

THÔNG TIN HỘI ĐỒNG CHẤM KHÓA LUẬN TỐT NGHIỆP

Hội đồng chấm khóa luận tốt nghiệp, thành lập theo Quyết định số
ngày của Hiệu trưởng Trường Đại học Công nghệ Thông tin.

LỜI CẢM ƠN

Để hoàn thành được khóa luận tốt nghiệp này trước hết chúng em xin gửi lời cảm ơn chân thành đến quý thầy cô giáo trong khoa Mạng máy tính & Truyền thông của trường Đại học Công nghệ Thông tin đã chỉ dạy, truyền đạt vốn kiến thức quý báu và kỹ năng cần thiết cho chúng em trong suốt quá trình học tập tại trường cũng như bước vào kỳ khóa luận này. Chúng em kính chúc mọi người sức khỏe, hạnh phúc và thành công trên con đường sự nghiệp giảng dạy.

Đặc biệt, chúng em xin gửi lời cảm ơn sâu sắc đến Tiến sĩ Lê Kim Hùng và Thạc sĩ Nguyễn Huỳnh Quốc Việt đã gắn bó và hướng dẫn tận tình chúng em từ Đồ án chuyên ngành đến Khóa luận tốt nghiệp. Trong quá trình học tập và làm việc với thầy, chúng em đã nhận được nhiều sự quan tâm giúp đỡ, những kiến thức học thuật bổ ích và quan trọng hết là tâm huyết của thầy dành cho khóa luận của chúng em. Mặc dù có đôi lúc chúng em còn sai sót trong công việc và làm trễ tiến độ luận văn do kiến thức một phần bị hạn chế, mong thầy thông cảm và bỏ qua cho. Ngoài ra, chúng em xin cảm ơn Thạc sĩ Nguyễn Khánh Thuật đã hỗ trợ ý kiến cho luận văn của chúng em.

Cuối cùng, chúng em cũng xin cảm ơn đến gia đình, người thân, cũng như đến tất cả bạn bè đã động viên và hỗ trợ tụi em trên con đường học tập suốt 4 năm học vừa qua.

Sinh viên

Phạm Nguyễn Hoàng Oanh

Sinh viên

Trương Thúc Khanh

MỤC LỤC

THÔNG TIN HỘI ĐỒNG CHẤM KHÓA LUẬN TỐT NGHIỆP	1
LỜI CẢM ƠN	2
TÓM TẮT KHÓA LUẬN	1
1 TỔNG QUAN	2
1.1 Giới thiệu vấn đề	2
1.1.1 Vấn đề phân loại rác thải sinh hoạt	2
1.1.2 Phân loại sử dụng Internet of Things	6
1.2 Previous work	8
1.3 Mục tiêu nghiên cứu	9
2 CƠ SỞ LÍ THUYẾT	12
2.1 Giới thiệu bài toán phân loại ảnh	12
2.2 Giới thiệu CNN	13
2.3 Giới thiệu TensorFlow và TensorFlow Lite	13
2.4 Giới thiệu công nghệ LoRaWan	14
2.5 LoRa và LoRaWAN khác nhau như thế nào	15
2.6 The Things Networks	15
2.6.1 Network Server	15
2.6.2 Application Server	16
2.6.3 Join Network	16
2.7 LoRa Dragino LG02 dual channel gateway	16
2.8 End-Device	17
2.9 Sử dụng Nodejs và Python Server	18
2.10 Giới thiệu về LSTM và bài toán dự đoán chuỗi dữ liệu đa bước thời gian	19
2.11 Giới thiệu bài toán TSP (Travelling Salesman Problem)	21

2.12	Giới thiệu về Mapbox	22
2.13	Giới thiệu về Thingsboard	22
3	MÔ HÌNH HỆ THỐNG	24
4	XÂY DỰNG THIẾT BỊ TỰ ĐỘNG PHÂN LOẠI RÁC	31
4.1	Xây dựng model phân loại rác dựa trên CNN	31
4.1.1	Tối ưu model để có thể chạy trên được chip nhúng ESP32	31
4.2	Evaluation	32
4.2.1	Data Description	32
4.2.2	Index of Performance	33
4.2.3	Kết quả hiện thực được	33
4.3	Build và kiểm thử model phân loại rác trên device	36
4.3.1	Kiểm thử chương trình phân loại rác trên device	36
4.3.2	Kết quả phân loại khi chạy trên device	36
4.4	Setup Gateway	39
4.4.1	Access LG02	39
4.4.2	Cài đặt network	39
4.4.3	Tạo gateway trên TTN Server	42
4.4.4	Configure LG02 Gateway	44
4.5	Setup End-Device	46
4.5.1	Thu gom rác	46
4.5.2	Đăng kí trên TTN Server	47
4.5.3	So sánh khi sử dụng Cayenne LPP và Custom	47
4.5.4	Lưu ý khi nạp code cho node	47
5	XÂY DỰNG MÔ HÌNH SERVER XỬ LÝ VÀ HIỂN THỊ DỮ LIỆU RÁC	49
5.1	Nodejs Server	49
5.1.1	Giới thiệu tổng quát	49
5.1.2	Modules và packages được sử dụng	49
5.1.3	API	50
5.1.4	Cải thiện hiệu suất với Clustering	60
5.2	Python Server	64
5.2.1	Giới thiệu tổng quát	64
5.2.2	Modules và packages	64

5.2.3	API	65
5.2.4	Đánh giá dữ liệu	67
5.3	Thingsboard	71
5.3.1	Giới thiệu tổng quát	71
5.3.2	Assets và Devices	71
5.3.3	Rule Chains	73
5.3.4	Dashboard	75
6	Khó khăn và hướng phát triển đề tài	79
6.1	Khó khăn	79
6.1.1	Về thiết bị	79
6.1.2	Về server	79
6.2	Hướng phát triển đề tài	80
6.2.1	Về thiết bị	80
6.2.2	Về server	80

DANH MỤC HÌNH

1.1	Sơ đồ chôn lắp	6
2.1	LoRaWAN Technology Stack	15
2.2	Mô hình LoRaWAN Network	16
2.3	LG02 Dragino Gateway	17
2.4	Kit RF thu phát wifi Blue esp32 + lora sx1278 oled heltec	18
2.5	Sơ đồ biểu diễn kiến trúc bên trong tế bào LSTM	20
2.6	Mô hình tổng quan Thingsboard	23
3.1	Mô hình tổng quan	24
3.2	Mô hình device	26
3.3	Mô hình server từ bước khởi tạo đến gửi dữ liệu	28
3.4	Mô hình server từ bước gửi dữ liệu đến xử lý bản đồ	29
4.1	Minh họa kiến trúc mạng đã xây dựng	31
4.2	Biểu đồ thể hiện loss và accuracy của mô hình trong quá trình train	34
4.3	Xác thực tại địa chỉ 10.130.1.1	39
4.4	Scan mạng Wireless	40
4.5	Tham gia vào wifi và nhập mật khẩu	40
4.6	Tắt wifi mặc định	41
4.7	Cấu hình địa chỉ IP của cổng Ethernet (Ubuntu 20.04)	42
4.8	Lấy ID của Gateway	43
4.9	Giao diện Gateway đã được tạo (chưa connect)	43
4.10	Nhập ID vào Frequency plan	44
4.11	Config LG02	45
4.12	Chọn server address thích hợp	45
4.13	Thông số configure	46
5.1	Sơ đồ miêu tả mô hình khởi tạo thiết bị và cảm biến	51
5.2	Sơ đồ miêu tả mô hình gửi dữ liệu tiên đoán và kiểm tra mức rác . .	54

5.3	Sơ đồ miêu tả luồng xuất mảng tọa độ đường đi tối ưu tới thùng rác đầy	56
5.4	Sơ đồ miêu tả luồng tạo dữ liệu giả cho thiết bị ảo	58
5.5	Hiệu suất của Nodejs khi nhận 10000 request không Clustering	62
5.6	Hiệu suất của Nodejs khi nhận 10000 request Clustering	62
5.7	Hiệu suất của Nodejs khi nhận 100 request không Clustering	63
5.8	Hiệu suất của Nodejs khi nhận 100 request Clustering	63
5.9	Sơ đồ mô tả tổng quát mô hình Deep Learning dự đoán lượng rác .	65
5.10	Biểu đồ lượng rác tái chế và không tái chế thu thập vào tháng 6 . .	68
5.11	Giá trị Train loss và Val loss	69
5.12	So sánh actual với predicted của tập test tái chế	70
5.13	So sánh actual với predicted của tập test không tái chế	70
5.14	Biểu đồ mức rác tái chế và không tái chế sau 4 tiếng tiếp theo . . .	71
5.15	Danh sách thùng rác	72
5.16	Danh sách cảm biến	72
5.17	Thuộc tính thùng rác	73
5.18	Mối quan hệ của thùng rác với cảm biến	73
5.19	Root Rule Chain	74
5.20	Delta-calculation Rule Chain	75
5.21	Bản đồ theo dõi vị trí thùng rác	76
5.22	Polygon các tọa độ đường đi trên bản đồ	77
5.23	Danh sách thùng rác hiện có	77
5.24	Giao diện chi tiết của thùng rác thông minh	78
5.25	Biểu đồ thể hiện dữ liệu của cảm biến	78

DANH MỤC BẢNG

1.1	Các loại rác truyền thống	3
1.2	Chức năng	11
2.1	So sánh giữa OTAA và ABP	16
4.1	Phân loại hình ảnh trong bộ dataset	33
4.2	Confusion matrix của model khi thử lại trên tập test	35
4.3	Kết quả phân loại của một số hình ảnh	37
4.4	Một số kết quả sai trong quá trình phân loại	38
4.5	Loại rác	47
5.1	Bảng mô tả định dạng dữ liệu được gửi từ TTN	59
5.2	Tổng quan mô hình mạng LSTM	66
5.3	Cấu trúc file CSV	67

TÓM TẮT KHÓA LUẬN

Hello đây là tóm tắt

Chương 1 TỔNG QUAN

1.1 Giới thiệu vấn đề

1.1.1 Vấn đề phân loại rác thải sinh hoạt

Phân loại rác bao gồm nhận diện, mô tả các loại chất thải dựa trên nguồn gốc và đặc điểm và phân loại chúng vào những danh mục rác khác nhau và theo những phương pháp phân loại khác nhau, có thể diễn ra theo phương thức thủ công tại nhà hoặc được thu gom bởi dịch vụ hoặc được phân loại tự động bằng máy. Mỗi danh mục rác đã phân loại sẽ được xử lý bằng những cách khác nhau, miễn là có lợi cho môi trường và cho cuộc sống người dân. Việc phân loại đóng vai trò thiết yếu trong một hệ thống quản lý và thu gom rác thải, đơn giản và thuận tiện hóa quá trình xử lý rác của các cơ sở thu gom rác, nâng cao ý thức người dân và giúp nhà nước quản lý những rủi ro thiệt hại tới sức khỏe con người và môi trường sống.

Nhìn chung, rác thải sẽ được phân loại làm 3 loại chính, được thể hiện như bảng 1.1

Loại rác	Khái niệm	Nguồn gốc	Ví dụ
Rác hữu cơ	Rác hữu cơ là loại rác thực phẩm sau khi chế biến đồ ăn như rau, củ, quả, Có đặc tính dễ phân hủy và có thể đưa vào tái chế để đưa vào sử dụng cho việc chăm bón và làm thức ăn cho động vật.	Phần bỏ đi của thực phẩm sau khi lấy đi phần chế biến được thức ăn cho con người. Phần thực phẩm hư hỏng hoặc không thể sử dụng cho con người. Các loại hoa, lá, cây cỏ không được con người sử dụng sẽ trở thành rác thải trong môi trường.	Các loại rau, củ quả, trái cây đã bị hư, thối... Cơm canh, thức ăn thừa hoặc bị thiêu.... Các loại bã chè, bã cafe Cỏ cây bị xén/chặt bỏ, hoa rụng... Mica trung quốc
Rác vô cơ	Rác vô cơ là những loại rác không thể sử dụng được nữa cũng không thể tái chế được mà chỉ có thể xử lý bằng cách mang ra các khu chôn lấp rác thải.	Các loại vật liệu xây dựng không thể sử dụng hoặc đã qua sử dụng và được bỏ đi. Các loại bao bì bọc bên ngoài hộp/chai thực phẩm. Các loại túi nilon được bỏ đi sau khi con người dùng đựng thực phẩm. Một số loại vật dụng/thiết bị trong đời sống hàng ngày của con người.	Gạch đá, đồ sanh, sứ vỡ hoặc không còn giá trị sử dụng. Ly, cốc, bình thủy tinh vỡ... Các loại vỏ sò, vỏ ốc, vỏ trứng... Đồ da, đồ cao su, đồng hồ hỏng, băng đĩa nhạc, radio... không thể sử dụng.
Rác tái chế	Rác tái chế là loại rác khó phân hủy nhưng có thể đưa vào tái chế để sử dụng nhằm mục đích phục vụ cho con người.	Các loại giấy thải. Các loại hộp/chai/vỏ lon thực phẩm bỏ đi.	Thùng carton, sách báo cũ. Hộp giấy, bì thư, bưu thiếp đã qua sử dụng. Các loại vỏ lon nước ngọt/lon bia/vỏ hộp trà.... Các loại ghế nhựa, thau/chậu nhựa, quần áo và vải cũ...

Bảng 1.1 Các loại rác truyền thống

Trong 3 loại chính, người ta còn định nghĩa cụ thể những loại rác khác:

- Rác thải văn phòng: những văn phòng phẩm không còn được sử dụng (giấy báo cũ, bút hết mực, ...)
- Rác thải công nghiệp: loại rác có thành phần cực kỳ độc thải ra từ các nhà máy, công xuồng,... (chất ngâm tẩm, tẩy rửa, chất hóa học, phế liệu công nghiệp, thuốc trừ sâu bọ, thuốc kích thích tăng trưởng...)
- Rác thải xây dựng: loại rác được thải ra môi trường từ những công trình xây dựng, sửa chữa, còn được gọi là xà bần (gạch, đá, vụn đất, ...)
- Rác thải y tế: loại rác được thải ra từ các cơ sở y tế, bệnh viện, bệnh xá, được phân loại cụ thể như sau:
 - Chất thải y tế lây nhiễm: gồm các chất thải lây nhiễm sắc nhọn (kim tiêm, luồng dao mổ, đinh, cưa, kim chọc dò, ...) và không sắc nhọn (chất thải thẩm, dính máu hoặc dịch sinh học cơ thể, chất thải phát sinh từ buồng cách ly, ...), chất thải có nguy cơ lây nhiễm cao (mẫu bệnh phẩm, dụng cụ đựng, dính mẫu bệnh phẩm phát sinh từ các phòng xét nghiệm an toàn sinh học ...) và chất thải giải phẫu (mô, bộ phận cơ thể người thải bỏ và xác động vật thí nghiệm)
 - Chất thải y tế nguy hại không lây nhiễm: gồm hóa chất, dược phẩm thải bỏ có các thành phần nguy hại, thiết bị y tế vỡ, hỏng, đã qua sử dụng có chứa thủy ngân hoặc kim loại nặng, chất hàn răng amalgam thải bỏ
 - Chất thải y tế thông thường: gồm chất thải rắn trong sinh hoạt thường ngày của con người, chất thải ngoại cảnh trong cơ sở y tế và chất thải lỏng không nguy hại.

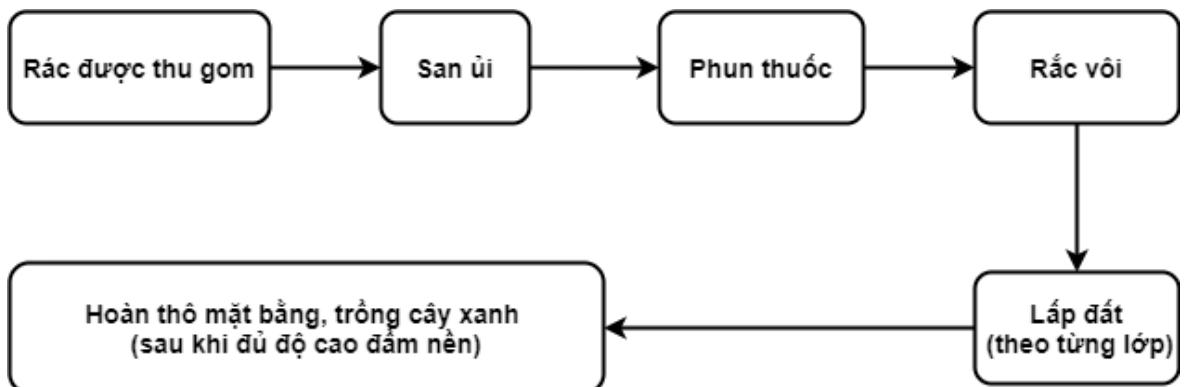
Vì thế, việc phân loại rác mang một vài ý nghĩa quan trọng đối với môi trường tự nhiên và cộng đồng xã hội:

- Nhằm giảm thiểu tổng lượng rác thải từ cộng đồng ra môi trường do đó góp phần làm không khí trong lành và giảm thiểu nguy cơ ô nhiễm môi trường.
- Giảm thiểu nguy cơ phát tán các tác nhân gây bệnh, các yếu tố độc hại, nguy hiểm.
- Hạn chế nước rỉ rác góp phần làm giảm ô nhiễm nguồn đất, nguồn nước ngầm, nước mặt, giảm nhiều diện tích chôn lấp rác sinh hoạt.

- Ngoài ra, phân loại rác còn có ảnh hưởng đến chất lượng cuộc sống của con người đem lại một lượng lớn các sản phẩm tái chế, mang lại hiệu quả kinh tế cho chính người thải rác bằng cách bán các nguyên, phế liệu có thể tái chế được, tận dụng các nguyên liệu hữu cơ sản xuất phân bón vi sinh.
- Góp phần nâng cao ý thức cuộc của cộng đồng về bảo vệ môi trường cũng như sử dụng tài nguyên hợp lý, nhất là ở trẻ nhỏ.
- Xây dựng một xã hội với môi trường xanh-sạch-đẹp.
- Nhằm giảm tải công tác xử lý, nhất là trong phương pháp đốt chất thải, đồng thời có thể lựa chọn phương pháp xử lý chất thải rắn phù hợp nhất.
- Giảm thiểu tổng lượng rác thải ra môi trường tiết kiệm chi phí thu gom, vận chuyển và xử lý.

Trước đây, quá trình phân loại rác thải sẽ do các cơ sở thu gom rác thực hiện nhưng vì tổng lượng rác thải từ nhiều nguồn được thải ra ngày càng nhiều nên việc huy động người dân phân loại rác tại nguồn là cách được sử dụng rộng rãi nhất. Cách phân loại sẽ tùy thuộc vào mỗi địa phương quy định chi tiết, ở Việt Nam sẽ chia thành 3 danh mục: rác vô cơ, rác hữu cơ và rác tái chế. Các cơ sở thu gom rác sẽ tiến hành thu gom tận nơi và vận chuyển đến điểm tập trung, lượng rác được phân loại sẽ được xử lý theo 3 phương pháp sau:

- Chế biến rác thải thành phân compost: chế biến rác hữu cơ dễ phân hủy thành phân compost dùng trong nông nghiệp, chia thành 2 quy mô chế biến:
 - Quy mô chế biến tập trung: Rác hữu cơ dễ phân hủy được tách ly, nghiền, ủ hiếu khí để tạo ra phân vi sinh. Việc thành lập nhà máy chế biến phân compost cần vốn đầu tư lớn, chi phí vận hành cũng tương đối cao.
 - Quy mô chế biến hộ gia đình: Rác hữu cơ dễ phân hủy được ủ thành phân compost ngay trong sân vườn
- Chôn lấp hợp vệ sinh: Rác thải được rải thành từng lớp dưới hố, đầm nén để giảm thể tích và phủ đất lên (phun hóa chất để tăng hiệu quả xử lý nhanh và hạn chế côn trùng) với sơ đồ quy định như hình 1.1:



Hình 1.1 Sơ đồ chôn lấp

- Thiêu đốt: Rác thải được phân hủy ở nhiệt độ cao ($1000 - 1100^{\circ}\text{C}$), giảm đáng kể thể tích chất thải phải chôn lấp (xỉ, tro), tuy nhiên chi phí đầu tư, vận hành nhà máy đốt rác khá cao, phù hợp với các nước tiên tiến, phát triển.

Ngoài ra, một số khu vực khác sẽ có cách phân loại rác theo 4 loại lớn:

- Rác dễ cháy (rác thải từ nhà bếp, giấy vụn, vải quần áo).
- Rác khó cháy (kim loại, thủy tinh, sành sứ).
- Rác tái chế (chai, bình, can, giấy báo).
- Rác cỡ lớn (đồ gia dụng cỡ lớn).

Đồng nghĩa với việc rác thải được phân chia vào từng loại khác nhau, là tính chất của các loại rác cũng sẽ khác nhau. Như vậy, từng loại phải có cách xử lý riêng.

Mỗi địa phương sẽ quy định chi tiết cách phân loại rác khác nhau, nhưng cách truyền thống nhất và được sử dụng rộng rãi nhất là chia rác thành 3 danh mục: rác vô cơ, rác hữu cơ và rác tái chế. Ngoài ra, một số khu vực khác sẽ có cách phân loại rác theo 4 loại lớn: rác dễ cháy (rác thải từ nhà bếp, giấy vụn, vải quần áo), rác khó cháy (kim loại, thủy tinh, sành sứ), rác tái chế (chai, bình, can, giấy báo) và rác cỡ lớn (đồ gia dụng cỡ lớn)

1.1.2 Phân loại sử dụng Internet of Things

Internet of Things (IoT) là một hệ sinh thái (được gọi là “thiết bị kết nối” và “thiết bị thông minh”), nhà cửa và các trang thiết bị khác được nhúng với các bộ phận điện tử, phần mềm, cảm biến, cơ cấu chấp hành cùng với khả năng kết nối mạng máy

tính giúp cho các thiết bị này có thể thu thập và truyền tải dữ liệu Internet of Things đang dần trở nên quen thuộc và trở thành một trong số các trào lưu của nền công nghiệp hóa 4.0. Không khó để có thể bắt gặp những sản phẩm thuộc lĩnh vực này, từ những sản phẩm gần gũi trong đời sống gia đình như đèn điều khiển bằng âm thanh, cửa thông minh...cho đến các sản phẩm giúp ích trong nông nghiệp như hệ thống tưới tiêu cho hoa màu từ xa, theo dõi các yếu tố về môi trường Chưa dừng lại ở đó, khi xã hội càng phát triển, việc kết hợp các phần kiến thức độc lập với nhau thành một thể thống nhất để nâng cao chất lượng của sản phẩm tạo ra là điều hết sức cần thiết. Điều đó được chứng minh rõ ràng qua việc bắt đầu có các sản phẩm IoT sử dụng kỹ thuật Machine Learning được ra đời và ứng dụng vào các lĩnh vực, vấn đề mà xã hội đang quan tâm. Trong đó, bằng cách tạo ra các sản phẩm sử dụng công nghệ tiên tiến nói trên vào mục đích bảo vệ môi trường đang là một trong các ý tưởng thiết thực, đáp ứng nhu cầu của xã hội.

Ở các nước khác, việc cung cấp các loại thùng rác với từng loại rác đã trở nên phổ biến, và yêu cầu mọi người phải tự giác thực hiện việc bỏ rác đúng quy định, đúng thùng rác. Một số ví dụ cụ thể như: Ở Nhật, việc bỏ rác được quy định ngay trên các tấm lịch, người dân cần phân loại rác và bỏ rác đúng loại rác vào ngày được quy định trên lịch nếu trường hợp ở hộ gia đình. Và các thùng rác công cộng cũng được kí hiệu phân loại rác, người dân phải bỏ đúng loại rác vào thùng rác có kí hiệu phân loại. Hoặc ở một số nước khác, như Singapore đã áp dụng các quy định cứng rắn đối với người dân và cả khách du lịch nếu họ vứt rác bừa bãi, mức phạt có thể lên đến 1,000 SGD (17 triệu đồng). Tuy nhiên, ở Việt Nam lại khá khó áp dụng như thế, về khách quan mà nói thì ta dễ dàng nhận thấy hệ thống thùng rác công cộng được bố trí trên đường khá thưa thớt, nên người dân khó tìm được nơi bỏ rác khi tham gia giao thông, và một mặt hạn chế mà chúng ta cần nhìn nhận là có một bộ phận không nhỏ người dân có ý thức chưa tốt, chưa thực hiện bỏ rác đúng nơi quy định cũng như chưa hình thành thói quen phân loại rác. Đa số chúng ta đều để chung tất cả các loại rác thải vào một bao sau đó để nhân viên dọn vệ sinh gồm về nhà máy xử lý. Quy trình và các ảnh hưởng xấu của việc xử lý rác thải truyền thống như đã nêu ở mục trên, vì thế việc thiết kế thùng rác kết hợp IoT tự động phân loại rác và hệ thống quản lý vị trí thùng rác cũng như lượng rác trong mỗi thùng được cập nhật liên tục mỗi giờ sẽ mang lại rất nhiều lợi ích: Giảm bớt quy trình xử lý, giảm công sức lao động, giảm tiền của, và mang tính quản lý chủ động cao.

