# MỤC LỤC

# DANH MỤC HÌNH ẢNH

# DANH MỤC BẢNG

# DANH MỤC CÁC TỪ VIẾT TẮT

| | | |
|---|---|---|
| CAN | Controller Area Network | Chuẩn truyền thông mạng |
| COM | Common | Tiếp điểm chung |
| DAC | Digital-to-Analog | Bộ chuyển đổi tín hiệu số sang tương tự |
| GND | Ground | Nối đất |
| GPIO | General Purpose Input/Output | Đầu vào/ra đa năng trên vi điều khiển |
| I2C | Inter-Integrated Circuit | Vi mạch tích hợp truyền thông nối tiếp |
| IC | Integrated Circuit | Mạch tích hợp |
| IoT | Internet of Things | Internet vạn vật |
| NC | Normally Closed | Thường đóng |
| NO | Normally Open | Thường mở |
| PWM | Pulse Width Modulation | Điều chế độ rộng xung |
| RAM | Random Access Memory | Bộ nhớ truy cập ngẫu nhiên |
| RX | Receive | Chân nhận |
| TX | Transmit | Chân truyền |
| UART | Universal Asynchronous Receiver / Transmitter | Truyền dữ liệu nối tiếp bất đồng bộ |
| VCC | Voltage Common Collector | Điện áp cung cấp |

# TÓM TẮT

Trong giai đoạn phát triển nông nghiệp hiện nay, việc áp dụng công nghệ vào sản xuất ngày càng đóng vai trò quan trọng khi Việt Nam đang phát triển mô hình nông nghiệp sử dụng công nghệ hiện đại. Một trong những điểm quan trọng quyết định năng suất và chất lượng cây trồng chính là tình trạng của đất. Tuy nhiên, các phương pháp kiểm tra chất lượng đất hiện nay vẫn còn thủ công, hiệu quả chưa cao và thiếu tính cập nhật theo thời gian thực.

Xuất phát từ những lý do đó, chúng tôi đã quyết định chọn thực hiện đề tài: **"Ứng dụng IoT và Machine Learning trong giám sát môi trường đất và đề xuất cây trồng phù hợp"**. Dự án được thiết kế tập trung vào vi điều khiển ESP32 và tích hợp các cảm biến chuyên dụng nhằm đo lường các chỉ số môi trường đất như nhiệt độ và độ ẩm đất, các thông số dinh dưỡng trong đất như là hàm lượng NPK, độ pH. Các dữ liệu cảm biến đo được hiển thị trực tiếp trên màn hình TFT và được gửi về hệ thống trung tâm thông qua giao thức MQTT để lưu trữ và được giám sát trên Website. Ngoài ra, nhóm còn sử dụng thuật toán học máy (Machine Learning) để phân tích dữ liệu đầu vào và cho ra kết quả dự đoán cây trồng thích hợp với điều kiện đất hiện tại. Đề tài không chỉ cho phép theo dõi các thông số dinh dưỡng của môi trường đất một cách trực quan và liên tục, mà còn giúp họ đưa ra quyết định trồng trọt hợp lý, khoa học hơn.

# CHƯƠNG 1: TỔNG QUAN

## 1.1. ĐẶT VẤN ĐỀ

Nước ta hiện nay đang không ngừng phát triển kết hợp với quá trình chuyển đổi số diễn ra trên toàn quốc. Điều này đã thúc đẩy ngành nông nghiệp nước nhà không ngừng cải tiến theo hướng hiện đại, công nghiệp hóa, áp dụng các giải pháp thông minh và phát triển mạnh mẽ. Trong xu thế hiện nay, việc áp dụng những tiến bộ khoa học kỹ thuật tiên tiến, nổi bật như là hệ thống Internet of Things (IoT) và trí tuệ nhân tạo (AI) vào sản xuất nông nghiệp đã là nhu cầu thiết yếu. Việc sử dụng công nghệ giúp tăng năng suất canh tác, cắt giảm chi phí sản xuất, giảm thiểu các rủi ro tiềm ẩn.

Trong quá trình sản suất nông nghiệp thì chất lượng đất đóng vai trò quan trọng trong việc quyết định năng suất và chất lượng cây trồng. Những chỉ số quan trọng như là độ ẩm, nhiệt độ, độ pH, hàm lượng chất dinh dưỡng (NPK) và lượng mưa. Tuy nhiên, hiện nay việc đánh giá và theo dõi các chỉ số đất vẫn chủ yếu thực hiện thủ công hoặc dựa vào kinh nghiệm, gây tốn thời gian và thiếu chính xác.

Vì những lý do mà nhóm chúng tôi quyết định thực hiện đề tài: **"Ứng dụng IoT và Machine Learning trong giám sát môi trường đất và đề xuất cây trồng phù hợp"**. Đề tài tập trung phát triển một hệ thống có chức năng chính là theo dõi chất lượng đất theo thời gian thực, sử dụng ESP32 là vi điều khiển chính và các cảm biến để đo lường các thông số quan trọng của đất. Dữ liệu mà cảm biến thu thập được sẽ truyền về máy chủ qua giao thức MQTT để hiển thị lên Website và lưu trữ phục vụ phân tích. Hệ thống còn ứng dụng Machine Learning cho việc xử lý dữ liệu và khuyến nghị cây trồng phù hợp với đất trồng hiện tại. Việc xây dựng hệ thống không chỉ giúp chúng ta dễ dàng theo dõi tình trạng đất canh tác mà còn hỗ trợ đưa ra các quyết định sản xuất chính xác, góp phần nâng cao năng suất, tiết kiệm tài nguyên và thúc đẩy quá trình hiện đại hóa nông nghiệp theo định hướng thông minh và bền vững.

## 1.2. MỤC TIÊU

Nhóm đặt ra mục tiêu nghiên cứu và tìm hiểu về vi điều khiển ESP32, các cảm biến đo chất lượng đất như độ ẩm đất, nhiệt độ, pH, hàm lượng NPK, cùng các module điều khiển thiết bị ngoại vi. Sau đó thiết kế và xây dựng hệ thống theo dõi môi trường đất ứng dụng IoT, có chức năng thu thập dữ liệu theo thời gian thực và điều khiển thiết bị.

Đề tài còn tích hợp Machine Learning nhằm phân tích dữ liệu cảm biến, phát hiện sớm các điều kiện bất lợi và đề xuất cây trồng phù hợp. Hệ thống hướng đến mục tiêu hỗ trợ người dùng trong việc nâng cao hiệu quả, tối ưu hóa tài nguyên và tăng năng suất cây trồng.

## 1.3. NỘI DUNG NGHIÊN CỨU

- Nội dung 1: Nghiên cứu lý thuyết về vi điều khiển ESP32 DEVKIT V1, cảm biến đo đạc thông số đất 7 trong 1 (ES-SOIL-7-IN-1), mạch chuyển đổi RS485 qua chuẩn RS485 to TTL.
- Nội dung 2: Tìm hiểu và áp dụng Machine Learning trong việc phân tích dữ liệu từ cảm biến, dự đoán cây trồng phù hợp. Từ đó sẽ thiết kế, xây dựng mô hình thực tế, tích hợp các thiết bị ngoại vi với vi điều khiển ESP32.
- Nội dung 3: Xây dựng lưu đồ thuật toán và phát triển phần mềm điều khiển cho ESP32. Thiết kế giao diện người dùng để theo dõi và điều khiển hệ thống.
- Nội dung 4: Tiến hành kiểm thử, hiệu chỉnh phần cứng và phần mềm để tăng hiệu suất hệ thống và giảm thiểu lỗi. Đối chiếu kết quả với dữ liệu thực tế, đánh giá kết quả thu được.
- Nội dung 5: Viết báo cáo thực hiện đồ án.
- Nội dung 6: Phân tích, đánh giá tổng thể kết quả đạt được.

## 1.4. GIỚI HẠN

- Mô hình lấy thông số lượng mưa bằng cách sử dụng API của các Website thời tiết thay vì cảm biến lượng mưa.
- Kích thước mô hình nhỏ.

- Hệ thống không có nguồn điện dự phòng.
- Mã nguồn chương trình chưa được tối ưu.

## 1.5. BỐ CỤC

Chương 1: *Tổng quan:* Phân tích tính cấp thiết của đề tài trong thời đại khoa học công nghệ phát triển và nhu cầu thực tiễn của ngành nông nghiệp hiện đại. Trình bày các xu hướng công nghệ liên quan và lý do lựa chọn đề tài, đồng thời nêu rõ mục tiêu.

Chương 2: *Cơ sở lý thuyết:* Giới thiệu các giao thức, linh kiện phần cứng của hệ thống, chức năng và nguyên lý hoạt động của từng module. Trình bày các kiến thức nền tảng để làm cơ sở cho việc thiết kế mô hình hoàn chỉnh.

Chương 3: *Thiết kế và xây dựng hệ thống:* Trình bày quy trình thiết kế hệ thống dựa trên yêu cầu đề tài, bao gồm sơ đồ tổng quan và các cách thức xử lý dữ liệu được sử dụng. Mô tả chi tiết quá trình chế tạo mô hình sản phẩm.

Chương 4: *Kết quả thực hiện:* Đánh giá hiệu suất hệ thống dựa trên các kết quả thu được từ phần cứng và phần mềm. So sánh với mục tiêu đặt ra ban đầu.

Chương 5: *Kết luận và hướng phát triển:* Tổng hợp những kết quả đạt được và những hạn chế chưa giải quyết được của đề tài. Đề xuất hướng nghiên cứu để hoàn thiện và nâng cao hiệu quả hệ thống.

# CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

## 2.1. TỔNG QUAN VỀ THUẬT TOÁN LIGHTGBM

### 2.1.1. Giới thiệu

LightGBM (Light Gradient Boosting Machine) là một thư viện học máy được phát triển bởi Microsoft, dựa trên thuật toán Gradient Boosting. Mục tiêu chính của LightGBM là xử lý nhanh và hiệu quả các bộ dữ liệu lớn, phức tạp, vượt trội hơn một số thư viện khác như CatBoost hay XGBoost.

Khác với các phương pháp truyền thống xây dựng cây quyết định theo chiều tầng (level-wise), LightGBM áp dụng kỹ thuật xây dựng theo chiều lá (leaf-wise), nghĩa là tiếp tục phân tách tại nhánh có sai số lớn nhất. Cách tiếp cận này giúp mô hình nhanh chóng cải thiện độ chính xác và giảm sai số. Tuy nhiên, nếu không kiểm soát tốt, mô hình cũng rất dễ bị quá khớp (overfitting).

### 2.1.2. Nguyên lý hoạt động

LightGBM triển khai thuật toán Gradient Boosting thông qua các bước như sau:

**a. Khởi tạo mô hình ban đầu**

Mô hình khởi đầu bằng việc đưa ra một giá trị dự đoán sơ bộ, thường là giá trị trung bình của nhãn trong bài toán hồi quy hoặc xác suất phân phối trong trường hợp bài toán phân loại.

**b. Tính toán Gradient**

Ở mỗi vòng lặp, mô hình tính toán gradient của hàm mất mát nhằm xác định mức độ chênh lệch giữa giá trị dự đoán và giá trị thực tế. Dựa vào thông tin này, mô hình sẽ xây dựng cây kế tiếp.

**c. Xây dựng cây quyết định mới**

LightGBM áp dụng kỹ thuật histogram để nhóm các giá trị liên tục thành từng khoảng (bin), từ đó giúp giảm chi phí tính toán. Cụ thể như sau:

- Tìm điểm chia (split point) tốt nhất dựa trên gradient và thông tin tăng (information gain).
- Phát triển cây theo hướng leaf-wise( theo lá) – chia tiếp nhánh có gradient lỗi lớn nhất để giảm sai số nhanh chóng.[1]

**d. Cập nhật mô hình dự đoán**

Sau mỗi lần huấn luyện, kết quả từ cây mới sẽ được kết hợp với kết quả dự đoán hiện tại thông qua một hệ số gọi là tốc độ học (learning rate). Sử dụng hệ số này giúp điều chỉnh tác động của cây mới lên mô hình, nhằm đảm bảo quá trình học diễn ra từ từ và giảm nguy cơ mô hình bị quá khớp (overfitting).

**e. Lặp lại quá trình huấn luyện**

Việc huấn luyện được lặp lại nhiều lần, trong đó các cây quyết định mới liên tục được thêm vào. Quá trình này sẽ dừng lại khi số lượng cây đạt giới hạn thiết lập hoặc khi mô hình không còn đạt được sự cải thiện rõ ràng về hiệu suất.[1]

### 2.1.3. Ưu điểm và nhược điểm

**a. Ưu điểm**

- Tốc độ vượt trội: LightGBM sử dụng Histogram-based Learning, giúp tăng tốc độ huấn luyện nhanh hơn so với XGBoost và các phương pháp boosting khác.
- Tiêu thụ ít bộ nhớ: Nhờ vào cách chia tập dữ liệu thành các bin nhỏ, LightGBM giảm đáng kể dung lượng bộ nhớ cần thiết.
- Xử lý dữ liệu lớn tốt: Có thể xử lý hàng triệu điểm dữ liệu mà vẫn đảm bảo tốc độ huấn luyện nhanh.
- Hiệu suất cao: Thường cho ra kết quả dự đoán có độ chính xác tốt hơn so với các thuật toán boosting khác trên cùng một tập dữ liệu.
- Hỗ trợ dữ liệu phân loại (categorical data): Không cần mã hóa one-hot, LightGBM có thể xử lý trực tiếp các đặc trưng phân loại, tiết kiệm thời gian và tài nguyên.
- Khả năng mở rộng: Hỗ trợ tính toán song song và GPU, phù hợp với các hệ thống phân tán. [2]

**b. Nhược điểm**

- Dễ xảy ra hiện tượng overfitting: Nếu các tham số không được điều chỉnh hợp lý, mô hình có nguy cơ học quá sát dữ liệu huấn luyện, làm giảm khả năng tổng quát.
- Không phù hợp với tập dữ liệu nhỏ: LightGBM cho kết quả hiệu suất tốt hơn với tập dữ liệu lớn, nhưng với dữ liệu nhỏ, nó có thể không hiệu quả bằng một số thuật toán khác như Random Forest.
- Khó thiết lập tham số: Mô hình yêu cầu điều chỉnh nhiều tham số để đạt được hiệu suất tối ưu, đòi hỏi người dùng cần có kinh nghiệm và hiểu biết nhất định.

### 2.1.4. Các tham số quan trọng

Khi triển khai mô hình LightGBM, người dùng có thể tinh chỉnh nhiều tham số khác nhau nhằm tối ưu hiệu suất. Một số tham số quan trọng bao gồm:

- **objective:** Xác định loại bài toán cần giải quyết. Ví dụ, 'binary' được dùng cho phân loại nhị phân, 'multiclass' cho phân loại nhiều lớp và 'regression' cho các bài toán hồi quy.
- **metric:** Đại diện cho thước đo đánh giá mô hình. Chẳng hạn, 'binary_error' là một chỉ số dùng trong phân loại nhị phân.
- **num_leaves:** Số lượng lá trong mỗi cây quyết định. Tham số này ảnh hưởng trực tiếp đến độ phức tạp và khả năng học của mô hình.
- **learning_rate:** Tốc độ học, quyết định mức độ thay đổi của mô hình sau mỗi lần cập nhật.
- **feature_fraction:** Tỷ lệ đặc trưng được chọn ngẫu nhiên để huấn luyện trong mỗi cây, giúp tăng khả năng tổng quát.
- **bagging_fraction và bagging_freq:** Hai tham số này điều khiển quá trình bagging – tức chọn ngẫu nhiên dữ liệu huấn luyện để xây dựng mô hình, từ đó hỗ trợ giảm hiện tượng overfitting.[2]

### 2.1.5. Cách sử dụng

Các bước sử dụng mô hình LightGBM trong một bài toán học máy bao gồm:

- **Chuẩn bị dữ liệu:** Dữ liệu cần được chuyển đổi về dạng phù hợp như mảng Numpy, bảng pandas DataFrame, hoặc dữ liệu có cấu trúc tương tự.
- **Tạo Dataset cho LightGBM:** Cần khởi tạo một đối tượng Dataset từ dữ liệu ban đầu, LightGBM yêu cầu dữ liệu đầu vào phải ở định dạng đặc biệt để có thể tối ưu hóa quá trình huấn luyện.
- **Xây dựng và huấn luyện mô hình:** Lựa chọn các tham số phù hợp với mô hình như objective, metric, num_leaves, learning_rate,…và tiến hành huấn luyện.
- **Dự đoán và đánh giá mô hình:** Sau khi hoàn thành huấn luyện, sử dụng mô hình đã huấn luyện để thực hiện dự đoán và đánh giá hiệu quả của mô hình.[2]

## 2.2. CÁC GIAO THỨC VÀ CHUẨN TRUYỀN THÔNG

### 2.2.1. Giao thức truyền thông Modbus RTU

#### a. Giới thiệu

Modbus RTU là một trong những giao thức truyền thông phổ biến hiện nay trong lĩnh vực công nghiệp. Giao thức này nằm trong họ Modbus, bao gồm cả Modbus TCP và Modbus ASCII. Modbus RTU vận hành theo mô hình Master-Slave, trong đó tín hiệu được truyền thông qua các cổng vật lý như RS-232 hoặc RS-485. Nhờ sự linh hoạt trong triển khai, Modbus RTU được sử dụng rộng rãi trong các hệ thống như quản lý tòa nhà (BMS), tự động hóa công nghiệp, điện lực,…

#### b. Nguyên lý hoạt động

Trong mạng Modbus RTU, chỉ có duy nhất một thiết bị Master điều khiển việc giao tiếp trên bus, còn các thiết bị Slave chỉ được phép phản hồi khi được Master yêu cầu. Cơ chế này giúp tránh xung đột trong truyền dữ liệu và đảm bảo quá trình truyền thông được tuần tự, rõ ràng. Modbus RTU truyền dữ liệu ở dạng nhị phân (binary), giúp giảm dung lượng và tăng tốc độ xử lý.

Cấu trúc khung dữ liệu tiêu chuẩn trong Modbus RTU bao gồm: 1 byte địa chỉ, 1 byte mã hàm, nhiều byte dữ liệu tùy thuộc yêu cầu, và cuối cùng là 2 byte mã kiểm tra CRC.

Modbus RTU Frame Format

| Start | Address | Function | Data | CRC | End |
|---|---|---|---|---|---|
| ≥3.5 char | 8 bit | 8 bit | N * 8 bit | 16 bits | ≥3.5 char |

**Hình 2.1:** Cấu trúc của khung truyền Modbus RTU[11]

Chức năng và vai trò :

- Byte địa chỉ: Đây là byte cho biết địa chỉ của Slave mà Master muốn giao tiếp. Nó có nhiệm vụ xác định thiết bị sẽ gửi hoặc nhận dữ liệu. Địa chỉ này có thể nằm trong khoảng từ 0 đến 247.
- Byte mã hàm: Có chức năng xác định yêu cầu của Master muốn Slave thực hiện.
- Byte dữ liệu: Là vùng lưu trữ thông tin của dữ liệu, có thể bao gồm các thông tin như địa chỉ thanh ghi, số lượng thanh ghi, giá trị ghi,... tùy thuộc vào mã chức năng đã chỉ định. Đây là vùng quan trọng để thực hiện việc truyền và nhận dữ liệu giữa Master và Slave.
- Byte CRC: Là mã kiểm tra lỗi (Cyclic Redundancy Check). CRC được tính trên toàn bộ khung dữ liệu, trừ phần CRC này. Nhờ có CRC, thiết bị nhận có thể phát hiện lỗi trong quá trình truyền và bỏ qua gói tin lỗi.[11]

**c. Đặc điểm kỹ thuật của Modbus RTU**

- Tốc độ baud rate: Thường sử dụng các giá trị tiêu chuẩn như 9600, 19200, 38400, lên đến 115200 bps.
- Cấu hình khung truyền: 8 bit dữ liệu, 1 bit stop, không có bit chẵn/lẻ (hoặc có tùy cấu hình).
- Chiều truyền: Half-duplex (truyền và nhận trên cùng một cặp dây, nhưng không đồng thời).
- Độ dài truyền: Tối đa khoảng 256 bytes trong 1 khung.
- Khoảng cách truyền: Lên đến 1200 mét khi sử dụng giao tiếp vật lý RS485.

**Ưu điểm của Modbus RTU:**

- Đơn giản, dễ triển khai, không yêu cầu phần mềm phức tạp.
- Tương thích với nhiều thiết bị công nghiệp và các hệ thống nhúng.

- Có thể mở rộng mạng với nhiều slave (tối đa 247 thiết bị).
- Độ tin cậy cao, có kiểm tra lỗi CRC.
- Tối ưu cho truyền dữ liệu trong khoảng cách xa và môi trường nhiễu.[11]

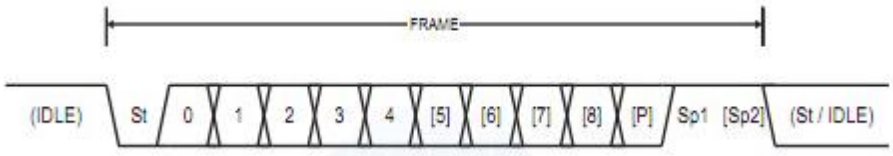
### 2.2.2. Chuẩn truyền thông UART

#### a. Giới thiệu

UART là một loại giao thức truyền thông nối tiếp, hoạt động theo chế độ không đồng bộ và được sử dụng phổ biến trong việc kết nối và trao đổi dữ liệu giữa các thiết bị điện tử. Đây là một trong những chuẩn giao tiếp cơ bản nhất, thường thấy trong các hệ thống kết nối với module không dây (Wi-Fi, Bluetooth), thiết bị đọc RFID hoặc các hệ thống nhúng và công nghiệp.

