Report Final

HỆ THỐNG TỰ ĐỘNG PHÂN LOẠI TÁO DỰA TRÊN MÀU SẮC

PBL5: Hệ thống nhúng và IoT

Ngày 13 tháng 2 năm 2025

Sinh viên thực hiện
106200284 - Hồ Đức Vũ - 20KTMT2
106200241 - Nguyễn Minh Phương - 20KTMT1
106200271 - Lê Tuấn Nhật - 20KTMT2
106200242 - Nguyễn Văn Vĩnh Quang - 20KTMT1

Giáo viên hướng dẫn
Ths. Hồ Viết Việt

Môn học
PBL5: Hệ thống nhúng và IoT

Đề tài
HỆ THỐNG TỰ ĐỘNG PHÂN LOẠI TÁO DỰA TRÊN MÀU SẮC

Xuất bản
Đà Nẵng, Ngày 13 tháng 2 năm 2025

Số trang
13

Mục lục

Nội dung báo cáo **3**

1 Giới thiệu **3**

2 Use case Diagram **3**

3 Activity Diagram **5**

4 Prediction system **6**

5 Sơ đồ kết nối phần cứng **7**

6 Ứng dụng FreeRTOS đối với ESP32 **9**
6.1 Các đoạn code quan trọng 9
6.2 Biểu đồ thời gian 9

7 Biểu diễn kết quả **10**

Tài liệu tham khảo **12**

Phụ lục **13**

1 Giới thiệu

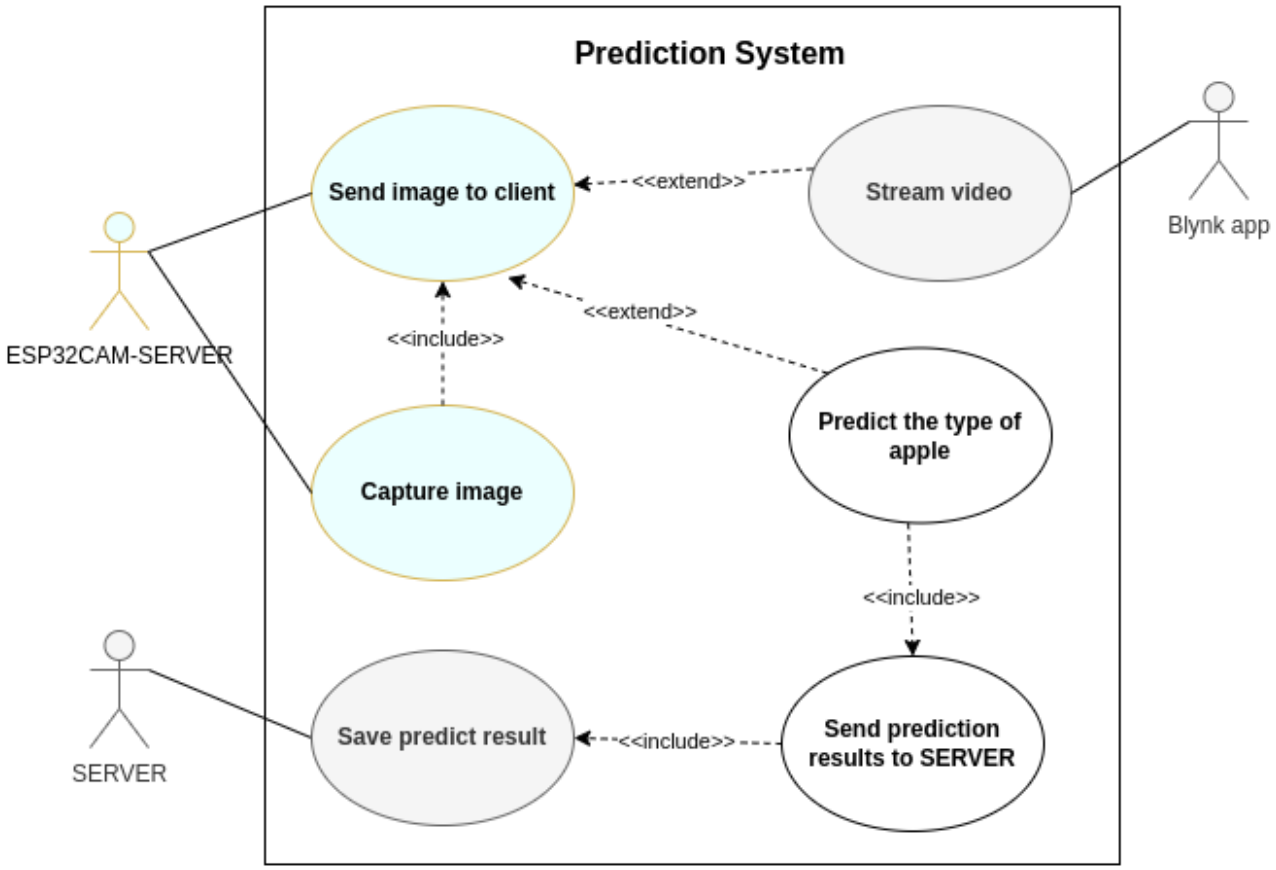
Ngày nay, việc tự động hóa máy móc trong sản xuất và đóng gói sản phẩm đã trở nên phổ biến trong các nhà máy, xí nghiệp. Cùng với đó là sự phát triển thần tốc của công nghệ *Internet of Things (IoT)* và *Trí tuệ nhân tạo* càng thúc đẩy sự phát triển mạnh mẽ nền công nghiệp sản xuất tự động, nơi mà hệ thống máy móc có thể hoạt động một cách độc lập và tương tác với nhau mà không hoặc ít cần sự can thiệp của con người.