1.2 Previous work

Trong công bố [1], hai tác giả đã cho rằng:

Dùng thị giác máy tính để phân loại tái chế rác sẽ mang lại nhiều hiệu quả đối với việc xử lý rác thải. Đối tượng của bài toán này là dùng hình ảnh của một loại rác tái chế hoặc rác thải thông thường và phân loại chúng vào 6 lớp: thủy tinh, giấy, kim loại, nhựa, bìa cứng, và rác thải khác. Mindy Yang và Gary Thung cũng tạo ra một dataset bao gồm mỗi lớp khoảng 400 - 500 hình ảnh được thu thập một cách thủ công. Họ dự kiến công khai tập dataset cho cộng đồng. Hai model được sử dụng là: SVM (Support Vector Machine) với SIFT (Scale-Invariant Feature Transform) và CNN (Convolutional Neural Network). Các thử nghiệm của họ chỉ ra rằng SVM có hiệu quả cao hơn CNN. Tuy nhiên, CNN chưa được huấn luyện một cách tốt nhất do gặp nhiều khó khăn trong việc xác định siêu tham số.

Trong công bố [2], hai tác giả đã cho rằng:

Hiện nay, những khu đô thị, thành phố lớn dần được chuyển hóa để trở nên “thông minh” hơn. Bên cạnh một số hệ thống quản lý thông minh phổ biến (giao thông, đèn, năng lượng, ...) thì hệ thống xử lý rác thải thông minh là một phần không thể thiếu cho bất kỳ một thành phố thông minh nào. Một hệ thống rác thải thông minh nhằm vào việc giải quyết lượng rác trong những khu công cộng hoặc ngoại ô. Ngoài việc mất mỹ quan đô thị của một khu vực, rác thải có thể gây ô nhiễm và ảnh hưởng trầm trọng đến chất lượng cuộc sống của người dân trong khu vực đó. Luận văn tập trung vào việc phát triển phần mềm có thể nhận diện lượng rác thải thông qua việc phân tích các luồng video ở thời gian thật. Mô hình mạng cải tiến YOLOv3 được triển khai để thực hiện chức năng phát hiện rác. Hệ thống đã được điều chỉnh thích hợp theo dataset đã được sưu tầm cho mục đích này.

Kết quả cho thấy cách tiếp cận như trên đã đóng góp một phần to lớn vào việc quản lý rác thải hiệu quả hơn ở những thành phố thông minh.

Trong công bố [3], tác giả cho rằng dự báo chuỗi thời gian là một vấn đề quan trọng trong nhiều lĩnh vực, một dự đoán tốt chính là bằng chứng cốt yếu để mọi người đưa ra quyết định.

Đối với những mô hình thực hiện phương pháp dựa trên kiến thức thông kê, chúng hoạt động rất tốt với việc dự đoán một bước tiếp theo, nhưng khi tiến hành dự

đoán nhiều bước tiếp, hiệu suất của một vài mô hình không đạt được như mong đợi. Vì thế, tác giả đã gợi ý sử dụng mô hình mạng LSTM (Long-Short Term Memory) để giải quyết bài toán mà không cần bận tâm đến nhiều bước thủ công trong các phương pháp mô hình hóa truyền thống, chẳng hạn như kiểm tra độ ổn định, hàm tự tương quan, hàm tự tương quan từng phần,...

Khi xét về chuỗi thời gian, có rất nhiều dữ liệu trong cuộc sống hằng ngày của chúng ta có thể được mô hình hóa thành chuỗi thời gian, như chỗ đậu xe còn lại trong bãi đậu xe thay đổi theo giờ, số lượng khách đến bảo tàng mỗi ngày, chi phí điện cho trung tâm mua sắm hằng tháng,... Các chuỗi thời gian khác nhau có các đặc điểm khác nhau, một số có thể có giá trị trung bình xung quanh 0, một số có tính chất tuần hoàn, một số có tính chất tăng/giảm, cho nên xây dựng một mô hình phù hợp với nhiều loại chuỗi dữ liệu khác nhau là một nhiệm vụ khó khăn.

Kết quả bài báo cho thấy mô hình LSTM khá phù hợp với phạm vi rộng lớn các mẫu dữ liệu, nhưng các mẫu dữ liệu trong bài báo tương đối đơn giản và vẫn còn rất nhiều không gian khai thác trong lĩnh vực dự đoán chuỗi thời gian đi trước nhiều bước.

1.3 Mục tiêu nghiên cứu

Năm bắt xu thế trên, bài toán được đặt ra là phải tạo ra sản phẩm vừa đáp ứng được yêu cầu triển khai trên diện rộng, nhằm giảm sức người, tiết kiệm, mà người quản lý dễ kiểm soát, dễ sử dụng, sửa chữa. Từ đó, đề tài về “Xây dựng mô hình thùng rác thông minh dựa trên công nghệ trí tuệ nhân tạo” đã được ra đời để giải quyết bài toán đó. Thùng rác tự động phân loại rác áp dụng kỹ thuật Machine Learning để giảm thiểu thời gian, sức người cũng như các dây chuyển xử lý phân loại rác như trước đây, từ đó tiết kiệm được các khoản chi phí. Tuy nhiên, việc thực hiện hóa đề tài trên đã gặp không ít những khó khăn, để giảm thiểu chi phí thiết kế, cũng như hướng đến việc khả thi khi đưa vào thực tế.

Nhóm đưa ra phương hướng thiết kế phải đảm bảo các tiêu chí: giá thành hợp lý, tiết kiệm điện năng, và có thể sử dụng ở nhiều môi trường (đặc biệt là các khu vực khó cung cấp nguồn điện). Để thực hiện các phương hướng đã đưa ra, nhóm chọn sử dụng mạch ESP32 CAM với giá thành cạnh tranh hơn rất nhiều so với sản phẩm dùng raspberry pi. Ngoài ra, hướng đến nguồn năng lượng tiết kiệm và thân thiện

hơn với môi trường, nhóm sử dụng pin năng lượng mặt trời để nạp vào nguồn pin dự trữ. Như vậy, thùng rác vẫn có thể hoạt động được vào cả ban ngày lẫn ban đêm, và có khả năng nạp nguồn từ việc chuyển hóa từ quang năng sang điện năng để cung cấp cho thùng rác hoạt động.

Khác với bộ xử lý mạnh mẽ có thể lên đến 16GB của raspberry pi cùng với các công cụ hỗ trợ phong phú, thì ESP32 CAM chỉ có bộ xử lý vỏn vẹn 4MB. Cũng vì sự khác biệt đó cộng thêm việc sử dụng Machine Learning vào các thiết bị nhúng vẫn là một xu thế khá mới mẻ, dẫn đến việc nhóm đã gặp rất nhiều khó khăn và thách thức khi thực hiện đề tài này. Đối với các microcontroller như ESP32, việc có thể thực hiện các tác vụ như chụp hình và trả kết quả phân loại rác đã phải mất nhiều thời gian. Thách thức lớn nhất là phải đưa được model đã train vào mạch có bộ nhớ thấp nhưng vẫn đảm bảo độ chính xác ở mức có thể chấp nhận.

Để giải quyết thách thức đó, nhóm em đã sử dụng Google Colab để train model sau đó sử dụng TensorFlow Lite để có thể đưa model đã train vào thiết bị nhúng bằng môi trường Espress IF. Với thiết kế tối giản, thùng rác sẽ vận hành theo các bước như sau:

- Bước 1: Thiết bị sẽ chụp hình vật thể sau khi vật thể đó được để vào khay đựng rác
- Bước 2: Hình ảnh sẽ được chuyển sang một chuỗi byte sau đó áp dụng model đã train để đưa ra kết quả phân loại. Nếu là nhựa, thủy tinh thì sẽ đem tái chế, còn nếu là giấy, và các loại rác khác thì sẽ không tái chế.
- Bước 3: Rác sẽ rơi vào một trong hai khay đựng rác bên trong theo đúng kết quả đã đưa ra.

Như vậy, việc phân loại đã trải qua ba bước như trên. Có thể thấy rằng, so với việc sử dụng raspberry pi, các microcontroller như ESP32 CAM đã giải quyết bài toán một cách nhanh chóng hơn bằng cách tự thực hiện quá trình xử lý ảnh và phân loại thay vì đưa hình lên server để xử lý sau đó trả kết quả ngược về cho raspberry pi. Ngoài ra, nhóm còn sử dụng công nghệ LoRaWan để truyền dữ liệu về mực rác còn lại trong thùng.

Bên cạnh đó, cần xây dựng một nơi để quản lý tất cả thùng rác và data cảm biến, dữ liệu realtime từ thiết bị gửi đến sẽ được xử lý linh hoạt thông qua 3 Backend Server: 1 open-source server từ Thingsboard cung cấp các API liên quan đến việc

quản lý thiết bị, cảm biến (bao gồm tạo, sửa, xóa, ...) và việc hiển thị, lưu trữ dữ liệu trên dashboard, 1 Python server để tiên đoán lượng rác thông qua Deep Learning và 1 Nodejs server để xử lý chính các dữ liệu thiết bị gửi tới và gọi API từ 2 server kia.

Server chính sẽ nhận dữ liệu từ cả thiết bị thật và ảo để xử lý và gửi lên Thingsboard để hiển thị, dữ liệu từ thiết bị thật sẽ được gửi từ gateway The Things Network (TTN) với một format nhất định, vì thế thiết bị ảo cũng sẽ được lập trình để gửi dựa theo format đó. Thiết bị thật và ảo đều có những chức năng giống nhau (tạo, liên kết, gửi dữ liệu, ...), nhưng cách xử lý thì khác nhau, cụ thể theo bảng 1.2

Bảng 1.2 Chức năng

Chức năng	Thiết bị thật	Thiết bị ảo
Tạo thiết bị	Được tạo thủ công trực tiếp trên Thingsboard hoặc gửi request body đến API của Server để tạo tự động	
Tạo cảm biến	Các cảm biến cũng được tạo tự động sau khi thiết bị được tạo và được liên kết mỗi quan hệ tự động với thiết bị	
Gửi dữ liệu	Dữ liệu được gửi thông qua gateway TTN theo format cố định	Dữ liệu được random từ file .csv, tính toán lại và được định dạng theo format của thiết bị thật và được gửi về server

Sau khi dữ liệu được gửi về server thì các chức năng sẽ được xử lý chung:

- Dữ liệu được gửi về sẽ được gửi lên Thingsboard để hiển thị, đồng thời được lưu trữ vào file csv của từng thiết bị.
- Các file csv sẽ được gửi qua Python server để tiên đoán lượng rác sau mỗi khoảng thời gian trước khi quá trình thu gom rác bắt đầu, kết quả sẽ trả về Node server và được xử lý tiếp.

Chương 2 CƠ SỞ LÍ THUYẾT

2.1 Giới thiệu bài toán phân loại ảnh

Phân loại ảnh là một bài toán phổ biến trong lĩnh vực thị giác máy tính. Mục tiêu của bài toán là làm cho máy tính có khả năng xác định nhãn của hình ảnh. Cụ thể hơn bài toán có đầu vào và đầu ra như sau:

- Đầu vào: Ảnh chứa đối tượng cần xác định nhãn và danh sách các nhãn (labels).
- Đầu ra: nhãn tương ứng với đối tượng trong ảnh đầu vào.

Hình minh họa đầu vào và đầu ra của bài toán. Bài toán này được áp dụng trong nhiều lĩnh vực như: phân loại biển báo giao thông, phân loại chữ viết tay,... Các phương pháp giải quyết bài toán được chia làm hai loại: tiếp cận dựa trên máy học và học sâu. Đối với cách tiếp cận dựa trên máy học trước tiên cần phải xác định các đặc trưng từ hình ảnh bằng một số phương pháp như: Scale-invariant feature transform (SIFT), Histogram of oriented gradients (HOG). Sau đó sử dụng thêm một số kỹ thuật phân lớp như thuật toán Support vector machine (SVM) hoặc Decision Tree để phân loại đối tượng. Cách tiếp cận dựa trên học sâu sử dụng kiến trúc mạng Convolution neural network cho cả việc trích xuất đặc trưng và phân loại đối tượng. Ngoài ra ta có thể kết hợp cả hai phương pháp bằng cách sử dụng mạng Convolution neural network để trích xuất đặc trưng sau đó sử dụng các kỹ thuật phân lớp để phân loại đối tượng.

Trong những năm gần đây, sự phát triển của khoa học kỹ thuật và lượng dữ liệu ngày càng lớn đã tạo điều kiện cho các kiến trúc mạng CNN được áp dụng ngày càng nhiều trong việc giải quyết các bài toán phân loại ảnh. Các mô hình này cũng mang lại kết quả cao với thời gian ngắn để đáp ứng yêu cầu áp dụng trong các bài toán real-time của con người. Chính vì vậy chúng tôi đã sử dụng thuật toán CNN cho khóa luận này.

2.2 Giới thiệu CNN

CNN là một kiến trúc mạng bao gồm các lớp: convolution layer + nonlinear layer, pooling layer, fully connected layer liên kết với nhau theo một thứ tự nhất định. Thông thường, ảnh được truyền qua lớp convolution + nonlinear sau đó đến pooling layer. Bộ ba convolution + nonlinear và pooling layer được lặp lại nhiều lần trong mạng. Sau đó các giá trị tính toán được lan truyền qua tầng fully connected và softmax để phân loại đối tượng trong ảnh. Hình minh họa mạng CNN cơ bản.

Trong kiến trúc mạng này ảnh đầu vào được biểu diễn dưới dạng ma trận. Lớp Convolution được sử dụng để phát hiện đặc trưng cụ thể của ảnh, từ các đặc trưng cơ bản như góc, cạnh đến các đặc trưng phức tạp như texture của ảnh. Một ma trận có kích thước nhỏ (3×3 hoặc 5×5) được gọi là filter sẽ lướt qua toàn bộ ảnh từ trái sang phải và từ trên xuống dưới để phát hiện các đặc trưng trong ảnh. Hình ... minh họa cụ thể phép tính convolution.

Kích thước của filter tỉ lệ thuận với số tham số cần học và thường là số lẻ. Sau khi áp dụng phép convolution thì ma trận đầu vào sẽ nhỏ dần dần đến số layer của mô hình CNN bị giới hạn. Vì vậy cần phải lưu ý đến tham số stride, thể hiện số pixel cần phải dịch chuyển mỗi khi trượt filter trên ảnh và thêm padding có giá trị bằng 0 ở phần khung của ảnh. Tương tự như mạng neural network, CNN cũng sử dụng hàm kích hoạt như ReLU hoặc tanh đặt ngay sau tầng convolution. Đối với dữ liệu ảnh hàm kích hoạt thường được sử dụng là ReLU, hàm này gán những giá trị âm bằng 0 và giữ nguyên các giá trị lớn hơn 0.

Pooling layer được xếp sau các lớp convolution để giảm tham số. Các loại pooling được sử dụng chủ yếu là max pooling và average. Các lớp này sẽ được lặp lại theo thứ tự để tạo ra feature map cuối cùng sau đó truyền vào tầng fully connected. Tầng này chuyển ma trận nhận được thành vector và phân loại vector đó tương tự như mạng neural network.

2.3 Giới thiệu TensorFlow và TensorFlow Lite

TensorFlow Lite là giải pháp gọn nhẹ của TensorFlow cho thiết bị di động và thiết bị nhúng. Nó cho phép suy luận học máy trên thiết bị với độ trễ thấp và kích thước nhị phân nhỏ.

2.4 Giới thiệu công nghệ LoRaWan

LoRa(long-range) sử dụng kỹ thuật điều chế gọi là Chirp Spread Spectrum. Có thể hiểu nôm na nguyên lý này là dữ liệu sẽ được băm bằng các xung cao tần để tạo ra tín hiệu có dãy tần số cao hơn tần số của dữ liệu gốc (cái này gọi là chipped); sau đó tín hiệu cao tần này tiếp tục được mã hoá theo các chuỗi chirp signal (là các tín hiệu hình sin có tần số thay đổi theo thời gian; Có 2 loại chirp signal là up-chirp có tần số tăng theo thời gian và down-chirp có tần số giảm theo thời gian; và việc mã hoá theo nguyên tắc bit 1 sẽ sử dụng up-chirp, và bit 0 sẽ sử dụng down-chirp) trước khi truyền ra anten để gửi đi.

Theo Semtech công bố thì nguyên lý này giúp giảm độ phức tạp và độ chính xác cần thiết của mạch nhận để có thể giải mã và điều chế lại dữ liệu; hơn nữa LoRa không cần công suất phát lớn mà vẫn có thể truyền xa vì tín hiệu Lora có thể được nhận ở khoảng cách xa ngay cả độ mạnh tín hiệu thấp hơn cả nhiều môi trường xung quanh. Bằng tần làm việc của LoRa từ 430MHz đến 915MHz cho từng khu vực khác nhau trên thế giới:

430MHz cho châu Á

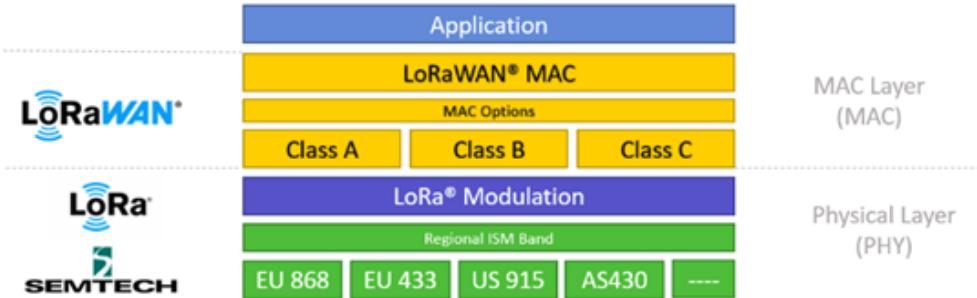
780MHz cho Trung Quốc

433MHz hoặc 866MHz cho châu Âu

915MHz cho USA

LoRaWAN là giao thức mạng năng lượng thấp, diện rộng (LPWA) được phát triển bởi Liên minh LoRa, kết nối không dây ‘hoạt động’ với internet trong các mạng khu vực, quốc gia hoặc toàn cầu, nhằm mục tiêu các yêu cầu chính của Internet of Things (IoT) như bì thông tin liên lạc hai chiều, dịch vụ bảo mật đầu cuối, di động và nội địa hóa. LoRaWAN sử dụng phổ không được cấp phép trong các dải ISM để xác định giao thức truyền thông và kiến trúc hệ thống cho mạng trong khi lớp vật lý LoRa tạo ra các liên kết giao tiếp tầm xa giữa các cảm biến từ xa và các cổng kết nối với mạng. Giao thức này giúp thiết lập nhanh chóng các mạng IoT công cộng hoặc riêng tư ở bất cứ đâu bằng phần cứng và phần mềm.

2.5 LoRa và LoRaWAN khác nhau như thế nào



Hình 2.1 LoRaWAN Technology Stack

Cả hai thuật ngữ thường được sử dụng đồng nghĩa, nhưng chúng có ý nghĩa khác nhau. Nhìn vào hình 2.1, ta có thể thấy:

- LoRa là chip (Physical layer) để giao tiếp không dây giữa Gateway và node thông qua tần số
- LoRaWan nằm ở MAC layer là giao thức mạng bao gồm việc sử dụng chip LoRa để liên lạc

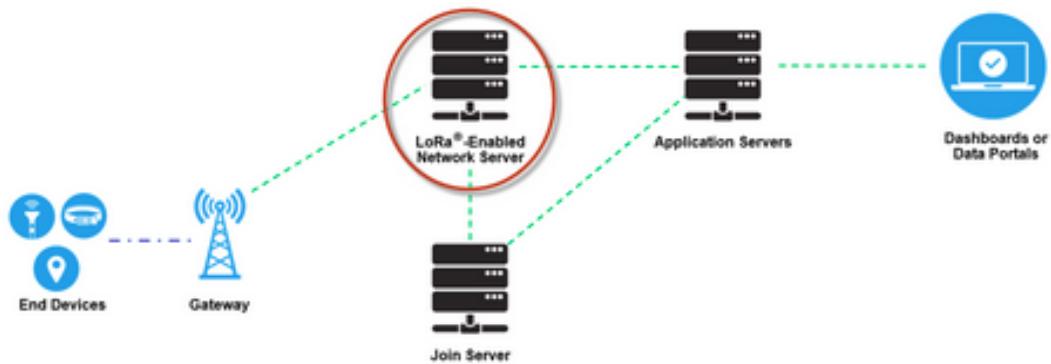
2.6 The Things Networks

Là hệ thống bao gồm các vai trò Network Server, Application Server, join server

2.6.1 Network Server

Tư hình 2.2, ta suy được:

- Nhận data từ sensor của các end-devices thông qua Gateway
- Thực tế, các Network sẽ:
 - Kiểm tra device address
 - Đếm và quản lí số frame thực tế



Hình 2.2 Mô hình LoRaWAN Network

2.6.2 Application Server

Quản lí và hiển thị data, thực hiện các thao tác tạo ra down-link payload để kết nối đến end-device

2.6.3 Join Network

Trên the things network, có 2 hình thức join: OTAA (over-the-air activation) và ABP(Activation by Personalization) có đặc điểm như bảng 2.1

Bảng 2.1 So sánh giữa OTAA và ABP

OTAA	ABP
- Bảo mật hơn	- Ít bảo mật
- Tự chọn channel join vào	- Setup channel ở end-device

2.7 LoRa Dragino LG02 dual channel gateway

Thiết bị được mô tả ở hình ?? với những đặc tính sau:

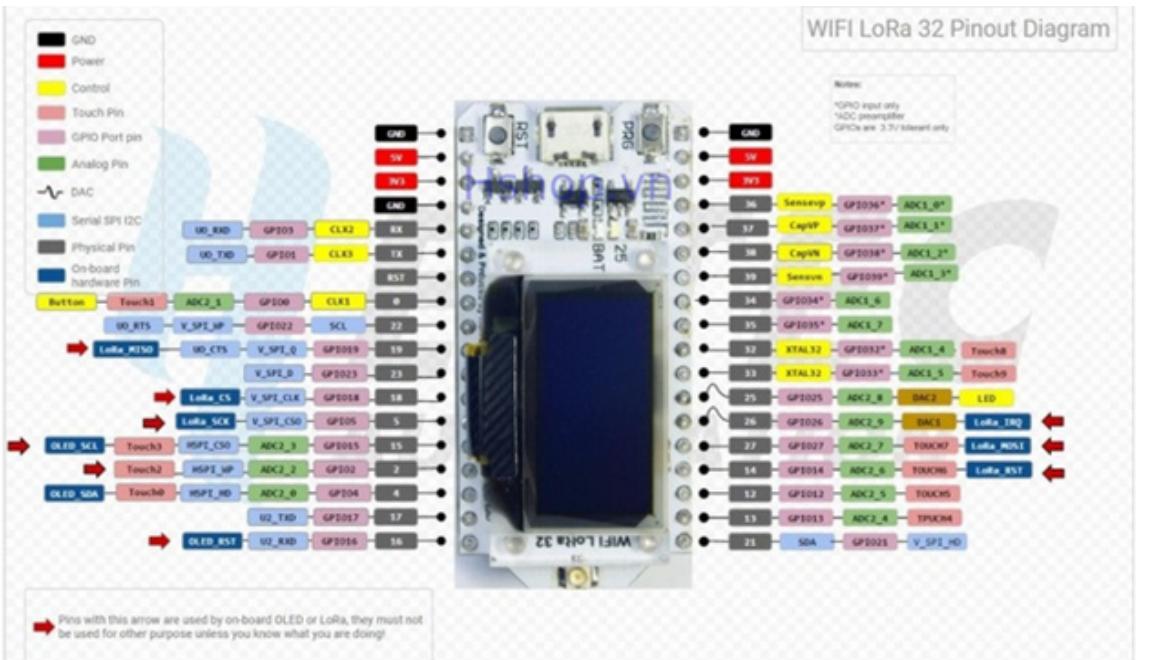


Hình 2.3 LG02 Dragino Gateway

- LG02 là open source dual channels LoRa Gateway. Là cầu nối dễ dàng cho LoRa wireless network và IP network: wifi, Ethernet, 3g/4g....
- Hỗ trợ giải quyết từ 50-300 sensor nodes
- Frequency 433Mhz

2.8 End-Device

Kit RF thu phát wifi Blue esp32 + lora sx1278 oled heltec được mô tả như hình 2.4 với những đặc điểm sau:



mềm, ...