Hai thiết bị UART giao tiếp trực tiếp thông qua hai đường tín hiệu độc lập: Tx (Transmitter) và Rx (Receiver). Dữ liệu được chuyển đổi từ dạng song song sang nối tiếp khi truyền, và ngược lại khi nhận. Quá trình này không yêu cầu tín hiệu đồng hồ chung giữa hai thiết bị, mà sử dụng bit start để đánh dấu điểm bắt đầu frame dữ liệu và bit stop để xác định điểm kết thúc của mỗi gói dữ liệu, đảm bảo thiết bị nhận có thể phân biệt chính xác thời điểm cần bắt đầu và kết thúc việc đọc dữ liệu.

Tốc độ truyền dữ liệu là một yếu tố quan trọng. Tốc độ này được đo bằng đơn vị bit/giây (bps), phổ biến trong khoảng 9600-115200 bps. Để đảm bảo giao tiếp hiệu quả, cả hai thiết bị UART phải được cấu hình cùng một tốc độ truyền và cấu trúc gói dữ liệu tương tự nhau.




**Hình 2.2:** Khung dữ liệu truyền của UART[10]

***Chú thích:***

- St: Bit bắt đầu luôn ở mức thấp
- n: 8 bit dữ liệu

- P: Bit kiểm tra chẵn lẻ
- Sp: Bit kết thúc luôn ở mức cao

Trong giao tiếp UART, dữ liệu được truyền đi dưới dạng các gói, mỗi gói gồm 1 bit bắt đầu, 5 đến 9 bit dữ liệu, 1 bit chẵn lẻ tùy chọn, và 1 hoặc 2 bit kết thúc.

Bit bắt đầu (Start bit): Khi không thực hiện truyền dữ liệu, đường TX duy trì ở mức HIGH. Để bắt đầu truyền, UART kéo đường TX xuống mức LOW trong 1 chu kỳ baud, tạo sườn xuống giúp bộ nhận đồng bộ.

Khung dữ liệu: Chứa dữ liệu chính với chiều dài từ 5-9 bit (tùy cấu hình). Dữ liệu được truyền từ bit có trọng số thấp nhất (LSB) trước.

Bit kiểm tra chẵn lẻ: Phát hiện lỗi, có thể cấu hình chế độ chẵn hoặc lẻ. Nếu tổng số bit 1 không khớp với chế độ đã chọn, hệ thống sẽ phát hiện có lỗi truyền.

Bit kết thúc: Kết thúc frame bằng cách kéo đường TX lên mức cao trong 1-2 chu kỳ baud, báo hiệu hoàn tất truyền frame hiện tại.[10]

**b. Nguyên lý hoạt động**

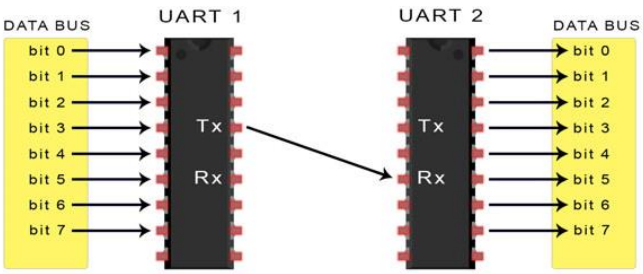
Giao tiếp UART hoạt động dựa trên quá trình truyền và nhận dữ liệu giữa các thiết bị thông qua hai chân Tx và Rx.

Ở bên truyền: Dữ liệu song song từ vi điều khiển được UART đóng gói thêm các bit điều khiển rồi chuyển thành tín hiệu nối tiếp phát qua chân TX.

Ở bên nhận: UART phát hiện start bit, lấy mẫu các bit dữ liệu, kiểm tra parity, loại bỏ các bit điều khiển rồi chuyển đổi về dạng song song cho vi điều khiển xử lý. Toàn bộ quá trình được đồng bộ bằng cùng tốc độ baud đã cấu hình trước.

UART hỗ trợ ba chế độ giao tiếp:

- Full duplex: Giao tiếp hai chiều diễn ra đồng thời giữa hai thiết bị.
- Half duplex: Dữ liệu được truyền theo một hướng tại mỗi thời điểm.
- Simplex: Giao tiếp chỉ diễn ra một chiều, không có chiều ngược lại.




**Hình 2.3:** Nguyên lý hoạt động của UART

### 2.2.3. Giao thức truyền thông MQTT

**a. Giới thiệu**

MQTT (Message Queuing Telemetry Transport) là giao thức nhắn tin tiêu chuẩn dùng để truyền dữ liệu giữa các thiết bị có băng thông thấp và tài nguyên hạn chế. Giao thức này hoạt động theo mô hình publish/subscribe (xuất bản/đăng ký), giúp hỗ trợ việc trao đổi thông tin giữa các thiết bị IoT (Internet of Things) và hệ thống Server.

Tác giả phát triển MQTT là Andy Stanford-Clark (IBM) và Arlen Nipper (Eurotech) vào năm 1999 với mục tiêu tạo ra một giao thức nhẹ, tiết kiệm băng thông và năng lượng để giám sát đường ống dầu khí qua vệ tinh. Năm 2013, MQTT được chuẩn hóa bởi tổ chức OASIS và trở thành một giao thức phổ biến trong lĩnh vực IoT.[12]

**b. Nguyên lý hoạt động**

MQTT hoạt động dựa trên mô hình máy chủ môi giới (broker), nơi các thiết bị máy khách (client) có thể gửi hoặc nhận dữ liệu thông qua các chủ đề (topics). Quá trình giao tiếp trong MQTT bao gồm các bước:

- Kết nối: Máy khách thiết lập kết nối với broker thông qua giao thức TCP/IP.
- Xác thực: Máy khách có thể xác thực bằng tên người dùng/mật khẩu hoặc chứng chỉ bảo mật.
- Giao tiếp: Máy khách có thể xuất bản (publish) dữ liệu lên một chủ đề hoặc đăng ký (subscribe) để nhận dữ liệu từ chủ đề đó.
- Kết thúc: Máy khách có thể ngắt kết nối khi không cần trao đổi dữ liệu nữa.

**c. Đặc điểm**

- Gọn nhẹ: Tiêu thụ ít tài nguyên, phù hợp với các thiết bị nhúng.

- Hiệu quả: Tối ưu hóa băng thông mạng, giúp truyền dữ liệu nhanh chóng.
- Mức độ tin cậy cao: Đa dạng nhiều khung chất lượng dịch vụ (QoS) để đảm bảo dữ liệu được truyền với tỉ lệ lỗi thấp nhất.
- Bảo mật: Có thể sử dụng SSL/TLS để mã hóa dữ liệu và xác thực thiết bị.[12]

**Ứng dụng của MQTT:**

- Nhà thông minh: Điều khiển và theo dõi thiết bị của bạn từ xa.
- Giám sát công nghiệp: Theo dõi máy móc và hệ thống sản xuất.
- Y tế: Truyền dữ liệu từ thiết bị đeo thông minh đến hệ thống phân tích.
- Giao thông: Quản lý dữ liệu thu thập được từ cảm biến trên xe và hệ thống giao thông thông minh.

## 2.3. GIỚI THIỆU PHẦN CỨNG

### 2.3.1. Cảm biến đất 7 trong 1

Cảm biến đất 7 trong 1 ES-SOIL-7-IN-1 (RS485 Modbus RTU) là thiết bị đo lường đa năng, có khả năng giám sát đồng thời nhiều thông số môi trường đất quan trọng như: nhiệt độ, độ ẩm đất, giá trị pH, độ dẫn điện của đất và chất dinh dưỡng NPK. Thiết bị có độ nhạy cao, phản hồi nhanh và đầu ra dữ liệu ổn định, phù hợp với nhiều loại đất khác nhau.

Cảm biến đo hằng số điện môi của đất, từ đó cung cấp dữ liệu chính xác về độ ẩm thực tế và phần trăm thể tích nước trong đất theo chuẩn quốc tế. Với thiết kế bền bỉ, thiết bị có thể được chôn trực tiếp trong đất mà không bị ảnh hưởng bởi hiện tượng ăn mòn hay điện phân, đảm bảo độ bền và độ chính xác trong thời gian lâu dài.

**Hình 2.4:** Cảm biến đất 7 trong 1

**Bảng 2.1:** Thông số của cảm biến đất 7 trong 1[3]

| | | | |
|---|---|---|---|
| Nguồn cung cấp | 12-24V DC | | |
| Tín hiệu đầu ra | RS485 | | |
| Nhiệt độ hoạt động | -20℃ ~ +60℃ | | |
| Áp suất hoạt động | 0.9-1.1atm | | |
| Thời gian phản hồi | <1s | | |
| | Phạm vi | Độ phân giải | Độ chính xác |
| Nhiệt độ | -40℃~80℃ | 0.1℃ | ±0.5℃ |
| Độ ẩm | 0-100% | 0.1% | ±3% (0-53%),<br>±5% (53-100%) |
| Độ dẫn điện EC | 0−10000us/cm | 10us/cm | ±5%FS |
| Độ PH | 3~9 pH | 0.01pH | ±0.3pH |
| NPK | 0-1999mg/kg | 1mg/kg | ±2%F.s |

### 2.3.2. ESP32 DEVKIT V1

ESP32 DEVKIT V1 là một vi điều khiển mạnh mẽ được phát triển bởi Espressif, tích hợp sẵn Wi-Fi và Bluetooth, với khả năng kết nối không dây ổn định nó thích hợp cho các ứng dụng IoT. Với khả năng xử lý mạnh mẽ và tiết kiệm năng lượng, ESP32 là khối xử lý trung tâm trong hệ thống, giúp điều khiển và giám sát các thiết bị, cảm biến và module relay thông qua giao thức Wi-Fi.

Điểm nổi bật đáng chú ý của ESP32 DEVKIT V1 là khả năng hỗ trợ nhiều chuẩn truyền thông như I2C, SPI, UART, và CAN, giúp giao tiếp linh hoạt với các cảm biến và thiết bị ngoại vi. Điều này làm tăng khả năng kết nối của hệ thống và dễ dàng tích hợp với các phần cứng khác. Bên cạnh đó, ESP32 hỗ trợ nhiều chân GPIO, cho phép người dùng kết nối và điều khiển các thiết bị ngoài như mô-đun relay, đèn LED, cảm biến và các thiết bị điện tử khác. ESP32 DEVKIT V1 có 34 chân GPIO có thể lập trình được, ESP32 DEVKIT V1 có đủ khả năng đáp ứng được nhu cầu của các ứng dụng yêu cầu số lượng kết nối lớn và điều khiển đồng thời nhiều thiết bị.

Với khả năng lập trình linh hoạt, hỗ trợ Wi-Fi, Bluetooth, nhiều chuẩn truyền thông, nhiều chân GPIO, và bộ nhớ Flash và RAM lớn, ESP32 DEVKIT V1 không chỉ giúp

theo dõi và điều khiển nhiều thiết bị từ xa mà còn mở rộng khả năng kết nối và bảo mật cho các hệ thống tự động hóa, ứng dụng IoT và các dự án điện tử phức tạp.[4]

**Hình 2.5:** ESP32 DEVKIT V1

**Bảng 2.2:** Thông số kỹ thuật của ESP32[4]

| | |
|---|---|
| Vi điều khiển | ESP32 |
| Bộ nhớ Flash | 16 MB |
| Bộ nhớ RAM | 520 KB RAM tích hợp, 8 KB SRAM cho dữ liệu và ngăn xếp, 32 KB SRAM bổ sung cho hệ thống |
| Kết nối không dây | Wi-Fi 802.11 b/g/n (2.4 GHz), Bluetooth v4.2 (BR/EDR và BLE) |
| GPIO | 34 chân GPIO (có thể lập trình với chức năng ADC, DAC, PWM, UART, SPI, I2C, CAN, Touch) |
| ADC | 15 chân ADC |
| DAC | 2 chân DAC |
| UART | 2 chân UARTs |
| SPI | Hỗ trợ giao thức SPI |
| I2C | Hỗ trợ giao thức I2C |
| PWM | Tạo tín hiệu PWM với độ phân giải 8 bit |
| CAN | Hỗ trợ giao thức CAN 2.0 |
| Điện áp hoạt động | 5V |
| Dòng tiêu thụ | Khoảng 240 mA |
| Kích thước | 54mm × 28mm |
| Cổng lập trình | Micro USB cấp nguồn vả nạp chương trình |
| Phần mềm hỗ trợ | Arduino IDE, Espressif IDF, PlatformIO |

### 2.3.3. Module relay 5V

Module relay 5V được sử dụng để điều khiển các thiết bị điện công suất lớn, như đèn hoặc động cơ quạt, bằng cách hoạt động như một công tắc điện. Trong hệ thống giám sát năng lượng, module này có chức năng ngắt điện các thiết bị khi có sự cố, chẳng

hạn như khi xảy ra quá áp, quá dòng, hoặc khi năng lượng tiêu thụ vượt ngưỡng giới hạn đã được cài đặt.

Module relay 5V có thể chịu được hiệu điện thế 250V và dòng 10A, không gây nguy hiểm cho thiết bị trong hệ thống. Khi bất kỳ giá trị như điện áp, dòng điện, công suất vượt quá ngưỡng an toàn, relay sẽ tự động ngắt điện, giúp bảo vệ thiết bị và tránh các rủi ro về sự cố điện.[5]

**Hình 2.6:** Module relay 5V

**Bảng 2.3:** Thông số kỹ thuật của Module relay 5V

| STT | Thông số | Giá trị | Đơn vị |
|---|---|---|---|
| **1** | Điện áp hoạt động | 5 | VDC |
| **2** | Dòng điện tiêu thụ | 80 | mA |
| **3** | Tiếp điểm | 250VAC – 10A<br>30VDC-10A | |

**Bảng 2.4:** Chức năng các chân của Module relay 5V

| STT | Chân | Chức năng |
|---|---|---|
| **1** | VCC | Cung cấp nguồn điện cho module relay. Kết nối với nguồn 5V. |
| **2** | GND | Nối đất. |
| **3** | IN | Đầu vào để kích mở relay. Khi nhận tín hiệu HIGH hoặc LOW, relay sẽ được kích hoạt. |
| **4** | H/L | Chân dùng để thiết lập mức điều khiển relay. Có thể chọn mức HIGH hoặc LOW để điều khiển việc mở/đóng relay. |

| | | |
|---|---|---|
| **5** | COM | Chân tiếp điểm chung. Đây là tiếp điểm trung gian của relay. Kết nối với thiết bị mà chúng ta muốn điều khiển (thường kết nối với nguồn điện). |
| **6** | NO | Chân thường mở. Khi relay được kích hoạt, tiếp điểm này sẽ nối với COM. |
| **7** | NC | Chân thường đóng. Khi relay không hoạt động, tiếp điểm này sẽ nối với COM. Khi relay kích hoạt, tiếp điểm này sẽ ngắt kết nối. |

### 2.3.4. Động cơ bơm 365

Động cơ máy bơm 365 12V có lưu lượng 3 lít/phút sử dụng điện áp 12VDC và có kích thước nhỏ gọn. Máy bơm khi hoạt động không gây quá ồn, mức âm thanh phát ra chỉ dưới 30db.

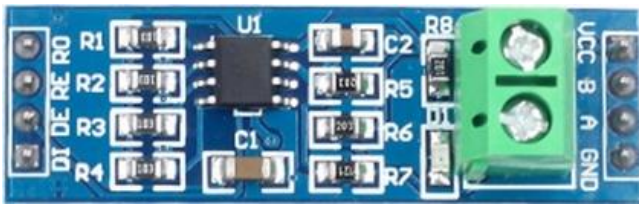
**Hình 2.7:** Động cơ bơm 12V

**Bảng 2.5:** Thông số kỹ thuật của động cơ bơm 12V[6]

| STT | Thông số | Giá trị | Đơn vị |
|---|---|---|---|
| **1** | Điện áp hoạt động | 12 | VDC |
| **2** | Dòng điện không tải | 0.23 | A |
| **3** | Lưu lượng | 2-3 | L/min |
| **4** | Áp suất đầu ra | 1-2.5 | kg |
| **5** | Khả năng đẩy cao | 2 | m |
| **6** | Độ sâu hút đạt được | 1-2.5 | m |
| **7** | Trọng lượng | 111 | g |

### 2.3.5. Mạch Chuyển Đổi RS485 To TTL

Mạch chuyển đổi RS485 to TTL là một module giao tiếp giúp chuyển đổi tín hiệu giữa chuẩn RS485 (chuẩn công nghiệp truyền thông nối tiếp) và tín hiệu TTL UART thường sử dụng trong các vi điều khiển như Arduino, STM32, ESP32,...Việc sử dụng mạch này giúp kết nối các vi điều khiển với thiết bị truyền thông sử dụng chuẩn RS485, vốn rất phổ biến trong các hệ thống công nghiệp nhờ khả năng truyền xa và chống nhiễu tốt.

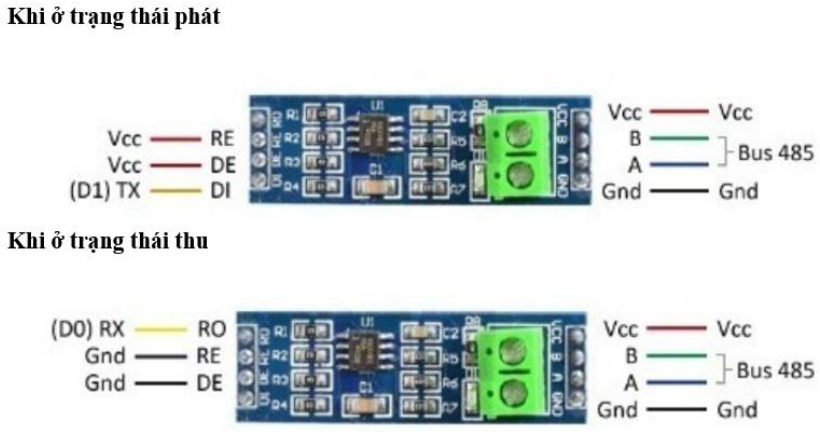



**Hình 2.8:** Mạch chuyển đổi RS485 to TTL

**Bảng 2.6:** Thông số kỹ thuật của mạch chuyển đổi RS485 to TTL[7]

| STT | Thông số | Giá trị | Đơn vị |
|---|---|---|---|
| **1** | Điện áp hoạt động | 3.3 - 5 | VDC |
| **2** | Dòng điện tiêu thụ | 10 | mA |
| **3** | Tốc độ truyền | 5 | Mbps |
| **4** | IC chính | MAX485 | |
| **5** | Khoảng cách truyền | 1200 | mét |

Sơ đồ chân Module RS485 to TTL:




**Hình 2.9:** Sơ đồ kết nối chân Mạch chuyển đổi RS485 to TTL ở trạng thái phát

- **A:** đầu vào bộ thu không đảo và đầu ra bộ phát không đảo.
- **B:** đầu vào bộ thu đảo và đầu ra bộ phát đảo.
- **RE (Receiver Output Enable – Kích hoạt đầu ra bộ thu):** cho phép module nhận dữ liệu từ đường truyền RS485 và xuất ra tín hiệu TTL khi ở mức HIGH., vô hiệu hóa đầu ra bộ thu khi ở mức LOW.
- **RO (Receiver Output – Đầu ra bộ thu):** Xuất tín hiệu TTL từ dữ liệu nhận được trên đường truyền RS485, chân này thường nối với chân RX của vi điều khiển.
- **DE (Driver Output Enable – Kích hoạt đầu ra bộ phát):** cho phép module truyền dữ liệu từ TTL sang RS485 khi ở mức HIGH, vô hiệu hóa bộ phát khi ở mức LOW.
- **DI (Driver Input – Đầu vào bộ phát):** nhận tín hiệu TTL từ vi điều khiển và truyền lên đường RS485, chân này thường nối với chân TX của vi điều khiển.

*Tóm lại:*

- Nếu muốn nhận dữ liệu từ RS485, đặt RE = 0, DE = 0 và đọc dữ liệu từ RO.
- Nếu muốn cần gửi dữ liệu lên RS485, đặt RE = 1, DE = 1 và truyền dữ liệu từ DI.[7]

### 2.3.6. Module LCD TFT ILI9341

Module LCD TFT ILI9341 là một màn hình hiển thị có kết cấu dạng tinh thể lỏng (LCD) có nhiều loại trên thị trường, phổ biến nhất là loại 2.4 inch, 2.8 inch, 3.2 inch, 3.5 inch, 4.3 inch. Module này sử dụng chip điều khiển ILI9341 của hãng ITE, hỗ trợ hiển thị màu sắc 16 bit (65.536 màu) và độ phân giải cao (lên đến 320 x 240 pixel).