Dựa trên xu hướng đó, nhóm đã lựa chọn đề tài *Hệ thống tự động phân loại táo dựa trên màu sắc* với mong muốn áp dụng những kiến thức, kỹ năng mà bản thân đã được và đang học, cùng với đó là bắt kịp xu hướng về tự động hóa, IoT và trí tuệ nhân tạo.

Đề tài *Hệ thống tự động phân loại táo dựa trên màu sắc* được thực hiện với mục đích phân loại táo gồm táo xanh và táo đỏ một cách tự động trên băng chuyền. Với đầu vào của hệ thống là các loại táo cần phân loại, đầu ra của hệ thống là các giỏ táo với mỗi giỏ chứa 1 loại táo. Hệ thống sử dụng camera từ esp32cam để thu thập hình ảnh trái cây trên băng chuyền, sau đó hình ảnh sẽ được xử lý và đưa ra dự đoán về loại trái cây, quá trình xử lý sẽ được hệ thống dự đoán đảm nhiệm sau đó kết quả dự đoán được lưu trữ tại Server, kết quả được gửi về một bộ kit khác mà nhóm sử dụng là esp32 để thực hiện phân loại trái cây vào giỏ tương ứng. Nguyên lý hoạt động của hệ thống được thể hiện chi tiết tại §2.

2 Use case Diagram

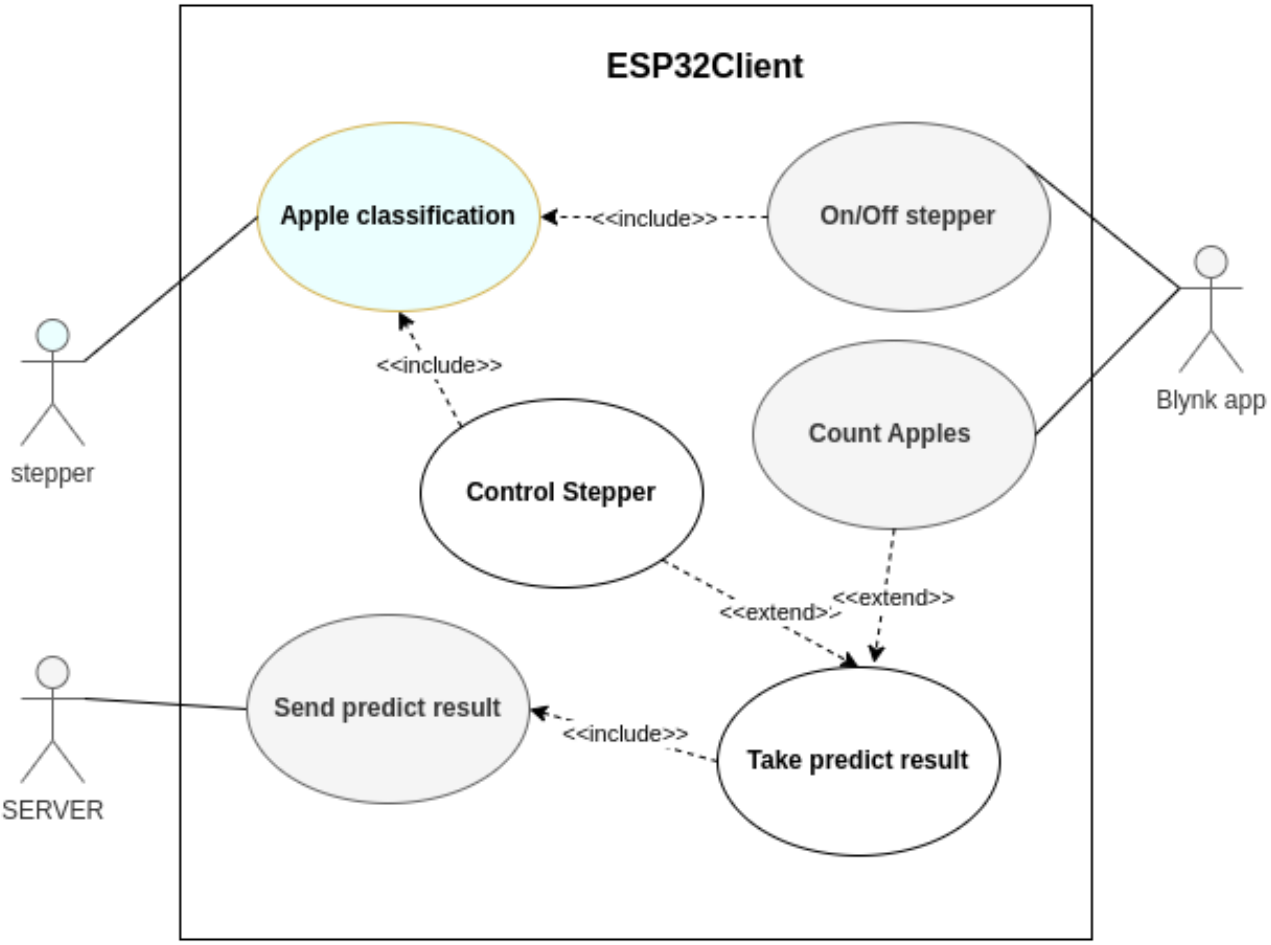
Hệ thống phân loại táo tự động được chia thành 2 phần chính: Prediction System và ESP32Client. Hình 2.1 và hình 2.2 lần lượt mô tả sơ đồ Use case của Prediction System và ESP32Client.