Việc sử dụng để tính toán nặng đòi hỏi nhiều tài nguyên CPU như Machine Learning hoặc Big Data không phải thế mạnh của Nodejs. Nên sử dụng Python sẽ phù hợp nhất với các dự án dựa trên AI và máy học vì tính đơn giản, linh hoạt và nhất quán, cũng như quyền truy cập vào nhiều thư viện và frameworks phong phú liên quan đến máy học, phân tích dữ liệu và tính toán khoa học, trực quan dữ liệu,

2.10 Giới thiệu về LSTM và bài toán dự đoán chuỗi dữ liệu đa bước thời gian

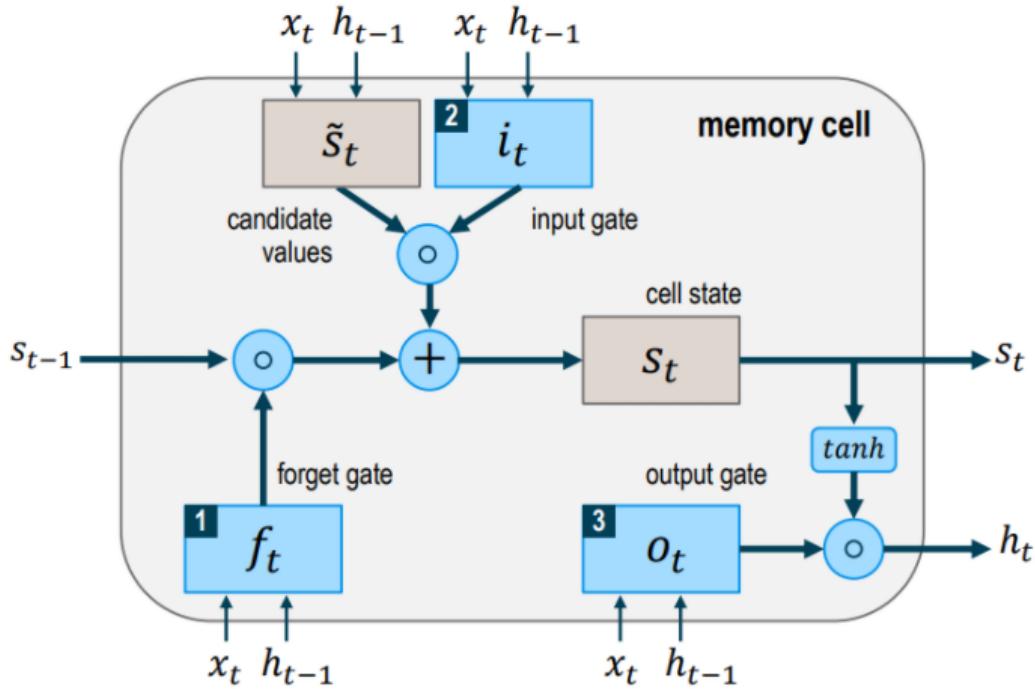
LSTM (Long Short-Term Memory) là phiên bản mở rộng của mạng RNN (Recurrent Neural Network), được thiết kế để giải quyết các bài toán về phụ thuộc dài hạn (long-term dependencies) trong mạng RNN do bị ảnh hưởng bởi vấn đề gradient biến mất.

Mạng LSTM có thể bao gồm nhiều LSTM memory cell liên kết với nhau và kiến trúc cụ thể của mỗi tế bào được biểu diễn như hình 2.5. Ý tưởng của mạng LSTM là bổ sung thêm trạng thái bên trong tế bào (cell internal state) s_t và ba cổng sàng lọc các thông tin đầu vào và đầu ra cho tế bào bao gồm forget state f_t (loại bỏ thông tin nhận được không cần thiết ra khỏi cell), input gate i_t (chọn lọc thông tin cần thiết thêm vào cell) và output gate o_t (xác định thông tin nào từ cell được sử dụng như đầu ra). Tại mỗi bước thời gian t , các cổng đều lần lượt nhận giá trị đầu vào x_t (đại diện cho một phần tử trong chuỗi đầu vào), sau đó dựa trên cơ chế hoạt động của cell, đưa qua quá trình lan truyền xuôi (forward pass) và giá trị h_{t-1} có được từ đầu ra của memory cell từ bước thời gian trước đó $t - 1$.

Dự đoán trước nhiều bước thời gian là công việc dự đoán một chuỗi các giá trị trong một chuỗi thời gian bằng việc áp dụng mô hình dự đoán từng bước và sử dụng giá trị dự đoán của bước thời gian hiện tại để xác định giá trị của nó trong bước thời gian tiếp theo. Hiện nay có ít nhất bốn chiến lược thường được sử dụng để giải quyết bài toán này:

- Dự báo trực tiếp:
 - Liên quan đến việc phát triển mô hình riêng biệt cho từng bước thời gian.
 - Không thích hợp trong việc tính toán và bảo trì vì số lượng bước thời

Hình 2.5 Sơ đồ biểu diễn kiến trúc bên trong tế bào LSTM



gian dự đoán đôi khi sẽ lên rất nhiều.

- Không có cơ hội để mô hình hóa sự phụ thuộc giữa các dự đoán.
- Dự báo đệ quy:
 - Liên quan đến việc sử dụng mô hình một bước nhiều lần trong đó dự đoán cho bước thời gian trước được sử dụng làm đầu vào để đưa ra dự đoán cho bước thời gian sau.
 - Cho phép tích lũy các lỗi dự đoán để hiệu suất có thể nhanh chóng suy giảm khi thời gian dự đoán tăng lên.
- Dự báo kết hợp:
 - Kết hợp giữa trực tiếp và đệ quy để cung cấp các lợi ích của cả hai phương pháp và khắc phục hạn chế của mỗi chiến lược.
 - Phát triển mô hình cho từng bước thời gian dự đoán, nhưng mỗi mô hình có thể sử dụng các bước dự đoán được thực hiện bởi các mô hình ở các bước thời gian trước đó làm giá trị đầu vào.

- Dự báo nhiều đầu ra:
 - Liên quan đến việc phát triển một mô hình có khả năng dự đoán toàn bộ chuỗi dự báo theo cách một lần.
 - Phức tạp hơn vì chúng có thể tìm hiểu cấu trúc phụ thuộc giữa đầu vào và đầu ra cũng như giữa các đầu ra.
 - Mô hình đào tạo sẽ chậm hơn và yêu cầu nhiều dữ liệu hơn để tránh Overfitting cho vấn đề.

2.11 Giới thiệu bài toán TSP (Travelling Salesman Problem)

TSP (bài toán người du lịch) là một bài toán tối ưu tổ hợp thuộc lớp phức tạp NP-hard, trình bày vấn đề: "Cho sẵn danh sách các thành phố và khoảng cách giữa những thành phố đó, tìm đường đi khả thi ngắn nhất từ điểm bắt đầu đến các thành phố và quay lại điểm bắt đầu, với điều kiện mỗi thành phố chỉ được đi qua một lần".

Hiện tại đã có nhiều phương pháp giải quyết bài toán này thông qua các thuật toán chính xác, heuristic:

- The Brute-Force Approach: Tính toán và so sánh tất cả các hoán vị có thể có của các tuyến đường để xác định đường đi ngắn duy nhất.
- The Branch and Bound Method: Tách bài toán lớn thành các bài toán nhỏ theo dạng nhánh, mỗi nhánh sẽ được tính toán và kiểm tra dựa trên các giới hạn ước tính trên và dưới và đưa ra giải pháp riêng. Những giải pháp sẽ được loại bỏ nếu nó không tốt hơn giải pháp tốt nhất được tìm thấy cho đến nay.
- The Nearest Neighbor Method: Chọn tuyến đường bắt đầu và đến những tuyến đường gần nhất của nó và sau đó quay trở về khi nó đã đến hết tất cả tuyến đường trên bản đồ.

Ngoài ra, nhiều giải pháp học thuật cũng được đưa ra để giải quyết những vấn đề phụ mà các phương pháp phổ biến hiện nay đang gặp phải: Zero Suffix Method, Biogeography-based Optimization Algorithm, Multi-Agent System, Multi-Objective Evolutionary Algorithm, ...

Về mặt lý thuyết, việc giải quyết TSP sẽ dễ dàng hơn vì bạn phải tìm ra con đường ngắn nhất cho mỗi chuyến đi trong thành phố. Nhưng nó trở nên khó khăn

để giải quyết TSP bằng tay khi số lượng thành phố tăng lên, bất kì giải thuật nào cho bài toán TSP cũng sẽ tăng theo hàm mũ với số lượng thành phố. Ngoài ra, một số ràng buộc làm cho TSP khó giải quyết hơn (giao thông, thay đổi tuyến đường đột ngột, phương tiện di chuyển,...). Vì thế trong lý thuyết của độ phức tạp tính toán, phiên bản quyết định của bài toán TSP thuộc lớp NP-Complete, tức là không có giải thuật hiệu quả duy nhất nào cho việc giải bài toán.

2.12 Giới thiệu về Mapbox

Mapbox là nền tảng đám mây hỗ trợ định vị và xử lý dữ liệu bản đồ thông qua các API dễ dàng tích hợp vào trang web hoặc ứng dụng của mình với giá cả rất hợp lý. Không những thế, Mapbox cho phép sử dụng miễn phí API tối đa ở 100.000 request 1 API, vì thế nâng cao trải nghiệm của người lập trình cho đến khi họ thực sự muốn xài dịch vụ.

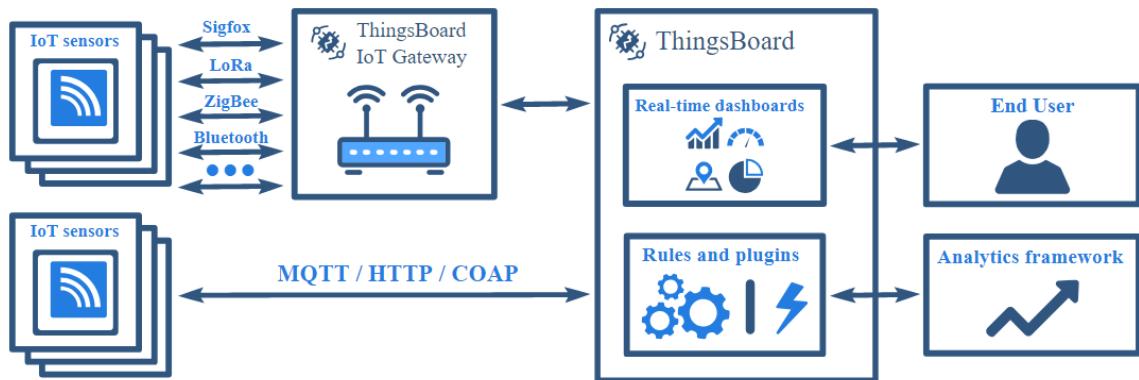
Mapbox sở hữu một trang docs với tất cả các API hỗ trợ từ vẽ, thiết kế bản đồ, quản lý các điểm, cho đến hỗ trợ tìm đường nhanh, ma trận đường, geocoding, ... Vì để giải quyết bài toán thu gom rác trong dự án, Mapbox API, cụ thể là Route-matrix và Optimization API sẽ được sử dụng để tìm thời gian đi và đường đi tối ưu nhất từ một điểm đến các điểm còn lại.

2.13 Giới thiệu về Thingsboard

Thingsboard là một nền tảng IoT mã nguồn mở, giúp phát triển, quản lý và mở rộng các dự án về Iot. Với Thingsboard, việc thu thập, xử lý, hiển thị trực quan và quản lý thiết bị sẽ trở nên thuận lợi hơn thông qua việc kết nối các thiết bị bằng các giao thức IoT tiêu chuẩn công nghiệp - MQTT, CoAP và HTTP, hỗ trợ triển khai đám mây và tại chỗ. Ngoài ra, Thingsboard cho phép tích hợp các thiết bị được kết nối với các máy chủ (OPC-UA, MQTT Broker, Sigfox Backend, Modbus slaves, ...) và bên thứ ba qua IoT Gateway bằng các giao thức hiện có (Sigfox, LoRa, ZigBee, Bluetooth, ...). Hình 2.6 hiển thị mô hình tổng quát của Thingsboard

Với các tính năng ưu việt như thu thập dữ liệu từ xa, quản lý thiết bị và các báo động, Thingsboard cung cấp hơn 30 tiện ích có sẵn (Google map, Realtime chart, HTML tab, ...), cho phép tạo các bảng điều khiển (Dashboard) phong phú để hiển thị

Hình 2.6 Mô hình tổng quan Thingsboard

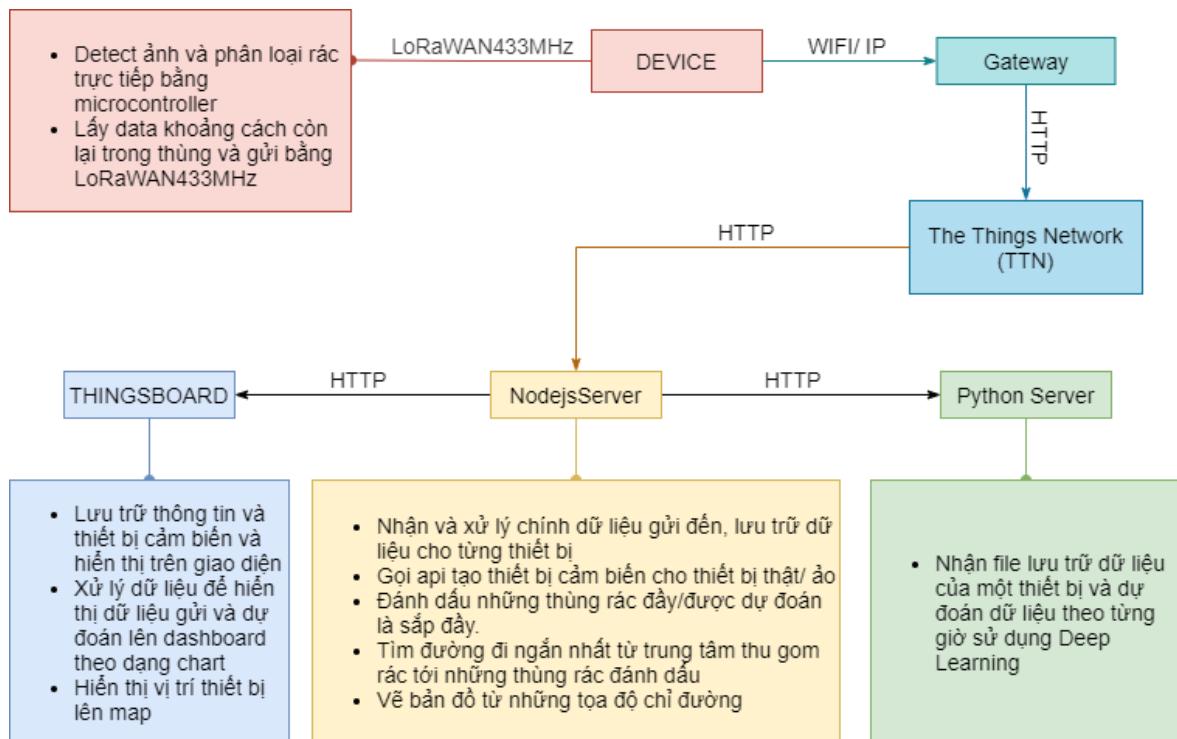


trực quan dữ liệu và điều khiển từ xa trong thời gian thực.

Bên cạnh đó, Thingsboard cho phép tạo các chuỗi quy tắc (Rule chains) kéo thả thân thiện để xử lý dữ liệu thiết bị dựa trên thuộc tính thực thể hoặc nội dung tin nhắn, tùy chỉnh logic xử lý phù hợp với các trường hợp sử dụng ứng dụng cụ thể.

Chương 3 MÔ HÌNH HỆ THỐNG

Mô hình tổng quan của đề tài hình 3.1



Hình 3.1 Mô hình tổng quan

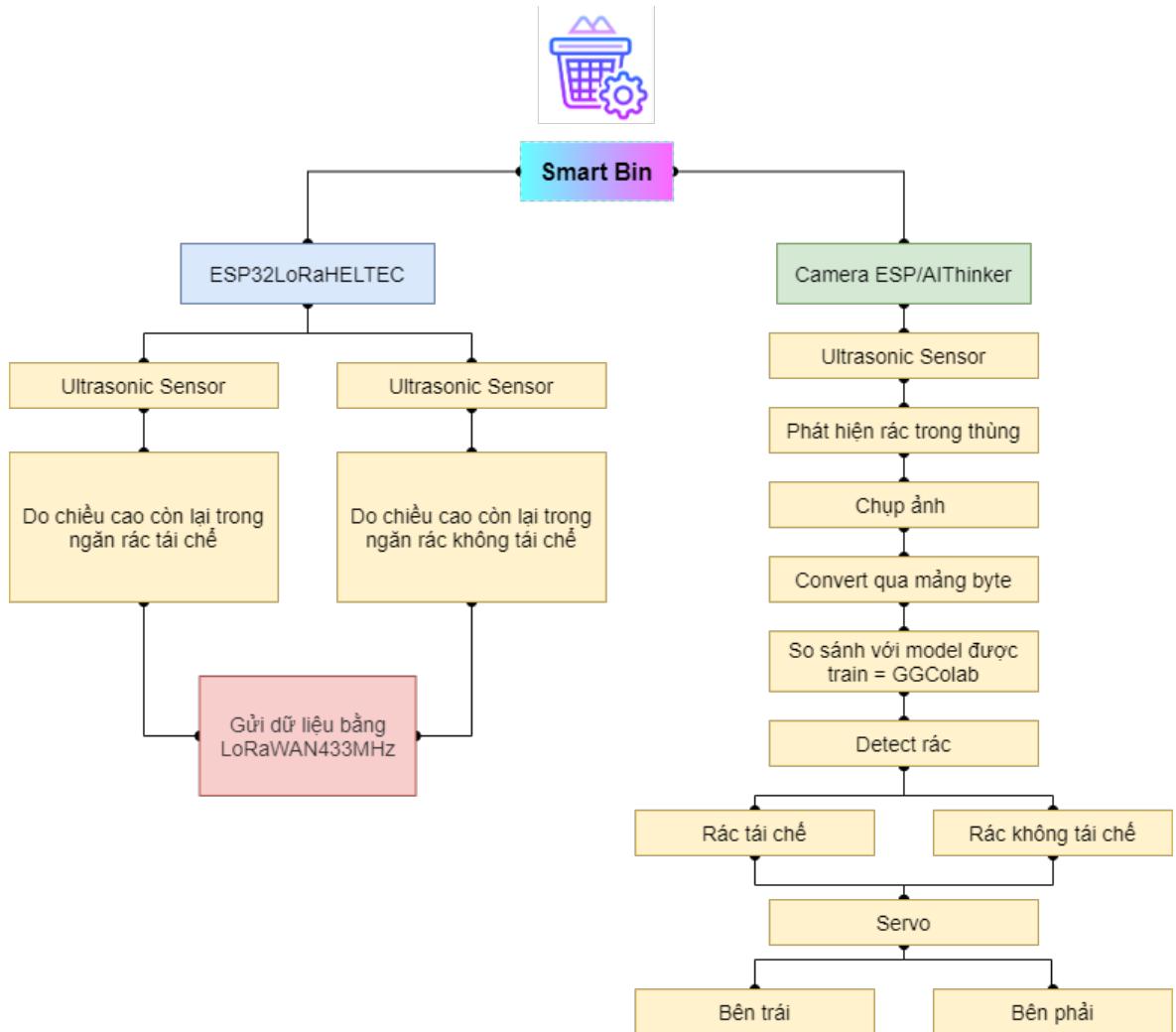
Ở mô hình tổng quan đề tài, chúng tôi chia làm 2 phần, về thiết bị và về server.

- Về thiết bị, chúng tôi gắn 2 loại board và 3 ultrasonic sensor, 1 servo vào thùng rác để thực hiện 2 mục đích khác nhau, cụ thể:
 - ESP32 camera AI Thinker, dùng vào mục đích chụp ảnh và phục vụ chức năng phân loại rác thải.
 - ESP32 Lora heltec, dùng trong mục đích thu thập data và gửi data cho server bằng công nghệ LoRaWAN.
 - 2 Ultrasonic sensor thực hiện nhiệm vụ đo độ dài khoảng trống còn lại

trong thùng rác. Và 1 Ultrasonic sensor có nhiệm vụ phát hiện có rác trong thùng để camera được kích hoạt. 1 servo để tác động làm mặt đựng rác quay trái hoặc quay phải.

- Về server, sử dụng Nodejs và Flask làm server cung cấp API, Thingsboard để làm giao diện hiển thị và một số server thứ ba khác, chức năng được mô tả cụ thể như sau:
 - Nodejs Server là server nhận và xử lý chính dữ liệu, lưu trữ dữ liệu cho từng thùng rác. Từ đó, sử dụng dữ liệu để hiển thị, dự đoán và tìm đường đi tối ưu nhất để thu gom rác.
 - Flask Server là server sử dụng Deep Learning để dự đoán dữ liệu được gửi từ Nodejs Server.
 - Thingsboard là giao diện lưu trữ thông tin thùng rác, hiển thị dữ liệu theo nhiều dạng và vẽ đường đi trên bản đồ.
 - Ngoài ra, còn sử dụng Thingsboard server hỗ trợ API thao tác đến Thingsboard và Mapbox server hỗ trợ API liên quan đến địa lý và đường đi.

Hình 3.2 mô tả chi tiết về cấu tạo của thùng rác, và các thức hoạt động của 2 loại board được gắn bên trong thùng.



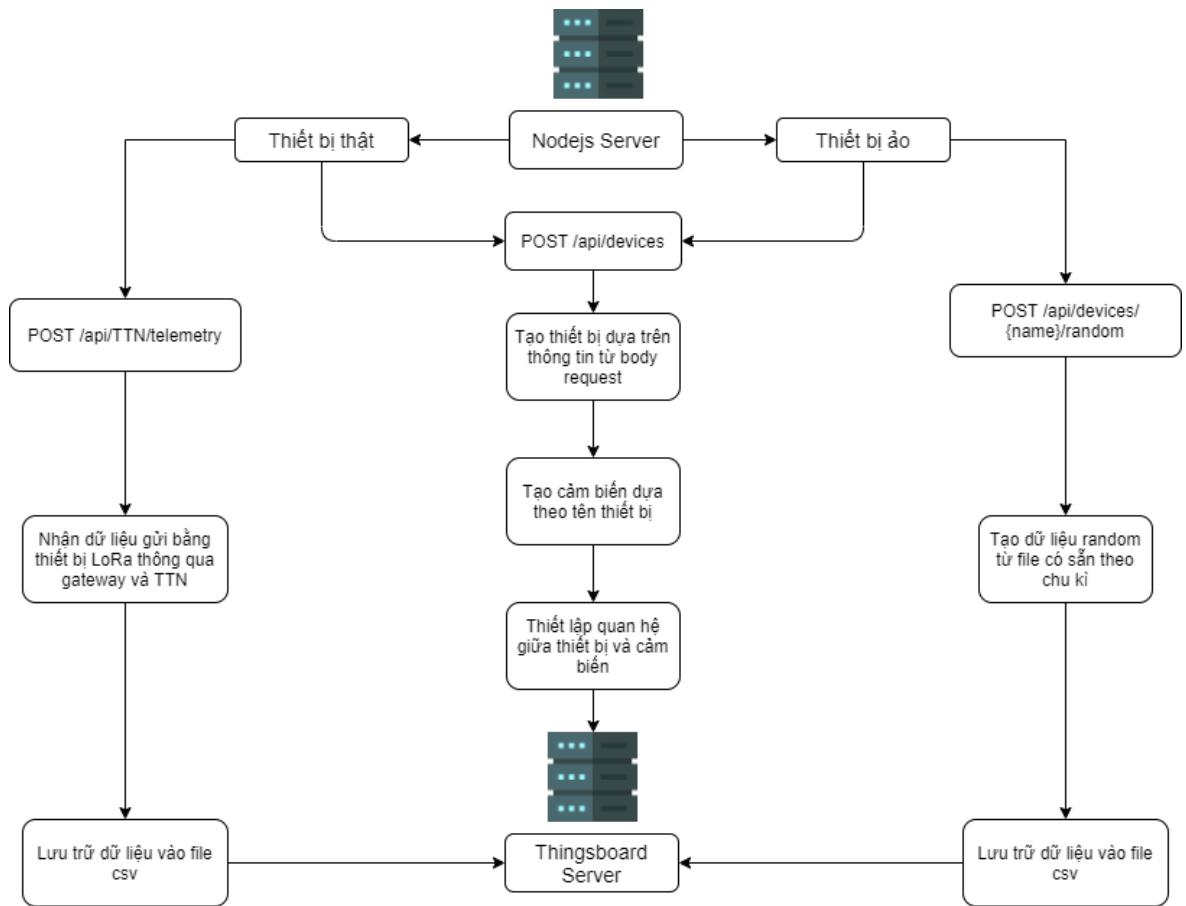
Hình 3.2 Mô hình device

Như đã nói ở sơ đồ 3.1, Smart bin sẽ bao gồm 2 board, hoạt động cụ thể của chúng như sau:

- Phân loại rác (ESP32Cam AI Thinker): Phía trong mặt trên của thùng rác được gắn 1 hộp thiết bị, bao gồm mạch ESP32Cam AI Thinker và 1 ultrasonic sensor.
 1. Khi bỏ rác vào thùng, rác sẽ làm cho chiều cao của thùng thay đổi, cụ thể là khoảng cách từ mặt trên của thùng đến phần tiếp xúc của rác sẽ giảm. Từ đó, đáp ứng điều kiện để kích hoạt camera chụp ảnh.
 2. Camera chụp ảnh.
 3. Hình ảnh đã chụp được mạch thực hiện thao tác convert qua 1 mảng byte.