**Hình 2.10:** Màn hình LCD TFT ILI9341

**Bảng 2.7:** Thông số kỹ thuật của màn hình LCD TFT ILI9341[8]

| STT | Thông số | Giá trị | Đơn vị |
|---|---|---|---|
| **1** | Điện áp hoạt động | 3.3 - 5 | VDC |
| **2** | Dòng điện tiêu thụ | 120 | mA |
| **3** | Độ phân giải | 240 x 320 | pixels |
| **4** | IC chính | ILI9341 | |
| **5** | Tốc độ refresh | 60 | Hz |
| **6** | Kích thước | 2.4 | inch |
| **7** | Số màu hiển thị | 262.144 | màu |
| **8** | Kích thước toàn màn hình | 86 x 50 | mm |

**Bảng 2.8:** Chức năng của các chân LCD[8]

| Chân | Tên | Chức năng |
|---|---|---|
| 1 | VCC | Nguồn điện cung cấp (3.3V hoặc 5V) |
| 2 | GND | Ground |
| 3 | CS | Chip Select - Chọn chip |
| 4 | RESET | Đặt lại - Reset |
| 5 | DC (D/CX) | Data/Command - Chọn giữa dữ liệu và lệnh |
| 6 | MOSI (SDA) | Master Out Slave In - Dữ liệu đầu vào SPI |
| 7 | SCK (SCL) | Serial Clock - Xung clock SPI |
| 8 | LED | Đèn nền - Backlight (thường kéo lên VCC qua điện trở hoặc điều khiển bằng vi điều khiển) |
| 9 | MISO | Master In Slave Out - Dữ liệu đầu ra SPI (ít dùng, thường để trống hoặc dùng trong chế độ 2 chiều) |
| 10 | T_CLK | Touch Panel Clock - Xung clock cho cảm ứng (nếu module có cảm ứng) |
| 11 | T_CS | Touch Panel Chip Select - Chọn chip cho cảm ứng (nếu module có cảm ứng) |

| | | |
|---|---|---|
| 12 | T_DIN | Touch Panel Data In - Dữ liệu đầu vào cho cảm ứng (nếu module có cảm ứng) |
| 13 | T_DO | Touch Panel Data Out - Dữ liệu đầu ra cho cảm ứng (nếu module có cảm ứng) |
| 14 | T_IRQ | Touch Panel Interrupt - Ngắt cho cảm ứng (nếu module có cảm ứng) |

### 2.3.7. Động cơ giảm tốc DC JGB37-520 12V 66RPM

Động cơ giảm tốc DC là thiết bị một chiều có hộp số giảm tốc, thường được sử dụng trong các ứng dụng yêu cầu mô-men xoắn lớn và tốc độ quay thấp. Đây là thiết bị phổ biến được sử dụng trong các mô hình robot, băng chuyền mini, hệ thống tự động hóa,…

**Hình 2.11:** Động cơ giảm tốc JGB37-520

**Bảng 2.9:** Thông số kỹ thuật của động cơ giảm tốc JGB37-520[9]

| STT | Thông số | Giá trị | Đơn vị |
|---|---|---|---|
| **1** | Điện áp hoạt động | 12 | VDC |
| **2** | Dòng điện tiêu thụ không tải | 0.12 | A |
| **3** | Dòng điện tiêu thụ tải tối đa | 1 | A |
| **4** | Tỉ số truyền | 1:90 | |
| **5** | Đường kính trục | 6 | mm |
| **6** | Kích thước động cơ | 37 | mm |
| **7** | Trọng lượng | 200 | gam |

### 2.3.8. Một số linh kiện khác

**a. Led đơn**

**Hình 2.12:** Led đơn

**Bảng 2.10:** Thông số kỹ thuật của led đơn

| STT | Thông số | Giá trị | Đơn vị |
|---|---|---|---|
| **1** | Điện áp hoạt động | 3 | VDC |
| **2** | Dòng điện tiêu thụ | 10-20 | mA |
| **3** | Đường kính | 5 | mm |
| **4** | Chiều dài chân | >20 | mm |

**b. Nút nhấn**

Công tắc nhấn 4 chân B3F-4055 là linh kiện điện tử dạng công tắc nhỏ gọn, thường được sự dụng phổ biến trong các thiết bị điện tử.

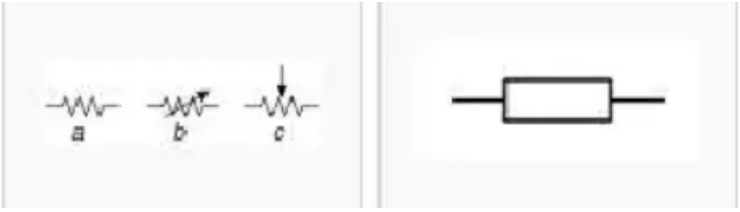
**Hình 2.13:** Nút nhấn

**Bảng 2.11:** Thông số kỹ thuật nút nhấn

| STT | Thông số | Giá trị | Đơn vị |
|---|---|---|---|
| **1** | Điện áp hoạt động | 5 | VDC |
| **2** | Dòng chịu tải | 50 | mA |
| **3** | Mức kích | Cao/Thấp | mm |

**c. Điện trở**

Điện trở hay Resistor là một linh kiện điện tử thụ động trong một mạch điện. Nhiệm vụ chính của nó là hạn chế dòng điện. Theo định luật Ohm, điện áp giữa hai đầu điện trở tỉ lệ thuận với dòng điện chạy qua nó.




**Hình 2.14:** Ký hiệu điện trở

**d. Tụ gốm**

Tụ gốm là loại tụ điện có giá trị điện dung cố định. Vật liệu cấu tạo nên tụ bao gồm chất điện môi là gốm, được xen kẽ giữa các lớp kim loại dẫn điện để tạo thành hai bản cực. Do không có cực tính rõ ràng, tụ gốm có thể lắp đặt theo bất kỳ chiều nào trong mạch mà không ảnh hưởng đến chức năng hoạt động.


**Hình 2.15:** Tụ gốm

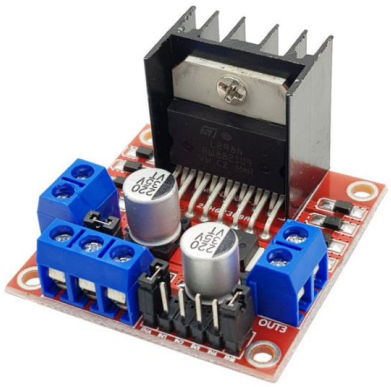
**Đặc điểm:**

- Dung sai chính xác.
- Kích thước nhỏ gọn.
- Công suất cao và điện áp cao.

**e. Mạch điều khiển động cơ DC L298N**

Là một module điều khiển động cơ thường được sử dụng trong các dự án như là robot, xe điều khiển từ xa và các thiết bị sử dụng vi điều khiển như Arduino hoặc ESP32. Mạch này sử dụng IC L298N, một mạch tích hợp dạng H-Bridge kép, cho phép điều khiển 2 động cơ DC riêng biệt hoặc 1 động cơ bước.

Ưu điểm chính của L298N là khả năng cho phép người dùng điều khiển chiều quay và tốc độ quay của động cơ thông qua 2 chân điều khiển logic (IN1, IN2 cho động cơ A; IN3, IN4 cho động cơ B).



**Hình 2.16:** Mạch điều khiển động cơ DC L298N

**f. Quạt tản nhiệt 12V**

Quạt tản nhiệt 12V là một trong những thiết bị quan trọng trong hệ thống làm mát, được sử dụng trong máy tính, thiết bị điện tử, tủ điện, biến tần và nhiều ứng dụng công nghiệp khác.


**Hình 2.17:** Quạt tản nhiệt 12V

**Bảng 2.12:** Thông số kỹ thuật của quạt tản nhiệt 12V

| STT | Thông số | Giá trị | Đơn vị |
|---|---|---|---|
| **1** | Điện áp hoạt động | 12 | VAC |
| **2** | Dòng điện tiêu thụ | 180 | mA |
| **3** | Kích thước | 5 x 5 | cm |
| **4** | Tiếng ồn | 18 | Db |
| **5** | Tham chiếu tốc độ | 4000 +/- 10% | RPM |

**g. Đèn 12V**

Đèn 12V là loại đèn có điện áp hoạt động thấp, được sử dụng nhiều trong điện dân dụng hoặc sử dụng nguồn năng lượng mặt trời.

**Hình 2.18:** Đèn 12V

**Bảng 2.13:** Thông số kỹ thuật của đèn 12V

| STT | Thông số | Giá trị | Đơn vị |
|---|---|---|---|
| **1** | Điện áp hoạt động | 12 | VAC |
| **2** | Công suất | 5 | W |
| **3** | Tuổi thọ | 30000 | h |

**h. Công tắc hành trình**

Công tắc hành trình (limit switch) là một thiết bị cơ điện tử dùng để phát hiện vị trí hoặc sự di chuyển của một đối tượng. Nó được sử dụng để giới hạn hành trình của các bộ phận cơ khí, đóng/mở mạch điện và chuyển đổi tín hiệu điện.

**Hình 2.19:** Công tắc hành trình

# CHƯƠNG 3: THIẾT KẾ VÀ XÂY DỰNG HỆ THỐNG

## 3.1. YÊU CẦU CỦA HỆ THỐNG

Hệ thống bao gồm phần cứng và phần mềm với các chức năng sau:

**Phần cứng:**

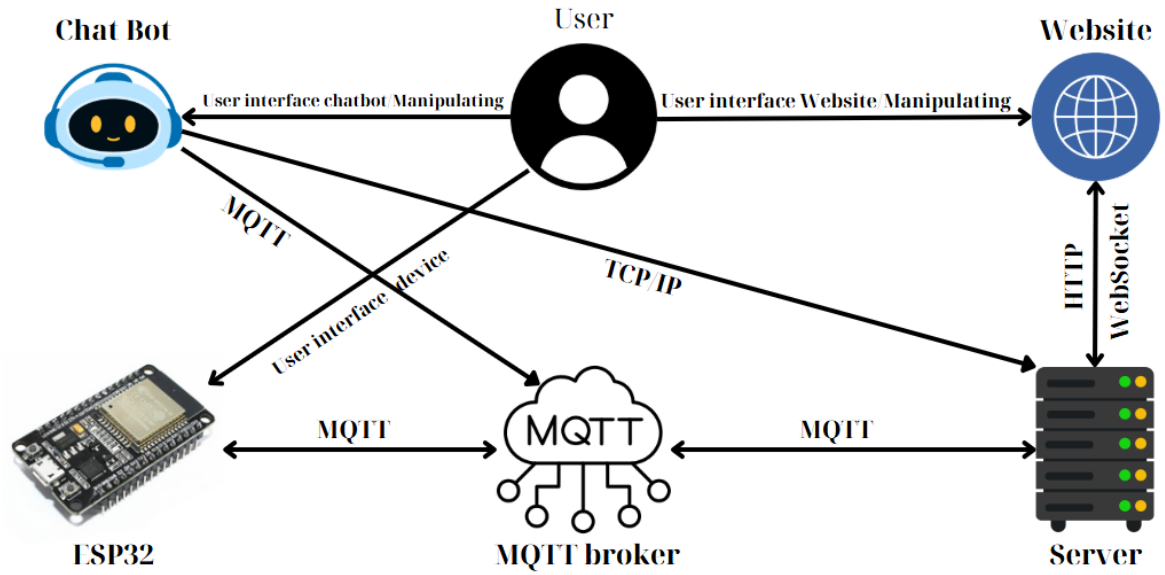
- Thu thập thông số nhiệt độ, độ ẩm, độ PH, hàm lượng NPK trong đất.
- Điều khiển các thiết bị đèn, quạt, máy bơm nước, mái che bằng nút nhấn.
- Hiển thị những thông số thu thập được và trạng thái thiết bị lên màn hình TFT.
- Gửi dữ liệu thu thập được và trạng thái thiết bị lên MQTT broker và nhận lệnh điều khiển thiết bị từ MQTT broker.

**Phần mềm:**

- Hiển thị dữ liệu nhiệt độ, độ ẩm, pH, hàm lượng NPK, lượng mưa hàng ngày và lượng mưa tháng trước theo thời gian thực.
- Hiển thị và cập nhật định kỳ dữ liệu trên biểu đồ.
- Điều khiển và hẹn giờ bật/tắt thiết bị thông qua giao diện.
- Đưa ra khuyến nghị cây trồng phù hợp dựa trên các thông số nitơ, phốt pho, kali, nhiệt độ, độ ẩm, pH và lượng mưa tháng trước.
- Đưa ra đề xuất điều chỉnh thông số hiện tại thay đổi phù hợp với loại cây mà người dùng muốn trồng.
- Chatbot có thể theo dõi dữ liệu và chức năng tương tự Website.

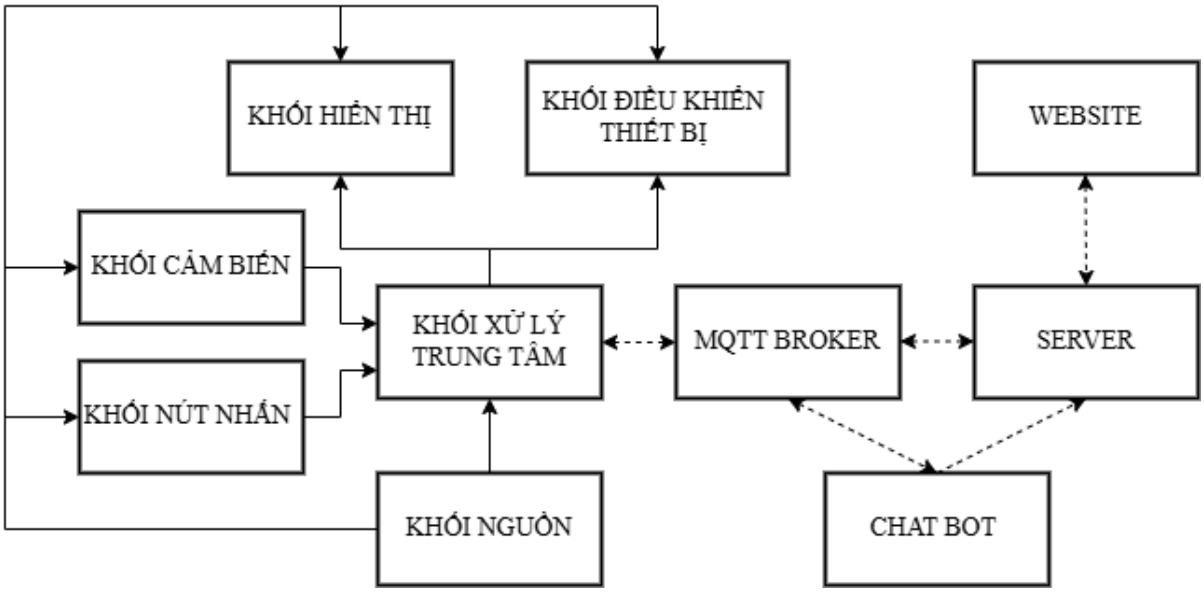
## 3.2. SƠ ĐỒ ĐẶC TẢ HỆ THỐNG

Hệ thống bao gồm thành phần chính: ChatBot, Website, ESP32, MQTT broker và server. Người dùng có thể tương tác với hệ thống thông qua giao diện Website, ChatBot hoặc thiết bị phần cứng. Các thành phần trong hệ thống sẽ giao tiếp với nhau thông qua các giao thức như MQTT, HTTP, Websocket và TCP/IP để thu thập dữ liệu, xử lý và phản hồi các yêu cầu của người sử dụng.




**Hình 3.1:** Sơ đồ đặc tả hệ thống

## 3.3. SƠ ĐỒ KHỐI HỆ THỐNG




**Hình 3.2:** Sơ đồ khối hệ thống

**Chức năng của từng khối:**

- **Khối cảm biến:** thu thập các thông số của đất và gửi về khối xử lý trung tâm.
- **Khối nút nhấn:** cung cấp tín hiệu thay đổi trạng thái thiết bị cho khối xử lý trung tâm.

- **Khối hiển thị:** hiển thị những thông số mà khối cảm biến thu thập được và trạng thái thiết bị do khối xử lý trung tâm gửi đến.
- **Khối điều khiển thiết bị:** nhận tín hiệu điều khiển từ khối xử lý trung tâm để thay đổi trạng thái các thiết bị.
- **Khối xử lý trung tâm:** nhận dữ liệu từ khối cảm biến; gửi tín hiệu điều khiển thiết bị; gửi dữ liệu đo được từ khối cảm biến cùng với trạng thái thiết bị đến khối hiển thị và MQTT broker; nhận lệnh điều khiển thiết bị từ MQTT broker.
- **Khối nguồn:** cấp nguồn cho các khối xử lý trung tâm, khối cảm biến, khối hiển thị và khối điều khiển thiết bị.
- **MQTT broker:** đóng vai trò là máy chủ trung gian, hỗ trợ giao tiếp giữa khối xử lý trung tâm, Server và ChatBot.
- **Server:** nhận dữ liệu từ MQTT broker và đồng thời gửi lệnh điều khiển đến MQTT broker, lưu và truy xuất cơ sở dữ liệu, hiển thị dữ liệu lên Website, xử lý dữ liệu từ Website như điều khiển thiết bị, khuyến nghị cây trồng,...
- **Website:** hiển thị giao diện người dùng, gửi yêu cầu REST API đến server để lấy dữ liệu và điều khiển thiết bị.
- **ChatBot:** truy vấn cơ sở dữ liệu từ server, nhận lệnh từ người dùng và đưa ra phản hồi, gửi lệnh điều khiển đến MQTT broker và theo dõi trạng thái thiết bị từ MQTT broker.

**Hoạt động của hệ thống:**

- **Bước 1:** Khi cấp nguồn, toàn bộ hệ thống sẽ được kích hoạt, khối Xử lý trung tâm sẽ khởi tạo khối hiển thị, cảm biến và thiết bị, kết nối Wifi và MQTT broker.
- **Bước 2:** Khối Xử lý trung tâm tiến hành đọc và xử lý dữ liệu từ khối cảm biến, hiển thị dữ liệu lên khối hiển thị và gửi lên MQTT broker.
- **Bước 3:** Khối xử lý trung tâm đọc trạng thái khối nút nhấn để gửi lệnh đến khối điều khiển thiết bị, cập nhật trạng thái thiết bị lên khối hiển thị và MQTT broker.
- **Bước 5:** Khối xử lý trung tâm phản hồi yêu cầu thử kết nối, yêu cầu gửi trạng thái thiết bị và lệnh điều khiển thiết bị từ MQTT broker.

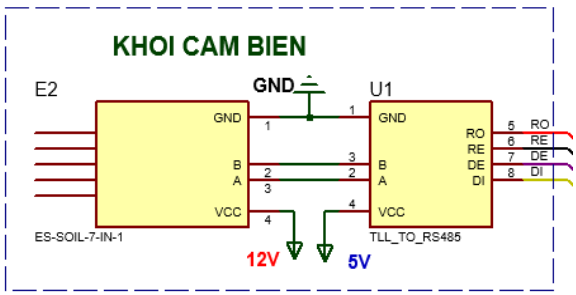
- **Bước 6:** Server nhận dữ liệu cảm biến và theo dõi trạng thái thiết bị từ MQTT broker, lưu dữ liệu vào cơ sở dữ liệu, sau đó Server sẽ xử lý và đẩy dữ liệu lên Website, đồng thời Server cũng xử lý và phản hồi các yêu cầu từ Website.
- **Bước 7:** Website đóng vai trò là giao diện người dùng với nhiệm vụ nhận và hiển thị dữ liệu từ server, gửi các yêu cầu về server để lấy dữ liệu.
- **Bước 8:** Người dùng gửi các lệnh cho Chatbot trong kênh chat, tuỳ theo lệnh mà ChatBot sẽ truy vấn cơ sở dữ liệu, gửi lệnh điều khiển thiết bị và theo dõi trạng thái thiết bị từ MQTT broker hoặc xử lý khuyến nghị cây trồng và phản hồi kết quả cho người dùng.

## 3.4. THIẾT KẾ HỆ THỐNG PHẦN CỨNG

### 3.4.1. Khối cảm biến

Khối cảm biến bao gồm cảm biến đất 7 trong 1 và module chuyển đổi TTL to RS485:

- Chân A của hai thiết bị được nối chung.
- Chân B của hai thiết bị được nối chung.
- Chân GND của hai thiết bị được nối đất.
- Chân VCC của cảm biến đất 7 trong 1 nối với nguồn 12V.
- Chân VCC module chuyển đổi TTL to RS485 nối với nguồn 5V.

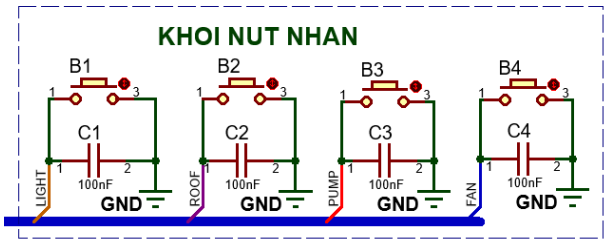



**Hình 3.3:** Sơ đồ kết nối khối cảm biến

### 3.4.2. Khối nút nhấn

Khối nút nhấn có 4 nút cung cấp tín hiệu thay đổi trạng thái tương ứng với 4 thiết bị: đèn , quạt, máy bơm, mái che. Mỗi nút nhấn sẽ mắc song song với một tụ điện 100nF

để chống dội khi nhấn nút, giúp khối xử lý trung tâm có thể đọc tín hiệu một cách chính xác. Một chân nút nhấn nối với một GPIO của khối Xử lý trung tâm, chân còn lại được nối đất. Khi không nhấn nút, GPIO được kéo lên 3.3V nhờ điện trở pull-up bên trong khối Xử lý trung tâm, lúc này đọc được mức HIGH. Khi nhấn nút, khối Xử lý trung tâm đọc mức LOW do GPIO bị nối thẳng xuống đất qua nút nhấn.