Hình 2.1: Sơ đồ Use case Prediction system

Đối với Prediction system, tại phần này ta có 6 chức năng và 3 actor. Các chức năng bao gồm "Capture Image"và "Send image to client", do actor ESP32CAM-SERVER đảm nhiệm, lúc này ESP32CAM đóng vai trò như một server chụp ảnh đầu vào của hệ thống và gửi ảnh đến các clients, chức năng khác là "Stream video", chức năng này được dùng để theo dõi hệ thống thông qua Blynk app, 2 chức năng dự "Predict the type of apple"và "Send prediction result to SERVER"được sử dụng để dự đoán loại táo từ dữ liệu ảnh có được từ ESP32CAM-SERVER và Gửi kết quả dự đoán đó đến SERVER. SERVER khác với ESP32CAM-SERVER, SERVER được dùng để lưu trữ kết quả dự đoán của Prediction System (chức năng "Save predict result") và gửi dự đoán đó đến ESP32Client (hình 2.2).

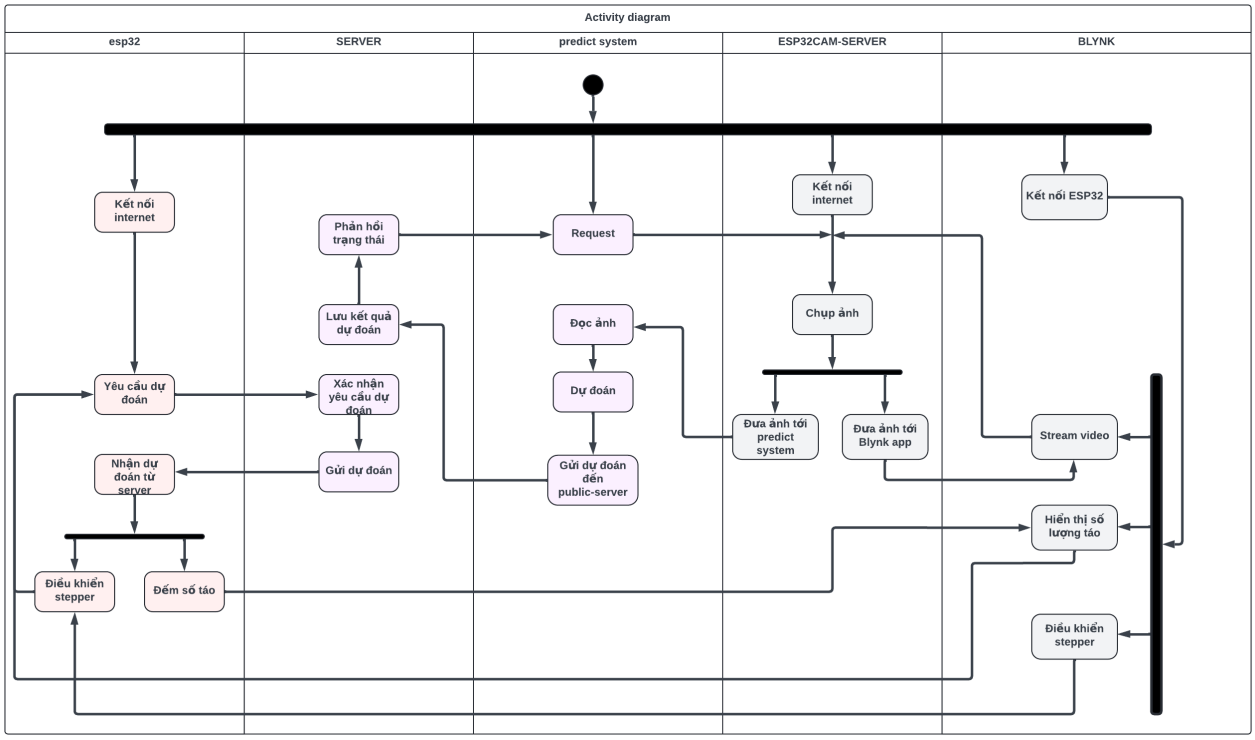
Đối với ESP32Client System, tại đây ta cũng có 6 chức năng và 3 actor. SERVER gửi kết quả dự đoán về ESP32Client bởi "Send predict result", hệ thống lấy kết quả dự đoán thông qua chức năng "Take predict result", chức năng "Control Stepper"là chức năng xử lý dữ liệu dự đoán có được để đưa ra quyết định điều khiển stepper, chức năng "Count Apples"được dùng để giám sát số lượng táo đã được phân loại (số lương táo đỏ và táo xanh) và chức năng "On/Off stepper"được dùng để điều khiển bật tắt stepper mà không cần quan tâm đến kết quả dự đoán, chức năng còn lại là "Apple classiication"do stepper thực hiện, đây là chức năng mà stepper thực hiện gạt trái cây vào giỏ thích hợp.



Hình 2.2: Sơ đồ Use case ESP32Client

3 Activity Diagram

Hình 3.1 thể hiện Activity Diagram mô tả luồng hoạt động của hệ thống.



