HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA KỸ THUẬT ĐIỆN TỬ I

----🖂 🎖 ★----

ĐỒ ÁN
HỆ THỐNG NHÚNG

Đề tài: Xây dựng hệ thống nhận diện giọng nói dựa trên TinyML

Giảng viên hướng dẫn: Th.S CHU VĂN CƯỜNG

Sinh viên thực hiện : Thái Minh Vũ-B21DCDT250

Nguyễn Văn Hải Anh-B21DCDT039

Nguyễn Quang Anh-B21DCDT036

Trần Thanh Tùng-B21DCDT240

Lớp :

Khóa : 2021 – 2026

Hệ : ĐẠI HỌC CHÍNH QUY

NHẬN XÉT CỦA GIÁO VIÊN HƯỚNG DẪN

Điểm: (bằng chữ:……..)

…… , ngày … tháng … năm 2023

CÁN BỘ - GIẢNG VIÊN HƯỚNG DẪN

(ký, họ tên)

Chu Văn Cường

LỜI CẢM ƠN

Để hoàn thành tốt đề tài tốt nghiệp này, ngoài sự nỗ lực của bản thân, chúng em còn nhận được rất nhiều sự giúp đỡ quý báu từ các thầy cô cùng các bạn.

Chúng em xin chân thành cảm ơn các thầy cô trong khoa Kỹ thuật Điện tử 1 - Học viện Công nghệ Bưu chính Viễn thông đã tận tình giảng dạy và truyền đạt kiến thức cho chúng em trong suốt quá trình học tập tại trường. Những kiến thức và kỹ năng chúng em có được là nhờ sự hướng dẫn nhiệt tình của thầy cô, và đó là nền tảng giúp chúng em hoàn thành đồ án tốt nghiệp này.

Chúng em đặc biệt cảm ơn thầy **Chu Văn Cường** đã dành thời gian chỉ bảo, hỗ trợ và đưa ra những góp ý quý báu trong quá trình thực hiện đề tài " Xây dựng hệ thống nhận diện giọng nói dựa trên TinyML ". Những lời khuyên của thầy đã giúp em có định hướng rõ ràng và khắc phục các khó khăn gặp phải.

Chúng em xin trân trọng cảm ơn tất cả những sự giúp đỡ quý báu đó!

Hà Nội, tháng 4 năm 2025

TÓM TẮT

Hệ thống nhúng nhận diện giọng nói hiện đang được xem là một trong những công nghệ tiên phong nhờ tính ứng dụng đa dạng và tiềm năng phát triển to lớn. Khi được tích hợp trong các thiết bị thông minh, hệ thống này cho phép người dùng tương tác bằng lời nói, từ đó mang lại trải nghiệm thuận tiện, tiết kiệm thời gian, đồng thời mở ra cơ hội cho nhiều giải pháp sáng tạo hơn. Điển hình, trong lĩnh vực nhà thông minh, hệ thống nhận diện giọng nói có thể thay thế hoàn toàn các phương thức điều khiển truyền thống như công tắc hay remote, giúp người dùng bật/tắt đèn, điều chỉnh nhiệt độ điều hòa, hoặc mở đóng rèm cửa bằng mệnh lệnh đơn giản. Điều này đặc biệt hữu ích với những người khuyết tật hoặc người cao tuổi, khi họ thường gặp khó khăn trong việc thao tác trên các thiết bị nhỏ hẹp hoặc phải di chuyển nhiều để tìm công tắc. Việc áp dụng công nghệ nhận diện giọng nói không chỉ nâng cao tính tiện dụng, mà còn góp phần xây dựng môi trường sống hiện đại và an toàn hơn.

Bên cạnh ứng dụng trong nhà thông minh, công nghệ này còn có khả năng mở rộng sang nhiều lĩnh vực khác. Chẳng hạn, trong ngành ô tô, hệ thống nhận diện giọng nói có thể hỗ trợ người lái thực hiện các tác vụ như điều khiển hệ thống giải trí, tra cứu bản đồ, hay thậm chí kiểm tra tình trạng xe – tất cả chỉ bằng câu lệnh. Đối với môi trường công nghiệp, công nghệ nhận diện giọng nói hỗ trợ công nhân hoặc kỹ sư vận hành máy móc, thu thập dữ liệu, giám sát quá trình sản xuất mà không cần phải dừng tay thao tác trên bảng điều khiển. Qua đó, hiệu suất lao động được cải thiện và giảm thiểu các sai sót có thể phát sinh.