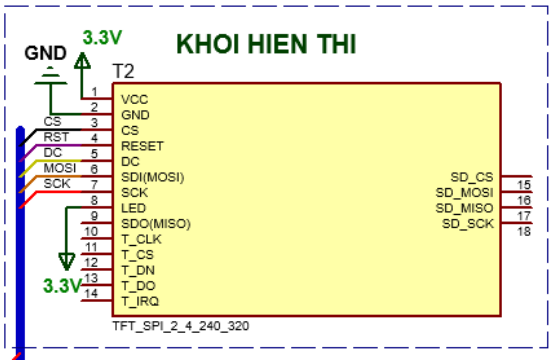



**Hình 3.4:** Sơ đồ kết nối khối nút nhấn

### 3.4.3. Khối hiển thị

Trong khối hiển thị là một module LCD TFT SPI, khối hiển thị kết nối như sau:

- Chân nguồn VCC và LED nối với nguồn 3.3V.
- Chân GND được nối đất.
- Các chân còn lại là CS, RESET, DC, SDI và SCK được nối với các GPIO của khối Xử lý trung tâm.




**Hình 3.5:** Sơ đồ kết khối hiển thị

### 3.4.4. Khối điều khiển thiết bị

Khối điều khiển thiết bị được thiết kế như sau:

- Mái che:

  - Hai cực của motor được nối vào chân OUT1, OUT2 của L298N.
  - Chân +12V của L298N nối với nguồn 12V.
  - Chân GND của L298N nối đất.
  - Chân IN1 và IN2 của L298N nối với điện trở 10k kéo xuống đất.
  - Chân NO của cả hai công tắc nối với điện trở 10k kéo lên nguồn 3.3V.
  - Chân COM của cả hai công tắc hành trình nối đất.

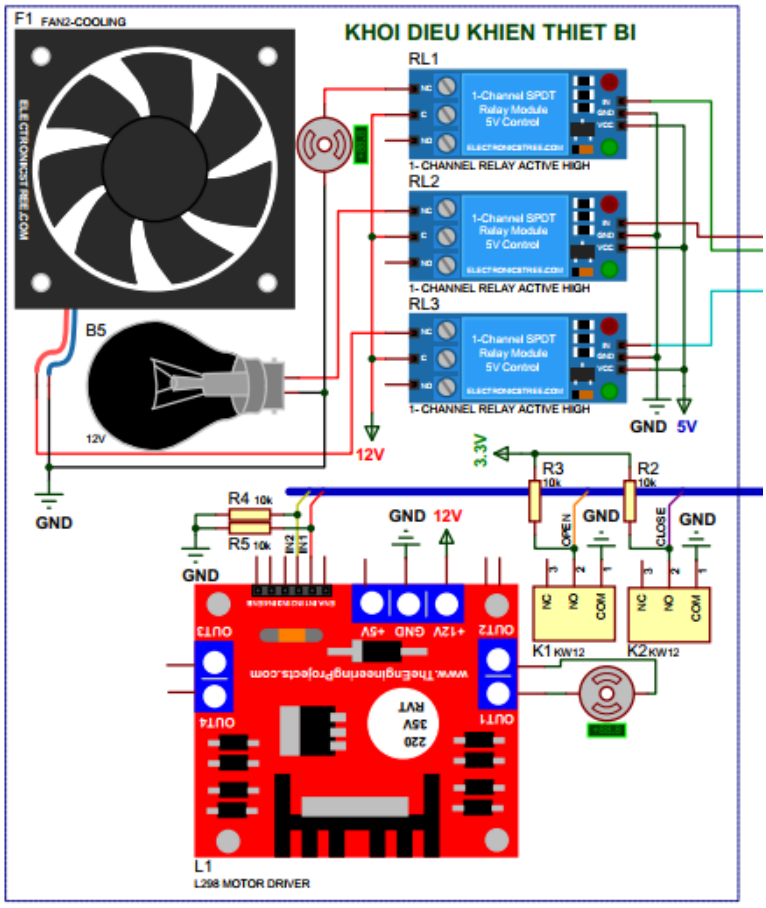
- Chân (+) của máy bơm nối với chân NC của relay1.

- Chân (+) của đèn nối với chân NC của relay2.

- Chân (+) của quạt nối với chân NC của relay3.

- Chân (-) của máy bơm, đèn và quạt nối đất.

- Chân COM của cả 3 relay nối với nguồn 12V.
- Chân GND của cả 3 của relay nối đất.
- Chân VCC của cả 3 nối với nguồn 5V.




**Hình 3.6:** Sơ đồ kết nối khối điều khiển thiết bị

### 3.4.5. Khối xử lý trung tâm

*Kết nối với khối cảm biến:*

- GPIO 32 nối với chân RO của module chuyển đổi TTL to RS485.

- GPIO 26 nối với chân RE của module chuyển đổi TTL to RS485.

- GPIO 25 nối với chân DE của module chuyển đổi TTL to RS485.

- GPIO 33 nối với chân DI của module chuyển đổi TTL to RS485.

*Kết nối với khối nút nhấn:*

- GPIO 5 nối với nút nhấn điều khiển đèn.

- GPIO 17 nối với nút nhấn điều khiển máy bơm.

- GPIO 16 nối với nút nhấn điều khiển quạt.

- GPIO 4 nối với nút nhấn điều khiển mái che.

*Kết nối với khối hiển thị:*

- GPIO 18 nối với chân CS của module LCD TFT.

- GPIO 19 nối với chân RESET của module LCD TFT.

- GPIO 21 nối với chân DC của module LCD TFT.

- GPIO 22 nối với chân MOSI của module LCD TFT.

- GPIO 23 nối với chân SCK của module LCD TFT.

*Kết nối với khối thiết bị:*

- GPIO 27 nối với chân IN của relay điều khiển đèn.

- GPIO 14 nối với chân IN của relay điều khiển máy bơm.

- GPIO 13 nối với chân IN của relay điều khiển quạt.

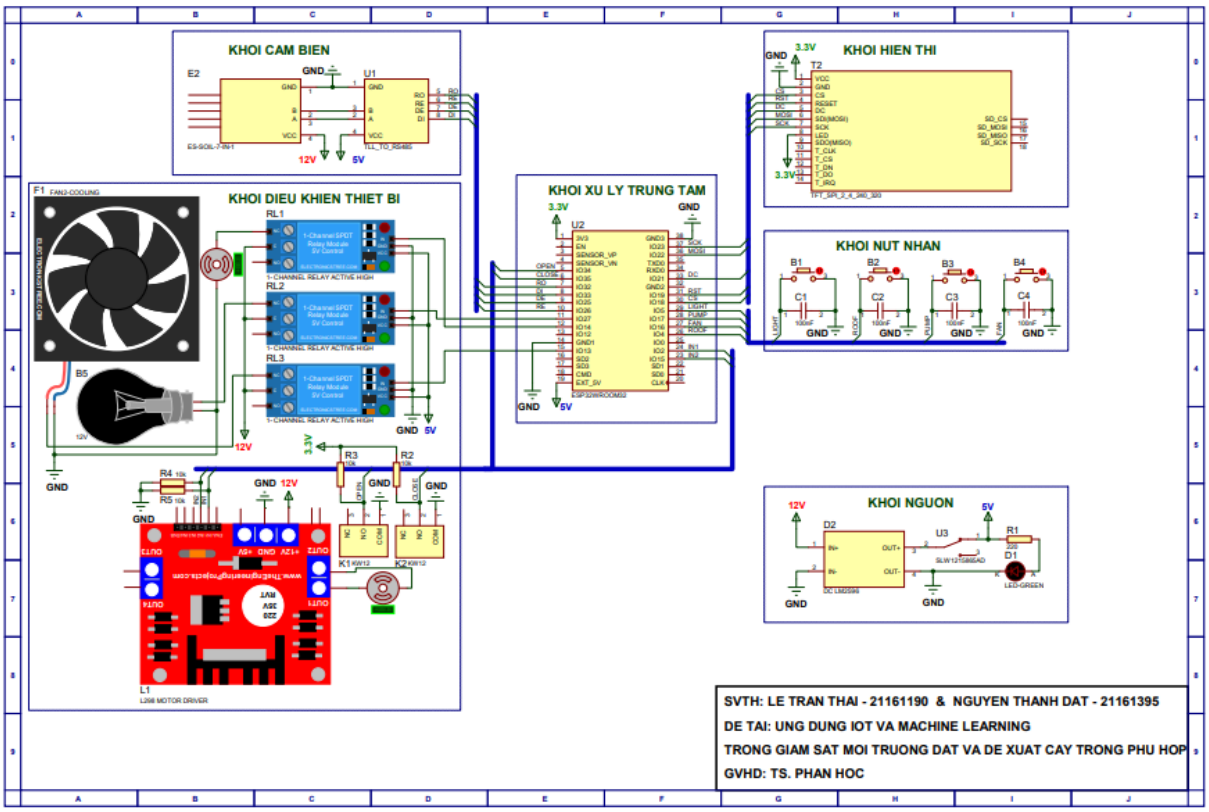
- GPIO 2 nối với chân IN1 của L298N.

- GPIO 15 nối với chân IN2 của L298N.

- GPIO 34 nối với chân NO của công tắc hành trình mở mái che.

- GPIO 35 nối với chân NO của công tắc hành trình đóng mái che.

- Chân V5 được cấp nguồn 5V từ khối nguồn bằng cách nối với chân số 1 của công tắc.




**Hình 3.7:** Sơ đồ nguyên lý toàn mạch

### 3.4.6. Khối nguồn

**Bảng 3.1:** Dòng tiêu thụ và điện áp các linh kiện sử dụng nguồn 5V

| STT | Thiết bị | Dòng tiêu thụ | Điện áp |
|---|---|---|---|
| **1** | Module chuyển đổi TTL to RS485 | 10mA | 5V |
| **2** | Màn hình 2.4 TFT SPI 240 ×320 | 120mA | 5V |
| **3** | ESP32 DEVKIT V1 | 240mA | 5V |
| **4** | Module relay 5V | 80mA | 5V |

Tính toán tổng dòng điện tiêu thụ nguồn 5V:

$$I_{5V} = I_{TTL\ to\ RS485} + I_{TFT} + I_{ESP32} + 3I_{RELAY} = 10 + 120 + 240 + 3 \times 80 = 610\ (\text{mA})$$

Trong dự án này, sử dụng mạch giảm áp DC LM2596 3A để giảm áp từ 12V xuống 5V để cấp nguồn cho các thiết bị hoạt động ổn định. Với công suất 15W, dòng tối

đa là 3A và hiệu suất chuyển đổi của mạch giảm áp LM2596 là 92% , ta có thể tính được dòng đầu vào như sau:

$$I_{LM2596} = \frac{V_{5V} \times I_{5V}}{V_{12V} \times \eta} = \frac{5 \times 0.61}{12 \times 0.92} = 0.276\ (A)$$


**Hình 3.8:** Mạch giảm áp DC LM2596

**Bảng 3.2:** Dòng tiêu thụ và điện áp các linh kiện sử dụng nguồn 12V

| STT | Thiết bị | Dòng tiêu thụ | Điện áp |
|---|---|---|---|
| **1** | Mạch điều khiển động cơ DC L298N | 2A | 5-30V |
| **2** | Cảm biến đất 7 trong 1 (RS485 Modbus RTU) | 42mA | 4.5-30V |
| **3** | Bóng đèn led ampoule 12V 5W | 0.42A | 12V |
| **4** | Quạt tản nhiệt | 0.18A | 12V |
| **5** | Động cơ bơm 365 | 0.23A | 12V |

Tính toán tổng dòng điện tiêu thụ nguồn 12V:

$I_{12V} = I_{L298N} + I_{CẢM\ BIẾN} + I_{ĐÈN} + I_{QUẠT} + I_{MÁY\ BƠM}$

= 2 + 0.042 + 0.42 + 0.18 + 0.23 = 2.872 (A)

Tổng dòng điện nguồn 12V cần cấp cho hệ thống:

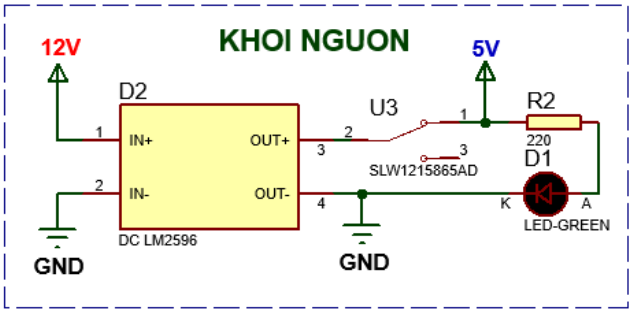
I = $I_{12V} + I_{LM2596}$ = 2.872 + 0.276 = 3,148 (A)

Do vậy, cần sử dụng nguồn 12V có dòng tối thiểu 3.2A. Ở dự án này, sử dụng nguồn tổ ong 12V 30A giúp đảm bảo hệ thống hoạt động một cách ổn định.

**Hình 3.9:** Nguồn 12V 30A

*Khối nguồn được kết nối như sau:*

- Chân IN+ của DC LM2596 nối với chân (+) của nguồn 12V.
- Chân OUT+ của DC LM2596 nối với chân số 2 của công tắc.
- Chân IN- và OUT- của DC LM2596 nối đất.
- Điện trở 220 và LED được mắc nối tiếp với chân số 1 của công tắc.




**Hình 3.10:** Sơ đồ khối nguồn

## 3.5. HUẤN LUYỆN MÔ HÌNH KHUYẾN NGHỊ CÂY TRỒNG

### 3.5.1. Chuẩn bị dữ liệu

Dữ liệu được lấy từ bộ dữ liệu công khai trên Kaggle [13], tệp dữ liệu bao gồm 22 loại cây trồng khác nhau, phần lớn các cây phổ biến ở Việt Nam như lúa, dưa hấu, cà phê, xoài,… Mỗi hàng trong tệp là một mẫu dữ liệu với các đặc trưng sau: nitơ (mg/kg), phốt pho (mg/kg), kali (mg/kg), nhiệt độ (°C), độ ẩm (%), độ pH, lượng mưa tháng trước (mm) và nhãn đầu ra tương ứng là tên loại cây trồng phù hợp. Tổng cộng có 2200 mẫu

dữ liệu, tức là 100 mẫu cho mỗi loại cây trồng, giúp cho mô hình được huấn luyện trên tập dữ liệu cân bằng, tăng khả năng tổng quát hoá và mức độ chính xác khi dự đoán thực tế. Dữ liệu được lưu theo định dạng CSV, các cột cách nhau bởi dấu phẩy, dễ dàng xử lý bằng ngôn ngữ Python với thư viện pandas.

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | N,P,K,temperature,humidity,ph,rainfall,label | | | | | | |
| 2 | 90,42,43,20.87974371,82.00274423,6.502985292000001,202.9355362,rice | | | | | | |
| 3 | 85,58,41,21.77046169,80.31964408,7.038096361,226.6555374,rice | | | | | | |
| 4 | 60,55,44,23.00445915,82.3207629,7.840207144,263.9642476,rice | | | | | | |
| 5 | 74,35,40,26.49109635,80.15836264,6.980400905,242.8640342,rice | | | | | | |
| 6 | 78,42,42,20.13017482,81.60487287,7.628472891,262.7173405,rice | | | | | | |
| 7 | 69,37,42,23.05804872,83.37011772,7.073453503,251.0549998,rice | | | | | | |
| 8 | 69,55,38,22.70883798,82.63941394,5.70080568,271.3248604,rice | | | | | | |
| 9 | 94,53,40,20.27774362,82.89408619,5.718627177999999,241.9741949,rice | | | | | | |
| 10 | 89,54,38,24.51588066,83.53521629999999,6.685346424,230.4462359,rice | | | | | | |

**Hình 3.11:** Một số mẫu dữ liệu cho cây lúa

### 3.5.2. Tiền xử lý dữ liệu

Trước khi đưa dữ liệu vào huấn luyện, tệp dữ liệu cần được xử lý nhằm đảm bảo tính chính xác và hiệu quả. Quá trình đó gọi là tiền xử lý dữ liệu bao gồm các bước sau:

- **Mã hoá nhãn đầu ra:** cột tên cây trồng trong tệp dữ liệu ở dạng văn bản, mà mô hình yêu cầu đầu ra ở dạng số nên cần mã hoá nhãn bằng kỹ thuật LabelEncoder, sau khi mã hoá mỗi cây sẽ tương ứng với một số nguyên như: 0→rice, 1→maize, 2→chickpea,…bảng ánh xạ này sẽ được lưu lại để sử dụng trong giai đoạn giải mã kết quả dự đoán.
- **Chuẩn hoá dữ liệu đầu vào:** các đặc trưng đầu vào có dải giá trị khác nhau, nếu không chuẩn hoá mô hình có thể sẽ bị lệch trọng số về các đặc trưng có giá trị lớn, do đó dữ liệu đã được chuẩn hoá bằng kỹ thuật StandardScaler, đưa các đặc trưng về phân phối với trung bình 0 và độ lệch chuẩn 1, giúp mô hình học nhanh và hiệu quả hơn.
- **Phân chia tập dữ liệu:** sau khi chuẩn hoá, chúng ta sẽ chia ngẫu nhiên dữ liệu thành hai phần là tệp huấn luyện (80%) và tệp kiểm tra (20%), dữ liệu được phân chia

ngẫu nhiên đảm bảo mô hình không bị trùng lặp dữ liệu và đánh giá một cách khách quan.

### 3.5.3. Huấn luyện mô hình

Sau khi hoàn thành quá trình tiền xử lý dữ liệu, lúc này mô hình học máy sẽ được huấn luyện bằng thuật toán LightGBM. Thuật toán này sử dụng phương pháp leaf-wise để xây dựng cây quyết định, cho phép tăng độ chính xác đáng kể cũng như tốc độ học, đặc biệt hiệu quả với dữ liệu dạng bảng.

- **Cấu hình dữ liệu đầu vào:** tệp dữ liệu sau khi được xử lý bao gồm X_train chứa các đặc trưng đầu vào (N, P, K, nhiệt độ, độ ẩm, pH, lượng mưa) và y_train chứa nhãn đầu ra (nhãn cây trồng đã mã hoá số).
- **Khởi tạo mô hình LightGBM:** chúng ta sẽ khởi tạo mô hình với các tham số:
    - n_estimators = 200: Số lượng cây quyết định
    - learning_rate = 0.05: Tốc độ học
    - max_depth = 10: Độ sâu tối đa của mỗi cây
- **Huấn luyện mô hình:** quá trình huấn luyện được thực hiện bằng cách truyền tệp dữ liệu vào mô hình qua hàm .fit(), LightGBM sẽ liên tục điều chỉnh trọng số dựa trên sai số từ vòng học trước, từ đó cải thiện độ chính xác qua từng vòng.
- **Kết quả huấn luyện:** sau quá trình huấn luyện hoàn tất, mô hình đã học được mối quan hệ giữa các dữ liệu và loại cây trồng tương ứng, lúc này mô hình đã sẵn sàng nhận dữ liệu từ cảm biến để đưa ra dự đoán cây trồng phù hợp nhất.

### 3.5.4. Đánh giá mô hình

- **Dự đoán trên tập kiểm tra:** mô hình thực hiện dự đoán trên X_test, kết quả trả về là danh sách các mã số tương ứng với loại cây trồng dự đoán.
- **Độ chính xác:** mô hình đạt độ chính xác khoảng 98.64% trên tập kiểm tra cho thấy mô hình học tốt quy luật phân loại dựa vào các đặc trưng của đất.
- **Báo cáo phân loại:** báo cáo chi tiết về precision, recall và f1-score được tạo bằng hàm classification_report, cung cấp đánh giá hiệu suất của mô hình cho từng loại cây trồng. Kết quả cho thấy hầu hết các lớp đạt precision và recall gần 100%, một

số lớp như 'rice' và 'pigeonpeas' có hiệu suất thấp hơn với recall lần lượt là 89% và 91%).

- **Đánh giá bằng cross-validation:** mô hình được đánh giá thêm bằng kỹ thuật cross-validation 5-fold trên tập huấn luyện, đạt độ chính xác trung bình 98.81% với độ lệch chuẩn 0.87%, cho thấy mô hình có khả năng tổng quát hóa tốt trên các tập dữ liệu khác nhau.
- **Ma trận nhầm lẫn (Confusion Matrix):** được sử dụng để phân tích chi tiết các trường hợp nhầm lẫn giữa các lớp cây trồng, mỗi hàng trong ma trận đại diện cho nhãn thực tế, cột đại diện cho nhãn dự đoán, các ô nằm ngoài đường chéo chính phản ánh các lỗi mà mô hình mắc phải khi phân loại.

### 3.5.5. Lưu mô hình

Sau khi đã hoàn thành huấn luyện và đánh giá, mô hình lưu thành 3 tệp như sau:

- **scaler.pkl:** bộ chuẩn hoá dữ liệu đầu vào, đảm bảo dữ liệu đầu vào lúc dự đoán được chuẩn hoá giống với lúc huấn luyện.
- **lgbm_crop_model.pkl:** mô hình LightGBM đã huấn luyện, bao gồm toàn bộ cấu trúc cây và các thông số đã học.
- **label_encoder.pkl:** lưu bộ mã hoá nhãn cây trồng, dùng để chuyển đổi giữa số và tên cây trồng trong quá trình dự đoán.

