```
#include <Cabinet_item_detection_inferencing.h>
#include "edge-impulse-sdk/dsp/image/image.hpp"
#include "esp_camera.h"
#include <WiFi.h>

// ---------------- Camera Model ----------------
#define CAMERA_MODEL_AI_THINKER // Has PSRAM

#if defined(CAMERA_MODEL_AI_THINKER)
  #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
#else
  #error "Camera model not selected"
#endif

// ---------------- Constants ----------------
#define EI_CAMERA_RAW_FRAME_BUFFER_COLS 320
#define EI_CAMERA_RAW_FRAME_BUFFER_ROWS 240
#define EI_CAMERA_FRAME_BYTE_SIZE 3

// ---------------- WiFi Setup ----------------
const char* ssid = "PL_507";
const char* password = "uiu54321";
// Replace this with the ESP32 main IP printed in its Serial Monitor
const char* TARGET_IP = "192.168.1.XXX";  // <-- CHANGE THIS to the main
ESP32 IP (e.g. 192.168.1.123)
const uint16_t TARGET_PORT = 80; // use HTTP port 80 (main ESP32 runs
WebServer on port 80)

// ---------------- Detection Settings ----------------
const float DETECTION_THRESHOLD = 0.90f;
const unsigned long MIN_SEND_INTERVAL_MS = 3000;
String lastSentLabel = "";
unsigned long lastSentTime = 0;

// ---------------- State Variables ----------------
static bool debug_nn = false;
static bool is_initialised = false;
uint8_t *snapshot_buf;
```

```
// ---------------- Camera Configuration ----------------
static camera_config_t camera_config = {
  .pin_pwdn = PWDN_GPIO_NUM,
  .pin_reset = RESET_GPIO_NUM,
  .pin_xclk = XCLK_GPIO_NUM,
  .pin_sscb_sda = SIOD_GPIO_NUM,
  .pin_sscb_scl = SIOC_GPIO_NUM,
  .pin_d7 = Y9_GPIO_NUM,
  .pin_d6 = Y8_GPIO_NUM,
  .pin_d5 = Y7_GPIO_NUM,
  .pin_d4 = Y6_GPIO_NUM,
  .pin_d3 = Y5_GPIO_NUM,
  .pin_d2 = Y4_GPIO_NUM,
  .pin_d1 = Y3_GPIO_NUM,
  .pin_d0 = Y2_GPIO_NUM,
  .pin_vsync = VSYNC_GPIO_NUM,
  .pin_href = HREF_GPIO_NUM,
  .pin_pclk = PCLK_GPIO_NUM,
  .xclk_freq_hz = 20000000,
  .ledc_timer = LEDC_TIMER_0,
  .ledc_channel = LEDC_CHANNEL_0,
  .pixel_format = PIXFORMAT_JPEG,
  .frame_size = FRAMESIZE_QVGA,
  .jpeg_quality = 12,
  .fb_count = 1,
  .fb_location = CAMERA_FB_IN_PSRAM,
  .grab_mode = CAMERA_GRAB_WHEN_EMPTY,
};

// ---------------- Function Prototypes ----------------
bool ei_camera_init(void);
void ei_camera_deinit(void);
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t
*out_buf);
static int ei_camera_get_data(size_t offset, size_t length, float
*out_ptr);
bool sendCommandToESP32(const char* cmd);

// ---------------- Setup ----------------
void setup() {
  Serial.begin(115200);
  while (!Serial) { delay(10); }
  Serial.println("Edge Impulse Object Detection Demo (ESP32-CAM)");

  // Connect to WiFi
  WiFi.begin(ssid, password);
  Serial.print("Connecting to WiFi");
  unsigned long start = millis();
  while (WiFi.status() != WL_CONNECTED && (millis() - start) < 10000) {
    Serial.print(".");
    delay(500);
  }
  if (WiFi.status() == WL_CONNECTED) {
    Serial.println("\nWiFi connected. IP: " + WiFi.localIP().toString());
  } else {
    Serial.println("\nWiFi connection failed (will continue offline).");
  }

  if (!ei_camera_init()) {
    ei_printf("Failed to initialize Camera!\r\n");
  } else {
    ei_printf("Camera initialized\r\n");
  }

  ei_printf("\nStarting continuous inference in 2 seconds...\n");
  ei_sleep(2000);
}

// ---------------- Main Loop ----------------
void loop() {
  if (ei_sleep(5) != EI_IMPULSE_OK) return;

  snapshot_buf = (uint8_t*)malloc(EI_CAMERA_RAW_FRAME_BUFFER_COLS *
                                  EI_CAMERA_RAW_FRAME_BUFFER_ROWS *
                                  EI_CAMERA_FRAME_BYTE_SIZE);
  if (snapshot_buf == nullptr) {