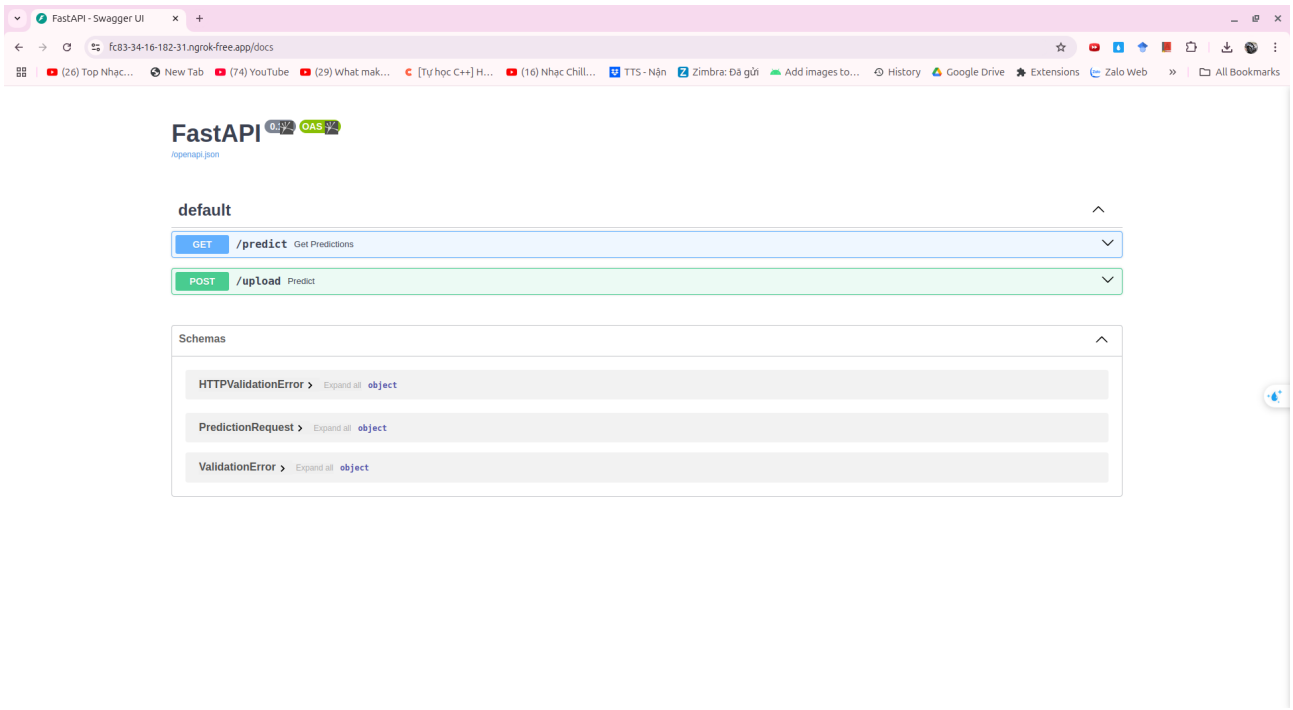
Hình 3.1: Activity diagram

Khi hệ thống hoạt động, các bộ xử lý ESP32 và ESP32CAM sẽ kết nối Internet, lúc này SERVER, Predict System và Blynk đều đã hoạt động, để có thể miêu tả cách hệ thống hoạt động, ta sẽ miêu tả nó từ đầu vào đến đầu ra của hệ thống, được chia thành 2 phần: Prediction System và ESP32Client System. Đầu tiên đối với Predict System, đầu vào của hệ thống sẽ được ESP32CAM (lúc này đóng vai trò như một server) chụp lại, hình ảnh được chụp lại sẽ được gửi đến clients khi nhận được request bao gồm predict system và blynk, hình ảnh được gửi đến blynk sẽ được stream để giám sát hệ thống, còn đối với hình ảnh được gửi đến predict system, khi predict system request đến ESP32CAM-SERVER, hình ảnh sẽ được gửi đến predict system, tại đây ta sẽ bắt đầu đọc hình ảnh dưới dạng byte, sau đó sử dụng các thư viện bao gồm numpy, pillow, opencv và pytorch (sử dụng Python) với model đã được huấn luyện từ trước để phân loại đầu vào là "Background", "GreenApple"hay "RedApple", sau đó kết quả dự đoán được gửi đến SERVER, lúc này SERVER sẽ nhận kết quả dự đoán và lưu nó vào danh sách lưu trữ, sau khi lưu trữ xong thì SERVER sẽ phản hồi lại predict system rằng kết quả dự đoán được gửi đến SERVER thành công hay không.

Đối với ESP32Client System, ta có thể miêu ta cách hệ thống này hoạt động như sau: Đầu tiên ESP32 sẽ gửi request đến SERVER, lúc này SERVER sẽ truy xuất danh sách kết quả dự đoán và lấy kết quả cuối cùng, sau đó phản hồi lại cho ESP32, ESP32 sẽ nhận dữ liệu dự đoán này để thực hiện 2 công việc, thức nhất sử dụng kết quả dự đoán để điều khiển stepper phân loại trái cây, công việc thứ hai là đếm số lượng táo xanh và táo đỏ và gửi dữ liệu này đến Blynk app để giám sát số lượng táo trong mỗi giỏ. Tại Blynk, ta có thể thực hiện điều khiển stepper bằng các switch, tín hiệu điều khiển được gửi đến ESP32. Sau khi thực hiện phân loại táo, quá trình trên được lặp lại tạo thành một hệ thống khép kín và hoàn chỉnh.