Các tệp này sẽ được tích hợp vào server để xử lý dự đoán cây trồng từ Website và ChatBot.

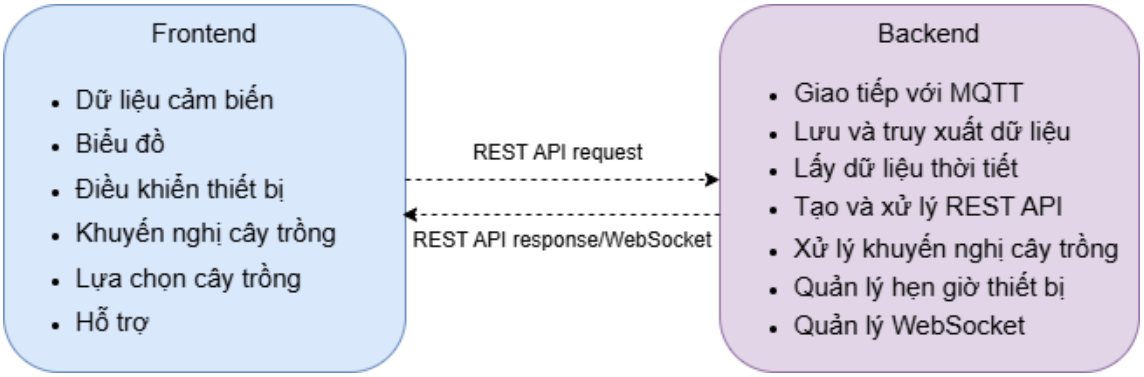
## 3.6. THIẾT KẾ PHẦN MỀM

### 3.6.1. Thiết kế Website

Thiết kế hệ thống gồm 2 phần chính:

**- Frontend (giao diện người dùng):** được kết hợp từ HTML, CSS và JavaScript, đảm bảo hiển thị dữ liệu thời gian thực, cung cấp giao diện điều khiển thiết bị, khuyến nghị cây trồng và một số chức năng khác. Frontend giao tiếp với backend qua REST API request để gửi yêu cầu và nhận dữ liệu phản hồi.

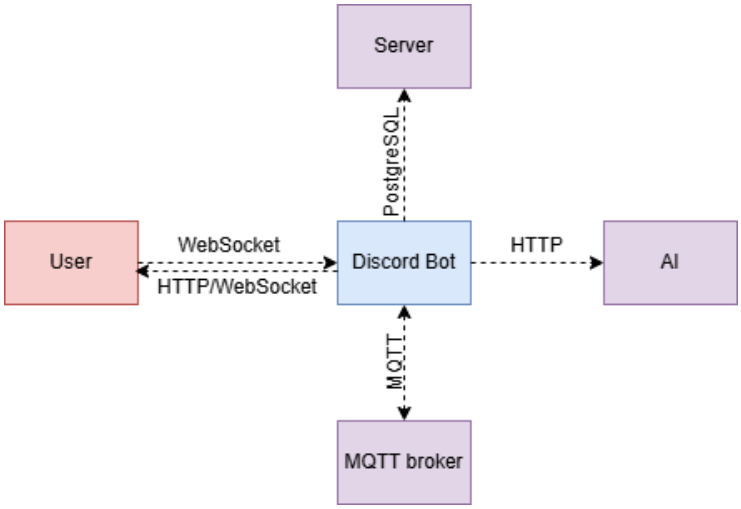
**- Backend (máy chủ xử lý):** được lập trình bằng ngôn ngữ Python, sử dụng framework FastAPI để đảm bảo vai trò của trung tâm xử lý dữ liệu. Backend sẽ giao tiếp với các thiết bị IoT thông qua MQTT broker, lưu trữ và truy xuất dữ liệu từ cơ sở dữ liệu PostgreSQL, đồng thời cung cấp các API phục vụ cho frontend. Ngoài ra, backend còn xử lý các chức năng khác như xử lý khuyến nghị cây trồng, quản lý hẹn giờ, đẩy dữ liệu thời gian thực đến frontend thông qua WebSocket,...




**Hình 3.12:** Mô hình phân chia chức năng giữa frontend và backend

### 3.6.2. Thiết kế ChatBot

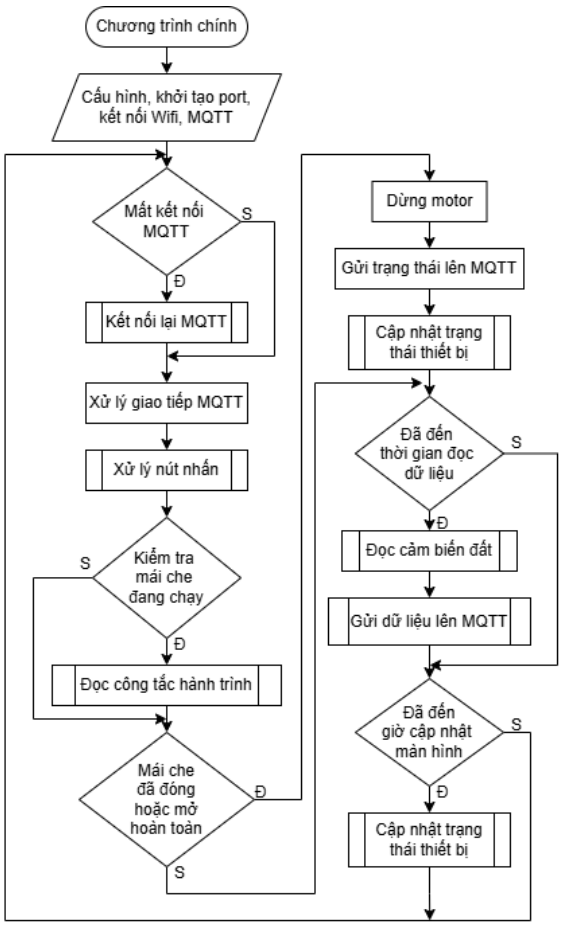
ChatBot có vai trò hỗ trợ người dùng giao tiếp với hệ thống nông nghiệp thông minh thông qua nền tảng Discord. Việc sử dụng nền tảng Discord giúp tận dụng sẵn hạ tầng ổn định, bảo mật và trải nghiệm người dùng quen thuộc. ChatBot được lập trình bằng ngôn ngữ Python, sử dụng đồng thời các giao thức MQTT, HTTP, WebSocket và TCP/IP giúp hệ thống xử lý nhanh chóng và dễ dàng mở rộng. Để tương tác với ChatBot, chỉ cần gõ các từ khoá như !sensor, !predict, !timer, !device,… hoặc trực tiếp hỏi các câu hỏi liên quan đến nông nghiệp. Tuỳ theo lệnh yêu cầu từ người dùng mà hệ thống sẽ truy vấn cơ sở dữ liệu để lấy về dữ liệu mới nhất, giao tiếp với MQTT broker để gửi lệnh điều khiển hoặc theo dõi trạng thái của thiết bị, gửi yêu cầu từ người dùng đến mô hình AI để xử lý và phản hồi.




**Hình 3.13:** Sơ đồ luồng hoạt động của ChatBot

## 3.7. LƯU ĐỒ GIẢI THUẬT CHO ESP32

### 3.7.1. Chương trình chính




**Hình 3.14:** Lưu đồ chương trình chính

Các bước hoạt động của chương trình chính:

**Bước 1:** Cấu hình, khởi tạo các cảm biến, GPIO, màn hình TFT, kết nối Wifi và MQTT, hiển thị màn hình khởi động.

**Bước 2:** Tiến hành kiểm tra kết nối với MQTT, nếu mất kết nối thì gọi hàm "Kết nối lại MQTT" để thử kết nối lại đến khi thành công rồi chuyển sang bước 3, nếu đã kết nối thành công thì chuyển sang bước 3.

**Bước 3:** Xử lý giao tiếp MQTT bằng cách gọi client.loop() để duy trì kết nối với MQTT, xử lý tin nhắn nhận được từ MQTT và gọi hàm "Xử lý sự kiện MQTT" khi cần thiết.

**Bước 4:** Gọi hàm "Xử lý nút nhấn" cho từng nút để điều khiển các thiết bị.

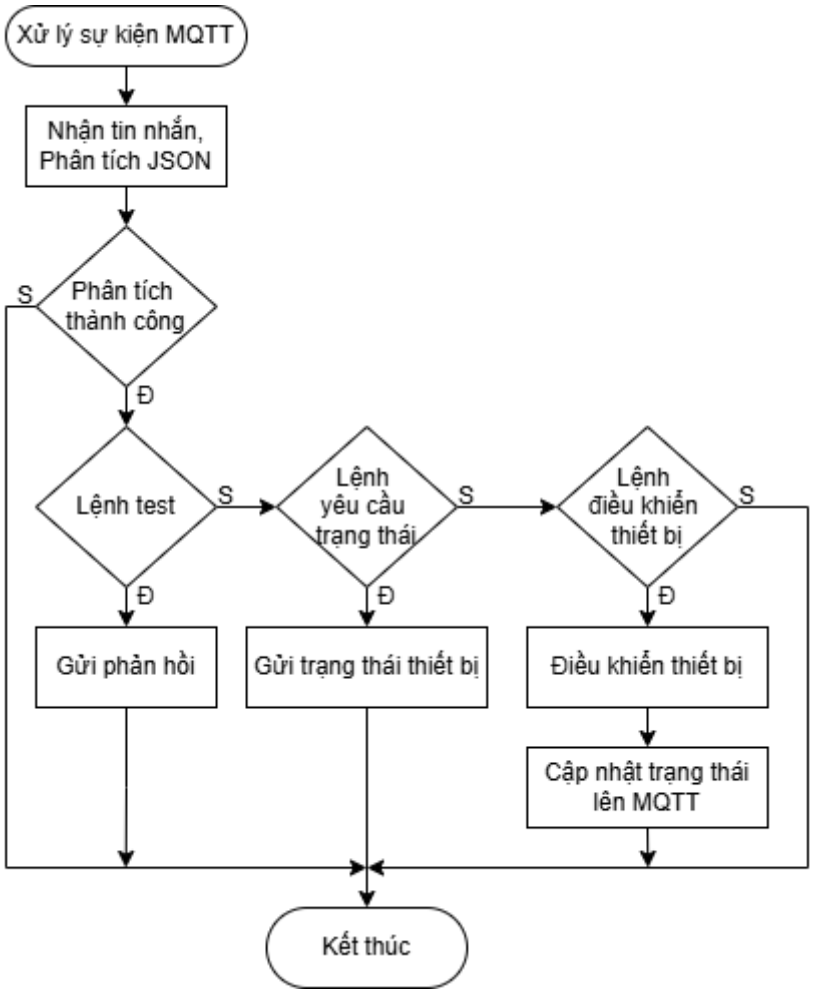
**Bước 5:** Kiểm tra mái che có đang chạy hay không, nếu mái che không chạy thì chuyển sang bước tiếp theo, nếu mái che đang chạy thì gọi hàm "Đọc công tắc hành trình" để lấy trạng thái ổn định của công tắc.

**Bước 6:** Kiểm tra trạng thái của công tắc hành trình để xem mái che đã được đóng/mở hoàn toàn chưa, nếu chưa thì chuyển sang bước tiếp theo, nếu mái che đã được đóng/mở hoàn toàn thì dừng motor và gửi trạng thái mái che lên MQTT.

**Bước 7:** Kiểm tra đã qua 5s kể từ lần cuối đọc dữ liệu từ cảm biến, nếu đã đến thời gian thì gọi hàm "Đọc cảm biến đất" để đọc dữ liệu mới.

**Bước 8:** Gọi hàm "Cập nhật trạng thái thiết bị" để hiển thị trạng thái của các thiết bị lên màn hình TFT, sau đó quay lại bước 2 để tiếp tục vòng lặp.

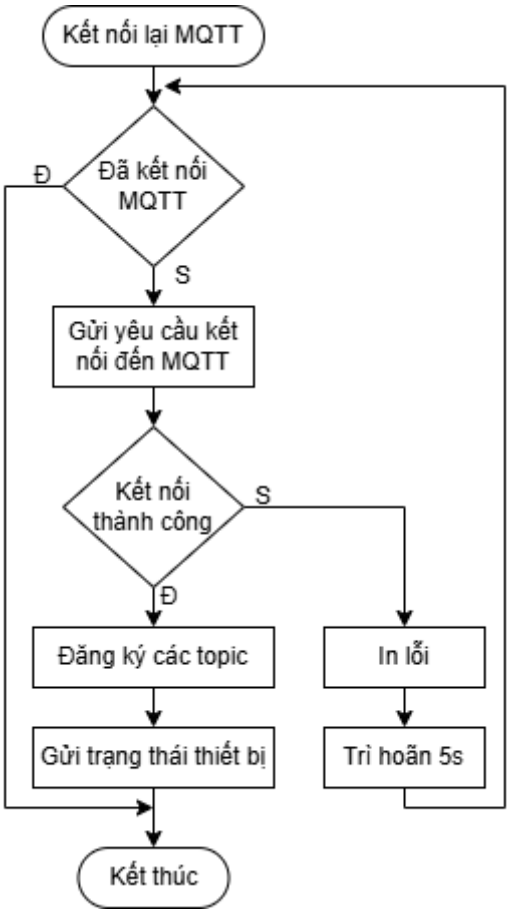
### 3.7.2. Xử lý sự kiện MQTT




**Hình 3.15:** Lưu đồ chương trình Xử lý sự kiện MQTT

Hàm này được gọi bởi client.loop() khi nhận được tin nhắn từ MQTT, sau khi nhận được tin nhắn, nó sẽ phân tích topic và payload. Nếu topic là test thì nó phản hồi trạng thái kết nối, nếu topic yêu cầu trạng thái thiết bị nó trả về trạng thái của thiết bị, nếu topic là lệnh điều khiển thiết bị thì nó sẽ gửi lệnh bật/tắt thiết bị tương ứng và cập nhật trạng thái thiết bị lên MQTT.

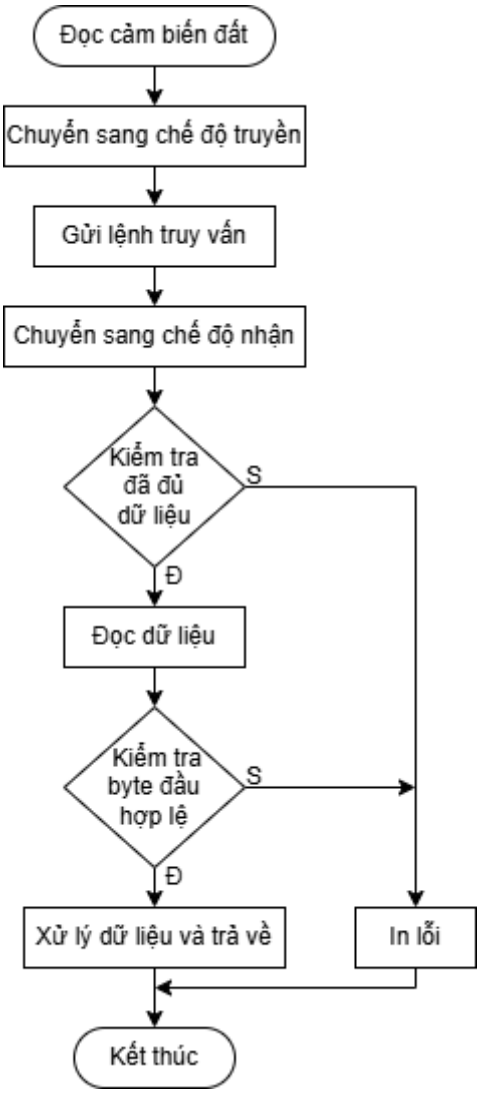
### 3.7.3. Kết nối lại MQTT




**Hình 3.16:** Lưu đồ chương trình Kết nối lại MQTT

Hàm này kiểm tra đã bị kết nối với MQTT chưa, nếu đang kết nối thì thoát khỏi chương trình, nếu chưa kết nối thì gửi thông tin xác thực đến MQTT để yêu cầu kết nối lại. Kết nối thành công thì sẽ đăng ký lại các topic và gửi trạng thái thiết bị lên MQTT sau đó thoát khỏi chương trình, ngược lại nếu kết nối thất bại được thì in lỗi ra màn hình và chờ 5 giây rồi thử kết nối lại.

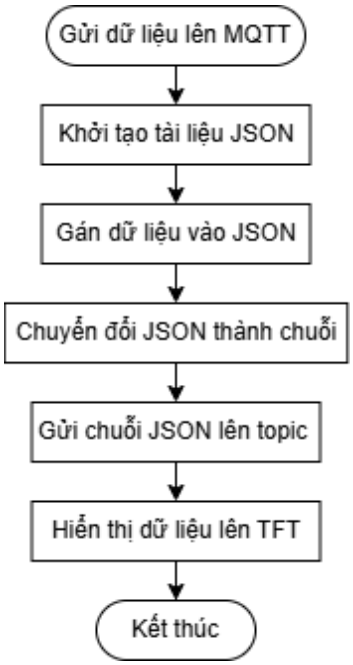
### 3.7.4. Đọc cảm biến đất




**Hình 3.17:** Lưu đồ chương trình Đọc cảm biến đất

Hàm này đọc các thông số từ cảm biến đất 7 trong 1, đầu tiên sẽ kích hoạt chế độ truyền bằng cách đặt chân DE và RE ở mức HIGH, tiếp theo sẽ gửi lệnh truy vấn đến cảm biến, sau đó chuyển về chế độ nhận bằng cách đặt chân DE và RE về mức LOW. Hàm sẽ đợi phản hồi từ cảm biến và thực hiện kiểm tra dữ liệu, nếu dữ liệu không đủ thì in lỗi ra màn hình và thoát khỏi chương trình, nếu dữ liệu đủ thì đọc dữ liệu. Tiếp theo kiểm tra byte đầu tiên để xem dữ liệu có hợp lệ không, nếu không hợp lệ thì in lỗi ra màn hình và thoát khỏi chương trình, nếu hợp lệ thì tiến hành xử lý và trả dữ liệu về.

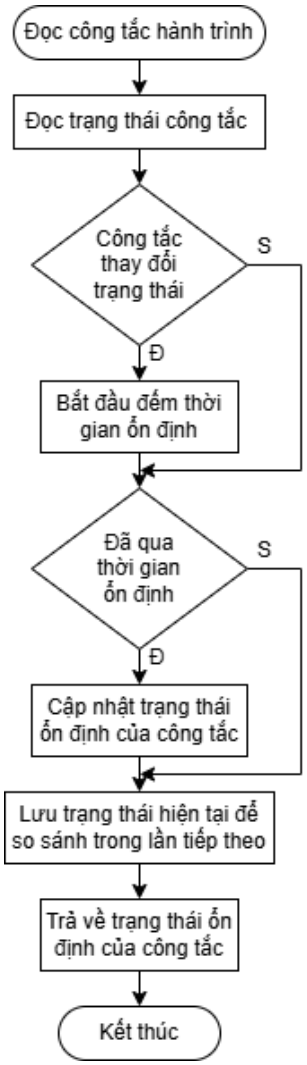
### 3.7.5. Gửi dữ liệu lên MQTT




**Hình 3.18:** Lưu đồ chương trình Gửi dữ liệu lên MQTT

Hàm này có chức năng gửi dữ liệu lên MQTT và hiển thị dữ liệu lên màn hình TFT. Đầu tiên, tạo một JSON chứa nhiệt độ, độ ẩm, N, P, K, pH, sau đó chuyển JSON thành dạng chuỗi và gửi lên MQTT, đồng thời cập nhật dữ liệu lên màn hình TFT.

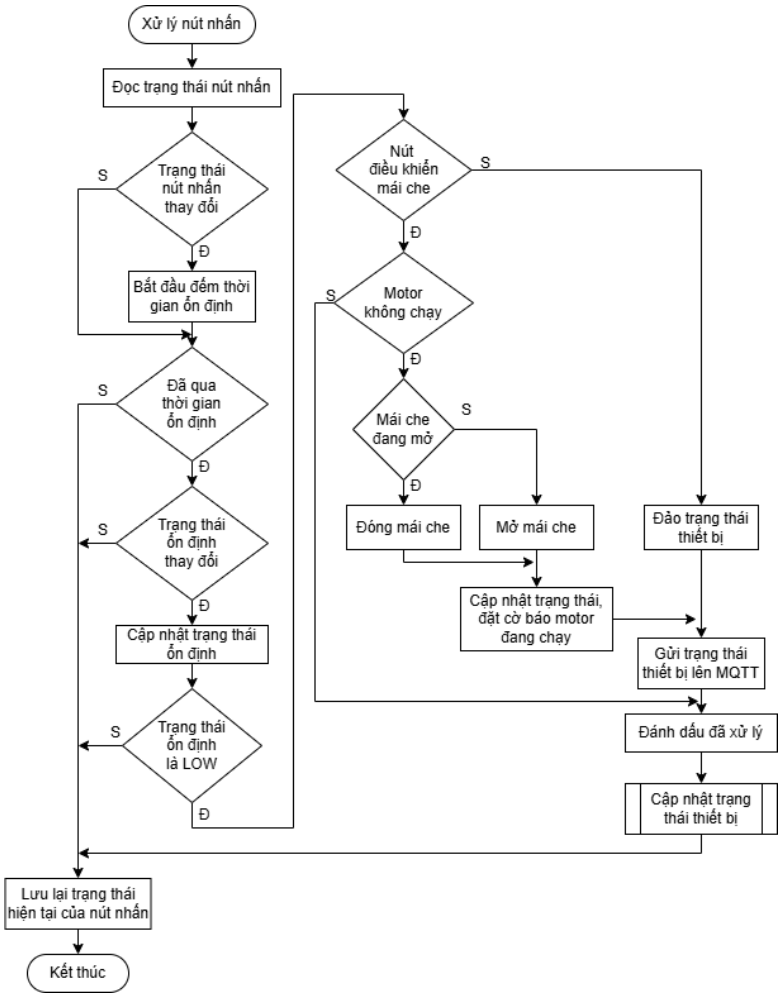
### 3.7.6. Đọc công tắc hành trình




**Hình 3.19:** Lưu đồ chương trình Đọc công tắc hành trình

Hàm này sử dụng cơ chế debounce để đọc trạng thái công tắc hành trình một cách ổn định, tránh nhiễu tín hiệu. Khi được gọi, nó sẽ đọc tín hiệu từ chân GPIO để biết trạng thái của công tắc hành trình và lưu lại trạng thái vừa đọc được. Tiếp theo kiểm tra xem công tắc hành trình có thay đổi trạng thái không bằng cách so sánh trạng thái mới đọc với trạng thái trước đó. Nếu phát hiện trạng thái công tắc hành trình thay đổi, bắt đầu đếm thời gian ổn định, nếu đã qua thời gian ổn định (50ms) thì lưu lại trạng thái của công tắc hiện tại để so sánh trong lần tiếp theo và trả về trạng thái công tắc đã ổn định.