4. Model detec ảnh đã được train từ Google Colab và đưa vào mạch cũng là 1 mảng byte. Nên mạch tiến hành so sánh 2 mảng byte đó và đưa ra kết quả detec.
 5. Sau khi có kết quả detec loại rác, mạch sẽ xét điều kiện xem loại rác đã được detec(glass/metal/plastic/cardboard/paper/other trash) thuộc loại rác: Tái chế hay Không tái chế.
 6. Sau khi có kết quả loại rác là "tái chế" hay "Không tái chế" thì mạch tiếp tục xét điều kiện:
 - a) Nếu rác thuộc loại "tái chế": servo sẽ quay 180 độ để làm rác rớt vào ngăn bên phải.
 - b) Nếu rác thuộc loại "tái chế": servo sẽ quay 0 độ để làm rác rớt vào ngăn bên trái.
- ESP32 Lora heltec, dùng trong mục đích thu thập data và gửi data cho server bằng công nghệ LoRaWAN.

Mô hình server từ bước khởi tạo đến gửi dữ liệu được vẽ ở hình 3.3 và được mô tả chi tiết như sau:



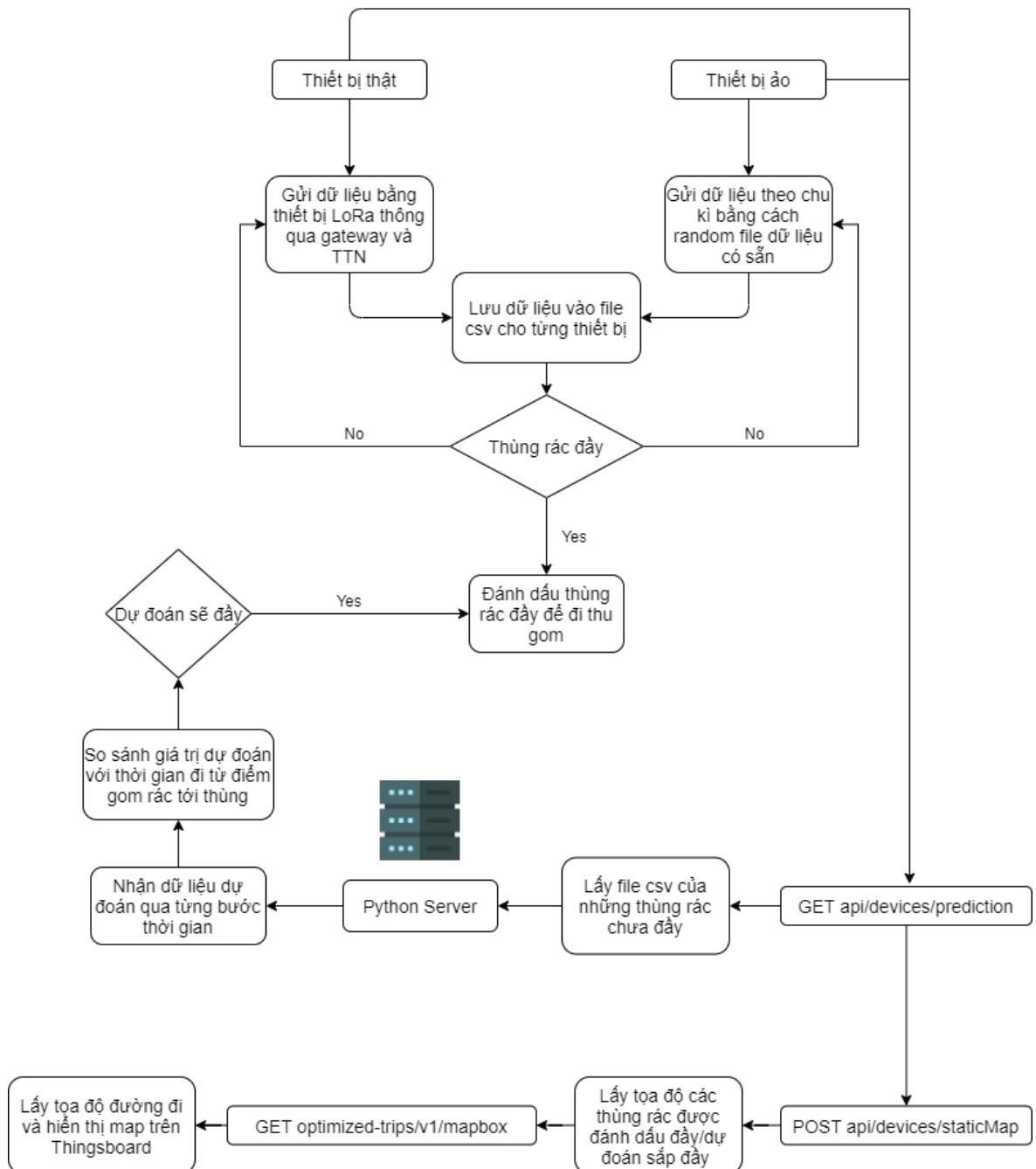
Hình 3.3 Mô hình server từ bước khởi tạo đến gửi dữ liệu

Chức năng chính của Nodejs Server là cung cấp API xử lý dữ liệu cho hai đối tượng chính là thùng rác thật và thùng rác ảo.

- **POST /api/devices:** Hỗ trợ tạo thùng rác và các cảm biến liên quan bằng cách gửi request body bao gồm các thông số như tên, chiều cao, tọa độ,... và tự động lưu vào database Thingsboard và cuối cùng là hiển thị trên dashboard. Thao tác này có thể thực hiện thủ công trên giao diện hoặc tự động thông qua API này.
- **POST /api/TTN/telemetry:** Nhận và xử lý dữ liệu được gửi từ các thùng rác thật thông qua gateway và TTN, format của dữ liệu bao gồm thông tin thùng rác và dữ liệu cảm biến. Dữ liệu được lưu vào file csv để dự đoán và được đưa lên Thingsboard để hiển thị trên dashboard.
- **POST /api/devices/{name}/random:** Tạo dữ liệu ngẫu nhiên cho thùng rác ảo với tên là {name} và được xử lý theo format dữ liệu giống với thiết bị thật. Dữ

liệu được lưu vào file csv để dự đoán và được đưa lên Thingsboard để hiển thị trên dashboard.

Mô hình server từ sau bước gửi dữ liệu đến xử lý bản đồ được vẽ ở hình 3.4 và được mô tả chi tiết như sau:



Hình 3.4 Mô hình server từ bước gửi dữ liệu đến xử lý bản đồ

Dữ liệu gửi tới sẽ được lưu trữ vào file csv cho đến khi thùng rác đầy, khi đó file csv sẽ ngừng ở giá trị lớn nhất và đợi đến khi nào thùng rác được gom sẽ tiếp tục nhận dữ liệu. Và khi thùng rác đầy, sẽ được đưa qua danh sách các thùng rác đầy để được đi thu gom.

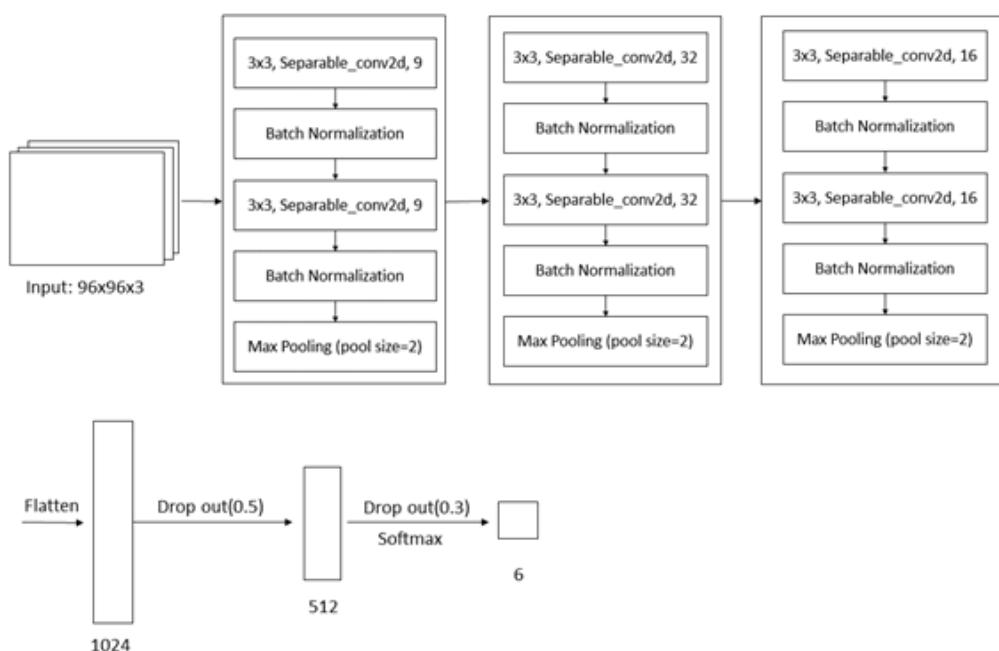
Ngoài ra, Nodejs còn cung cấp 2 API gửi dữ liệu tiên đoán và tìm đường đi tối ưu cho cả 2 thiết bị thật và ảo:

- GET api/devices/prediction: Đưa tất cả file csv dữ liệu của các thùng rác chưa đầy qua bên Python Server để xử lý dữ đoán dữ liệu qua các bước thời gian tiếp. Kết quả sẽ được trả về Nodejs Server, được so với thời gian di chuyển từ nơi gom rác đến thùng rác để chọn giá trị thích hợp và so với chiều cao thùng. Nếu giá trị lớn hơn 90% thùng thì thùng đó sẽ được đưa vào danh sách thùng rác đầy.
- GET api/devices/staticMap: Lấy tất cả tọa độ trong danh sách thùng rác đầy và đưa qua API của Mapbox để vẽ ra đường đi tối ưu nhất để gom các thùng rác. Kết quả trả về là một mảng các tọa độ đường đi, sau đó được đưa lên Thingsboard để hiển thị trên bản đồ.

Chương 4 XÂY DỰNG THIẾT BỊ TỰ ĐỘNG PHÂN LOẠI RÁC

4.1 Xây dựng model phân loại rác dựa trên CNN

4.1.1 Tối ưu model để có thể chạy trên được chip nhúng ESP32



Hình 4.1 Minh họa kiến trúc mạng đã xây dựng

Do sử dụng microcontroller cụ thể là board ESP32 camera AI Thinker, nên ram hỗ trợ rất khiêm tốn. chỉ có 4MB. nên chúng tôi không thể xây dựng mô hình với trọng số quá lớn và có chi phí tính toán cao được. Vì thế, chúng tôi chọn các sử dụng separable convolution thay cho lớp convolution thông thường để giảm bớt chi phí tính toán cho mạng. Ngoài ra ở lớp này, chúng tôi dùng regularization l2 để tránh overfitting và hàm kích hoạt relu – đây là hàm thường được sử dụng trong quá trình train model với dữ liệu dưới dạng ảnh. Chúng tôi cũng thêm các lớp batch normalization và drop out để tránh cho mô hình bị overfitting và loại bỏ sự kết nối chật chẽ giữa các lớp fully connected. Tổng số tham số của mô hình là 532,161. Sau

khi xây dựng mô hình, nhóm đã sử dụng optimization stochastic gradient descent với learning rate = 1e-4 và momentum = 0.8 để train mô hình. Cùng với việc giảm kích thước hình ảnh từ 512×384 pixels xuống còn 96×96 pixels để phù hợp với kích thước ram mà board hỗ trợ.

4.2 Evaluation

4.2.1 Data Description

Chúng tôi dùng bộ dataset TrashNet [1] để train và test với kích thước 512×384 . Gồm 2527 hình thuộc 6 lớp với sự phân bố ở mỗi lớp:

Cardboard: 403

Glass: 501

Metal: 410

Paper: 594

Plastic: 482

Trash: 137

Dữ liệu được chia làm 2 phần theo tỉ lệ 80% train và 20% test.

Train: 2024

Test: 503

Sau đây là hình ảnh minh họa của 6 lớp cần phân loại như trên, được hiển thị ở bảng 4.1.

Cardboard	Metal	Glass
		
Paper	Plastic	Other
		

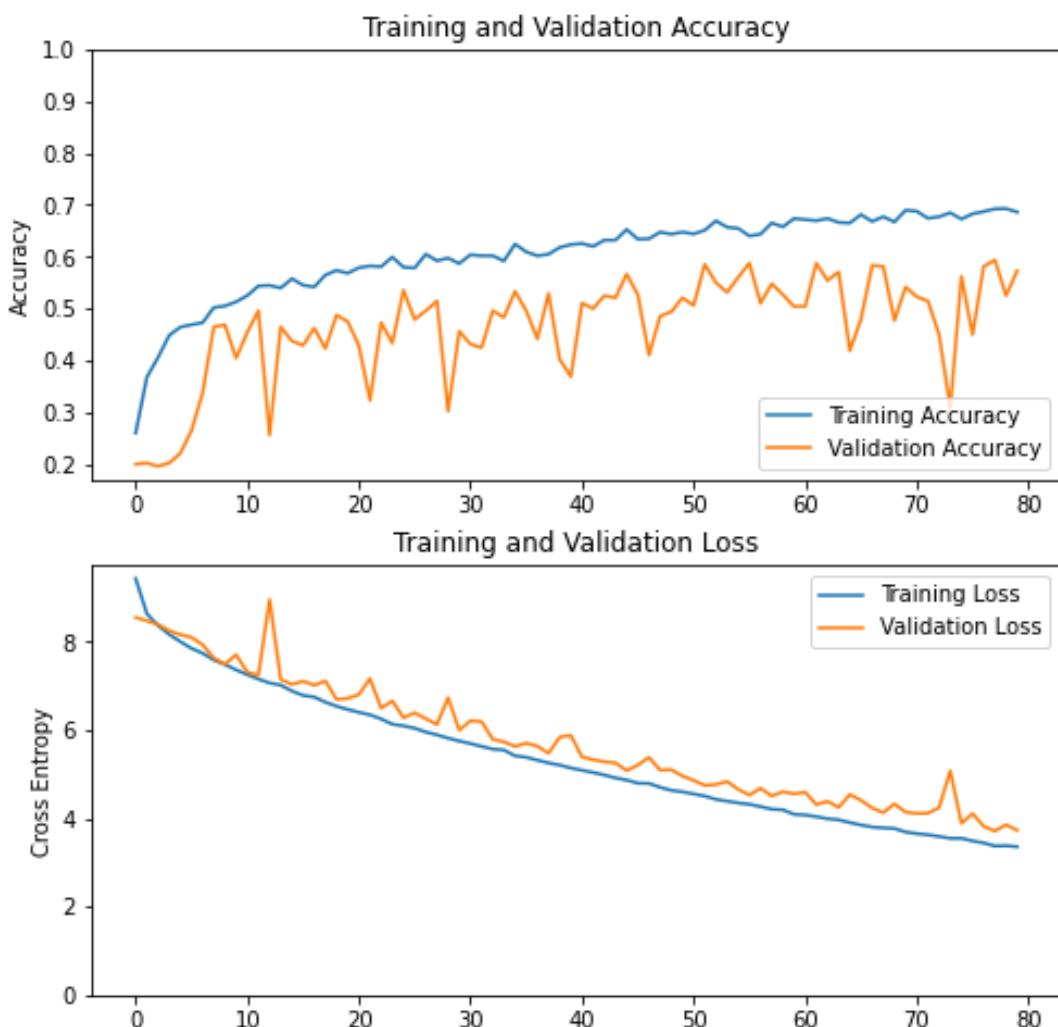
Bảng 4.1 Phân loại hình ảnh trong bộ dataset

4.2.2 Index of Performance

Chúng tôi sử dụng hai chỉ số là accuracy và F1-score để đánh giá model. Accuracy là tính tỉ lệ giữa số ảnh được dự đoán đúng và tổng số ảnh trong tập dữ liệu kiểm thử.

4.2.3 Kết quả hiện thực được

Kết quả train mô hình với 80 epoch



Hình 4.2 Biểu đồ thể hiện loss và accuracy của mô hình trong quá trình train

Accuracy cao nhất trên tập train là 0.6938 và trên tập test là 0.5938 Loss nhỏ nhất trên tập train là 3.3844 và trên tập test là 3.7116

Kết quả thử nghiệm mô hình trên tập test sau khi đã lưu mô hình tốt nhất từ quá trình train

CONFUSION MATRIX				
[[10 21 13 25 7 4] [10 43 12 25 5 5] [9 27 18 26 1 1] [6 48 25 24 11 4] [14 32 17 26 4 3] [2 10 4 7 4 0]]				
CLASSIFICATION REPORT				
	Precision	Recall	f1 - score	Support
Cardboard	0.20	0.12	0.15	80
Glass	0.24	0.43	0.31	100
Metal	0.20	0.22	0.21	82
Paper	0.18	0.20	0.19	118
Plastic	0.12	0.04	0.06	96
Trash	0.00	0.00	0.00	27
Accuracy			0.20	503
Macro Avg	0.16	0.17	0.15	503
Weighted Avg	0.18	0.20	0.18	503

Bảng 4.2 Confusion matrix của model khi thử lại trên tập test

Từ hình 4.2 ta có thể thấy accuracy và f1-score của mô hình còn khá thấp, tuy nhiên do mô hình được xây dựng chỉ có 532,161 tham số nên không đủ để phân loại chính xác các lớp được. Chúng tôi đã thử train thêm epoch nhưng mô hình dễ bị overfitting và kết quả khi in confusion matrix vẫn không thay đổi nhiều. Ngoài ra nếu tăng thêm trọng số thì không thể sử dụng model trên thiết bị được.

4.3 Build và kiểm thử model phân loại rác trên device

Sau khi model đã được train và evaluate kỹ lưỡng trên máy tính, công đoạn tiếp theo là build chương trình phân loại rác trên device dựa model đã train và thư viện TensorflowLite. Chương trình được build sẽ phải được kiểm thử để đảm bảo khả năng phân loại rác tương đương với khả năng phân loại của model đã được train trên ảnh từ PC.

4.3.1 Kiểm thử chương trình phân loại rác trên device

Để kiểm tra sự phân loại rác sau khi board ESP32 camera AI thinker hoàn thành việc chụp ảnh, tôi sử dụng 1 file python để laptop đóng vai trò như một server để nhận ảnh, detec ảnh, và trả kết quả detec. Việc trao đổi nhận ảnh từ board sang server sẽ sử dụng cable.

Việc kiểm thử này phải đáp ứng được gần đúng với yêu cầu mà mô hình thực tế hình 3.2. Quy trình thực hiện việc kiểm tra sẽ bao gồm các bước sau:

Khi build file C để nạp code vào board thành công. Tiếp tục build file python để tạo server. Lưu ý vẫn giữ kết nối giữa board và laptop qua cable.

Để thay thế yêu cầu, khi ultrasonic sensor gặp vật cản sau đó khởi động chụp ảnh, thì trong môi trường kiểm tra. Tôi dùng một input được nhập từ bàn phím (nhấn enter) để thay thế bước trên.

1. Nhận input từ người dùng: khi nhấn enter:
2. send "1" đến board.
3. nhận hình và kết quả dự đoán từ board .
4. save hình vào local (cùng thư mục vs python, tên file là thời điểm nhận hình).
5. show kết quả dự đoán trên console và bật app show ảnh (đã save ở 4)

4.3.2 Kết quả phân loại khi chạy trên device

Một số hình ảnh được chụp từ thiết bị:

Miếng sắt được phân vào lớp "Metal"	
Chai nước nhựa được phân vào lớp "Plastic"	
Chai nhựa được phân vào lớp "Plastic"	
Miếng bìa được phân vào lớp "Cardboard"	
Các loại rác vò lại méo mó được phân vào lớp "Trash"	

Bảng 4.3 Kết quả phân loại của một số hình ảnh

Tuy nhiên, do accuracy của model chỉ ở mức tương đối, và hình ảnh được train và hình ảnh được chụp từ thiết bị sẽ có thay đổi về màu sắc. Nên sẽ có một số loại rác bị detec sai. Dưới đây là các trường hợp sai trong quá trình kiểm thử.

Bao khăn giấy được phân vào lớp "Glass"	
Vỏ kẹo được phân vào lớp "Glass"	
Bàn tay người được phân vào lớp "Cardboard"	

Bảng 4.4 Một số kết quả sai trong quá trình phân loại

Như vậy, tuy có các trường hợp detec sai, nhưng có thể hình dung được mức độ đánh giá của model trong việc detec hình ảnh. Ví dụ như ở hình 4.4 và ??, mặc dù đã bị detec sai lớp, nhưng có thể thấy, thủy tinh nói chung đều có tính chất phản

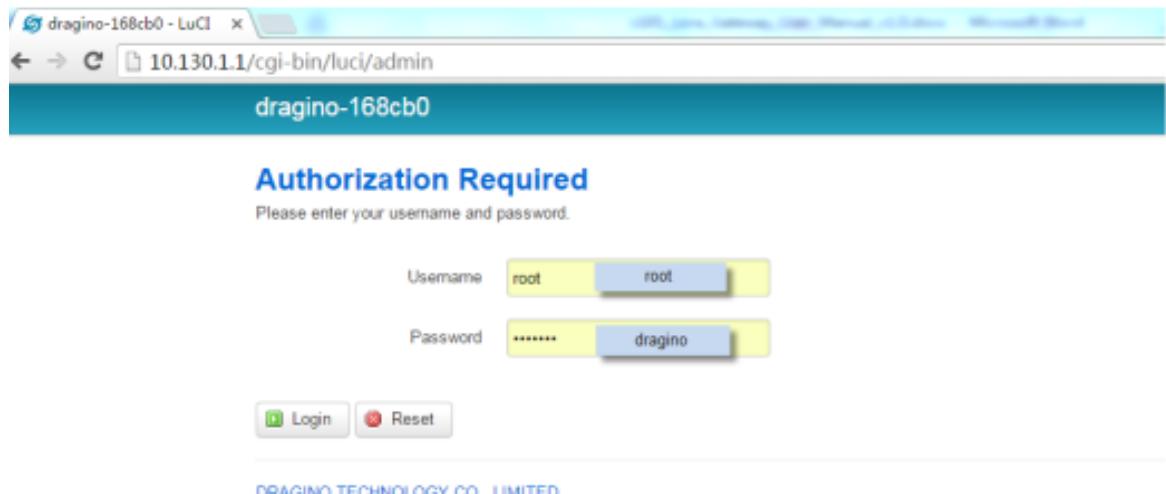
ánh sáng, nên khi hình ảnh khi chụp 1 vật làm bằng thủy tinh sẽ có một độ bóng nhất định. Vì thế ở hình 4.4 và ?? đều có tính chất phản ánh sáng như thủy tinh, nên thiết bị sẽ detect vào lớp "glass". Tương tự, các miếng bìa hầu hết trong tập dataset đều có các lăng vân nhấp nhô, và khi thiết bị chụp hình ảnh bàn tay người(hình ??), bàn tay cũng có các đường chỉ tay sâu, nên thiết bị sẽ detect vào lớp "cardboard".

4.4 Setup Gateway

4.4.1 Access LG02

Cấp nguồn điện cho gateway, sau đó dùng laptop để bật scan wifi, lúc này sẽ xuất hiện tên wifi : "dragino-168cb0"

Kết nối với wifi đó và truy cập địa chỉ IP: "10.130.1.1", lúc này sẽ đến giao diện để config gateway bằng cách nhập username và password như hình 4.3

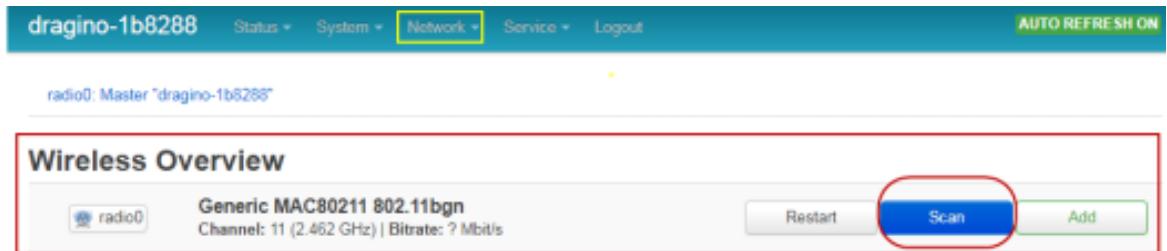


Hình 4.3 Xác thực tại địa chỉ 10.130.1.1

4.4.2 Cài đặt network

Truy cập Internet như Wifi Client theo các bước:

Bước 1: Network → Wireless, chọn radio0 và scan như hình 4.4



Hình 4.4 Scan mạng Wireless

Bước 2: Chọn wifi và tham gia vào mạng, sau đó nhập mật khẩu và Submit như hình 4.5

Select the wireless AP and join:

The screenshot shows the 'Join Network: Wireless Scan' page. It lists two networks: 'dragino-office' (Signal: 100%, SSID: dragino-office, Channel: 8, Mode: Master, BSSID: 60:64:2B:1A:B6:4D, Encryption: mixed WPA/WPA2 - PSK) and 'ChinaNet-gLnB' (Signal: 84%, SSID: ChinaNet-gLnB, Channel: 2, Mode: Master, BSSID: A4:29:40:66:F4:E7, Encryption: mixed WPA/WPA2 - PSK). To the right of each network entry is a 'Join Network' button, with the one for 'dragino-office' circled in red.