4 Prediction system

Cấu trúc SERVER được mô tả tại hình 4.1, với 2 phương thức là GET (/predict) và POST(/upload), phương thức /upload được dùng để nhận dữ liệu từ Predict System và lưu trữ dữ liệu đó, phương thức /predict được dùng để truy xuất dữ liệu từ danh sách lưu trữ và gửi dữ liệu đó đến ESP32 khi nhận được request. Công cụ được sử dụng để xây dựng SERVER là FastAPI framework và SERVER được deploy bằng Ngrok.



Hình 4.1: Cấu trúc SERVER.

ESP32CAM-SERVER được thiết lập tại 2 port: 80 và 81, cổng 80 được dùng để liên kết với Predict System, cổng 81 được liên kết với Blynk app.

```
WebServer captureServer(80);    // Server for predict system
WebServer streamServer(81);     // Server for streaming on Blynk app
....
void captureTask(void *pvParameters) {
  captureServer.on("/cam-hi.jpg", handleJpgHi);
  captureServer.begin();
  for (;;) {
    captureServer.handleClient();
  }
}

void streamTask(void *pvParameters) {
  streamServer.on("/monitor", []() {
    handleMjpeg();
  });
  streamServer.begin();
  for (;;) {
    streamServer.handleClient();
  }
}
....
void setup() {
    ...
```

```
    Serial.print("Capture server: http://");
    Serial.print(WiFi.localIP());
    Serial.println("/cam-hi.jpg");

    Serial.print("Stream server: http://");
    Serial.print(WiFi.localIP());
    Serial.println(":81/monitor");

    xTaskCreatePinnedToCore(captureTask, "Capture Task", 4096, NULL, 1, NULL, 0);  // Core 0
    xTaskCreatePinnedToCore(streamTask, "Stream Task", 4096, NULL, 1, NULL, 1);    // Core 1
}
```

Nhóm đã thực hiện huấn luyện 3 model dùng để dự đoán loại táo, kết quả được liệt kê tại bảng 1, dựa vào kết quả kiểm tra, nhóm đã chọn model MobileNetV2 để làm model phân loại táo cho hệ thống.

Name model	Accuracy Testing (%)	Accuracy Predict (%)	Time Predict (s)	Size (MB)
ResNet50	98.660714	97.001250	0.203992	89
ResNet18	97.321429	99.881893	0.178116	42
MobileNetV2	98.660714	99.923056	0.170975	8

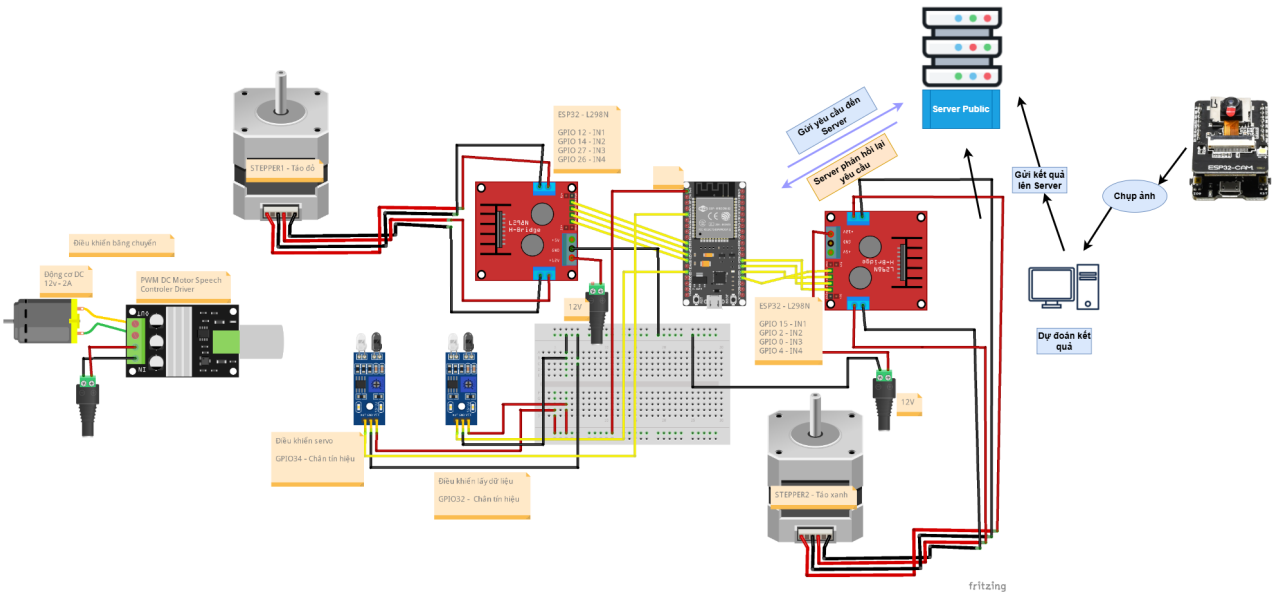
Bảng 1: Đánh giá các mô hình AI.