### 3.7.7. Xử lý nút nhấn




**Hình 3.20:** Lưu đồ chương trình Xử lý nút nhấn

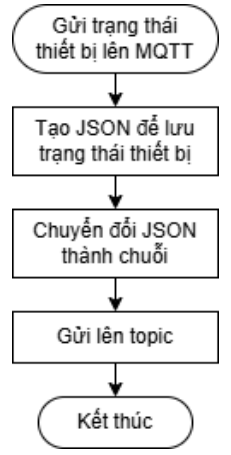
Hàm này sử dụng cơ chế debounce để đọc trạng thái nút nhấn một cách ổn định. Khi được gọi, hàm sẽ đọc trạng thái nút nhấn, nếu nút nhấn đổi trạng thái, bắt đầu đếm thời gian để chờ đến khi nút nhấn ổn định. Nếu đã qua thời gian ổn định mà nút nhấn vẫn giữ trạng thái ổn định không thay đổi thì cập nhật lại biến trạng thái và chờ đến khi nút nhấn xuống mức LOW để thực hiện các lệnh tiếp theo. Sau khi nút nhấn xuống mức LOW, kiểm tra xem nút nhấn nào vừa được nhấn, nếu là nút nhấn mái che thì tiến hành điều khiển mái che đóng/mở, nếu là nút nhấn của các thiết bị khác thì đảo trạng thái của thiết bị đó. Sau đó gửi trạng thái thiết bị lên MQTT, đánh dấu đã xử lý và cập nhật trạng thái thiết bị trên màn hình LCD TFT. Lưu lại trạng thái của nút nhấn để so sánh ở lần tiếp theo và thoát khỏi chương trình.

### 3.7.8. Cập nhật trạng thái thiết bị


**Hình 3.21:** Lưu đồ chương trình Cập nhật trạng thái thiết bị

Hàm này có chức năng cập nhật trạng thái thiết bị lên màn hình TFT, thường được gọi khi trạng thái thiết bị thay đổi. Khi được gọi, hàm sẽ gửi lệnh xoá vùng hiển thị trạng thái thiết bị, tiếp theo chọn kích thước chữ và hiển thị trạng thái của từng thiết bị tại các vị trí cố định.

### 3.7.9. Gửi trạng thái thiết bị lên MQTT



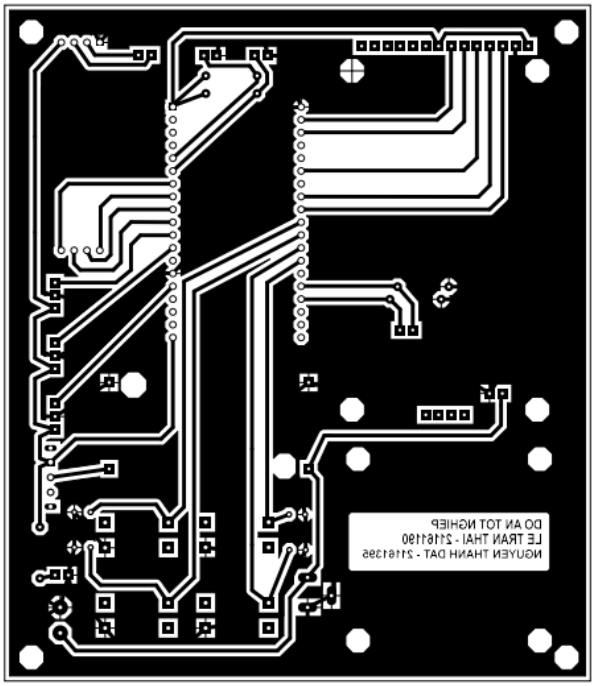

**Hình 3.22:** Lưu đồ chương trình Gửi trạng thái thiết bị lên MQTT

Hàm này có chức năng gửi trạng thái của tất cả các thiết bị lên MQTT khi được gọi. Đối với mỗi thiết bị, hàm sẽ tạo một JSON chứa trạng thái của thiết bị, sau đó chuyển JSON này thành chuỗi và gửi đến các topic MQTT, đồng thời in trạng thái của thiết bị ra màn hình.

# CHƯƠNG 4: KẾT QUẢ THỰC HIỆN

## 4.1. KẾT QUẢ MẠCH IN

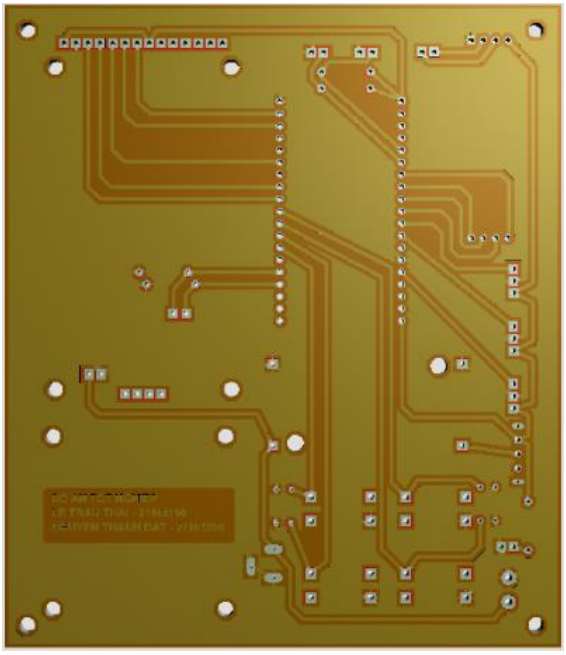
Sau khi thiết kế xong sơ đồ nguyên lý và tính toán các thông số cần thiết, nhóm kiểm tra hoạt động của hệ thống trên testboard nhằm đảm bảo mạch hoạt động một cách chính xác, phát hiện và điều chỉnh các sai sót trước khi tiến hành thiết kế mạch in bằng phần mềm Proteus. Các linh kiện được bố trí một cách một cách hợp lý, tối ưu không gian và thuận tiện cho quá trình thi công.



**Hình 4.1:** Kết quả vẽ mạch in bằng phần mềm Proteus.

Bước tiếp theo là in và thi công mạch. Để đảm bảo độ chính xác trong quá trình lắp ráp linh kiện thì sơ đồ bố trí linh kiện là rất cần thiết để xác định vị trí và chiều của linh kiện, giảm thiểu lỗi trong quá trình thi công.

**Hình 4.2:** Sơ đồ bố trí linh kiện mặt trên



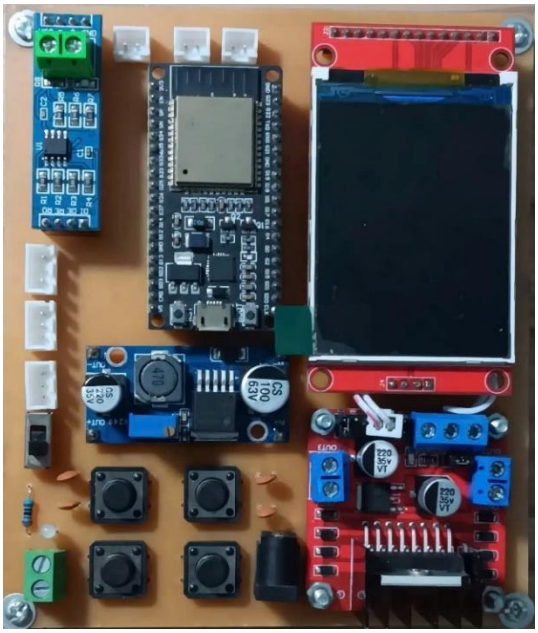
**Hình 4.3:** Sơ đồ bố trí linh kiện mặt dưới

Các linh kiện được sử dụng trên mạch in đều được lựa chọn để thích hợp với yêu cầu của hệ thống và tuân theo đúng thông số kỹ thuật đã được tính toán trước đó. Danh sách các linh kiện sử dụng trong mạch được liệt kê trong Bảng 4.1:

**Bảng 4.1:** Danh sách linh kiện sử dụng trong mạch in

| STT | Tên linh kiện | Số lượng |
|---|---|---|
| **1** | ESP32 DEV KIT V1 | 1 |
| **2** | Module LCD 2.4inch 240x320 TFT ILI9341 Giao Tiếp SPI | 1 |
| **3** | Mạch chuyển đổi RS485 to TTL | 1 |
| **4** | Mạch điều khiển động cơ DC L298N | 1 |
| **5** | Mạch Giảm Áp DC LM2596 | 1 |
| **6** | Nút nhấn | 4 |
| **7** | Tụ 100nF | 4 |
| **8** | LED | 1 |
| **9** | Điện trở 220R | 1 |
| **10** | Điện trở 10K | 4 |
| **11** | Đầu Cắm Nguồn Cái DC-005 | 1 |
| **12** | Công tắc gạt SS-12F44 | 1 |
| **13** | Domino 2Pin | 1 |
| **14** | XH2.54-2P Connector | 5 |
| **15** | XH2.54-3P Connector | 3 |

Các linh kiện được lắp ráp và hàn vào mạch in theo đúng thiết kế. Sau khi làm xong mạch, kiểm tra mạch để đảm bảo mạch không bị chập, đứt hay hở chân linh kiện. Cuối cùng là kiểm tra hoạt động của từng khối và kiểm tra hoạt động của cả mạch trước khi tích hợp vào mô hình.



**Hình 4.4:** Hình ảnh mạch in thực tế

## 4.2. KẾT QUẢ MÔ HÌNH

Sau khi làm xong mạch in, nhóm xây dựng mô hình thực tế để mô phỏng hoạt động của hệ thống. Mô hình được thiết kế mô phỏng một nhà kính thu nhỏ, được tích hợp các thiết bị như đèn, máy bơm nước, quạt thông gió, mái che tự động, module relay và cảm biến đất 7 trong 1 được kết nối với mạch in. Toàn bộ hệ thống được cấp nguồn bằng nguồn tổ ong 12V. Việc xây dựng mô hình có vai trò quan trọng để kiểm tra tính thực tế và ứng dụng của đề tài.


**Hình 4.5:** Mặt trước của sản phẩm

**Hình 4.6:** Mặt sau của sản phẩm

## 4.3. KIỂM TRA HOẠT ĐỘNG HỆ THỐNG

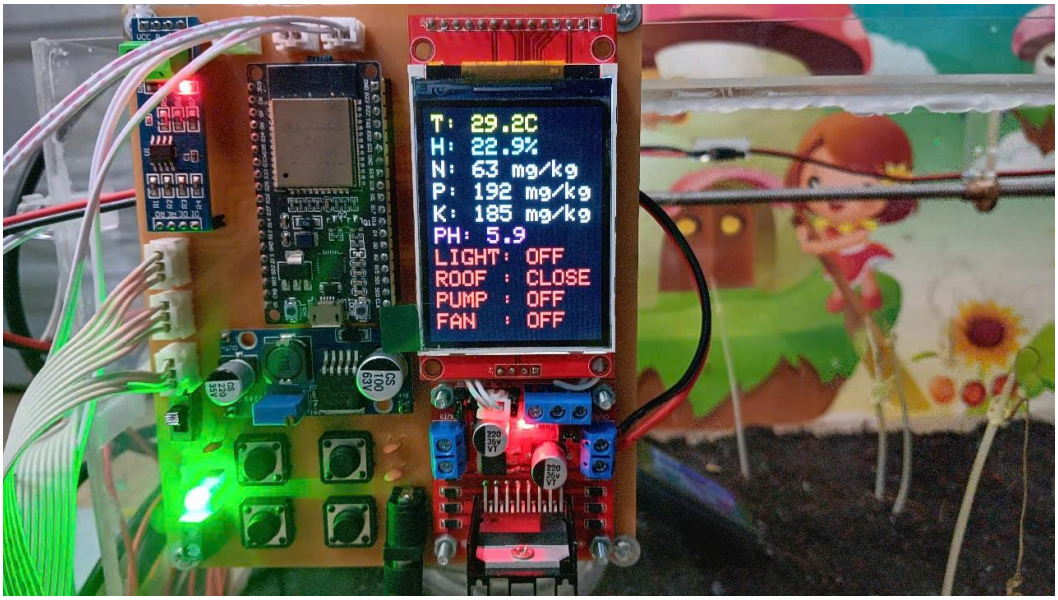
### 4.3.1. Phần cứng

**a. Thu thập dữ liệu nhiệt độ, độ ẩm, độ pH, hàm lượng NPK trong đất**

Tại thời điểm kiểm tra, cảm biến cho biết các chỉ số môi trường đất như sau:

- **Nhiệt độ:** 29.2°C
- **Độ ẩm:** 22.9%
- **Hàm lượng Nitơ:** 63 mg/kg
- **Hàm lượng Phốt pho:** 192 mg/kg
- **Hàm lượng Kali:** 185 mg/kg
- **pH:** 5.9

Các giá trị thu thập được đều nằm ngưỡng hợp lý cho thấy hệ thống đã hoạt động chính xác và dữ liệu là đáng tin cậy. Dữ liệu được hiển thị và cập nhật định kỳ trên màn hình TFT một cách ổn định.




**Hình 4.7:** Hiển thị dữ liệu cảm biến trên màn hình

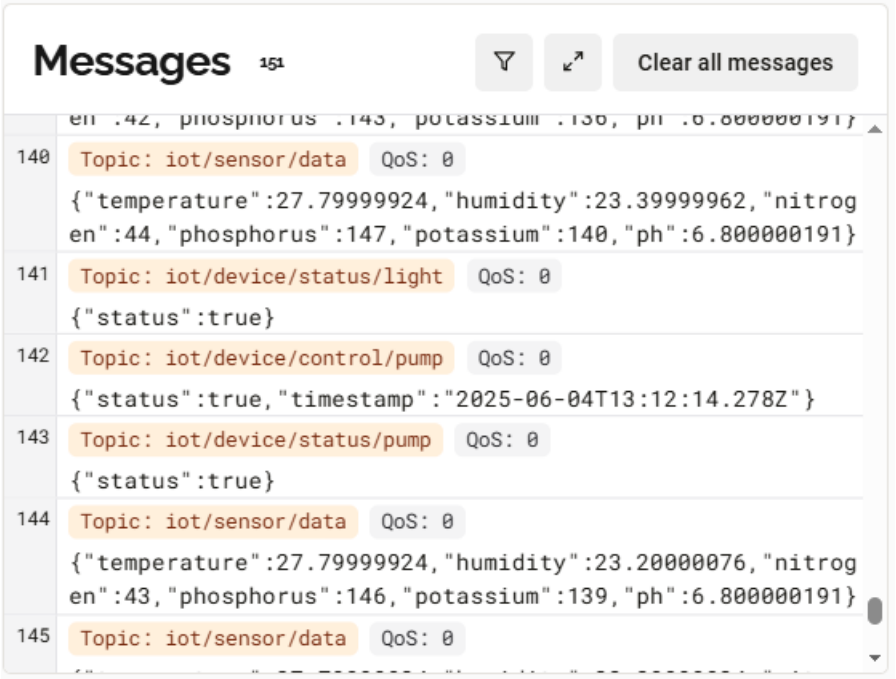
**b. Điều khiển các thiết bị bằng nút nhấn.**

Thử nhấn nút để bật các thiết bị, các thiết bị lập tức được bật và trạng thái được cập nhật trên màn hình LCD TFT một cách nhanh chóng.


**Hình 4.8:** Điều khiển thiết bị bằng nút nhấn

**c. Gửi dữ liệu thu thập được và trạng thái thiết bị lên MQTT broker, nhận lệnh điều khiển thiết bị từ MQTT broker**

Dữ liệu cảm biến được gửi lên MQTT broker bằng topic iot/sensor/data với đầy đủ các thông số. Khi nhấn nút trên mạch để bật đèn, ESP32 gửi topic iot/device/status/light với trạng thái "true" cho thấy ESP32 đã gửi đúng trạng thái đến MQTT broker. Khi nhấn bật máy bơm trên giao diện website, lệnh điều khiển máy bơm được server gửi qua topic iot/device/control/pump, ESP32 nhận lệnh và bật máy bơm, sau đó ESP32 gửi lại trạng thái của máy bơm đến MQTT broker để server xử lý và cập nhật giao diện website. Tóm lại, kết nối với MQTT broker hoạt động ổn định, dữ liệu gửi lên MQTT theo định dạng JSON một cách chính xác, các lệnh điều khiển được xử lý một cách nhanh chóng.




**Hình 4.9:** Dữ liệu trên MQTT broker

### 4.3.2. Phần mềm

**a. Hiển thị dữ liệu theo thời gian thực.**

Các thông số nhiệt độ, độ ẩm, pH, hàm lượng NPK được cập nhật một cách nhanh chóng và chính xác từ MQTT broker mỗi khi có dữ liệu mới. Dữ liệu lượng mưa hôm nay và lượng mưa tháng trước được lấy từ Open-Meteo cũng được cập nhật liên tục. Toàn bộ dữ liệu được lưu vào cơ sở dữ liệu PostgreSQL ở dạng bảng để thuận tiện cho việc quản lý. Người dùng có thể xem toàn bộ dữ liệu ở bảng Lịch sử, ngoài ra còn có thể lọc và xuất dữ liệu thành tệp Excel để thống kê và báo cáo.


**Hình 4.10:** Giao diện giám sát dữ liệu

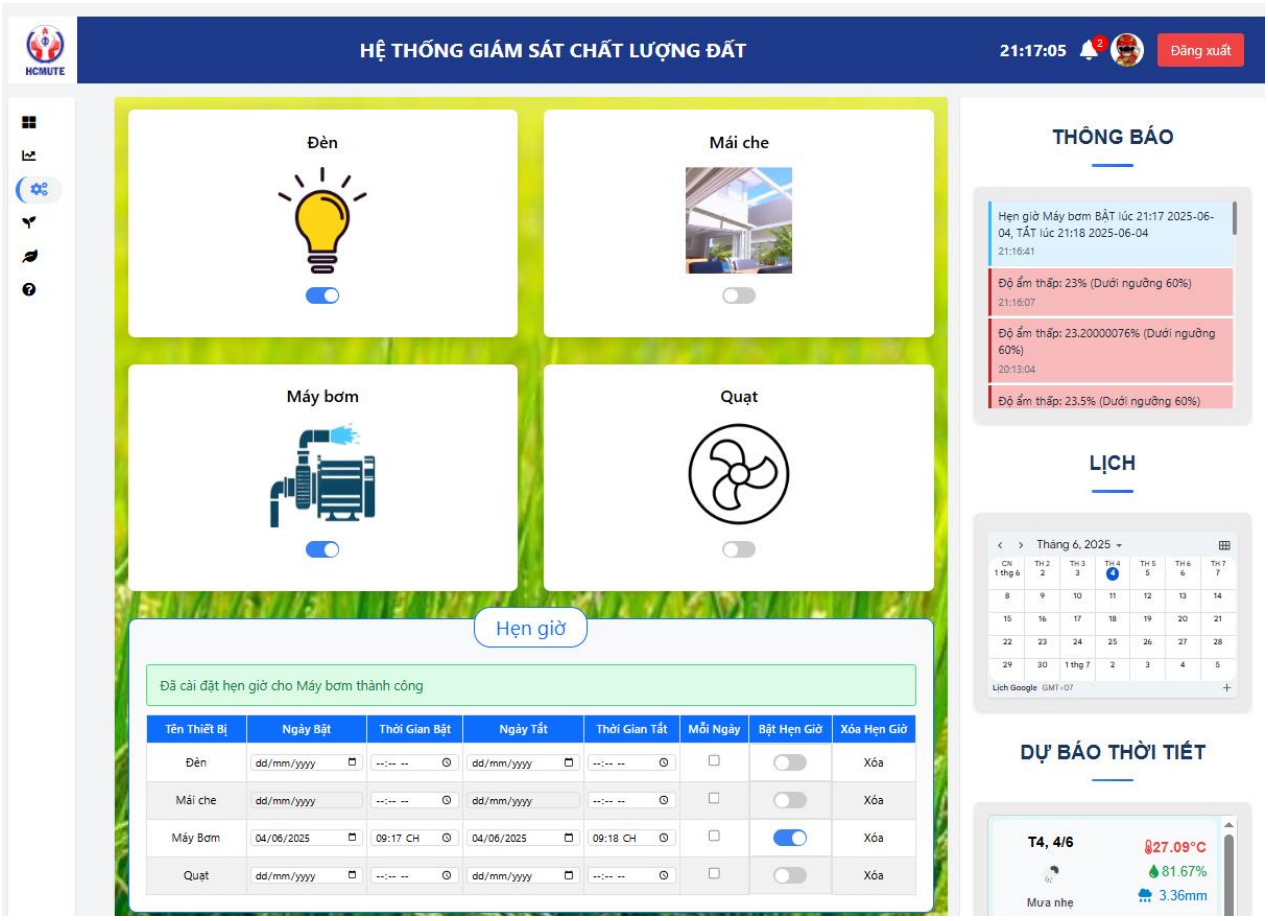
**b. Hiển thị và cập nhật định kỳ dữ liệu trên biểu đồ.**

Mỗi thông số được biểu diễn trực quan trên một biểu đồ giúp người dùng quan sát sự thay đổi của các thông số một cách dễ dàng. Dữ liệu trên biểu đồ được cập nhật mỗi 5 phút. Người dùng có thể sử dụng bộ lọc để xem biểu đồ ở khoảng thời gian mong muốn một cách thuận tiện.