The screenshot shows the 'Joining Network: "dragino-office"' configuration page. It includes fields for 'Replace wireless configuration' (checkbox), 'WPA passphrase' (text input field containing '*****'), 'Name of the new network' (text input field containing 'wwan'), 'Create / Assign firewall-zone' (dropdown menu set to 'wan: wan:'), and a note about allowed characters. At the bottom are 'Back to scan results' and 'Submit' buttons, with the 'Submit' button circled in red.

Hình 4.5 Tham gia vào wifi và nhập mật khẩu

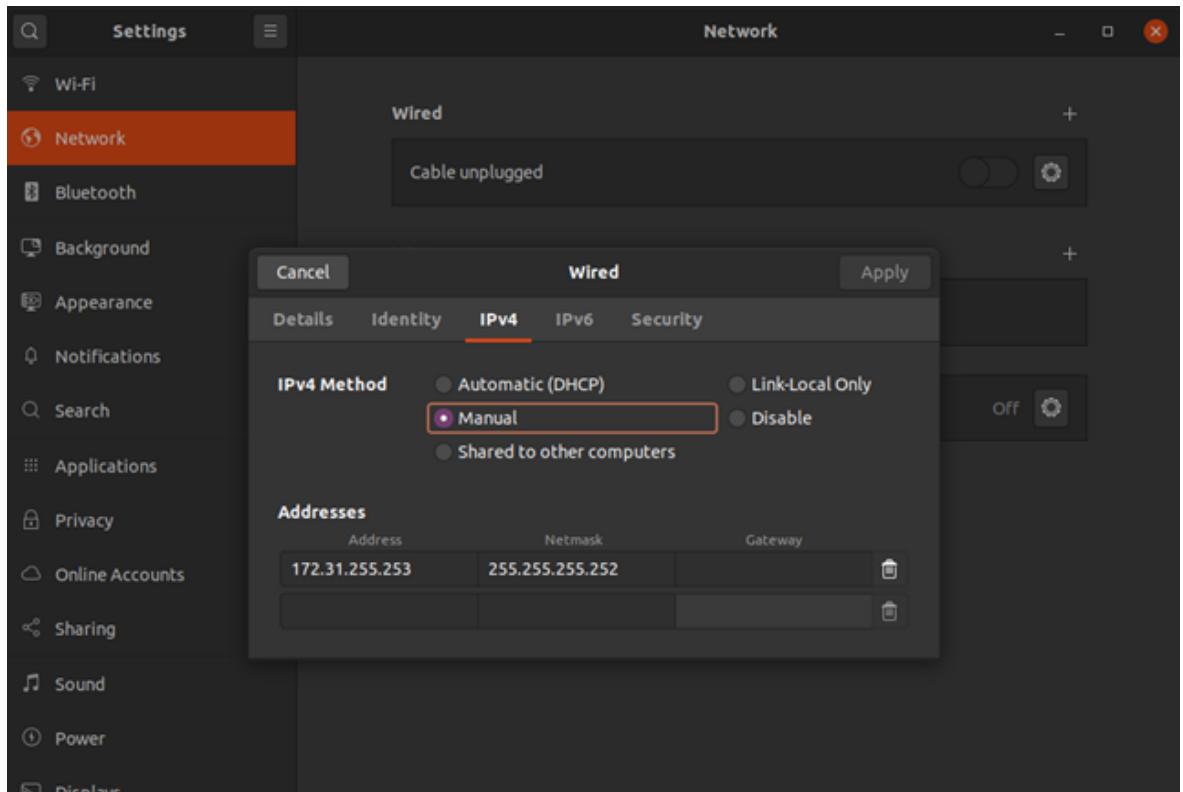
Bước 3: Network → Wireless, chọn disable wifi mặc định của gateway như hình 4.6

Hình 4.6 Tắt wifi mặc định

(lưu ý, sau khi thực hiện bước 3, kết nối sẽ bị mất, nếu laptop kết nối wifi mặc định lúc này)

Trong trường hợp không thể truy cập địa chỉ 10.130.1.1 nữa, ta sử dụng cổng LAN để kết nối:

1. Kết nối LAN Port
2. Configure Ethernet port có địa chỉ IP như hình 4.7



Hình 4.7 Cấu hình địa chỉ IP của cổng Ethernet (Ubuntu 20.04)

3. Dùng địa chỉ 172.31.255.254 để truy cập vào Web

4.4.3 Tạo gateway trên TTN Server

Bước 1: Vào Service → LoRa → Lấy ID của Gateway như hình 4.8

dragino-1dad80 Status ▾ System ▾ Network ▾ Service ▾ Logout

LoRa Gateway Settings

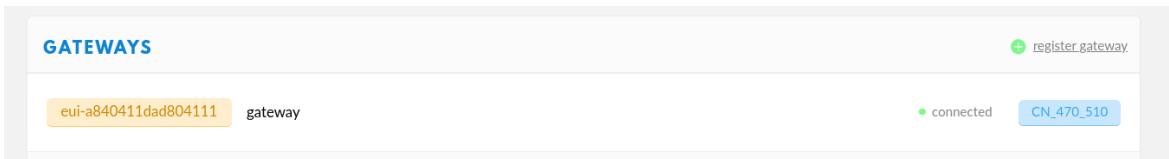
Configuration to communicate with LoRa devices and LoRaWAN server

LoRaWAN Server Settings

IoT Service	LoRaWan/RAW forwarder
Debug Level	No debug
Service Provider	The Things Network
Server Address	ttn-router-asia-se
Server Port	1700
Gateway ID	a840411dad804111
Mail Address	Mail address sent to Server
Latitude	Location Info
Longitude	Location Info
RadioMode	A for RX, B for TX
Radio Power (Unit: dBm)	20

Hình 4.8 Lấy ID của Gateway

Bước 2: Truy cập TTN: <https://console.thethingsnetwork.org/gateways>, giao diện hiển thị ở hình 4.9



Hình 4.9 Giao diện Gateway đã được tạo (chưa connect)

Bước 3: Nhập ID vào Frequency plan tùy thuộc vào Gateway mà chọn cho phù hợp như hình 4.10, và kết quả được hiển thị ở hình ?? (nếu power on gateway thì status sẽ hiện "connected")

Gateways > Register

Gateway ID
A unique, human-readable identifier for your gateway. It can be anything so be creative!

a840411dad804111 

ⓘ It looks like you are trying to register a gateway by EUI. This probably means you are using the legacy packet forwarder.

I'm using the legacy packet forwarder
Select this if you are using the legacy [Semtech packet forwarder](#).

Description
A human-readable description of the gateway

Frequency Plan
The [frequency plan](#) this gateway will use

China 470-510MHz 

Router
The router this gateway will connect to. To reduce latency, pick a router that is in a region which is close to the location of the gateway.

ttn-router-asia-se 

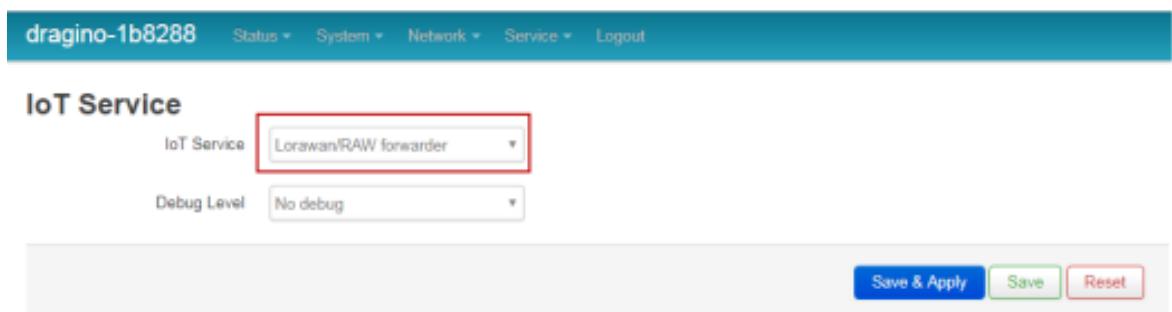
Hình 4.10 Nhập ID vào Frequency plan

**do Gateway của nhóm sử dụng frequency 433Mhz nhưng The Things Network không hỗ trợ nên khi đăng kí sẽ là : china 470-510Mhz (do tần số 433 trãi từ tần số 433 đến 510)

4.4.4 Configure LG02 Gateway

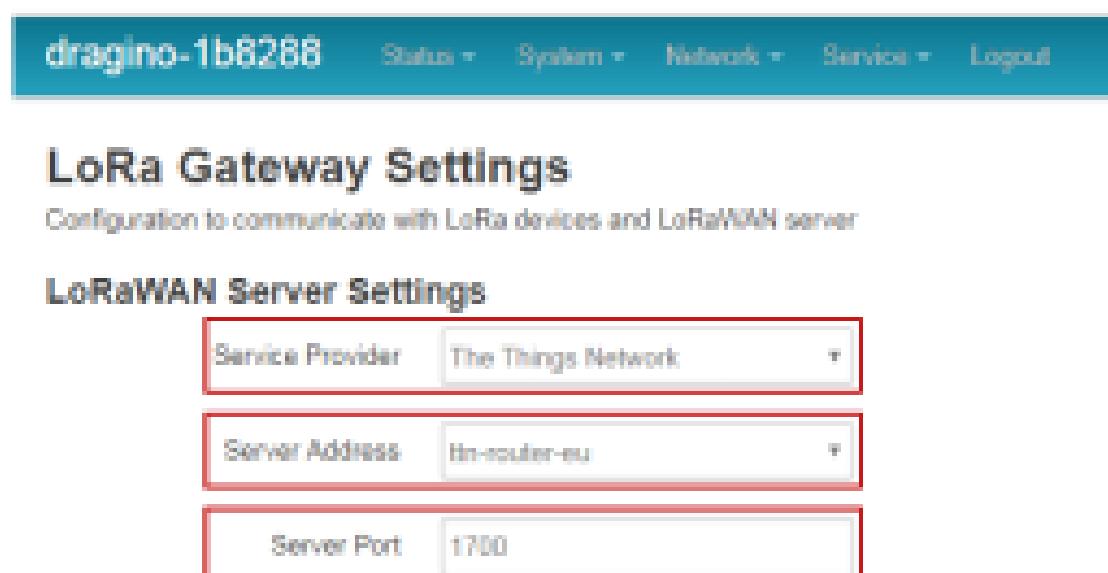
1. Configure to LoRaWAN server:

Bước 1: Config LG02 như hình 4.11



Hình 4.11 Config LG02

Bước 2: Chọn server address và gateway ID như hình 4.12 (nhóm chọn ttn-router-asia-se)



Hình 4.12 Chọn server address thích hợp

2. Configure LG02's RX frequency: Ở frequency 433Mhz, có 8 channel uplink nhưng nhóm chọn channel 433.175Mhz để end-device join vào, thông số như hình 4.13

Channel 2 Radio Settings

Radio settings for Channel 2

RadioB Frequency (Unit:Hz)	433175000
RadioB Spreading Factor	SF9
RadioB Coding Rate	4/5
RadioB Signal Bandwidth	125 kHz
RadioB Preamble Length	3 <small>Length range: 6 ~ 65536</small>
RadioB LoRa Sync Word	52 <small>Value 52(0x34) for LoRaWAN</small>
Encryption Key	Encryption Key

Hình 4.13 Thông số configure

4.5 Setup End-Device

4.5.1 Thu gom rác

Phần mềm sử dụng: Arduino IDE 1.8.2

Ngôn ngữ lập trình: C

Thiết bị Kit RF thu phát wifi Blue esp32 + Lora sx1278 oled Heltec.

Bước 1: Add lib esp32 từ link https://github.com/HelTecAutomation/Heltec_ESP32 để thêm thư viện esp32 cho Arduino

Bước 2: Add lib Lmic <https://github.com/matthijskooijman/arduino-lmic> để thêm vào thư viện Lmic cho Arduino

** Thư viện esp32 hỗ trợ hoạt động trên Heltech ESP32 develop framework.

** Thư viện Lmic cung cấp các giao tiếp LoRaWan ở classA và Class B. Tuy nhiên chỉ hỗ trợ ở band EU-868 và US 915. Nhưng thiết bị nhóm sử dụng là frequency 433Mhz nên cần sửa 1 số file để thiết bị có thể truyền data

Lúc này, có thể nạp code vào board như bình thường.

4.5.2 Đăng kí trên TTN Server

Bước 1: Tạo Application ID và handler, sau đó vào Devices → Register device → Điền Device ID

Bước 2: Sau khi đăng kí device thành công, The Things Network sẽ mặc định setup device join vào channel theo OTAA, nhưng nhóm chọn join theo ABP nên cần vào Setting → Activation method → ABP.

** Ở phần này Payload Formats sẽ tùy thuộc vào code nạp vào end-device

4.5.3 So sánh khi sử dụng Cayenne LPP và Custom

Ta có bảng 4.5

Bảng 4.5 Loại rác

Cayenne LLP	Custom
<ul style="list-style-type: none">Không cần decoder (do thư viện Cayenne LPP đã có những quy định về code ở end-device)Khó hiển thị thêm các trường data từ sensor khác (Vì chỉ hỗ trợ mặc định cho sensor nhiệt độ độ ẩm, các sensor khác phải qua pin Analog)Mặc định payload	<ul style="list-style-type: none">Cần decoder để TTN hiển thị thông tinCó thể thêm nhiều sensor dễ dàng bằng code trên node, và chỉ cần cài đặt Payload formats trên TTN để hiển thị thông tinCó thể thay đổi Payload

Để chuyển data cho backend, cần cấu hình ở phần Integrations.

4.5.4 Lưu ý khi nạp code cho node

Vì sử dụng phương thức join vào server là ABP nên cần khai báo các thông tin cần thiết vào file *.ino để node có thể send data đến application server

- Device EUI, Network Session key và App Session key
- Để tối ưu việc truyền nhận từ node đến gateway, nên disable các channels không cần thiết(chỉ enable channel đã khai báo ở RX của gateway)

Chương 5 XÂY DỰNG MÔ HÌNH SERVER XỬ LÝ VÀ HIỂN THỊ DỮ LIỆU RÁC

5.1 Nodejs Server

5.1.1 Giới thiệu tổng quát

Nodejs là nền tảng phù hợp nhất đối với các ứng dụng nhận và xử lý dữ liệu realtime từ các thiết bị IoT. Vì thế, sử dụng Nodejs Server sẽ hỗ trợ tốt trong quá trình gửi dữ liệu realtime từ The Things Network (TTN) đến server thông qua HTTP và nhận, xử lý dữ liệu nhanh chóng nhờ vào cơ chế và các thư viện, gói mà Nodejs sở hữu. Bên cạnh đó, những thiết bị ảo cũng được lập trình với chức năng gần giống như các thiết bị IoT thật để tăng khả năng mở rộng của dự án khi chỉ có một số lượng nhỏ thiết bị thật cũng như hỗ trợ quá trình kiểm tra lỗi và hiển thị trực quan.

Ngoài việc xử lý nội bộ bên trong, Nodejs còn hỗ trợ kết nối với nhiều server bên thứ ba để tận dụng những API liên quan đến dự án như quản lý, hiển thị, địa lý, tiên đoán, ...

5.1.2 Modules và packages được sử dụng

Nodejs sử dụng kiến trúc Module để đơn giản hóa việc tạo ra các ứng dụng phức tạp, mỗi module chứa một tập các hàm chức năng khác nhau tùy vào mục đích của đối tượng. Bên ngoài những module tích hợp sẵn trong Nodejs, những module bên ngoài có thể được cài đặt vào nhờ vào trình quản lý package của Nodejs (npm).

- Những module tích hợp được sử dụng:
 - http: biến server hoạt động như một máy chủ HTTP.
 - fs: xử lý hệ thống tệp.
 - os: cung cấp thông tin về hệ điều hành.

- cluster: chia nhiều luồng xử lý để tự động phân chia công việc.
- Những module bên ngoài được sử dụng:
 - dotenv: tải các biến môi trường từ tệp .env vào process.env.
 - body-parser: phân tích cú pháp các phần thân nội dung yêu cầu đến trong phần mềm trung gian trước trình xử lý.
 - express: framework được xây dựng trên nền tảng của Nodejs, cung cấp các tính năng mạnh mẽ để phát triển web hoặc mobile.
 - cors: cơ chế cho phép nhiều tài nguyên khác nhau (fonts, Javascript, v.v...) của một trang web có thể được truy vấn từ domain khác với domain của trang đó.
 - morgan: log trạng thái HTTP request.
 - async-waterfall: hỗ trợ chạy một loạt các hàm không đồng bộ nối tiếp nhau, mỗi hàm chuyển kết quả của chúng cho các hàm tiếp theo.
 - request: thực hiện HTTP request tới các server bên thứ ba.
 - lodash: thư viện chuyên xử lý logic các nhóm đối tượng.
 - random: trình tạo số ngẫu nhiên hỗ trợ nhiều phân phối phổ biến.
 - csvtojson: chuyển đổi tệp .csv thành định dạng json hoặc mảng cột.
 - moment: thư viện hỗ trợ xử lý thời gian.
 - csv-write-stream: lưu trữ giá trị vào file csv thông qua luồng mã hóa CSV.

Ngoài các module tích hợp và module bên ngoài, chương trình còn có một số module chức năng tự viết để giảm tải tính lặp lại khi lập trình.

5.1.3 API

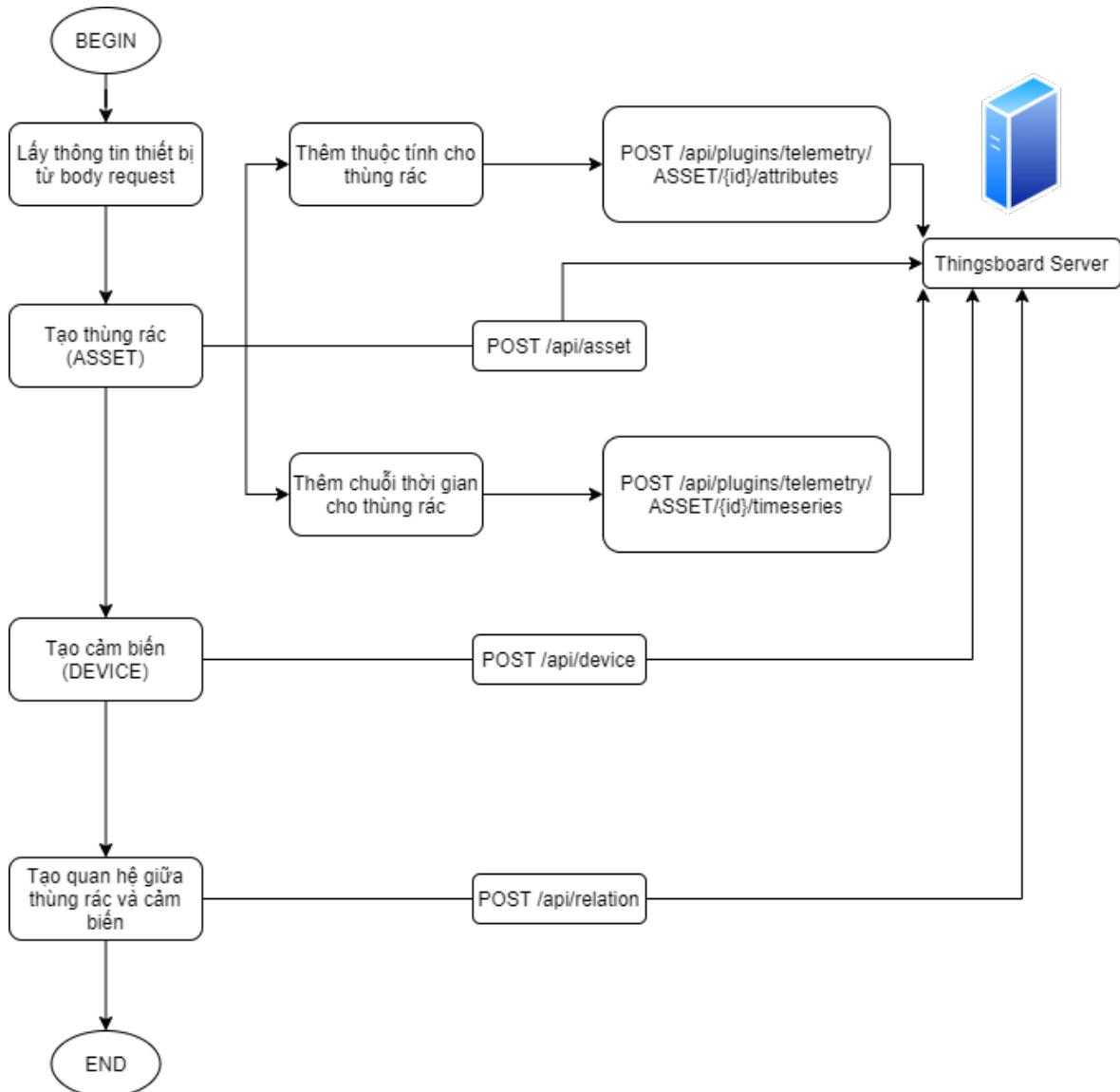
Nodejs Server là trung tâm xử lý chính luồng dữ liệu gửi đến, cung cấp những một loạt những API liên quan đến khởi tạo thiết bị, cảm biến thật và ảo, nhận và xử lý dữ liệu sau đó lưu vào database. Ngoài ra còn có API gọi tới Python server để xử lý Deep Learning cho những thùng chưa đầy, và API tìm đường đi tối ưu nhất đến các thùng đã đầy hoặc dự đoán sắp đầy.