Dữ liệu đầu vào được thực hiện dự đoán tại Predict System và gửi đến SERVER

```python
# Python
def predict():
    while True:
        try:
            # Request image from the camera feed
            img_resp = urllib.request.urlopen(cam_url)
            # Read image
            imgnp = np.array(bytearray(img_resp.read()), dtype=np.uint8)
            im = cv2.imdecode(imgnp, -1)
            # Try predict
            pre = model.make_predict(im)
            print(f"Predict: {pre}")
            # Prepare the data to send
            data = {
                "prediction": pre,
                "status": "200"
            }
            # Make a POST request to send the prediction to the SERVER
            response = requests.post(f'{url_server}', json=data, timeout=10)
            if response.status_code == 200:
                print("Prediction successfully posted!")
            else:
                print(f"Error posting prediction: {response.text}")
        except Exception as e:
            print(f"Error in predict function: {e}")
```

5 Sơ đồ kết nối phần cứng

Hình 5.1 biểu diễn sơ đồ phần cứng và kết nối chân của hệ thống, miêu tả chi tiết về vai trò và chức năng của từng linh kiện trong hệ thống được thể hiện tại bảng 2.



Hình 5.1: Sơ đồ chi tiết kết nối phần cứng

STT	**Số lượng**	**Tên**	**Tính năng, nhiệm vụ**	**Kết nối**
1	1	ESP32	Hệ thống vi điều khiển (MCU) có khả năng kết nối Wifi. ESP32 sẽ gửi yêu cầu đến server để nhận dự đoán từ server và điều khiển các stepper hoạt động dựa theo dữ liệu nhận được.	Được cấp nguồn 5V, cung cấp nguồn 3.3V và tín hiệu đến 2 thiết bị L298N, 2 cảm biến hồng ngoại.
2	1	ESP32-CAM	Là một thiết bị giám sát video sử dụng chip ESP32 và camera OV2640. Thiết bị này có thể kết nối với Wi-Fi và phát trực tuyến video qua mạng internet. ESP32-CAM sẽ liên tục chụp những hình ảnh của táo trên băng chuyền gửi về cho hệ thống thực hiện công việc dự đoán.	Cấp nguồn 5V.
3	2	L298N	Là một vi điều khiển tốc độ (Motor Driver) được sử dụng để điều khiển các stepper.	Cấp nguồn riêng 5V và nối đất. Kết nối với stepper1 thông qua GPIO 12, 14, 27, 26 với IN1, 2, 3, 4. Kết nối với stepper2 thông qua GPIO 15, 2, 0, 4 với IN1, 2, 3, 4.
4	2	Stepper STP-43D1046	Là một loại động cơ bước điện tử được thiết kế để di chuyển liên tục với bước nhảy cố định. Stepper sẽ quay các thanh trục để đẩy những quả táo ra khỏi băng chuyền để vào nơi chỉ định.	Cấp nguồn riêng 5V và nối chung đất nới ESP32. Stepper1 và 2 nhận tín hiệu thông qua L298N qua OUT1, 2, 3, 4 từ ESP32.
5	2	Cảm biến vật cản hồng ngoại	Cảm biến vật cản hồng ngoại dùng để phát đối tượng đi qua bằng cách phát ra tia hồng ngoại và đo sự thay đổi khi tia phản xạ trở lại.	Cấp nguồn 3.3V và nối chung đất với ESP32. Cảm biến 1 và 2 lấy dữ liệu từ GPIO 32 và GPIO 34.
6	1	PWM DC Motor Speed Controller	Điều chỉnh tốc độ động cơ DC để điều khiển tốc độ băng chuyền.	Cấp nguồn 12V-2A.

Bảng 2: Bảng dữ liệu phần cứng

6 Ứng dụng FreeRTOS đối với ESP32

6.1 Các đoạn code quan trọng

Dưới đây là các đoạn code mà nhóm đã sử dung FreeRTOS, bao gồm: Queue, Task, Semaphore.

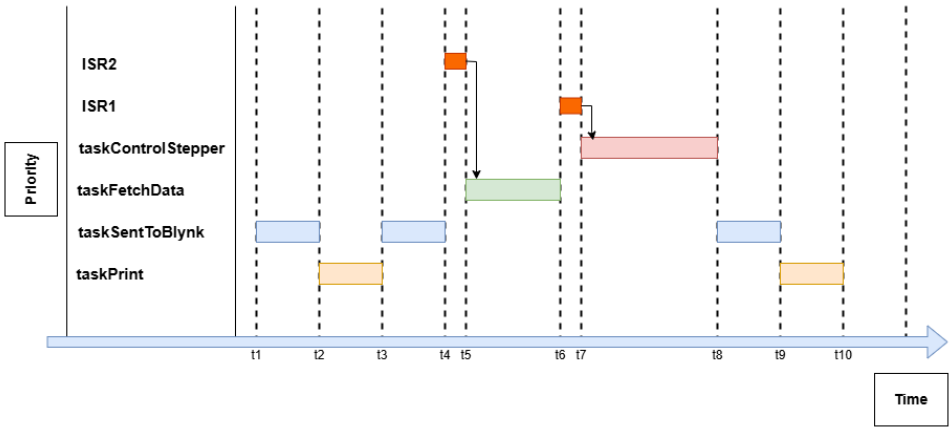
```
// declare semaphore
SemaphoreHandle_t cb1Semaphore;
SemaphoreHandle_t cb3Semaphore;
//setup
cb1Semaphore = xSemaphoreCreateBinary();
cb3Semaphore = xSemaphoreCreateBinary();
// establish
attachInterrupt(digitalPinToInterrupt(cb1Pin),
                debounceInterrupt1, FALLING);
attachInterrupt(digitalPinToInterrupt(cb3Pin),
                debounceInterrupt3, FALLING);
// apply
xSemaphoreGiveFromISR(cb1Semaphore, NULL);
xSemaphoreGiveFromISR(cb3Semaphore, NULL);
(xSemaphoreTake(cb3Semaphore, portMAX_DELAY) == pdTRUE)
(xSemaphoreTake(cb1Semaphore, portMAX_DELAY) == pdTRUE)
```

```
// declare task
void taskFetchData(void *parameter);
void taskControlStepper(void *parameter);
void taskSentToBlynk(void *parameter);
void taskPrint(void *parameter);
// initial
 xTaskCreatePinnedToCore(taskFetchData, "Fetch Server Data", 4096,
                         NULL, 3, NULL, 1);
 xTaskCreatePinnedToCore(taskControlServo, "Control Stepper", 2048,
                         NULL, 4, NULL, 1);
 xTaskCreatePinnedToCore(taskSentToBlynk, "sent to blynk", 4096,
                         NULL, 2, NULL, 1);
 xTaskCreatePinnedToCore(taskPrint, "Print Value", 2048,
                         NULL, 1, NULL, 1);
```

```
// declare queue
QueueHandle_t serverQueue;
//setup
serverQueue = xQueueCreate(5, sizeof(String));
//apply
xQueueSend(serverQueue, &data, portMAX_DELAY);
(xQueueReceive(serverQueue, &serverData, portMAX_DELAY) == pdTRUE)
```