    ei_printf("ERR: Failed to allocate snapshot buffer!\n");
    return;
  }

  ei::signal_t signal;
  signal.total_length = EI_CLASSIFIER_INPUT_WIDTH *
EI_CLASSIFIER_INPUT_HEIGHT;
  signal.get_data = &ei_camera_get_data;

  if (!ei_camera_capture(EI_CLASSIFIER_INPUT_WIDTH,
EI_CLASSIFIER_INPUT_HEIGHT, snapshot_buf)) {
    ei_printf("Failed to capture image\r\n");
    free(snapshot_buf);
    return;
  }

  ei_impulse_result_t result = { 0 };
  EI_IMPULSE_ERROR err = run_classifier(&signal, &result, debug_nn);
  if (err != EI_IMPULSE_OK) {
    ei_printf("ERR: Failed to run classifier (%d)\n", err);
    free(snapshot_buf);
    return;
  }

  ei_printf("Predictions (DSP: %d ms., Classification: %d ms., Anomaly:
%d ms.):\n",
             result.timing.dsp, result.timing.classification,
result.timing.anomaly);

  // ---------------- Object Detection OR Classification ----------------
#if EI_CLASSIFIER_OBJECT_DETECTION == 1
    ei_printf("Object detection bounding boxes:\r\n");
    for (uint32_t i = 0; i < result.bounding_boxes_count; i++) {
        ei_impulse_result_bounding_box_t bb = result.bounding_boxes[i];
        if (bb.value == 0) continue;
        ei_printf("  %s (%f) [ x: %u, y: %u, width: %u, height: %u
]\r\n",
                  bb.label, bb.value, bb.x, bb.y, bb.width, bb.height);
    }

    // Optional: send detected label to main ESP32 if high confidence
    for (uint32_t i = 0; i < result.bounding_boxes_count; i++) {
        ei_impulse_result_bounding_box_t bb = result.bounding_boxes[i];
        if (bb.value >= DETECTION_THRESHOLD) {
            String detectedLabel = String(bb.label);
            detectedLabel.toUpperCase();

            unsigned long now = millis();
            if (detectedLabel.length() &&
                (detectedLabel != lastSentLabel || (now - lastSentTime) >
MIN_SEND_INTERVAL_MS)) {
                if (sendCommandToESP32(detectedLabel.c_str())) {
                    lastSentLabel = detectedLabel;
                    lastSentTime = now;
                    ei_printf("Sent command to ESP32: %s\r\n",
detectedLabel.c_str());
                } else {
                    ei_printf("Failed to send command to ESP32: %s\r\n",
detectedLabel.c_str());
                }
            }
        }
    }

#else
    ei_printf("Predictions:\r\n");
    float best = 0.0f;
    int bestIx = -1;
    for (uint16_t i = 0; i < EI_CLASSIFIER_LABEL_COUNT; i++) {
        ei_printf("  %s: %.5f\r\n",
ei_classifier_inferencing_categories[i],
                                   result.classification[i].value);
        if (result.classification[i].value > best) {
            best = result.classification[i].value;
            bestIx = i;
        }
    }

    if (bestIx >= 0) {
        String topLabel =
String(ei_classifier_inferencing_categories[bestIx]);
        ei_printf("Top: %s (%.3f)\r\n", topLabel.c_str(), best);

        String tl = topLabel;
        tl.toLowerCase();
        unsigned long now = millis();
        if (best >= DETECTION_THRESHOLD) {
            String sendLabel = "";
            if (tl.indexOf("potato") >= 0) sendLabel = "POTATO";
            else if (tl.indexOf("onion") >= 0) sendLabel = "ONION";
            else if (tl.indexOf("garlic") >= 0) sendLabel = "GARLIC";

            if (sendLabel.length() && (sendLabel != lastSentLabel || (now
- lastSentTime) > MIN_SEND_INTERVAL_MS)) {
                if (sendCommandToESP32(sendLabel.c_str())) {
                    lastSentLabel = sendLabel;
                    lastSentTime = now;
                    ei_printf("Sent command to ESP32: %s\r\n",
sendLabel.c_str());