**Hình 4.11:** Giao diện biểu đồ

### c. Điều khiển và hẹn giờ thiết bị

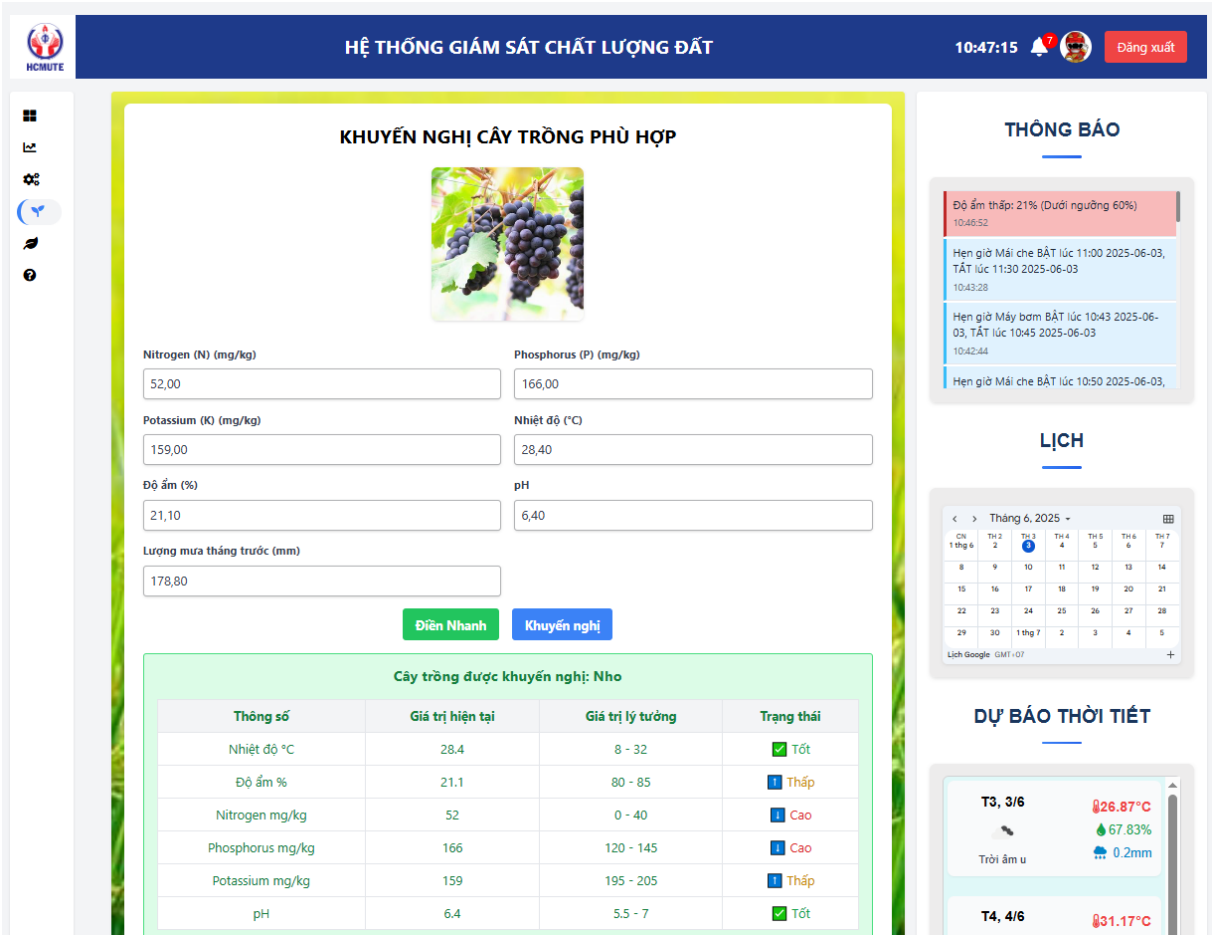
Các thiết bị được điều khiển một cách dễ dàng bằng cách kéo thanh trượt. Khi kéo thanh trượt để bật đèn, lệnh điều khiển được gửi đến MQTT broker, sau đó ESP32 nhận lệnh từ MQTT broker để điều khiển đèn và phản hồi lại trạng thái của đèn nếu thành công, giao diện cập nhật trạng thái đèn đang bật. Nếu không kết nối MQTT thành công hoặc không có phản hồi từ ESP32 thì sẽ báo lỗi không điều khiển được giúp hệ thống hoạt động một cách chính xác. Đến với tính năng hẹn giờ, tiến hành cài đặt thời gian bật máy bơm lúc 21:17, khi đến thời gian lệnh điều khiển được gửi từ server đến MQTT broker, ESP32 nhận lệnh và điều khiển thiết bị sau đó phản hồi trạng thái về MQTT broker, server đọc trạng thái từ MQTT broker và giao diện được cập nhật đúng với trạng thái của máy bơm. Tóm lại, tính năng điều khiển và hẹn giờ thiết bị hoạt động chính xác và ổn định.




**Hình 4.12:** Giao diện điều khiển thiết bị

**d. Đưa ra khuyến nghị cây trồng phù hợp.**

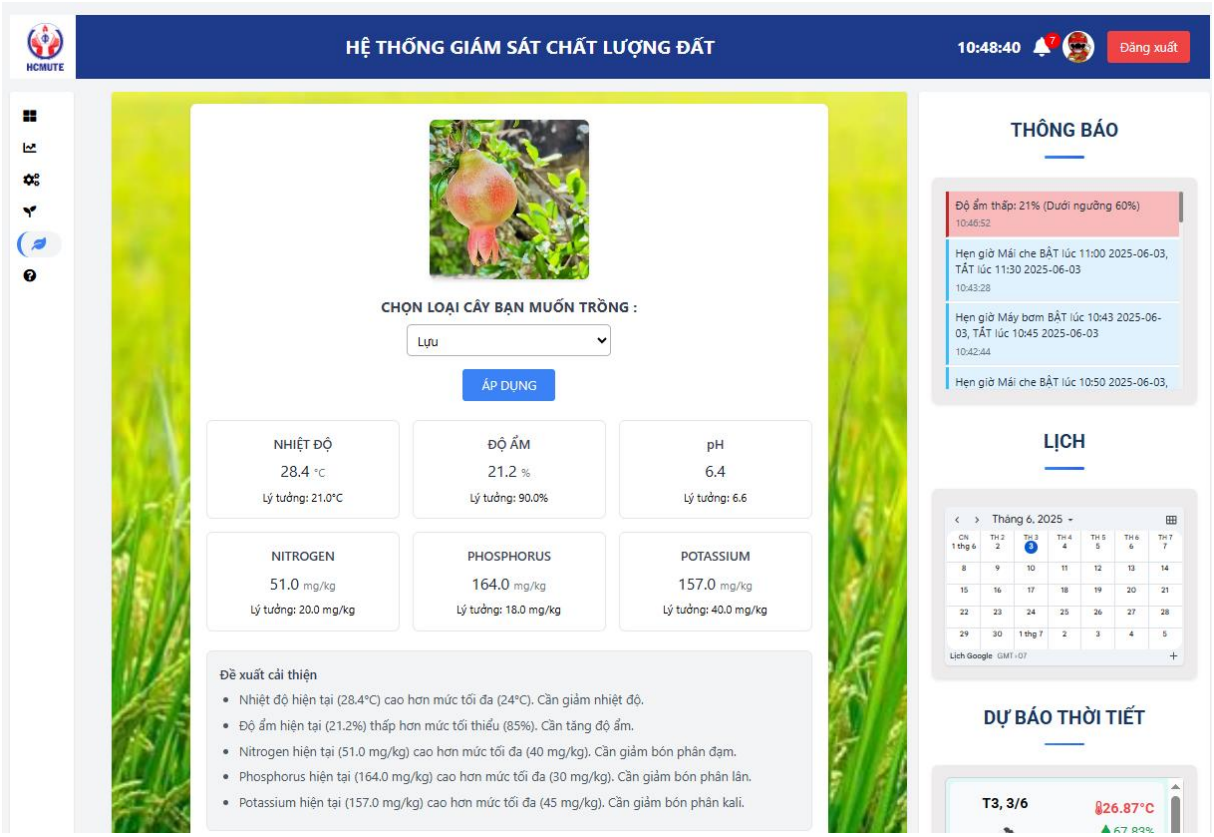
Tiến hành điền các thông số vào các ô N, P, K, nhiệt độ, độ ẩm, lượng mưa tháng trước với dữ liệu lần lượt là 52 (mg/kg); 166 (mg/kg); 159 (mg/kg); 28,4 (°C); 21,1 (%); 6,4; 178,8 (mm). Ấn nút Khuyến nghị, kết quả cây trồng phù hợp là cây nho cùng với thông số lý tưởng và các đề xuất để chăm sóc cây nho.




**Hình 4.13:** Giao diện khuyến nghị cây trồng

**e. Đề xuất thông số phù hợp cho cây trồng được lựa chọn**.

Người dùng lựa chọn loại cây muốn trồng và nhấn nút áp dụng để nhận được các đề xuất chăm sóc phù hợp với loại cây muốn trồng. Khi dữ liệu từ cảm biến vượt khoảng lý tưởng cho loại cây đã chọn thì giao diện sẽ hiện các cảnh báo. Thử nhấn chọn loại cây muốn trồng là cây lựu, hệ thống cho kết quả độ pH hiện tại là phù hợp, còn nhiệt độ, độ ẩm, NPK nằm ngoài khoảng lý tưởng cho cây lựu và đưa ra các đề xuất như giảm bón phân lân, phân đạm,…




**Hình 4.14:** Giao diện lựa chọn cây trồng

### f. Chatbot thông minh

Tiến hành gửi lệnh !sensor trong kênh chat, bot lập tức phản hồi dữ liệu cảm biến mới nhất được lấy từ cơ sở dữ liệu. Tiếp tục nhập lệnh !device và ấn bật đèn, bot gửi lệnh điều khiển đèn đến MQTT broker và phản hồi khi điều khiển thành công. Thử hẹn giờ bật quạt sau 5 phút bằng lệnh !timer, bot đếm ngược thời gian điều khiển quạt và phản hồi người dùng khi điều khiển thành công. Cuối cùng là thử nhận khuyến nghị cây trồng phù hợp bằng lệnh !predict, bot sẽ gửi phản hồi cây trồng phù hợp với dữ liệu hiện tại.


**Hình 4.15:** Giao diện chatbot

# CHƯƠNG 5: KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

## 5.1. KẾT LUẬN

Nhờ có giúp đỡ nhiệt tình và chu đáo từ giảng viên hướng dẫn, về cơ bản chúng tôi đã hoàn thành được những mục tiêu được đề ra là thiết kế hệ thống giám sát môi trường đất và đề xuất cây trồng phù hợp dựa trên dữ liệu thu thập được. Trong quá trình thực hiện tuy gặp không ít khó khăn nhưng nhờ vậy chúng tôi học được tính kiên trì và khả năng tự học, tích lũy thêm kinh nghiệm và kỹ năng mới như kỹ năng lên kế hoạch, kỹ năng thiết kế mạch, kỹ năng hàn mạch…

Tuy nhiên, trong quá trình thực hiện đề tài vẫn còn một số vấn đề chưa được giải quyết do hạn chế về thời gian và điều kiện triển khai. Hệ thống hiện tại lấy dữ liệu lượng mưa từ các trang dự báo thời tiết làm cho dữ liệu lượng mưa không chính xác tuyệt đối cho vị trí cụ thể nơi đặt hệ thống. Hệ thống chưa có nguồn dự phòng dẫn đến nguy cơ bị gián đoạn khi xảy ra sự cố mất điện. Ngoài ra, mã nguồn chương trình chưa được tối ưu, có thể gây chậm trễ trong quá trình xử lý dữ liệu khi có nhiều người sử dụng.

## 5.2. HƯỚNG PHÁT TRIỂN

Trước tiên cần cải thiện hạn chế về dữ liệu lượng mưa, nhóm dự kiến tích hợp cảm biến đo lưu lượng mưa để cải thiện độ chính xác cho hệ thống khi khuyến nghị cây trồng tại vị trí triển khai. Thứ hai, hệ thống cần được tích hợp thêm nguồn dự phòng như năng lượng mặt trời để có thể hoạt động khi bị mất điện. Cuối cùng, cần tối ưu hoá mã nguồn để hệ thống hoạt động một cách ổn định và hiệu quả hơn.

Xu hướng phát triển của công nghệ nông nghiệp thông minh đang mở ra nhiều cơ hội để mở rộng và nâng cấp hệ thống. Đề tài này có thể là nền tảng để xây dựng một hệ thống quản lý nông nghiệp toàn diện hơn – không chỉ dừng lại ở việc giám sát môi trường đất và đề xuất cây trồng mà còn có thể mở rộng sang việc tự động hóa hệ thống tưới tiêu, giám sát sâu bệnh, dự báo mùa vụ để tăng năng suất cây trồng. Việc ứng dụng trí tuệ nhân tạo, học máy và thị giác máy tính là rất cần thiết để hướng tới phát triển một hệ thống nông nghiệp thông minh toàn diện. Để thực hiện điều đó, cần thay thế ESP32

bằng Raspberry Pi để có thể xử lý dữ liệu trực tiếp mà không cần phụ thuộc máy chủ từ xa. Bên cạnh đó, có thể tích hợp thêm mô hình học sâu để cải thiện khả năng phân tích dữ liệu và đưa ra khuyến nghị chính xác hơn.

# TÀI LIỆU THAM KHẢO

## Trang web:


[1] Dutta, A. (2025). *LightGBM – Light Gradient Boosting Machine*. [online] GeeksforGeeks. Đăng tại: https://www.geeksforgeeks.org/lightgbm-light-gradient-boosting-machine/ [Truy cập ngày 12/03/2025].

[2] Nguyễn Cúc (2025). *Hướng dẫn đầy đủ về cách sử dụng LightGBM dành cho người mới*. [online] FUNiX. Đăng tại: https://funix.edu.vn/chia-se-kien-thuc/huong-dan-day-du-ve-cach-su-dung-lightgbm/ [Truy cập ngày 27/02/2025].

[3] EPCB Việt Nam, *Cảm biến đất 7 trong 1 ES-SOIL 7-IN-1 RS485 Modbus RTU*. [online] Đăng tại: https://epcb.vn/products/cam-bien-dat-7-trong-1-es-soil-7-in-1-rs485-modbus-rtu [Truy cập ngày 18/02/2025].

[4] PYWORLD Việt Nam, *ESP32 DevKit V1*. [online] Đăng tại: https://surl.li/tvjdlo [Truy cập ngày 18/02/2025].

[5] Thegioiic, *Module 1 Relay 5V Kích Mức Cao/Thấp*. [online] Đăng tại: https://www.thegioiic.com/module-1-relay-5v-kich-muc-cao-thap [Truy cập ngày 23/02/2025].

[6] Nshop Việt Nam, *Động cơ bơm 365 12VDC*. [online] Đăng tại: https://nshopvn.com/product/dong-co-bom-365-12vdc/ [Truy cập ngày 10/03/2025].

[7] Nshop Việt Nam, *Module giao tiếp TTL RS485*. [online] Đăng tại: https://nshopvn.com/product/module-giao-tiep-ttl-rs485/ [Truy cập ngày 15/03/2025].

[8] SEMTECH, *SX1278 datasheet*. [online] Alldatasheet. Đăng tại: https://www.alldatasheet.com/datasheet-pdf/pdf/800241/SEMTECH/SX1278.html [Truy cập ngày 15/03/2025].

[9] Nshop Việt Nam, *Động cơ giảm tốc JGB37-520 12V 66rpm*. [online] Đăng tại: https://nshopvn.com/product/dong-co-giam-toc-jgb37-520-12v-66rpm/ [Truy cập ngày 10/03/2025].

[10] Peña, E. and Legaspi, M.G. (2020). *UART: A Hardware Communication Protocol*. [online] Analog Devices. Đăng tại: https://www.analog.com/en/resources/analog-dialogue/articles/uart-a-hardware-communication-protocol.html#author [Truy cập ngày 20/03/2025].

[11] PTVinh (2023). *Tìm hiểu về chuẩn truyền thông Modbus và các dạng của nó*. [online]. Đăng tại: Giao thức Modbus là gì và các dạng của nó?

[12] Mesidas, *MQTT*. [online] Đăng tại: https://mesidas.com/mqtt/ [Truy cập ngày 25/02/2025].

[13] Harshit Gupta (2023), *Crop Recommendation with 99% Accuracy*. [online] Kaggle. Đăng tại: https://www.kaggle.com/code/casper6290/agriculturerecommendation-99-acc/notebook [Truy cập ngày 02/03/2025].