6.2 Biểu đồ thời gian

Hình 6.1 biểu diễn quá trình thực thi của các task với độ ưu tiên và thời gian hoạt động khác nhau. Có tất cả 4 task đó là 'taskPrint', 'taskSentToBlynk', 'taskFtchData', 'taskControlStepper' có độ ưu tiên lần lượt từ thấp lên cao. Có nghĩa là 'taskPrint' sẽ có độ ưu tiên thấp nhất là 1 và 'taskControlSetpper' có độ ưu tiên cao nhất là 4. Ngoài ra còn có 2 sự kiện ngắt ISR1 và ISR2.
Trong đó, 'taskPrint' có nhiệm vụ hiển thị kết quả số lượng táo xanh và táo đỏ trên Monitor. 'taskSentToBlynk' gửi kết quả lên App Blynk và hiển thị. 'taskFetchData' lấy kết quả dự đoán từ server và 'taskControlStepper' dùng để điều khiển đóng mở 2 con stepper dựa vào kết quả dự đoán mà server trả về. ISR1 và ISR2 cấp Semaphore cho 'taskControlStepper' và 'taskFetchData'.

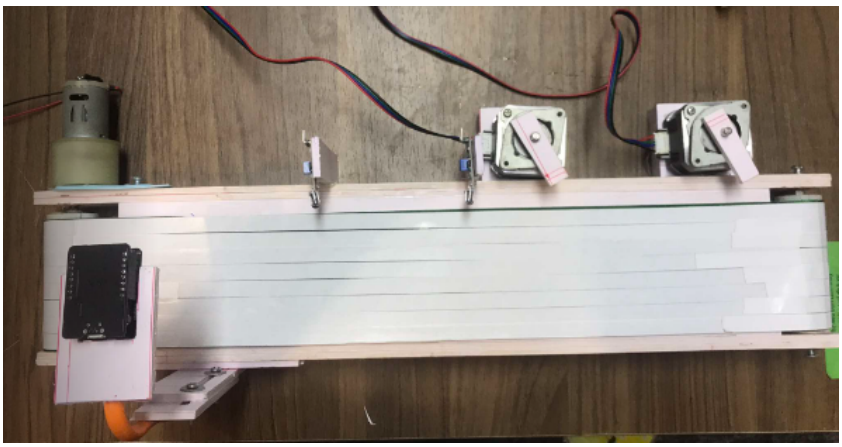


Hình 6.1: Biểu đồ thể hiện thời gian thực thi của các task với esp32.

Ta có, 'taskFetchData' và 'taskControlStepper' luôn được đặt ở trạng thái Blocked chờ nhận Semaphore từ 2 sự kiện ngắt ISR1, ISR2 để có thể được thực thi. Do đó, khi hệ thống hoạt động, 'taskSentToBlynk' với độ ưu tiên là 2 sẽ được thực thi trước và cho đến khi gặp 'vTaskDelay', task rơi vào trạng thái Blocked, sẽ bị dừng lại và không được CPU cấp thời gian thực thi cho đến khi hết thời gian delay.Lúc này, 'taskPrint' (có độ ưu tiên thấp hơn) sẽ được nhường tài nguyên để thực thi. Và Khi thời gian delay kết thúc, taskSentToBlynk sẽ chuyển từ trạng thái Blocked sang trạng thái Ready và chờ được CPU cấp quyền thực thi. Khi tín hiệu ISR2 được bật và cấp tín hiệu Semaphore cho 'taskFetchData'. Lúc này, 'taskFetchData' nhận được Semaphore, task sẽ chuyển từ trạng thái Blocked sang trạng thái Ready và được thực thi ngay lập tức bởi vì lúc này, task có độ ưu tiên là cao nhất. Sau khi chạy xong, task sẽ tiếp tục rơi vào trạng thái Blocked chờ nhận Semaphore. Đối với 'taskControlStepper' cũng tương tự như vậy. Khi có tín hiệu ngắt ISR1, task nhận được Semaphore và thực thi ngay. Sau khi chạy xong, task sẽ tiếp tục rơi vào trạng thái Blocked chờ nhận Semaphore. Khi đó, 'taskSenToBlynk' có độ ưu tiên cao nhất và tiếp tục được thực thi. Và quá trình hoạt động của hệ thống sẽ được lặp đi lặp lại như thế.