                } else {
                    ei_printf("Failed to send command to ESP32: %s\r\n",
sendLabel.c_str());
                }
            }
        }
    }
#endif

#if EI_CLASSIFIER_HAS_ANOMALY
    ei_printf("Anomaly prediction: %.3f\r\n", result.anomaly);
#endif

  free(snapshot_buf);
}

// ---------------- Camera Init ----------------
bool ei_camera_init(void) {
  if (is_initialised) return true;

  esp_err_t err = esp_camera_init(&camera_config);
  if (err != ESP_OK) {
    Serial.printf("Camera init failed with error 0x%x\n", err);
    return false;
  }

  sensor_t *s = esp_camera_sensor_get();
  if (s->id.PID == OV3660_PID) {
    s->set_vflip(s, 1);
    s->set_brightness(s, 1);
    s->set_saturation(s, 0);
  }

  is_initialised = true;
  return true;
}

// ---------------- Capture Image ----------------
bool ei_camera_capture(uint32_t img_width, uint32_t img_height, uint8_t
*out_buf) {
  if (!is_initialised) {
    ei_printf("ERR: Camera not initialized\r\n");
    return false;
  }

  camera_fb_t *fb = esp_camera_fb_get();
  if (!fb) {
    ei_printf("Camera capture failed\n");
    return false;
  }

  bool converted = fmt2rgb888(fb->buf, fb->len, PIXFORMAT_JPEG,
snapshot_buf);
  esp_camera_fb_return(fb);
  if (!converted) {
    ei_printf("Conversion failed\n");
    return false;
  }

  if (img_width != EI_CAMERA_RAW_FRAME_BUFFER_COLS ||
      img_height != EI_CAMERA_RAW_FRAME_BUFFER_ROWS) {
    ei::image::processing::crop_and_interpolate_rgb888(
      out_buf,
      EI_CAMERA_RAW_FRAME_BUFFER_COLS,
      EI_CAMERA_RAW_FRAME_BUFFER_ROWS,
      out_buf,
      img_width,
      img_height
    );
  }

  return true;
}

// ---------------- Camera Data Getter ----------------
static int ei_camera_get_data(size_t offset, size_t length, float
*out_ptr) {
  size_t pixel_ix = offset * 3;
  size_t pixels_left = length;
  size_t out_ptr_ix = 0;
  while (pixels_left != 0) {
    out_ptr[out_ptr_ix] =
        (snapshot_buf[pixel_ix + 2] << 16) +
        (snapshot_buf[pixel_ix + 1] << 8) +
        snapshot_buf[pixel_ix];
    out_ptr_ix++;
    pixel_ix += 3;
    pixels_left--;
  }
  return 0;
}

// ---------------- Send Command to ESP32 ----------------
bool sendCommandToESP32(const char* cmd) {
  // cmd is expected like "POTATO", "ONION", "GARLIC"
  if (WiFi.status() != WL_CONNECTED) {
    Serial.println("WiFi not connected (CAM).");
    return false;
  }

  String path = "";
  String up = String(cmd);
  up.toUpperCase();

  if (up.indexOf("POTATO") >= 0) path = "/play35";
  else if (up.indexOf("ONION") >= 0) path = "/play36";
  else if (up.indexOf("GARLIC") >= 0) path = "/play37";
  else {
    Serial.printf("No mapping for label: %s\n", cmd);
    return false;
  }

  WiFiClient client;
  if (!client.connect(TARGET_IP, TARGET_PORT)) {
    Serial.printf("Failed to connect to %s:%d\n", TARGET_IP,
TARGET_PORT);
    return false;
  }

  // send HTTP GET
  client.print(String("GET ") + path + " HTTP/1.1\r\n" +
               "Host: " + TARGET_IP + "\r\n" +
               "Connection: close\r\n\r\n");

  // wait for response (short timeout)
  unsigned long start = millis();
  while (!client.available() && (millis() - start) < 1000) {
    delay(5);
  }

  // read (optional) - consume response for cleanliness
  while (client.available()) {
    String line = client.readStringUntil('\n');
    // You can print response lines if you want:
    // Serial.println(line);
  }

  client.stop();
  return true;
}

#if !defined(EI_CLASSIFIER_SENSOR) || EI_CLASSIFIER_SENSOR !=
EI_CLASSIFIER_SENSOR_CAMERA
  #error "Invalid model for current sensor"
#endif
```