
Doorbell AI	Detect face → Unlock door (via relay)
Smart Garden Monitor	Detect motion → Capture wildlife
Surveillance	Detect intruder → Alert via Telegram
Garbage Sorting	Detect type of waste → Open correct bin

4. Add SD Card Logging & Time-Stamped Events

- Log captured images or detection events
- Organize by date/time
- Sync with Google Drive using Node-RED

5. Improve UX/UI

- Build a mobile app or web dashboard (React / Flutter)
- Live view + image history + detection overlays
- Real-time alert configuration (motion sensitivity, zones, etc.)

6. Multiple ESP32-CAMs in Mesh

- Create a Wi-Fi mesh network for large-area monitoring
- Sync multiple ESP32-CAMs to a central server
- Time-synced footage for security or traffic analysi

7. Energy Efficiency & Battery Optimization

- Sleep mode + motion interrupt wake-up (PIR sensor)
- Solar-powered ESP32-CAM + low-power data transmission (LoRa, MQTT)

REFERENCES

AI Camera Project Reference Links

1. **Techiesms YouTube Channel – ESP32-CAM Video Tutorial**
   A beginner-friendly tutorial on using the ESP32-CAM module for AI camera applications.
   Watch on YouTube
2. **ESP32-CAM Development Board – Espressif Systems**
   Official ESP-IDF hardware reference and getting started guide for the ESP32-CAMboard.
   ESP32-CAM Hardware Reference
3. **Mozilla Developer Network (MDN) – Base64 Encoding**
   Explanation and implementation of Base64 encoding, useful for handling image data in embedded systems.
   Base64 Glossary – MDN
4. **OpenAI – GPT-4o API Documentation**
   Official documentation for integrating GPT-4o in AI projects such as smart camera features.
   OpenAI GPT-4o API Docs
5. **GitHub – Source Code & Schematic Documentation**
   Complete source code and circuit diagram for your AI Camera project.
   View on GitHub

QR code