Về khía cạnh kỹ thuật, việc nhận diện giọng nói đòi hỏi khả năng xử lý phức tạp, bao gồm thu âm, tiền xử lý tín hiệu, trích xuất đặc trưng, và cuối cùng là nhận dạng hoặc phân loại âm thanh dựa trên mô hình học máy. Công nghệ TinyML ra đời như một giải pháp tối ưu, cho phép các nhà phát triển đưa những mô hình học máy nhỏ gọn lên vi điều khiển hoặc các phần cứng nhúng có tài nguyên hạn chế. Một trong những ưu điểm nổi bật của TinyML là giúp hệ thống có thể thực hiện nhận diện từ khóa (keyword spotting) ngay trên thiết bị, không cần duy trì kết nối internet liên tục. Điều này không chỉ nâng cao tốc độ xử lý, giảm độ trễ truyền tin, mà còn đảm bảo sự riêng tư của người dùng bởi dữ liệu giọng nói không phải gửi lên máy chủ để phân tích.

Hơn nữa, tính linh hoạt của TinyML phù hợp với xu hướng phát triển IoT (Internet of Things), khi mà hàng tỷ thiết bị đang được triển khai trên toàn cầu, đòi hỏi giải pháp điện toán biên (edge computing) hiệu quả và tiết kiệm năng lượng. Việc tích hợp mô hình nhận diện giọng nói lên các thiết bị nhỏ gọn giúp tiết kiệm pin, hạn chế tình trạng quá tải mạng và nâng cao tính độc lập của hệ thống. Từ đó, các giải pháp như loa thông minh, cảm biến môi trường, camera giám sát, cho đến robot dịch vụ đều có thể ứng dụng công nghệ này để nâng cao khả năng nhận diện và phản hồi theo thời gian thực.

Song song với tiềm năng, triển khai hệ thống nhúng nhận diện giọng nói cũng đặt ra một số thách thức. Do tài nguyên hạn chế, các mô hình học máy phải được thiết kế tối ưu về dung lượng bộ nhớ và khả năng tính toán, đòi hỏi sự kết hợp chặt chẽ giữa đội ngũ phát triển phần mềm và thiết kế phần cứng. Ngoài ra, việc xây dựng bộ dữ liệu huấn luyện (dataset) đầy đủ và đa dạng trong nhiều ngữ cảnh, điều kiện môi trường âm thanh khác nhau cũng là yếu tố quan trọng để hệ thống đạt hiệu năng cao.

Tóm lại, hệ thống nhúng nhận diện giọng nói và công nghệ TinyML đang ngày càng khẳng định vai trò quan trọng trong kỷ nguyên số. Với khả năng tương tác tự nhiên, tiết kiệm năng lượng và tiềm năng triển khai rộng rãi, chúng không chỉ cải thiện trải nghiệm người dùng, mà còn góp phần thúc đẩy sự phát triển của hệ sinh thái IoT và mở ra nhiều cơ hội ứng dụng đột phá trong tương lai.

LỜI CAM ĐOAN

Chúng em xin cam đoan nội dung trình bày trong đồ án "Xây dựng hệ thống nhận diện giọng nói dựa trên TinyML" này là quá trình nghiên cứu và tìm hiểu của chúng em dưới sự hướng dẫn của thầy Chu Văn Cường. Những nội dung nghiên cứu và kết quả trong đồ án là hoàn toàn trung thực, có trích dẫn đầy đủ. Nếu như không đúng như đã nêu trên, chúng em xin chịu hoàn toàn trách nhiệm về đề tài của mình.

Hà Nội, tháng 11 năm 2024

Sinh viên

MỤC LỤC

LÝ DO CHỌN ĐỀ TÀI

Hệ thống nhúng nhận diện giọng nói đang trở thành một trong những công nghệ có tiềm năng lớn nhờ tính ứng dụng cao. Hệ thống này có thể được sử dụng trong nhà thông minh, giúp người dùng điều khiển thiết bị điện bằng giọng nói, giảm sự phụ thuộc vào công tắc hay remote truyền thống. Đặc biệt, nó còn hỗ trợ người khuyết tật và người cao tuổi trong việc sử dụng các thiết bị điện một cách thuận tiện hơn.

TinyML là một xu hướng mới trong lĩnh vực AI nhúng, cho phép triển khai các mô hình học máy trên vi điều khiển có tài nguyên hạn chế. Một ưu điểm lớn của công nghệ này là khả năng nhận diện từ khóa mà không cần kết nối internet, giúp tăng tốc độ xử lý và đảm bảo tính bảo mật cho dữ liệu người dùng. Ngoài ra, TinyML đặc biệt phù hợp với các thiết bị IoT có công suất thấp, giúp tiết kiệm năng lượng hiệu quả.

CHƯƠNG 1: TỔNG QUAN ĐỀ TÀI

1.1 . Tổng quan về hệ thống

Hệ thống nhận diện giọng nói là một ứng dụng AI đặc biệt, có khả năng nhận diện và xử lý các lệnh giọng nói của người dùng để điều khiển thiết bị. Đồ án này tập trung vào việc phát triển một hệ thống nhận diện giọng nói đơn giản, có thể nhận diện 4 từ khóa: "bật đèn", "tắt đèn", "mở cửa" và "đóng cửa". Mục đích của hệ thống là giúp người dùng điều khiển các thiết bị như đèn và cửa chỉ bằng giọng nói mà không cần sử dụng các thiết bị điều khiển phức tạp.