Sau đây là tất cả API mà server cung cấp với bản mô tả chi tiết từng luồng chức năng:

- Xử lý chung:

- POST /api/devices: Tạo và cài đặt thùng rác, cảm biến

Mô hình luồng xử lý được mô tả tổng quát ở hình 5.1



Hình 5.1 Sơ đồ miêu tả mô hình khởi tạo thiết bị và cảm biến

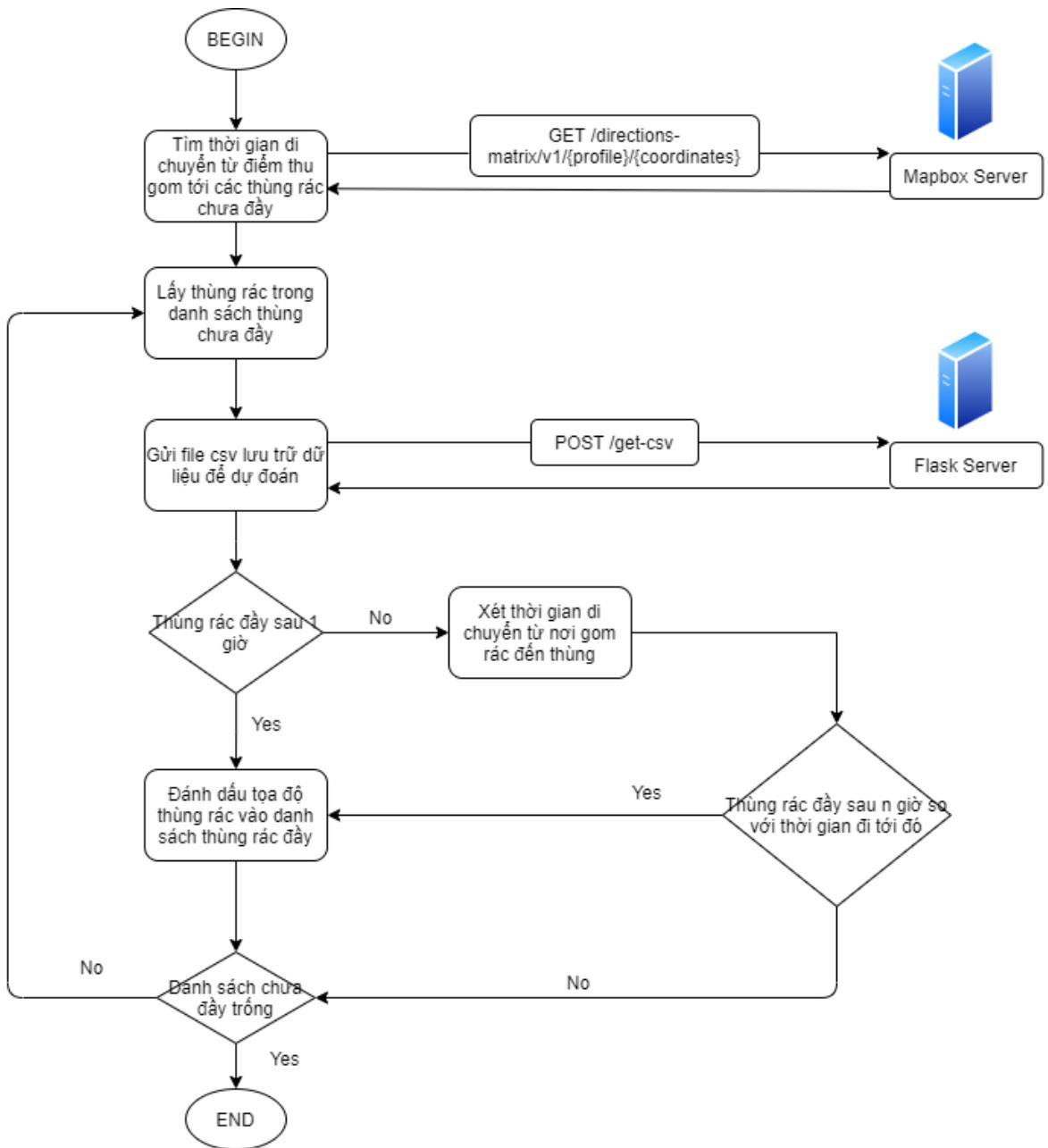
- * Trước khi tạo một thùng rác thật/ảo mới cũng phải điền đầy đủ những thông tin sau trong body:

- name (string): tên thùng rác.
 - type (string): loại thùng rác.
 - address (string): địa chỉ của thùng, nếu không có thì sẽ sử dụng API Geocoding từ Mapbox nhận vào tọa độ của thùng để định vị địa chỉ.
 - latitude (double): kinh độ.
 - longitude (double): vĩ độ.
 - nonrecycleHeight (double): chiều cao ngăn không tái chế.
 - recycleHeight (double): chiều cao ngăn tái chế.
- * Sau khi gửi body request như trên, một loạt hàm sẽ được chạy dựa trên cơ chế waterfall, sử dụng request để gọi API khởi tạo từ Thingsboard. Những API từ Thingsboard đều có JWT để bảo mật route, khi sử dụng phải cần có token đăng nhập lấy từ API POST /api/auth/log. Token sau đó được gắn vào header ở dạng 'Bearer {token}' ở trường 'X-Authorization'. Nếu không có token thì mọi API sẽ trả về status 401 unauthorized.
1. POST /api/asset để tạo mới thùng rác, nhận vào {name, type} của thùng rác. Nếu status trả về là 201 tức là tạo thùng rác thành công, còn 404 là báo lỗi trùng tên thùng rác. ID thùng rác trả về sẽ được lưu trữ.
 2. POST /api/device để tạo mới cảm biến, nhận vào {name, type} của cảm biến. Vì trong thùng luôn chứa hai cảm biến siêu âm để phân loại rác tái chế và không tái chế nên tên cảm biến sẽ được chuẩn hóa theo dạng '{name}_{function}', ví dụ: '{name}_taiche' và '{name}_khongtaiche'. Nếu status trả về 201 tức là tạo cảm biến thành công, còn 404 là báo lỗi trùng tên cảm biến. ID cảm biến trả về sẽ được lưu trữ.
 3. POST /api/plugins/telemetry/ASSET/{id thùng rác}/{attributes || timeseries} để thêm các thuộc tính hoặc giá trị theo thời gian cho thùng, những thuộc tính mặc định bao gồm {address, latitude, longitude, recycleHeight, nonrecycleHeight} còn giá trị

thời gian thì có mức rác 2 ngắn, giá trị tiên đoán và message thông báo. Nếu trường đã có thì sẽ thay đổi giá trị trường đó. Nếu status 201 tức là thêm thành công, 404 là báo lỗi thiếu dữ liệu.

4. POST /api/relation để tạo quan hệ một nhiều giữa thùng rác và cảm biến, nhận vào body dạng {from: {type: 'ASSET', id: {id thùng rác}}, to: {type: 'DEVICE', id: {id cảm biến}}}. Nếu status 201 tức là thành công, còn 404 là thiếu dữ liệu hoặc id không tồn tại.
 5. Sau khi thao tác thành công, thông tin thùng rác sẽ được lưu lại dưới server để xử lý dữ liệu và dự đoán.
- GET /api/devices/prediction: Xử lý dự đoán mức rác cho những thùng rác chưa đầy, được gọi vào 1 giờ trước khi thu gom rác.

Mô hình luồng xử lý được mô tả tổng quát ở hình 5.2



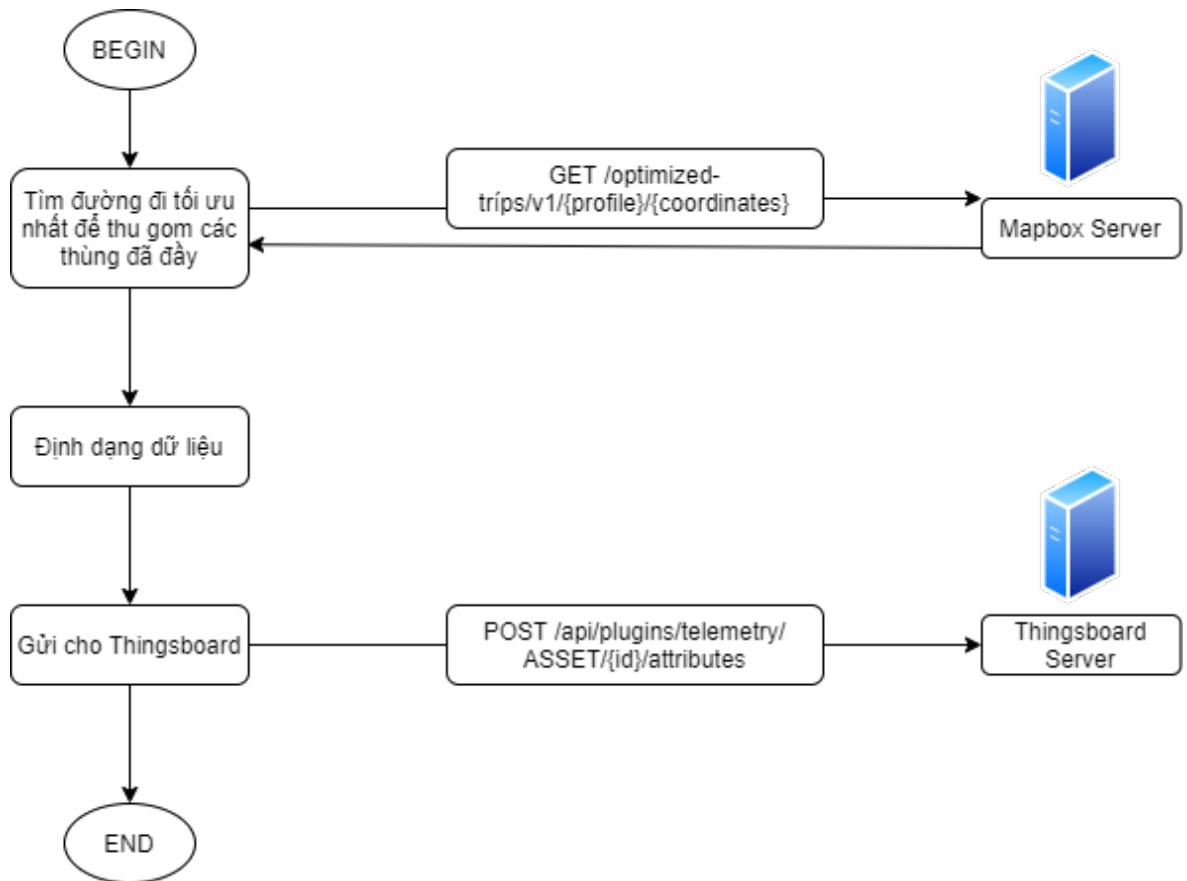
Hình 5.2 Sơ đồ miêu tả mô hình gửi dữ liệu tiên đoán và kiểm tra mức rác

1. Lấy tọa độ của những thùng rác chưa đầy, sau đó định dạng theo kiểu chuỗi '{lng_norigomrác},{lat_norigomrác};{lng_thùng1},{lat_thùng1};{lng_thùng2},{lat_thùng2} ... '.
2. Gọi API từ Mapbox GET /directions-matrix/v1/{profile}/{coordinates}?sources&annotations&access_token để lấy ma trận khoảng cách hoặc thời gian di chuyển giữa nhiều điểm. Những params và query được

mô tả chi tiết như sau:

- * profile (params): cấu hình ID phương tiện, giá trị bao gồm "driving", "walking", "cycling" và "driving-traffic".
 - * coordinates (params): danh sách tọa độ {kinh độ, vĩ độ} được phân tách bằng dấu chấm phẩy.
 - * sources (query): đánh chỉ tọa độ làm điểm nguồn tới tất cả tọa độ còn lại.
 - * annotations (query): dùng để chỉ định kết quả trả về, giá có thể có là "duration"(thời lượng) hoặc "distance"(khoảng cách) hoặc cả hai ngăn cách bằng dấu phẩy.
 - * access_token (query): token key được cung cấp bởi Mapbox
3. Kết quả trả về sẽ theo dạng ma trận 1xn (n là số thùng chưa đầy) [0, n1, n2, n3, ...], mỗi phần tử là thời gian di chuyển từ điểm nguồn tới từng tọa độ trong {coordinates}.
 4. Duyệt qua những thùng rác chưa đầy, khi file .csv lưu trữ dữ liệu quá ít thì sẽ không dự đoán, còn lại thì sẽ gửi file .csv qua Python Server để dự đoán mức rác vào một giờ sau (giờ gom rác).
 5. Kết quả trả về là một mảng một chiều, mỗi phần tử là lượng rác được dự đoán sau {index + 1} giờ. Nếu phần tử 0 (lượng rác vào lúc thu gom) lớn hơn hoặc bằng 90% chiều cao thùng tức là thùng đó dự đoán sẽ đầy sau một giờ và sẽ được đánh dấu thùng đầy.
 6. Đối với những thùng rác dự đoán chưa đầy vào một giờ, thì ta xét trong thời gian từ lúc bắt đầu tới khi đến được thùng rác, thùng có đầy không. Bằng cách lấy thời gian di chuyển đến thùng đó so với các khoảng giờ i (max=3), nếu thời gian nằm trong khoảng giờ i, thì sẽ lấy lượng rác dự đoán sau i + 1 giờ.
 7. Lấy kết quả dự đoán so với 90% của thùng, nếu được dự đoán sẽ đầy thì sẽ được đánh dấu vào những thùng đầy.
- GET /api/devices/static-map

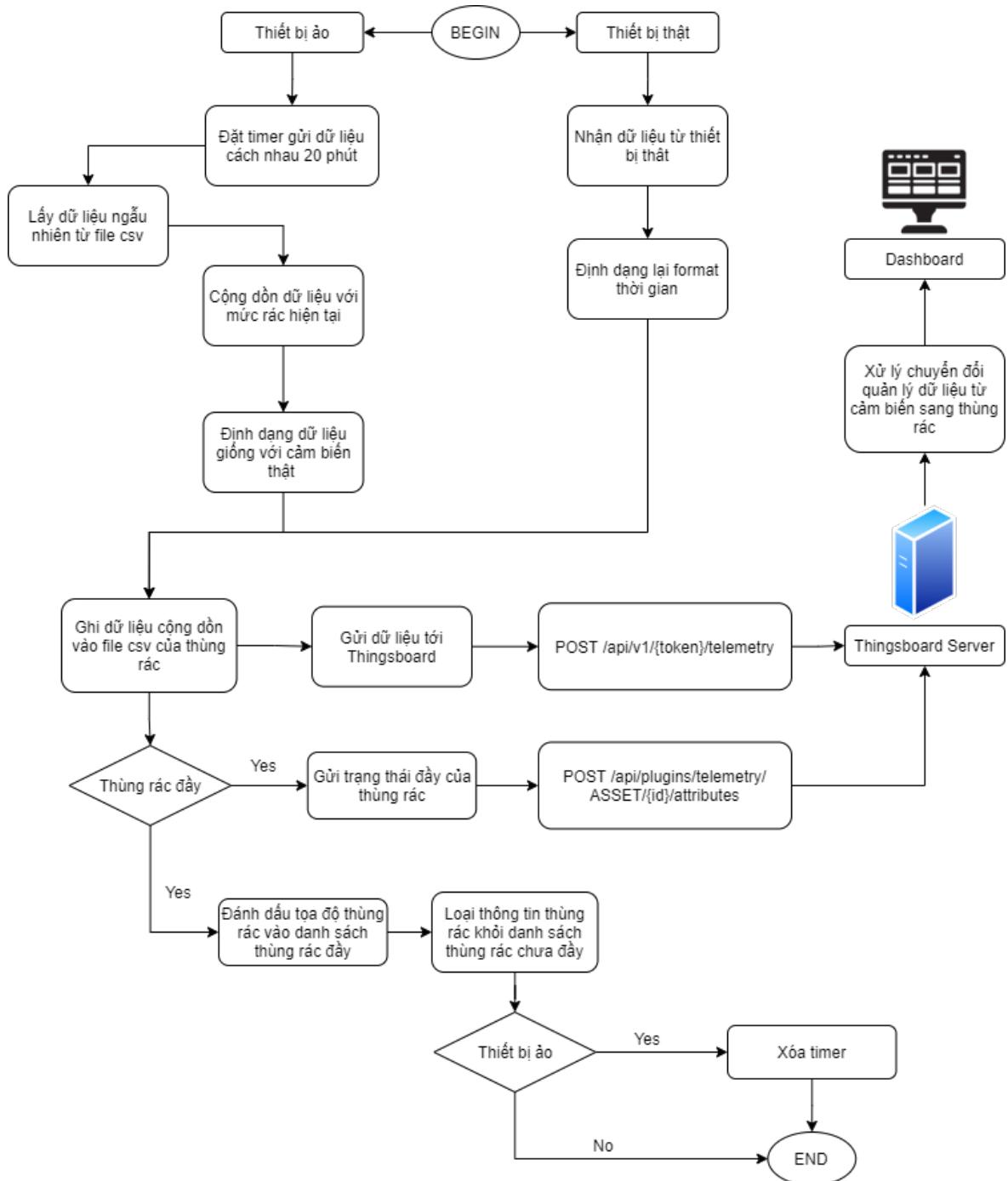
Mô hình luồng xử lý được mô tả tổng quát ở hình 5.3



Hình 5.3 Sơ đồ miêu tả luồng xuất mảng tọa độ đường đi tối ưu tới thùng rác đầy

1. Lấy tọa độ của những thùng rác đầy, sau đó định dạng theo kiểu chuỗi '{lng_nơiigomrác},{lat_nơiigomrác};{lng_thùng1},{lat_thùng1};{lng_thùng2},{lat_thùng2} ... '.
2. Gọi API từ Mapbox GET /optimized-trips/v1/{profile}/{coordinates}?language&roundtrip&geometries&steps&access_token để lấy ma trận khoảng cách hoặc thời gian di chuyển giữa nhiều điểm. Những params và query được mô tả chi tiết như sau:
 - * profile (params): cấu hình ID phương tiện, giá trị bao gồm "driving", "walking", "cycling" và "driving-traffic".
 - * coordinates (params): danh sách tọa độ {kinh độ, vĩ độ} được phân tách bằng dấu chấm phẩy.
 - * language (query): định dạng ngôn ngữ trả về ở văn bản hướng dẫn.

- * steps (query): trả về chi dẫn từng chặng chi tiết.
 - * roundtrip (query): cho biết tuyến đường có phải là khú hồi hay không.
 - * geometries (query): định dạng kiểu trả về, giá trị bao gồm "geojson", "polyline", "polyline6".
 - * access_token (query): token key được cung cấp bởi Mapbox.
3. Kết quả trả về là một mảng tọa độ chỉ đường đi đến tất cả thùng rác đầy, sau đó định dạng lại theo dạng `[[{lat1},{lng1}],[{lat2},{lng2}],...]`.
 4. Gửi kết quả định dạng lên Thingsboard thông qua API POST `/api/plugins/telemetry/ASSET/{id}/attribute` với id của nơi gom rác được tạo sẵn để hiển thị đường đi lên bản đồ Thingsboard.
- Xử lý dữ liệu cảm biến của thiết bị ảo và thiết bị thật:
 - Mô hình luồng xử lý được mô tả tổng quát ở hình 5.4



Hình 5.4 Sơ đồ miêu tả luồng tạo dữ liệu giả cho thiết bị ảo

1. Với thiết bị thật (POST /api/TTN/telemetry):

- * Những thiết bị thật sẽ gửi dữ liệu thông qua API này với định dạng là một object được tạo từ TTN được mô tả trong bảng 5.1

Key	Type	Value
app_id	string	Tên ứng dụng được đăng ký trên TTN
dev_id	string	Tên thiết bị được đăng ký trên TTN
hardware_serial	string	LoRaWAN DevEUI
port	number	LoRaWAN frame port
counter	number	LoRaWAN frame counter
payload_raw	string	Payload mã hóa bằng Base64
payload_fields	object	Đối tượng chứa kết quả của payload
metadata	object	Đối tượng chứa thông số của thiết bị và gateway
downlink url	string	URL đưa tín hiệu điều khiển trong khu vực, tên ứng dụng và khóa truy cập

Bảng 5.1 Bảng mô tả định dạng dữ liệu được gửi từ TTN

- * Trong payload_fields, đối tượng bao gồm tên cảm biến là key và giá trị cảm biến là value ({{ [sensorName]: [sensorValue] }}).
- * Trong metadata, đối tượng bao gồm các thông số của thiết bị và gateway (tần số, modulation, data_rate, coding_rate, ...), nhưng chú trọng nhất là time (thời gian server nhận được gói tin) và tọa độ của thiết bị.
- * Vì dữ liệu được gửi bằng cảm biến thật nên không cần thay đổi định dạng payload_fields của chúng, nhưng vì thời gian trong metadata là một chuỗi ISO string, chúng tôi phải định dạng lại kiểu thời gian thành dạng “DD/MM/YYYY hh:mm”, sau đó rồi mới đưa dữ liệu vào file csv.

2. Với thiết bị ảo (POST /api/devices/{deviceName}/random):
 - * Tiến hành tạo ngẫu nhiên dữ liệu của thùng rác có tên {deviceName}, cài đặt timer bằng cách setInterval với khoảng thời gian là 120.000ms (20 phút/1 lần).
 - * Dữ liệu được lấy ngẫu nhiên trong file random.csv với giá trị tái chế và không tái chế ngẫu nhiên từ 0 tới 13.
 - * Khởi tạo mức delta bằng 0, khi thùng rác chưa đầy ta lấy delta cộng với giá trị ngẫu nhiên đó và lấy chiều cao thùng rác trừ cho tổng giá trị sẽ ra được giá trị thật của cảm biến. Thực hiện đến khi nào thùng rác đầy thì giá trị ngẫu nhiên sẽ bằng 0 và khi đó giá trị sẽ dừng ở mức cao nhất của thùng rác.
 - * Định dạng dữ liệu giống với format payload_fields của thiết bị thật, đối với thời gian thì sử dụng moment để lấy thời gian thực.
3. Sau bước định dạng dữ liệu của 2 loại thiết bị, lưu dữ liệu vào file .csv của từng thùng với tên "{deviceName}_deltaHeight.csv", bên trong file bao gồm các cột timestamp (thời gian), recycleData (dữ liệu tái chế) và nonrecycleData (dữ liệu không tái chế). Đồng thời dữ liệu sẽ được gửi lên Thingsboard, chuyển đổi thực thể và hiển thị lên giao diện theo dạng biểu đồ đường.
4. Khi thùng rác đã đầy, ta sẽ gửi thông báo đến Thingsboard để hiển thị màu sắc, cùng lúc đó, đánh dấu thùng rác đó vào danh sách các thùng rác đầy và xóa khỏi danh sách các thùng rác chưa đầy. Đối với thiết bị ảo, sẽ dùng timer để giảm tải công việc cho server.

5.1.4 Cải thiện hiệu suất với Clustering

Khi chạy một quá trình của Nodejs, nó sẽ chạy trên một luồng duy nhất (single-thread), việc này có nghĩa là nó chỉ kết nối với một nhân (core) duy nhất trên máy chủ. Vì thế, để sử dụng một cách hiệu quả nhất sức mạnh các máy đa nhân (multi-core), ta sẽ phải chạy nhiều quá trình của Nodejs cùng một lúc. Các quá trình này có thể thực hiện ở ngoài Nodejs nhưng Cluster Module đã hỗ trợ cách phân luồng bên trong máy chủ để cải thiện thông lượng (request/s) và cập nhật ứng dụng

một cách dễ dàng vì đôi khi một số máy khách cần được phục vụ đồng thời.

Cluster Module cho phép khởi tạo các quy trình con (Workers) chạy đồng thời và chia sẻ cùng một port server. Mỗi Workers được tạo đều có vòng lặp sự kiện, bộ nhớ và phiên bản V8 của riêng nó, sử dụng IPC (Inter-process Communication) để giao tiếp với quy trình cha.

Để sử dụng chúng, ta phải cài đặt cluster trong ứng dụng Nodejs.

```
var cluster = require("cluster");
```

Sau đó, điều đầu tiên phải làm là xác định phần nào của đoạn mã dành cho Master Process và phần nào dành cho Workers Process.

```
if (cluster.isMaster) (...)
```

Nếu là Master Process, ta sẽ lấy số core có sẵn trên hệ thống và lặp lại số core để khởi tạo các Worker Process bằng phương thức fork().

```
let numCores = require("os").cpus().length;  
for (let i = 0; i < numCores; i++) { cluster.fork(); }
```

Một cluster module chứa một số sự kiện, hai trong số đó là sự kiện phổ biến liên quan đến thời điểm bắt đầu và kết thúc của Workers, đó là sự kiện “online” và “exit”. Ta gọi 2 sự kiện đó bằng cluster.on

- Sự kiện “online” được thực hiện khi Workers được phân nhánh và được gán id.
- Sự kiện “exit” được thực hiện nếu một Worker Process bị chết, sau đó bắt đầu quy trình mới bằng cách đơn giản là phân nhánh (fork) một quy trình khác.

Để đánh giá hiệu suất của Nodejs Server sử dụng Clustering Module khi xử lý một lượng lớn các request đến server, chúng tôi sử dụng loadtest để mô phỏng một số lượng lớn các kết nối đồng thời tới API để có thể đo lường hiệu suất của nó. Chúng tôi sử dụng loadtest vào API /api/devices/staticMap như sau:

```
$ loadtest http://localhost:1111/api/devices/staticMap -n 10000 -c 100 (với -n là tổng số request và -c là số request đồng thời trong đó)
```

Đầu tiên khi không sử dụng Clustering Module, chúng tôi được kết quả như hình 5.5:

```
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO Max requests: 10000
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO Concurrency level: 100
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO Agent: none
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO Completed requests: 10000
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO Total errors: 207
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO Total time: 1641.260661 s
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO Requests per second: 6
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO Mean latency: 16371.7 ms
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO Percentage of the requests served within a certain time
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO 50% 9511 ms
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO 90% 36757 ms
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO 95% 48883 ms
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO 99% 120004 ms
[Thu Jul 08 2021 16:02:12 GMT+0700 (Indochina Time)] INFO 100% 120127 ms (longest request)
```

Hình 5.5 Hiệu suất của Nodejs khi nhận 10000 request không Clustering

Sau đó, khi sử dụng Clustering Module, chúng tôi được kết quả như hình 5.6:

```
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO Max requests: 10000
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO Concurrency level: 100
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO Agent: none
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO Completed requests: 10000
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO Total errors: 196
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO Total time: 1612.6640393 s
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO Requests per second: 6
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO Mean latency: 15374.9 ms
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO Percentage of the requests served within a certain time
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO 50% 8309 ms
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO 90% 33943 ms
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO 95% 46590 ms
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO 99% 120002 ms
[Thu Jul 08 2021 15:30:38 GMT+0700 (Indochina Time)] INFO 100% 120633 ms (longest request)
```

Hình 5.6 Hiệu suất của Nodejs khi nhận 10000 request Clustering

Ta có thể thấy được khi Clustering số request/1 giây là 6 requests, bằng với khi không Clustering, nhưng độ trễ trung bình (mean latency) khi Clustering giảm một chút 15374.9 ms so với 16371.7 ms khi không Clustering.