7 Biểu diễn kết quả

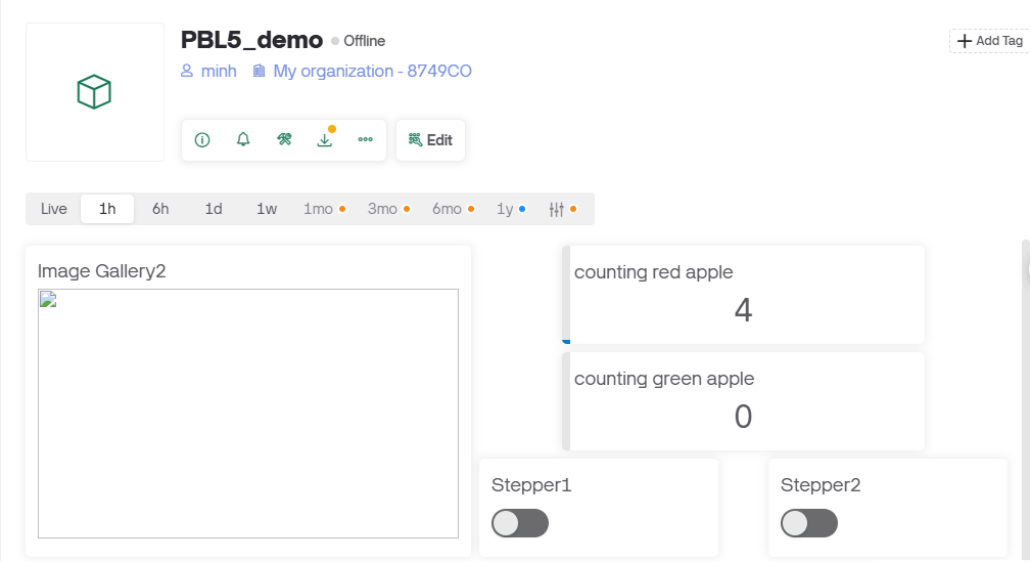
SERVER được build như hình 4.1. Hình 7.1 là mô hình băng chuyền phân loại mà nhóm đã lắp ráp được (bản demo).



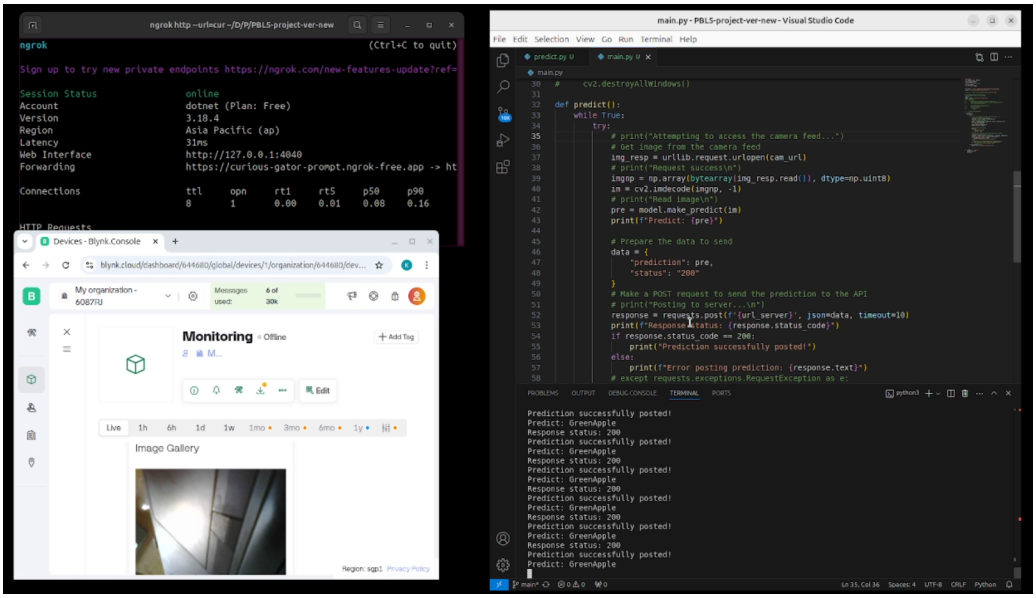
Hình 7.1: Demo mô hình băng chuyền phân loại

Hình 7.2 là một App Blynk được nhóm xây dựng với các nhiệm vụ như: Hiển thị video hình ảnh từ ESP32Cam,

hiển thị số lượng táo xanh và táo đỏ. Ngoài ra còn có 2 nút điều khiển đóng mở Stepper (để điều chỉnh khi cần thiết, phòng khi có sự cố về việc dự đoán, hay các con cảm biến không hoạt động).



Hình 7.2: Demo App Blynk



Hình 7.3: Thử nghiệm hệ thống

Hình 7.3 biểu diễn thử nghiệm hoạt động của hệ thống, với Blynk App dùng để giám sát hệ thống, Predict System dự đoán và Post đến Server.

Tài liệu tham khảo

[1] https://github.com/yoursunny/esp32cam

Phụ lục

[1] Group's github repository: https://github.com/HODUCVU/Automatic-fruit-sorting-conveyor-system.git

[2] Drive: https://drive.google.com/drive/folders/1rNSxV73FJjdwqtn1A5vkjFCKTb-1W1Sy?usp=drive_link

[3] FastAPI Tutorial: https://fastapi.tiangolo.com/tutorial/

[4] https://pytorch.org/

[5] Deploy IP Address: https://ngrok.com/