# PHỤ LỤC

**Mã nguồn ESP32:**

```
#include <WiFi.h>
#include <PubSubClient.h>
#include <WiFiClientSecure.h>
#include <SoftwareSerial.h>
#include <ArduinoJson.h>
#include <Adafruit_GFX.h>
#include <Adafruit_ILI9341.h>
#include <SPI.h>

#define RO_PIN 32
#define DI_PIN 33
#define DE_PIN 25
#define RE_PIN 26

#define LIGHT_PIN 27
#define PUMP_PIN 14
#define FAN_PIN 13

#define LIGHT_BTN_PIN 5
#define ROOF_BTN_PIN 4
#define PUMP_BTN_PIN 17
#define FAN_BTN_PIN 16

#define TFT_CS   18
#define TFT_RST  19
#define TFT_DC   21
#define TFT_MOSI 22
#define TFT_SCLK 23
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);

#define UPDATE_INTERVAL 5000
#define SERIAL_BAUD 115200
#define NPK_BAUD 4800
#define BUFFER_SIZE 256

const char* SSID = "Tan Dang";
const char* PASSWORD = "hoicauut";
const char* MQTT_SERVER =
"1f64ac3cc86c445ea655a706dc9c7a90.s1.eu.hivemq.cloud";
const int MQTT_PORT = 8883;
const char* MQTT_USERNAME = "A-IoT";
```

```
const char* MQTT_PASSWORD = "AIoT2025";

SoftwareSerial npkSerial(RO_PIN, DI_PIN);
WiFiClientSecure espClient;
PubSubClient client(espClient);

const byte QUERY_DATA[] = {0x01, 0x03, 0x00, 0x00, 0x00, 0x07, 0x04, 0x08};
const size_t RESPONSE_SIZE = 19;

const int IN1 = 2;
const int IN2 = 15;
const int limitSwitchClose = 35;
const int limitSwitchOpen = 34;

bool lightStatus = false;
bool pumpStatus = false;
bool isOpening = false;
bool isRunning = false;
bool fanStatus = false;

const unsigned long DEBOUNCE_DELAY = 50;
const unsigned long DISPLAY_INTERVAL = 1000;
unsigned long lastDisplayUpdate = 0;

struct ButtonState {
    bool lastReading;
    bool state;
    bool lastState;
    unsigned long lastDebounceTime;
    bool changed;
};

ButtonState buttons[4] = {
    {HIGH, HIGH, HIGH, 0, false}, // Đèn
    {HIGH, HIGH, HIGH, 0, false}, // Mái che
    {HIGH, HIGH, HIGH, 0, false}, // Máy bơm
    {HIGH, HIGH, HIGH, 0, false}  // Quạt
};

struct LimitSwitchState {
    bool lastReading;
    bool state;
    unsigned long lastDebounceTime;
};
LimitSwitchState limitSwitches[2] = {
```

```
    {HIGH, HIGH, 0}, // Mở
    {HIGH, HIGH, 0}  // Đóng
};

struct SoilData {
    float temperature;
    float humidity;
    uint16_t nitrogen;
    uint16_t phosphorus;
    uint16_t potassium;
    float ph;
};

struct DeviceStatus {
    bool light;
    bool roof;
    bool pump;
    bool fan;
};
DeviceStatus lastDeviceStatus = {false, false, false, false};
SoilData lastData = {0, 0, 0, 0, 0, 0};

void setup() {
    Serial.begin(SERIAL_BAUD);

    SPI.begin(TFT_SCLK, -1, TFT_MOSI, TFT_CS);
    tft.begin();
    tft.setRotation(2);
    tft.fillScreen(ILI9341_BLACK);

    pinMode(DE_PIN, OUTPUT);
    pinMode(RE_PIN, OUTPUT);
    digitalWrite(DE_PIN, LOW);
    digitalWrite(RE_PIN, LOW);

    pinMode(LIGHT_PIN, OUTPUT);
    pinMode(PUMP_PIN, OUTPUT);
    pinMode(FAN_PIN, OUTPUT);
    digitalWrite(LIGHT_PIN, LOW);
    digitalWrite(PUMP_PIN, LOW);
    digitalWrite(FAN_PIN, LOW);

    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(limitSwitchClose, INPUT);
```

```
    pinMode(limitSwitchOpen, INPUT);
    stopMotor();

    pinMode(LIGHT_BTN_PIN, INPUT_PULLUP);
    pinMode(ROOF_BTN_PIN, INPUT_PULLUP);
    pinMode(PUMP_BTN_PIN, INPUT_PULLUP);
    pinMode(FAN_BTN_PIN, INPUT_PULLUP);

    npkSerial.begin(NPK_BAUD);
    setupWiFi();
    espClient.setInsecure();
    client.setServer(MQTT_SERVER, MQTT_PORT);
    client.setCallback(mqttCallback);

    tft.setTextSize(2);
    tft.setTextColor(ILI9341_WHITE);
    tft.setCursor(10, 10);
    tft.println("Initializing...");

    tft.setTextSize(3);
    tft.fillRect(10, 190, 230, 30, ILI9341_BLACK);
    tft.setCursor(10, 190);
    tft.setTextColor(ILI9341_RED);
    tft.printf("LIGHT: OFF");

    tft.fillRect(10, 220, 230, 30, ILI9341_BLACK);
    tft.setCursor(10, 220);
    tft.setTextColor(ILI9341_RED);
    tft.printf("ROOF : CLOSE");

    tft.fillRect(10, 250, 230, 30, ILI9341_BLACK);
    tft.setCursor(10, 250);
    tft.setTextColor(ILI9341_RED);
    tft.printf("PUMP : OFF");

    tft.fillRect(10, 280, 230, 30, ILI9341_BLACK);
    tft.setCursor(10, 280);
    tft.setTextColor(ILI9341_RED);
    tft.printf("FAN  : OFF");
}

void setupWiFi() {
    WiFi.begin(SSID, PASSWORD);
    Serial.printf("Connecting to %s", SSID);
```

```
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
        Serial.print(".");
    }
    Serial.printf("\nConnected, IP: %s\n", WiFi.localIP().toString().c_str());
}
// Nhận lệnh từ MQTT
void mqttCallback(char* topic, byte* payload, unsigned int length) {
    String message = String((char*)payload, length);
    Serial.printf("Message arrived [%s]: %s\n", topic, message.c_str());
    StaticJsonDocument<256> doc;
    DeserializationError error = deserializeJson(doc, payload, length);
    if (error) {
        Serial.printf("Failed to parse JSON: %s\n", error.c_str());
        return;
    }
    // Xử lý tin nhắn test kết nối
    if (strstr(topic, "iot/test") != NULL) {
        String clientId = doc["clientId"].as<String>();
        String timestamp = doc["timestamp"].as<String>();

        StaticJsonDocument<128> response;
        response["type"] = "connection_response";
        response["clientId"] = clientId;
        response["deviceId"] = "ESP32";
        response["timestamp"] = timestamp;
        response["status"] = "connected";
        char responseBuffer[128];
        size_t n = serializeJson(response, responseBuffer);
        client.publish("iot/test_response", responseBuffer, n);
        Serial.printf("Sent test response: %s\n", responseBuffer);
        return;
    }
    // Xử lý yêu cầu trạng thái
    if (strstr(topic, "iot/device/status_request/") != NULL) {
        String device = String(topic).substring(String(topic).lastIndexOf('/')
+ 1);
        StaticJsonDocument<64> response;
        response["status"] = false;

        if (device == "light") {
            response["status"] = lightStatus;
        } else if (device == "roof") {
            response["status"] = isOpening;
        } else if (device == "pump") {
```

```
            response["status"] = pumpStatus;
        } else if (device == "fan") {
            response["status"] = fanStatus;
        }

        char responseBuffer[64];
        size_t n = serializeJson(response, responseBuffer);
        String responseTopic = "iot/device/status_response/" + device;
        client.publish(responseTopic.c_str(), responseBuffer, n);
        Serial.printf("Sent status response for %s: %s\n", device.c_str(),
responseBuffer);
    }
    // Xử lý lệnh điều khiển thiết bị
    else if (strstr(topic, "iot/device/control/") != NULL) {
        String device = String(topic).substring(String(topic).lastIndexOf('/')
+ 1);
        bool status = doc["status"].as<bool>();
        if (device == "light") {
            digitalWrite(LIGHT_PIN, status ? HIGH : LOW);
            lightStatus = status;
            Serial.printf("Light turned %s\n", status ? "ON" : "OFF");
        }
        else if (device == "roof") {
            if (status && !isRunning) {
                motorOpen();
                isOpening = true;
                isRunning = true;
                Serial.println("Roof opening");
            } else if (!status && !isRunning) {
                motorClose();
                isOpening = false;
                isRunning = true;
                Serial.println("Roof closing");
            }
        }
        else if (device == "pump") {
            digitalWrite(PUMP_PIN, status ? HIGH : LOW);
            pumpStatus = status;
            Serial.printf("Pump turned %s\n", status ? "ON" : "OFF");
        }
        else if (device == "fan") {
            digitalWrite(FAN_PIN, status ? HIGH : LOW);
            fanStatus = status;
            Serial.printf("Fan turned %s\n", status ? "ON" : "OFF");
        }
```

```
        // Gửi cập nhật trạng thái
        StaticJsonDocument<64> statusUpdate;
        statusUpdate["status"] = status;
        char statusBuffer[64];
        size_t n = serializeJson(statusUpdate, statusBuffer);
        String statusTopic = "iot/device/status/" + device;
        client.publish(statusTopic.c_str(), statusBuffer, n);
        Serial.printf("Published status update for %s: %s\n", device.c_str(),
statusBuffer);
        updateDeviceStatus();
    }
}
// Kết nối lại MQTT
void reconnectMQTT() {
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        if (client.connect("ESP32Client", MQTT_USERNAME, MQTT_PASSWORD)) {
            Serial.println("connected");
            // Đăng ký các topic
            client.subscribe("iot/device/control/light");
            client.subscribe("iot/device/control/roof");
            client.subscribe("iot/device/control/pump");
            client.subscribe("iot/device/control/fan");

            client.subscribe("iot/device/status_request/light");
            client.subscribe("iot/device/status_request/roof");
            client.subscribe("iot/device/status_request/pump");
            client.subscribe("iot/device/status_request/fan");

            client.subscribe("iot/test");
            client.subscribe("esp32/client");

            publishAllDeviceStatuses();
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" retrying in 5 seconds");
            delay(5000);
        }
    }
}
// Đọc cảm biến đất
bool readNPKData(SoilData& data) {
    digitalWrite(DE_PIN, HIGH);
    digitalWrite(RE_PIN, HIGH);
```

```
    delay(10);
    npkSerial.write(QUERY_DATA, sizeof(QUERY_DATA));
    npkSerial.flush();
    digitalWrite(DE_PIN, LOW);
    digitalWrite(RE_PIN, LOW);
    delay(100);
    if (npkSerial.available() < RESPONSE_SIZE) {
        Serial.println("Error: NPK sensor timeout or insufficient data");
        return false;
    }
    byte response[RESPONSE_SIZE];
    npkSerial.readBytes(response, RESPONSE_SIZE);
    if (response[0] != 0x01) {
        Serial.println("Error: Invalid response from NPK sensor");
        return false;
    }
    data.humidity = ((response[3] << 8) | response[4]) / 10.0;
    data.temperature = ((response[5] << 8) | response[6]) / 10.0;
    data.nitrogen = (response[11] << 8) | response[12];
    data.phosphorus = (response[13] << 8) | response[14];
    data.potassium = (response[15] << 8) | response[16];
    data.ph = ((response[9] << 8) | response[10]) / 10.0;
    return true;
}
// Gửi dữ liệu lên MQTT và cập nhật TFT
void publishData(const SoilData& data) {
    StaticJsonDocument<256> doc;
    doc["temperature"] = data.temperature;
    doc["humidity"] = data.humidity;
    doc["nitrogen"] = data.nitrogen;
    doc["phosphorus"] = data.phosphorus;
    doc["potassium"] = data.potassium;
    doc["ph"] = data.ph;

    char jsonBuffer[BUFFER_SIZE];
    size_t n = serializeJson(doc, jsonBuffer);
    client.publish("iot/sensor/data", (const uint8_t*)jsonBuffer, n, true);
    Serial.printf("Published JSON: %s\n", jsonBuffer);
    Serial.printf("T: %.2f, H: %.2f, N: %u, P: %u, K: %u, pH: %.2f\n",
                  data.temperature, data.humidity, data.nitrogen,
                  data.phosphorus, data.potassium, data.ph);
    // Cập nhật dữ liệu trên màn hình
    if (abs(data.temperature - lastData.temperature) >= 0.1) {
        tft.fillRect(10, 10, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 10);
```

```
        tft.setTextColor(ILI9341_YELLOW, ILI9341_BLACK);
        tft.setTextSize(3);
        tft.printf("T: %.1f", data.temperature);
        tft.print(" C");
        lastData.temperature = data.temperature;
    }
    if (abs(data.humidity - lastData.humidity) >= 0.1) {
        tft.fillRect(10, 40, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 40);
        tft.setTextColor(ILI9341_CYAN, ILI9341_BLACK);
        tft.setTextSize(3);
        tft.printf("H: %.1f", data.humidity);
        tft.print("%");
        lastData.humidity = data.humidity;
    }
    if (abs(data.nitrogen - lastData.nitrogen) >= 1) {
        tft.fillRect(10, 70, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 70);
        tft.setTextColor(ILI9341_WHITE);
        tft.setTextSize(3);
        tft.printf("N: %u mg/kg", data.nitrogen);
        lastData.nitrogen = data.nitrogen;
    }
    if (abs(data.phosphorus - lastData.phosphorus) >= 1) {
        tft.fillRect(10, 100, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 100);
        tft.setTextColor(ILI9341_WHITE);
        tft.printf("P: %u mg/kg", data.phosphorus);
        lastData.phosphorus = data.phosphorus;
    }
    if (abs(data.potassium - lastData.potassium) >= 1) {
        tft.fillRect(10, 130, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 130);
        tft.setTextColor(ILI9341_WHITE);
        tft.printf("K: %u mg/kg", data.potassium);
        lastData.potassium = data.potassium;
    }
    if (abs(data.ph - lastData.ph) >= 0.1) {
        tft.fillRect(10, 160, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 160);
        tft.setTextColor(ILI9341_MAGENTA);
        tft.setTextSize(3);
        tft.printf("PH: %.1f", data.ph);
        lastData.ph = data.ph;
    }
```

```
}
// Điều khiển motor
void motorOpen() {
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
}
void motorClose() {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, HIGH);
}
void stopMotor() {
    digitalWrite(IN1, LOW);
    digitalWrite(IN2, LOW);
}
// Đọc công tắc hành trình
bool readLimitSwitch(uint8_t pin, LimitSwitchState &ls) {
    bool reading = digitalRead(pin);
    if (reading != ls.lastReading) {
        ls.lastDebounceTime = millis();
    }
    if ((millis() - ls.lastDebounceTime) > DEBOUNCE_DELAY) {
        ls.state = reading;
    }
    ls.lastReading = reading;
    return ls.state;
}
// Xử lý nút nhấn
void handleButton(int index, uint8_t btnPin, uint8_t devicePin, bool
&deviceStatus, const char* mqttTopic) {
    bool reading = digitalRead(btnPin);
    ButtonState &btn = buttons[index];
    if (reading != btn.lastReading) {
        btn.lastDebounceTime = millis();
        btn.changed = false;
    }
    if ((millis() - btn.lastDebounceTime) > DEBOUNCE_DELAY) {
        if (reading != btn.state) {
            btn.state = reading;
            if (btn.state == LOW && !btn.changed) {
                if (index == 1) { // Mái che
                    if (!isRunning) {
                        if (isOpening) {
                            motorClose();
                            isOpening = false;
                            isRunning = true;
```

```
                        } else {
                            motorOpen();
                            isOpening = true;
                            isRunning = true;
                        }
                        if (client.connected()) {
                            StaticJsonDocument<64> doc;
                            doc["status"] = isOpening;
                            char buffer[64];
                            size_t n = serializeJson(doc, buffer);
                            client.publish(mqttTopic, buffer, n);
                            Serial.printf("Button %d pressed, Roof: %s\n",
index, isOpening ? "OPENING" : "CLOSING");
                        }
                    }
                } else { // Đèn hoặc máy bơm hoặc quạt
                    deviceStatus = !deviceStatus;
                    digitalWrite(devicePin, deviceStatus ? HIGH : LOW);
                    if (client.connected()) {
                        StaticJsonDocument<64> doc;
                        doc["status"] = deviceStatus;
                        char buffer[64];
                        size_t n = serializeJson(doc, buffer);
                        client.publish(mqttTopic, buffer, n);
                        Serial.printf("Button %d pressed, Device: %s, Status:
%s\n",
                                      index, mqttTopic, deviceStatus ? "ON" :
"OFF");
                    }
                }
                btn.changed = true;
                updateDeviceStatus();
            }
        }
    }
    btn.lastReading = reading;
}
// Cập nhật trạng thái thiết bị lên màn hình
void updateDeviceStatus() {
    if (lightStatus != lastDeviceStatus.light) {
        tft.fillRect(10, 190, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 190);
        tft.setTextColor(lightStatus ? ILI9341_GREEN : ILI9341_RED);
        tft.setTextSize(3);
        tft.printf("LIGHT: %s", lightStatus ? "ON " : "OFF");
```

```
        lastDeviceStatus.light = lightStatus;
    }
    if (isOpening != lastDeviceStatus.roof) {
        tft.fillRect(10, 220, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 220);
        tft.setTextColor(isOpening ? ILI9341_GREEN : ILI9341_RED);
        tft.printf("ROOF : %s", isOpening ? "OPEN " : "CLOSE");
        lastDeviceStatus.roof = isOpening;
    }
    if (pumpStatus != lastDeviceStatus.pump) {
        tft.fillRect(10, 250, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 250);
        tft.setTextColor(pumpStatus ? ILI9341_GREEN : ILI9341_RED);
        tft.printf("PUMP : %s", pumpStatus ? "ON " : "OFF");
        lastDeviceStatus.pump = pumpStatus;
    }
    if (fanStatus != lastDeviceStatus.fan) {
        tft.fillRect(10, 280, 230, 30, ILI9341_BLACK);
        tft.setCursor(10, 280);
        tft.setTextColor(fanStatus ? ILI9341_GREEN : ILI9341_RED);
        tft.printf("FAN  : %s", fanStatus ? "ON " : "OFF");
        lastDeviceStatus.fan = fanStatus;
    }
}
// Gửi trạng thái thiết bị lên MQTT
void publishAllDeviceStatuses() {
    StaticJsonDocument<64> lightDoc;
    lightDoc["status"] = lightStatus;
    char lightBuffer[64];
    size_t n = serializeJson(lightDoc, lightBuffer);
    client.publish("iot/device/status/light", lightBuffer, n);
    Serial.printf("Published light status: %s\n", lightBuffer);

    StaticJsonDocument<64> roofDoc;
    roofDoc["status"] = isOpening;
    char roofBuffer[64];
    n = serializeJson(roofDoc, roofBuffer);
    client.publish("iot/device/status/roof", roofBuffer, n);
    Serial.printf("Published roof status: %s\n", roofBuffer);

    StaticJsonDocument<64> pumpDoc;
    pumpDoc["status"] = pumpStatus;
    char pumpBuffer[64];
    n = serializeJson(pumpDoc, pumpBuffer);
    client.publish("iot/device/status/pump", pumpBuffer, n);
```

```
    Serial.printf("Published pump status: %s\n", pumpBuffer);

    StaticJsonDocument<64> fanDoc;
    fanDoc["status"] = fanStatus;
    char fanBuffer[64];
    n = serializeJson(fanDoc, fanBuffer);
    client.publish("iot/device/status/fan", fanBuffer, n);
    Serial.printf("Published fan status: %s\n", fanBuffer);
}
// Chương trình chính
void loop() {
    static unsigned long lastUpdate = 0;
    if (!client.connected()) {
        reconnectMQTT();
    }
    client.loop();

    handleButton(0, LIGHT_BTN_PIN, LIGHT_PIN, lightStatus,
"iot/device/status/light");
    handleButton(1, ROOF_BTN_PIN, 0, isOpening, "iot/device/status/roof");
    handleButton(2, PUMP_BTN_PIN, PUMP_PIN, pumpStatus,
"iot/device/status/pump");
    handleButton(3, FAN_BTN_PIN, FAN_PIN, fanStatus, "iot/device/status/fan");

    if (isRunning) {
        bool openState = readLimitSwitch(limitSwitchOpen, limitSwitches[0]);
        bool closeState = readLimitSwitch(limitSwitchClose, limitSwitches[1]);
        if (!openState && isOpening) {
            stopMotor();
            isRunning = false;
            if (client.connected()) {
                StaticJsonDocument<64> doc;
                doc["status"] = isOpening;
                char buffer[64];
                size_t n = serializeJson(doc, buffer);
                client.publish("iot/device/status/roof", buffer, n);
                Serial.println("Roof fully opened");
            }
            updateDeviceStatus();
        } else if (!closeState && !isOpening) {
            stopMotor();
            isRunning = false;
            if (client.connected()) {
                StaticJsonDocument<64> doc;
                doc["status"] = isOpening;
```

```
                char buffer[64];
                size_t n = serializeJson(doc, buffer);
                client.publish("iot/device/status/roof", buffer, n);
                Serial.println("Roof fully closed");
            }
            updateDeviceStatus();
        }
    }

    if (millis() - lastUpdate >= UPDATE_INTERVAL) {
        SoilData data;
        if (readNPKData(data)) {
            publishData(data);
        } else {
            tft.fillRect(0, 0, 240, 180, ILI9341_BLACK);
            tft.setCursor(10, 10);
            tft.setTextColor(ILI9341_RED);
            tft.setTextSize(2);
            tft.println("NPK Sensor Error");
        }
        lastUpdate = millis();
    }

    if (millis() - lastDisplayUpdate >= DISPLAY_INTERVAL) {
        updateDeviceStatus();
        lastDisplayUpdate = millis();
    }
}
```

**Mã nguồn học máy:**

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split, cross_val_score
from sklearn.preprocessing import StandardScaler, LabelEncoder
from lightgbm import LGBMClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt
import joblib
# Đọc dữ liệu
data = pd.read_csv('/kaggle/input/smart-agricultural-production-optimizing-
engine/Crop_recommendation.csv')

print("Thông tin dữ liệu:")
print(data.info())
print("\n5 dòng đầu tiên của dữ liệu:")
print(data.head())

# Tiền xử lý dữ liệu
X = data.drop('label', axis=1)
y = data['label']

label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

X_train, X_test, y_train, y_test = train_test_split(X, y_encoded,
test_size=0.2, random_state=42)

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

# Khởi tạo và huấn luyện mô hình
model = LGBMClassifier(
    n_estimators=200,
    learning_rate=0.05,
    max_depth=10,
    random_state=42
)
model.fit(X_train_scaled, y_train)

# Dự đoán trên tập kiểm tra
y_pred = model.predict(X_test_scaled)
```

```
# Đánh giá mô hình
accuracy = accuracy_score(y_test, y_pred)
print(f"\nĐộ chính xác trên tập kiểm tra: {accuracy * 100:.2f}%")

# Báo cáo chi tiết
print("\nBáo cáo phân loại:")
print(classification_report(y_test, y_pred,
target_names=label_encoder.classes_))

# Đánh giá bằng cross-validation
cv_scores = cross_val_score(model, X_train_scaled, y_train, cv=5)
print(f"\nĐộ chính xác trung bình từ cross-validation (5-fold):
{cv_scores.mean() * 100:.2f}% ± {cv_scores.std() * 100:.2f}%")

# Vẽ ma trận nhầm lẫn
cm = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(12, 10))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
            xticklabels=label_encoder.classes_,
            yticklabels=label_encoder.classes_)
plt.title('Confusion Matrix')
plt.xlabel('Dự đoán')
plt.ylabel('Thực tế')
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()

# Dự đoán cho một mẫu dữ liệu mới
new_data = pd.DataFrame(
    [[90, 42, 43, 20.879744, 82.002744, 6.502985, 202.935536]],
    columns=['N', 'P', 'K', 'temperature', 'humidity', 'ph', 'rainfall']
)
new_data_scaled = scaler.transform(new_data)
prediction = model.predict(new_data_scaled)
predicted_crop = label_encoder.inverse_transform(prediction)[0]
print(f"\nDự đoán loại cây trồng cho mẫu mới: {predicted_crop}")

# Lưu mô hình
joblib.dump(model, 'lgbm_crop_model.pkl')
joblib.dump(scaler, 'scaler.pkl')
joblib.dump(label_encoder, 'label_encoder.pkl')
print("\nĐã lưu mô hình, scaler và label encoder thành file .pkl")
```