Sau đó chúng tôi kiểm tra với số lượng request là 100 với request đồng thời là 100 cho API đó, kết quả khi không Clustering như hình 5.7 và khi Clustering như hình 5.8.

```
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO Max requests: 100
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO Concurrency level: 100
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO Agent: none
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO Completed requests: 100
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO Total errors: 0
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO Total time: 15.7770627 s
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO Requests per second: 6
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO Mean latency: 7237.4 ms
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO Percentage of the requests served within a certain time
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO 50% 6656 ms
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO 90% 10937 ms
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO 95% 12424 ms
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO 99% 15720 ms
[Thu Jul 08 2021 16:15:49 GMT+0700 (Indochina Time)] INFO 100% 15720 ms (longest request)
```

Hình 5.7 Hiệu suất của Nodejs khi nhận 100 request không Clustering

```
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO Max requests: 100
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO Concurrency level: 100
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO Agent: none
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO Completed requests: 100
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO Total errors: 0
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO Total time: 15.997002901 s
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO Requests per second: 6
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO Mean latency: 14194.7 ms
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO Percentage of the requests served within a certain time
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO 50% 14638 ms
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO 90% 15876 ms
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO 95% 15904 ms
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO 99% 15965 ms
[Thu Jul 08 2021 16:12:45 GMT+0700 (Indochina Time)] INFO 100% 15965 ms (longest request)
```

Hình 5.8 Hiệu suất của Nodejs khi nhận 100 request Clustering

Ở đây lượng request/1 giây của 2 trường hợp đều bằng nhau là 6 request nhưng lần này đối với độ trễ trung bình (mean latency) của trường hợp không Clustering lại nhỏ hơn rất nhiều 7237.4 ms so với khi Clustering 14194.7 ms.

Vì thế, chúng tôi kết luận rằng Cluster Module sẽ hoạt động tốt nhất đối với các thao tác xử lý một số lượng lớn request và trong đó, đoạn mã bao gồm nhiều vòng lặp lớn, tức là nó sẽ tỏa sáng khi liên quan đến các tác vụ đòi hỏi nhiều CPU. Tuy nhiên, nếu ứng dụng không chạy nhiều tác vụ đòi hỏi CPU thì việc tạo ra nhiều Workers sẽ khiến việc phân bổ tài nguyên bổ sung có thể không đáng bởi vì mỗi quy trình đều có bộ nhớ và phiên bản V8 của riêng nó. Vì hiện tại số lượng công việc tại Nodejs Server vẫn chưa đáng kể nên chúng tôi vẫn không khuyến khích sử dụng Clustering, nhưng khi mô hình được mở rộng với nhiều request hơn thì chúng tôi sẽ

cân nhắc sử dụng và kiểm tra hiệu suất chính xác.

5.2 Python Server

5.2.1 Giới thiệu tổng quát

Python là sự ưu tiên hàng đầu khi được lựa chọn ngôn ngữ cho các dự án liên quan đến AI. Vì thế sử dụng Python làm một server thứ ba cung cấp API xử lý chính các vấn đề về dự đoán lượng rác, sau đó việc trả kết quả về bên server chính để xử lý tiếp sẽ nhanh và hiệu quả hơn việc xử lý Deep Learning tại Nodejs.

Flask là framework được lựa chọn để xây dựng server vì tính năng hỗ trợ các yêu cầu RESTful và dễ dàng triển khai và tương tác với các server khác. Chức năng chính của server được mô tả tổng quát là cung cấp một API nhận và chuyển một tệp dạng .csv lưu trữ chuỗi thời gian đa biến thành các supervised learning. Sau đó sẽ chia tập lớn thành tập train và tập test, rồi đưa vào mô hình mạng LTSM để dự báo lượng rác sau từng khoảng thời gian.

5.2.2 Modules và packages

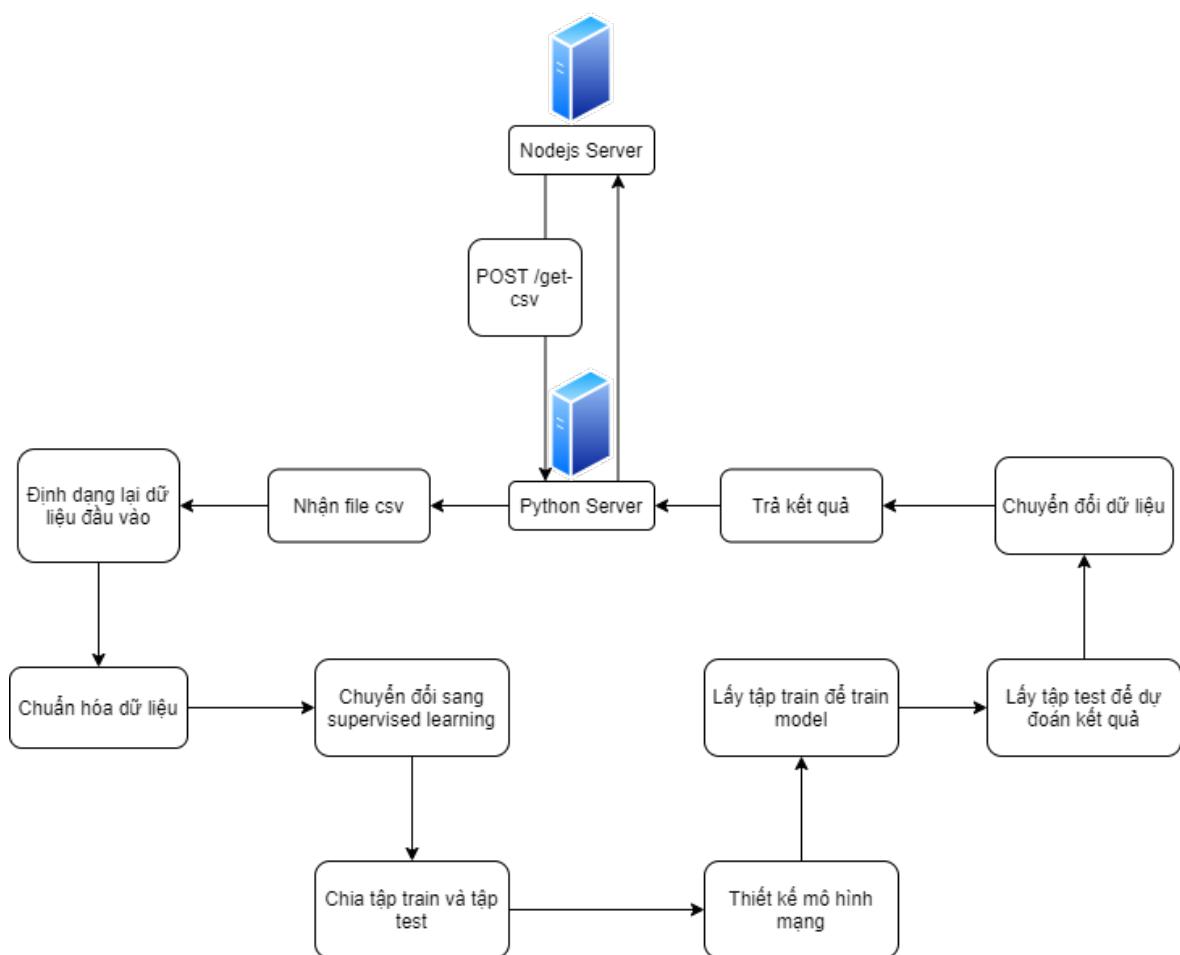
Flask có cung cấp thư viện mở rộng flask-restful hỗ trợ cho việc xây dựng các RESTful API một cách nhanh chóng, khuyến khích các phương pháp hay nhất với thiết lập tối thiểu. Ngoài ra, việc xử lý Deep Learning cần sử dụng một số thư viện quan trọng sau:

- tensorflow: thư viện mã nguồn mở dùng cho tính toán số học sử dụng đồ thị luồng dữ liệu, xử lý dựa trên sự thay đổi của dữ liệu.
- sklearn: thư viện cung cấp một tập các công cụ xử lý các bài toán máy học và mô hình thống kê.
- numpy: thư viện lõi của Python, hỗ trợ cho việc tính toán dãy số và ma trận nhiều chiều với tốc độ xử lý nhanh.
- math: thư viện hỗ trợ các hàm toán học tiêu chuẩn C.
- matplotlib: thư viện hỗ trợ vẽ đồ thị hai chiều, ba chiều mạnh mẽ.

- pandas: thư viện hỗ trợ xử lý và phân tích các dữ liệu có cấu trúc (dạng bảng, đa chiều, ...) và dữ liệu chuỗi thời gian nhanh, mạnh mẽ và linh hoạt.

5.2.3 API

Server cung cấp 1 API GET /get-csv với luồng dữ liệu được mô tả tổng quát ở hình 5.9



Hình 5.9 Sơ đồ mô tả tổng quát mô hình Deep Learning dự đoán lượng rác

Luồng dữ liệu được mô tả chi tiết như sau:

- Server nhận vào một tệp .csv được gửi từ Nodejs Server, bao gồm các cột [thời gian, lượng rác tái chế, lượng rác không tái chế].
- Tách riêng 2 cột tái chế và không tái chế thành 2 mảng 2 chiều riêng biệt dưới dạng [[a1],[a2],[a3],...].

3. Chuyển các mảng thành dạng mảng Numpy và định dạng kiểu float32 để thuận tiện sử dụng.
4. Chuẩn hóa các mảng bằng MinMaxScaler giúp thuật toán dễ dàng xử lý hơn.
5. Quy định lấy 75% chuỗi dữ liệu để làm tập train và phần còn lại để làm tập test. Chuyển đổi tất cả các tập sang supervised learning với số time_step tự quy định (số time_step trước đó được sử dụng để dự đoán giá trị ở time_step tiếp theo).
6. Kết quả trả về là một mảng 2 chiều có shape là (số lượng dữ liệu, time_steps). Reshape các tập train input lại thành kiểu (samples, timeseries, features) để đưa vào mô hình dự đoán.
7. Xây dựng mô hình mạng Stacked LSTM như bảng 5.2

Model: "sequential"		
Layer (type)	Output Shape	Param #
Istm (LSTM)	(None, 60, 128)	66560
Istm_1 (LSTM)	(None, 128)	131584
dense (Dense)	(None, 1)	129
Total params: 198, 273 Trainable params: 198, 273 Non-trainable params: 0		

Bảng 5.2 Tổng quan mô hình mạng LSTM

bằng cách tạo một mô hình Sequential và thêm nhiều layer vào trong, cụ thể là 2 LSTM layers và 1 Dense layer với các tham số. Sau đó model sẽ compile với các tham số để train model (loss, optimizer, metrics) và fit data vào để train (bao gồm tổng cộng 198.273 params có thể train được).

8. Ta lấy {time_step} phần tử cuối từ tập test với t là thời điểm của giá trị cuối cùng để dự đoán dữ liệu ở thời điểm t+1 ... và tiếp đó chạy vòng lặp để dự đoán

dữ liệu ở $t+2$, $t+3$, $t+4$... bằng cách lấy giá trị dự đoán ở $t+1$ cho vào mảng, dịch phần tử đầu tiên ra khỏi mảng và tiếp tục dự đoán $t+2$.

9. Trả kết quả dự đoán mức rác tái chế và không tái chế về Nodejs Server để tiếp tục xử lý.

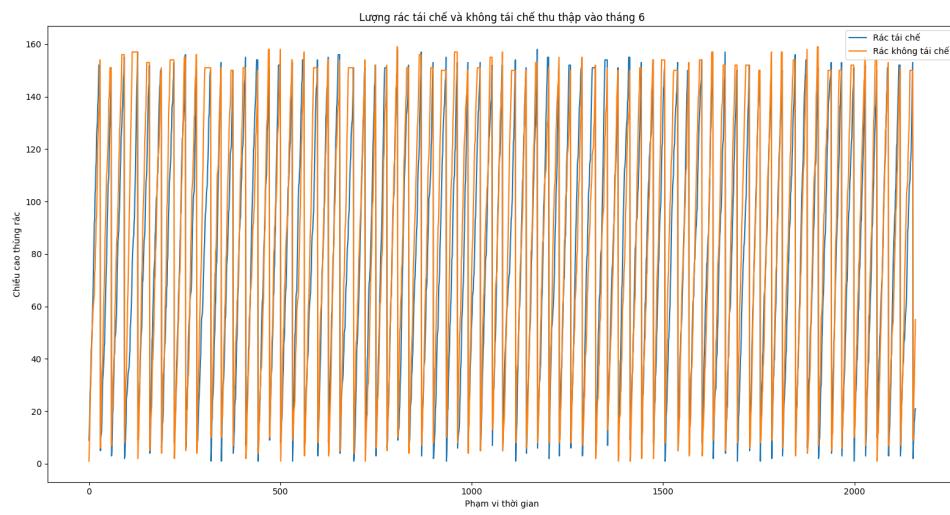
5.2.4 Đánh giá dữ liệu

Chúng tôi gửi một tệp .csv của một thùng rác có chuỗi dữ liệu thu thập mỗi 20 phút vào tháng 6 với tổng 2161 dữ liệu như bảng 5.3:

Timestamp	01/06/2021 00:00	01/06/2021 00:20	01/06/2021 00:40	01/06/2021 01:00	...	30/06/2021 22:40	30/06/2021 23:00	30/06/2021 23:20	30/06/2021 23:40
Recycle Data	9	12	18	27	...	16	17	20	21
Non-recycle Data	1	7	12	22	...	30	36	49	55

Bảng 5.3 Cấu trúc file CSV

hiển thị dưới dạng biểu đồ hình 5.10



Hình 5.10 Biểu đồ lượng rác tái chế và không tái chế thu thập vào tháng 6

timestamp: mốc thời gian

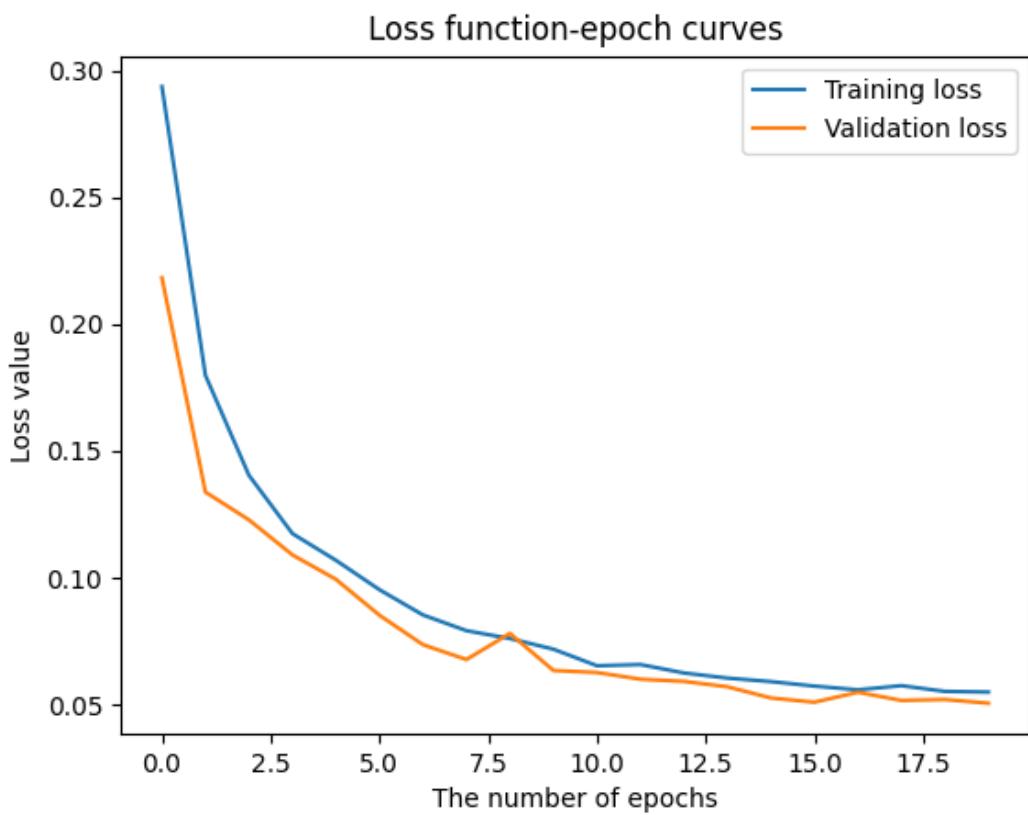
recycle data: lượng rác tái chế

non-recycle data: lượng rác không tái chế

Tập dữ liệu được chuyển thành supervised learning với time_steps là 60 (tức là lấy dữ liệu 20 tiếng trước để dự đoán) và chia thành 75% cho tập train (1620) và phần còn lại (541) cho tập test.

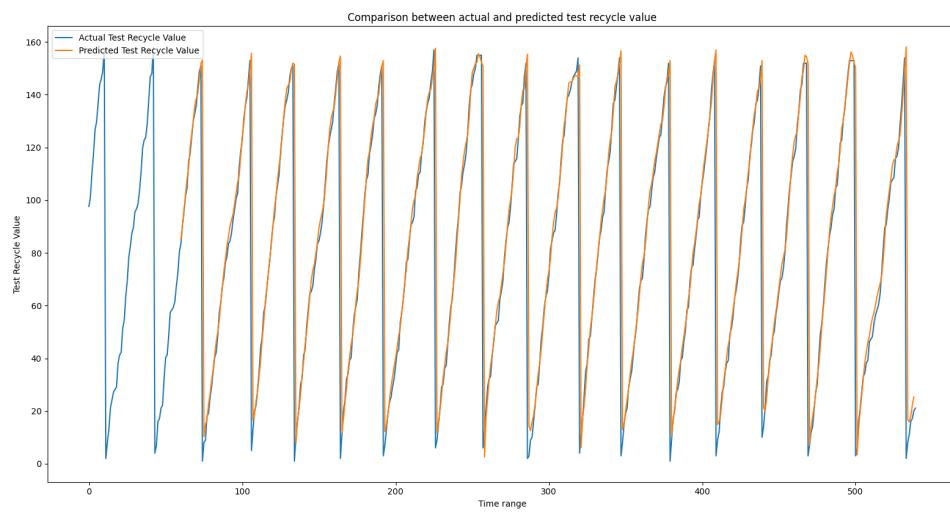
Sau khi train mô hình, chúng tôi đánh giá model bằng chỉ số loss để đo tỉ lệ chênh lệch giữa tập dữ liệu thực và tập dữ liệu model dự đoán.

Kết quả train mô hình với 20 epochs cho thấy loss nhỏ nhất trên tập train là 0.05242335423827171 và trên tập test là 0.047845833003520966, được biểu diễn ở biểu đồ 5.11

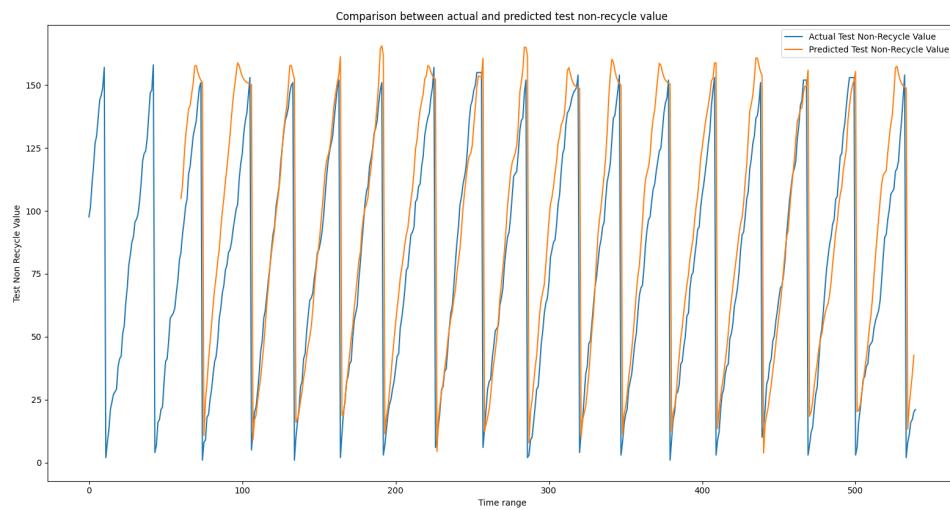


Hình 5.11 Giá trị Train loss và Val loss

Sau đó chúng tôi sử dụng tập test để dự đoán và so sánh kết quả dự đoán với dữ liệu thực như hình 5.12 cho mức rác tái chế và hình 5.13 cho mức rác không tái chế với kết quả RMSE và MSE cho tập test mức rác tái chế là 97.511 và 86.812, RMSE và MSE cho tập test mức rác không tái chế là 104.531 và 93.270.

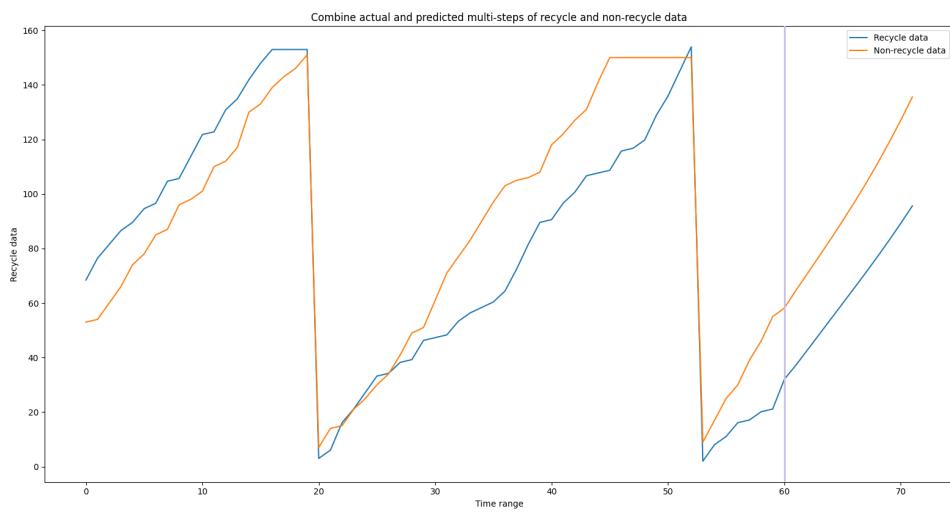


Hình 5.12 So sánh actual với predicted của tập test tái chế



Hình 5.13 So sánh actual với predicted của tập test không tái chế

Ở bước cuối cùng chúng tôi sử dụng model để dự đoán mức rác vào 12 time_steps (4 tiếng tiếp theo), kết quả hiển thị như hình 5.14



Hình 5.14 Biểu đồ mức rác tái chế và không tái chế sau 4 tiếng tiếp theo

5.3 Thingsboard

5.3.1 Giới thiệu tổng quát

Việc xử lý dữ liệu ở server vẫn chưa đủ, thông tin dữ liệu của thùng rác cần được hiển thị cho cả admin và người dùng quan sát. Vì thế, Thingsboard vừa được sử dụng như một Frontend đáp ứng việc này đồng thời cung cấp một doc cung cấp những API để hỗ trợ đưa thông tin lên giao diện.

Chúng tôi sử dụng bản demo của Thingsboard cho dự án để giảm bớt thời gian cài đặt cho bản Localhost và chi phí cho bản Premium, tuy vậy những tính năng mà bản demo mang lại vẫn đáp ứng đầy đủ những yêu cầu mà dự án cần.

5.3.2 Assets và Devices

Cả asset và device đều là một thực thể của Thingsboard, ở trong dự án này asset được quy định là thùng rác và device được quy định là cảm biến của thùng rác đó. Những thực thể có thể được tạo thủ công trên giao diện hoặc tự động bằng Thingsboard API, và sẽ được lưu trữ tại database PostgreSQL.

Thingsboard cung cấp 2 tab bao gồm danh sách tất cả asset và device đã tạo

núi hình 5.15 và 5.16.

The screenshot shows the ThingsBoard interface with the 'Assets' tab selected. The left sidebar includes 'Home', 'Rule chains', 'Customers', 'Assets' (selected), 'Devices', 'Device profiles', 'OTA updates', 'Entity Views', 'Edge instances', 'Edge management', 'Widgets Library', 'Dashboards', 'Audit Logs', 'API Usage', and 'System Settings'. The main content area is titled 'Assets' with a dropdown for 'Asset type' set to 'All'. A table lists 12 assets, each with a checkbox, name, asset type ('smart_bin'), label, customer, and public status. The assets are dated from June 16 to September 20, 2021.

	Created time	Name	Asset type	Label	Customer	Public	Actions
<input type="checkbox"/>	2021-06-16 14:06:49	Khanhh	smart_bin			<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:29:19	May	smart_bin			<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:29:11	Oct	smart_bin			<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:21:35	Jan	smart_bin			<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:21:26	Jun	smart_bin			<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:21:08	Apr	smart_bin			<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:21:01	Aug	smart_bin			<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:20:55	Sep	smart_bin			<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:20:47	Jul	smart_bin			<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:20:40	Feb	smart_bin			<input type="checkbox"/>	

Hình 5.15 Danh sách thùng rác

The screenshot shows the ThingsBoard interface with the 'Devices' tab selected. The left sidebar includes 'Home', 'Rule chains', 'Customers', 'Assets', 'Devices' (selected), 'Device profiles', 'OTA updates', 'Entity Views', 'Edge instances', 'Edge management', 'Widgets Library', 'Dashboards', 'Audit Logs', 'API Usage', and 'System Settings'. The main content area is titled 'Devices' with a dropdown for 'Device profile' set to 'All'. A table lists 12 devices, each with a checkbox, name, device profile ('ultra_sonic_sensor'), label, customer, and public status. The devices are dated from June 16 to September 20, 2021.

	Created time	Name	Device profile	Label	Customer	Public	Is gateway	Actions
<input type="checkbox"/>	2021-06-16 14:06:49	Khanhh_khongtaiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-16 14:06:49	Khanhh_taiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:29:19	May_khongtaiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:29:19	May_taiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:29:11	Oct_taiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:29:11	Oct_khongtaiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:21:35	Jan_khongtaiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:21:35	Jan_taiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:21:26	Jun_khongtaiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	
<input type="checkbox"/>	2021-06-09 13:21:26	Jun_taiche	ultra_sonic_sensor			<input type="checkbox"/>	<input type="checkbox"/>	

Hình 5.16 Danh sách cảm biến

Mỗi thực thể đều có những đặc điểm sau:

- Details: chứa thông tin cơ bản của thực thể và một id phân biệt.
- Attributes: thuộc tính của thực thể, có thể cập nhật thủ công hoặc tự động thông qua Thingsboard API (hình 5.17).
- Latest Telemetry: dữ liệu theo thời gian của thực thể, cập nhật thông qua Thingsboard API, mang giá trị gần đây nhất.
- Alarms: danh sách các báo động của thực thể.
- Events: các sự kiện giúp theo dõi tình trạng của thực thể.
- Relations: các mối quan hệ giữa các thực thể với nhau (hình 5.18).
- Audit Logs: nhật ký kiểm tra theo dõi hành động người dùng liên đến

các thực thể.

Key	Value	
Last update time	2021-06-16 14:06:50	
address	Nhà văn hóa Thiếu nhi	
latitude	10.851299	
longitude	106.767518	
nonrecycleHeight	190	
recycleHeight	190	

Hình 5.17 Thuộc tính thùng rác

Type	To entity type	To entity name	
Contains	Device	Khanhh_taiche	
Contains	Device	Khanhh_khongtaiche	

Hình 5.18 Mối quan hệ của thùng rác với cảm biến

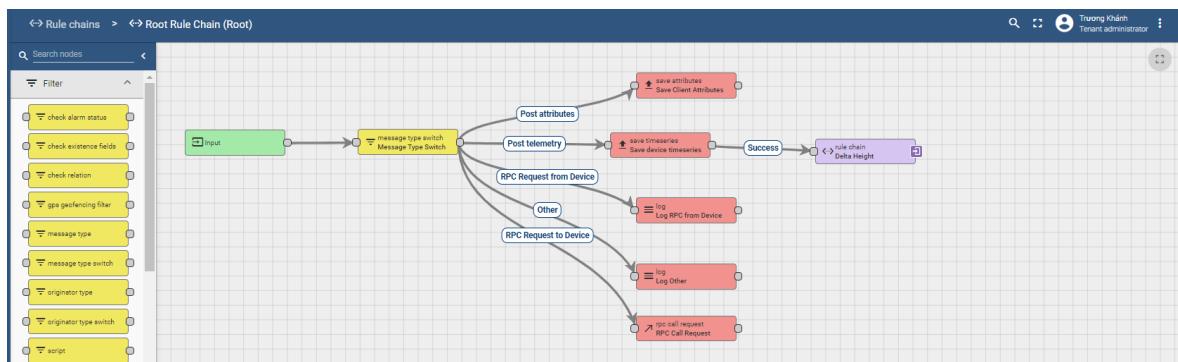
5.3.3 Rule Chains

Rule chains là những khung logic dạng kéo thả (nodes) được nối lại với nhau thành một chuỗi logic (rule chain) dùng để xử lý dữ liệu trực tiếp trên Thingsboard. Những nodes mà Thingsboard cung cấp đều có một chức năng khác nhau và được chia thành từng danh mục theo từng màu.

- Filter (vàng): lọc những gói tin tới bằng những điều kiện được lập trình.
- Enrichment (xanh lá): thêm thông tin lấy từ thực thể và gắn vào gói tin.
- Transformation (xanh dương): thay đổi thông tin gói tin bằng mã lập trình.

- Action (đô): tạo những action cho thực thể để xử lý dữ liệu.
- External (cam): tương tác với các hệ thống bên ngoài.
- Rule Chain (tím): gửi gói tin tới các rule chains khác.

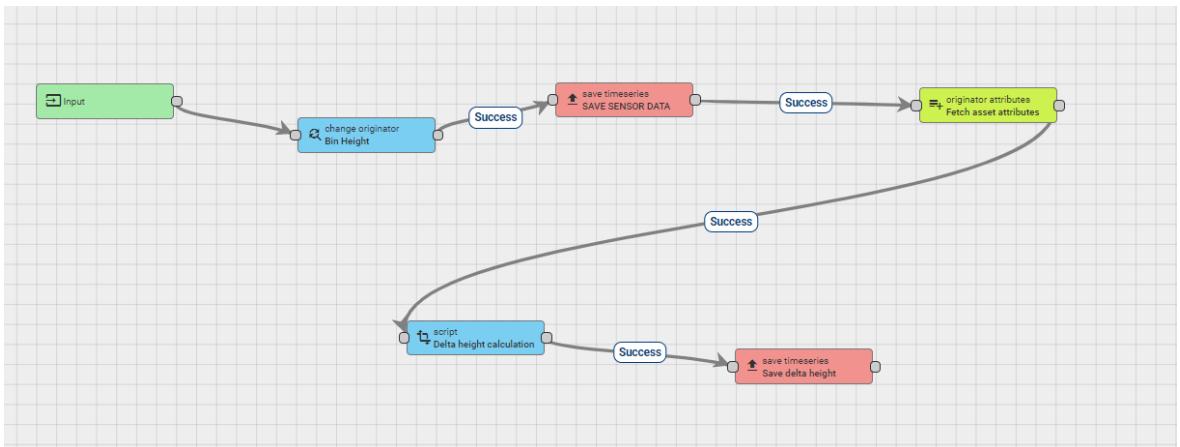
Vì dữ liệu lùi lượng rác được gửi từ các cảm biến và thùng rác phải sở hữu và định dạng lại dữ liệu để hiển thị lên biểu đồ đường, nên chúng tôi sử dụng rule chain để lưu dữ liệu vào cảm biến, đổi thực thể sở hữu sang thùng rác và định dạng lại dữ liệu thành chiều cao của thùng. Rule chain được mô tả ở hình 5.19 và Delta chain 5.20



Hình 5.19 Root Rule Chain

Root Rule Chain là luồng xử lý chính cho Thingsboard, tất cả gói tin input được gửi từ hệ thống bên ngoài sẽ được lưu vào khung input. Sau đó gói tin sẽ được lọc theo {MESSAGE_TYPE} bằng khung Message Type Switch, những {MESSAGE_TYPE} bao gồm những sự kiện khác nhau liên quan đến thực thể.

Những thuộc tính của thùng rác sẽ được lọc qua luồng Post attributes và được lưu vào thực thể, còn những dữ liệu thời gian của thùng rác và cảm biến sẽ được lọc qua luồng Post telemetry, được lưu vào thực thể và được gửi qua Delta Rule Chain. Những gói tin thuộc {MESSAGE_TYPE} khác sẽ đi vào luồng khác hoặc được log qua Audit Logs.



Hình 5.20 Delta-calculation Rule Chain

Gói tin được lọc từ type Post telemetry sẽ được lưu ở khung input, sau đó dữ liệu mức rác sẽ được chuyển thực thể từ cảm biến sang của thùng rác bằng khung Change originator và được lưu vào thùng rác.

Dữ liệu sau đó được định dạng lại giống chiều cao thùng rác bằng cách lấy chiều cao thùng trừ cho dữ liệu, vì thế sử dụng khung Originator attributes để fetch thuộc tính thực thể và đưa tất cả qua một Transformation script để xử lý logic.

Xử lý dữ liệu mức rác cũng khá đơn giản khi chỉ cần phân chia dữ liệu tái chế và không tái chế và lấy ngăn chiều cao trừ cho dữ liệu, kết quả gửi qua khung Save timeseries để lưu dữ liệu vào thùng rác.

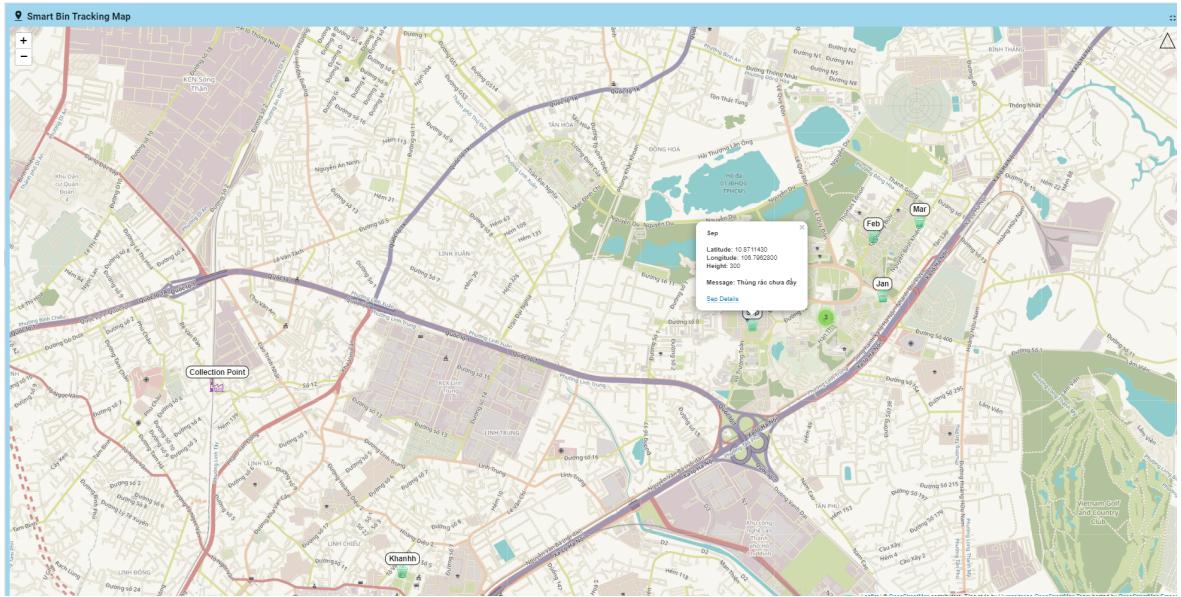
5.3.4 Dashboard

Dashboard là khung màn hình hiển thị chính của Thingsboard, được tạo bởi các widget hỗ trợ sẵn (biểu đồ, bản đồ, bảng, input tương tác, alarms, ...), hỗ trợ hiển thị, tương tác và trực quan hóa dữ liệu theo nhiều cách.

Dashboard cho phép chúng ta tạo ra nhiều các dashboard khác, trong một dashboard, ta có thể có nhiều state và action event để di chuyển giữa các state, nhiều Entity aliases để phân biệt và gom cụm các thực thể lại với nhau và thêm vào các widget để lấy dữ liệu dễ dàng hơn.

Ở trang dashboard chính, chúng tôi cho hiển thị bản đồ hiện tất cả những thùng rác được tạo như hình 5.21 và một danh sách các thùng rác gồm các thuộc

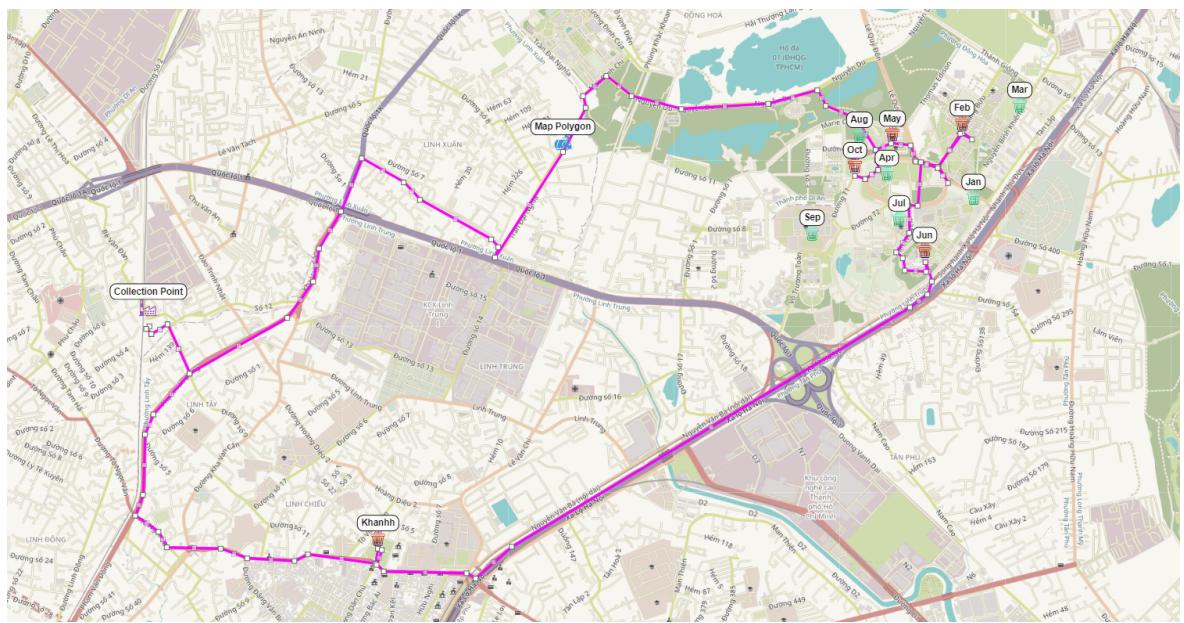
tính của nó như hình 5.23



Hình 5.21 Bản đồ theo dõi vị trí thùng rác

Bản đồ hiển thị tất cả các thùng rác và địa điểm thu gom rác, được phân biệt bằng icon khác nhau và các thùng rác được clustering để gom cụm từng vùng. Với mỗi thùng rác, khi nhận được trạng thái đầy sẽ chuyển thành màu đỏ để dễ nhận biết. Ngoài ra, mỗi thùng rác còn có tooltip hiện tên, tọa độ, chiều cao, chuỗi thông báo và một đường link dẫn đến state chi tiết của thùng rác.

Bản đồ này cũng tích hợp tính năng Polygon để vẽ tuyến đường, có thể thích ứng với chức năng tìm đường tối ưu nhất từ nơi gom rác đến tất cả thùng rác đầy như hình 5.22.



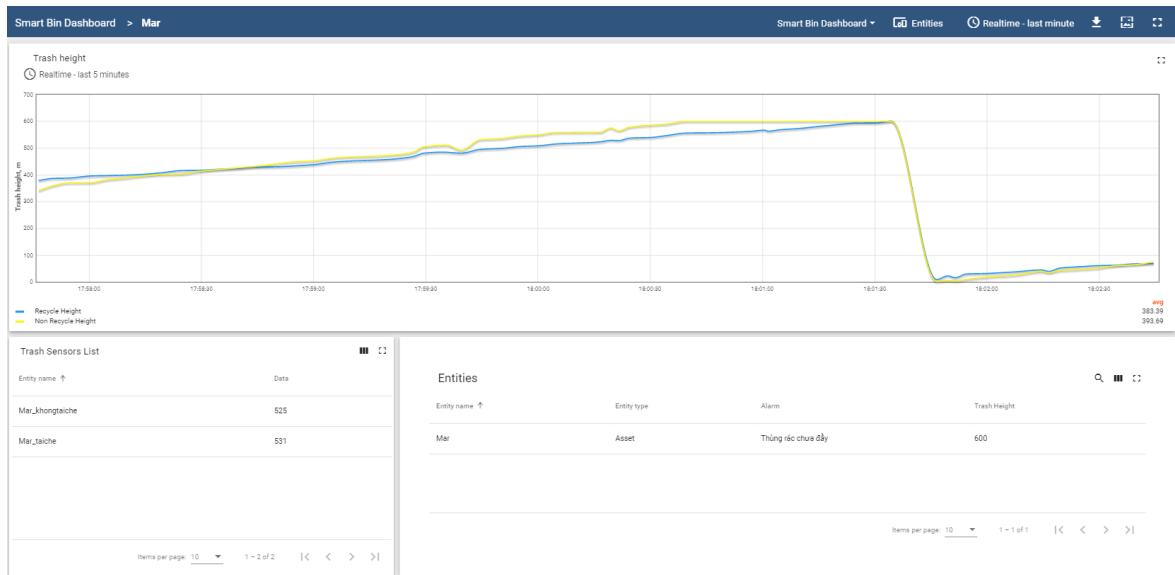
Hình 5.22 Polygon các tọa độ đường đi trên bản đồ

Entity name	Entity type	Address	Non-Recycle Height	Recycle Height	latitude	longitude
Apr	Asset	Nhà văn hóa SV	400	400	10.87502	106.80127
Aug	Asset	Phố thông năng khiếu	550	550	10.877918	106.799437
Feb	Asset	KTX Khu A	500	500	10.878273	106.806262
Jan	Asset	Dai hoc An ninh nhân dân	320	320	10.873467	106.807002
Jul	Asset	Dai hoc KHH & NV	150	150	10.872084	106.802031
Jun	Asset	Dai hoc CNTT	100	100	10.870034	106.803754
Khanhh	Asset	Nhà văn hóa Thiếu nhi	190	190	10.851299	106.767518
Mar	Asset	Cafe Heme	600	600	10.879462	106.810051
May	Asset	Dai hoc Quốc tế	270	270	10.877621	106.801622
Oct	Asset	Dai hoc KHTN	200	200	10.875556	106.799141

Hình 5.23 Danh sách thùng rác hiện có

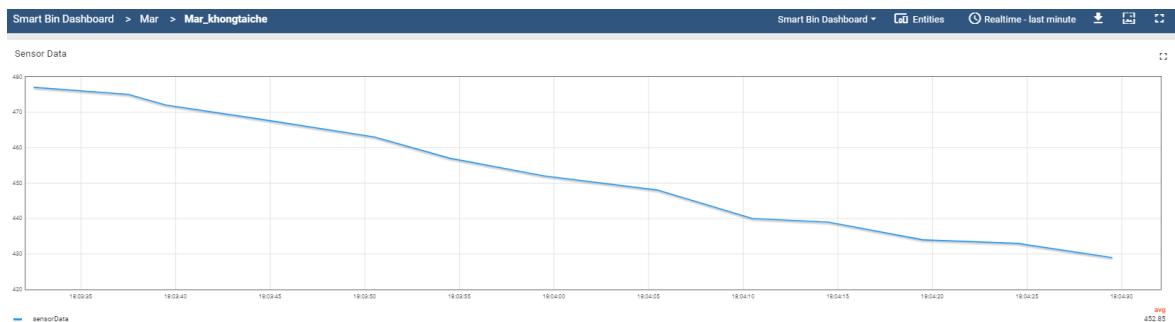
Danh sách các thùng rác bao gồm tên, loại, địa chỉ, tọa độ và chiều cao hai ngăn của thùng, mỗi thùng rác được gắn sự kiện event khi nhấn vào thì sẽ chuyển qua state dashboard khác, cụ thể là state hiển thị dữ liệu của thùng rác và các cảm biến của nó.

Ở state dashboard chi tiết từng thùng rác, giao diện được bố trí như hình 5.24, với một biểu đồ đường hiển thị mức rác tái chế và không tái chế của thùng, ở dưới bên trái là danh sách các cảm biến của thùng rác cùng với dữ liệu của cảm biến, còn ở bên phải là bảng thông báo tình trạng của thùng rác bao gồm tình trạng chưa đầy, đã đầy hoặc dự tính đã đầy trong một khoảng thời gian tới.



Hình 5.24 Giao diện chi tiết của thùng rác thông minh

Khi nhấn vào mỗi cảm biến của thùng rác, dashboard sẽ chuyển sang state mới hiển thị biểu đồ dữ liệu của cảm biến theo thời gian như hình 5.25.



Hình 5.25 Biểu đồ thể hiện dữ liệu của cảm biến

Chương 6 KHÓ KHĂN VÀ HƯỚNG PHÁT TRIỀN ĐỀ TÀI

6.1 Khó khăn

6.1.1 Về thiết bị

Khi setup node, nhóm gặp vấn đề về frequency, node không thể gửi data cho gateway.

6.1.2 Về server

- Việc xử lý dự đoán chuỗi dữ liệu rác của nhiều thùng rác mất thời gian dài để train vì model sử dụng nhiều layer, khi thực hiện với một số lượng lớn thùng rác thì có thể bị trễ giờ thu gom. Nếu mô hình sử dụng nhiều layer và epochs để train thì có thể mất càng nhiều thời gian hơn nữa.
- Với kiến thức hạn chế về Deep Learning nên còn gặp khá nhiều sai sót trong quá trình xây dựng và kiểm thử mô hình. Ngoài ra, dữ liệu rác thực tế vẫn còn rất ít và đơn giản, chưa đủ để kết luận và đánh giá chính xác được hiệu suất của mô hình.
- Server của Thingsboard rất dễ xảy ra sự cố timeout khi gửi nhiều request cùng một lúc, gây ra hiện tượng mất gói tin đến và dẫn đến sự thiếu hụt thông tin. Sau này việc mở rộng dự án với số lượng request thùng rác ngày càng tăng là một vấn đề cần được giải quyết.
- Những API miễn phí cung cấp từ Mapbox chỉ hạn chế số lượng request ở từng API như 100.000 Optimization, 100.000 Matrix, .../1 request. Vì thế, khi dự án kéo dài phải tính thêm việc trả phí để sử dụng những tính năng từ Mapbox.
- Những widget kéo thả cung cấp bởi Thingsboard rất tiện lợi và đa dạng nhưng mang lại phong cách khá đơn giản và một số cái khá rập khuôn khi sử dụng, ngoài ra, vẫn chưa tận dụng hết những tính năng mà Thingsboard mang lại.

6.2 Hướng phát triển đề tài

6.2.1 Về thiết bị

6.2.2 Về server

- Mô hình thu gom có hướng mở rộng khả quan, cụ thể là với mỗi quan hệ liên quan chặt chẽ giữa các đối tượng với nhau, dự án có thể phát triển thêm nhiều nơi gom rác khác nhau và một nơi gom rác có thể quản lý nhiều thùng rác trong bán kính cụ thể. Khi đạt tới một mức nhất định, chúng tôi cần nhắc sử dụng Cluster Module cho server để xử lý đa luồng nhiều request.
- Khi mở rộng mô hình, cần cải tiến thêm chức năng thu gom tọa độ để vẽ đường đi tối ưu cho từng cụm thu gom, hạn chế việc di chuyển đến các thùng rác ở khoảng cách xa, ngoài ra trong quá trình dự đoán các thùng rác còn trống cần lọc ra những thùng rác nào thực sự cần phải dự đoán để giảm thiểu công việc xử lý cho Python Server.
- Ngoài ra, khi số lượng thùng rác tăng lên đáng kể, việc quản lý các cụm thùng rác cần phải được phân bổ cho các Customers để quan sát và thông báo dữ liệu rác. Khi đó, việc xác định những thùng rác đầy sẽ là nhiệm vụ của các Customers và chúng tôi sẽ điều chỉnh lại chức năng tìm đường ngắn nhất sao cho phù hợp nhất.
- Tuy mô hình dự đoán vẫn chưa ổn định, song nếu có thời gian hoàn thiện và kiểm thử nhiều dạng dữ liệu với số lượng lớn hơn thì có thể đưa ra đánh giá hiệu suất chính xác và giải pháp cải tiến thuật toán cho mô hình.
- Về phần giao diện, vì tính tiện lợi, mở rộng và hiệu quả của Thingsboard cung cấp, chúng tôi cần nâng cấp giao diện lên bản Professional Edition vì tính ổn định và nhiều tính năng khác.

TÀI LIỆU THAM KHẢO

- [1] Gary Thung, Mindy Yang. *Classification of trash for recyclability status*. CS229 Project Report 2016, 2016.
- [2] De Carolis, Berardina, Francesco Ladogana, and Nicola Macchiarulo. *YOLO TrashNet: Garbage Detection in Video Streams*. 2020 IEEE Conference on Evolving and Adaptive Intelligent Systems (EAIS). IEEE, 2020.
- [3] Liu Yunpeng, Di Hou, Bao Junpeng, Qi Yong. *Multi-step Ahead Time Series Forecasting for Different Data Patterns Based on LSTM Recurrent Neural Network*. 2017 14th Web Information Systems and Applications Conference (WISA). IEEE 2017.
- [4] XiaoFeng Wang, Ying Zhang. *Multi-Step-Ahead Time Series Prediction Method with Stacking LSTM Neural Network*. 2020 3rd International Conference on Artificial Intelligence and Big Data (ICAIBD). IEEE, 2020.
- [5] Bipin Aasi, Syeda Aniqa Imtiaz, Hamzah Arif Qadeer, Magdalean Singarajah, Rasha Kashef. *Stock Price Prediction Using a Multivariate Multistep LSTM: A Sentiment and Public Engagement Analysis Model*. 2021 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS). IEEE, 2021.