Hệ thống sẽ sử dụng các công nghệ AI tiên tiến với mô hình TinyML để triển khai trên các thiết bị nhúng với tài nguyên hạn chế, đặc biệt là vi điều khiển ESP32-S3. Phần cứng ESP32-S3 có khả năng xử lý các tác vụ AI cơ bản mà không cần sự hỗ trợ của các máy chủ ngoài. Điều này không chỉ giúp hệ thống có thể hoạt động độc lập mà còn giảm bớt độ trễ và yêu cầu kết nối mạng, mang lại sự tiện lợi trong các ứng dụng thực tế.

Trong hệ thống này, âm thanh sẽ được thu nhận từ các từ khóa được phát ra bởi người dùng. Sau khi thu âm, các tín hiệu giọng nói sẽ được xử lý và phân tích thông qua các thuật toán đặc trưng để trích xuất các thông tin có thể sử dụng cho phân loại. Mô hình học máy sẽ dựa vào các đặc trưng này để xác định lệnh giọng nói người dùng đã đưa ra. Cuối cùng, hệ thống sẽ thực hiện hành động tương ứng để điều khiển thiết bị (bật/tắt đèn, mở/đóng cửa).

Mô hình hoạt động của hệ thống:

1. Thu thập âm thanh: Hệ thống sẽ sử dụng microphone.
2. Trích xuất đặc trưng âm thanh: Bằng cách sử dụng các thuật toán MFCC (Mel Frequency Cepstral Coefficients), hệ thống sẽ trích xuất các đặc trưng từ tín hiệu âm thanh để làm đầu vào cho mô hình học máy.
3. Huấn luyện mô hình TinyML: Dữ liệu âm thanh đã được xử lý sẽ được dùng để huấn luyện mô hình nhận diện giọng nói. Các mô hình này sẽ được xây dựng với các thuật toán học máy như Mạng nơ-ron đơn giản hoặc các thuật toán phân loại như SVM (Support Vector Machine).
4. Triển khai mô hình: Mô hình huấn luyện được nạp vào phần cứng ESP32-S3, cho phép hệ thống nhận diện các lệnh giọng nói và thực hiện hành động điều khiển thiết bị mà không cần kết nối mạng.

1.2 . Công nghệ sử dụng

Trong đồ án này, các công nghệ tiên tiến nhất hiện nay trong lĩnh vực AI và IoT đã được sử dụng để xây dựng và triển khai hệ thống nhận diện giọng nói. Các công nghệ chủ yếu bao gồm:

- ESP32-S3: ESP32-S3 là vi điều khiển mạnh mẽ với khả năng xử lý tín hiệu và hỗ trợ AI trực tiếp trên phần cứng. Với bộ vi xử lý này, hệ thống có thể thực hiện các tác vụ nhận diện giọng nói ngay trên thiết bị mà không cần kết nối đến máy chủ, giúp tiết kiệm băng thông và giảm độ trễ.
- TinyML: TinyML là công nghệ giúp chạy các mô hình học máy trên các thiết bị nhúng với tài nguyên hạn chế. Điều này rất quan trọng vì các vi điều khiển như ESP32-S3 có bộ nhớ và sức mạnh tính toán hạn chế, nhưng TinyML cho phép các mô hình nhận diện giọng nói có thể được triển khai trực tiếp trên thiết bị mà không cần máy chủ.
- MFCC (Mel Frequency Cepstral Coefficients): MFCC là phương pháp phổ biến trong xử lý âm thanh, đặc biệt là trong nhận diện giọng nói. MFCC giúp chuyển đổi tín hiệu âm thanh từ dạng sóng (waveform) sang dạng đặc trưng có thể sử dụng cho học máy. Các đặc trưng này sẽ được dùng làm đầu vào cho các mô hình học máy để nhận diện lệnh giọng nói.
- Edge Impulse: Edge Impulse là một nền tảng cho phép phát triển và huấn luyện các mô hình TinyML. Nền tảng này hỗ trợ thu thập dữ liệu, xử lý tín hiệu, huấn luyện mô hình và triển khai mô hình lên các thiết bị nhúng một cách dễ dàng và hiệu quả.
- Arduino trên VS Code với PlatformIO: Môi trường phát triển này cung cấp một nền tảng lập trình quen thuộc và mạnh mẽ cho việc phát triển ứng dụng trên ESP32-S3. Nó hỗ trợ nhiều thư viện và công cụ cần thiết để lập trình và tương tác với phần cứng một cách dễ dàng.

1.3 . Mục tiêu của nhóm

Mục tiêu của nhóm là xây dựng một hệ thống nhận diện giọng nói đơn giản nhưng hiệu quả, có thể nhận diện các lệnh giọng nói như "bật đèn", "tắt đèn", "mở cửa" và "đóng cửa" để điều khiển thiết bị. Mục tiêu cụ thể của nhóm bao gồm:

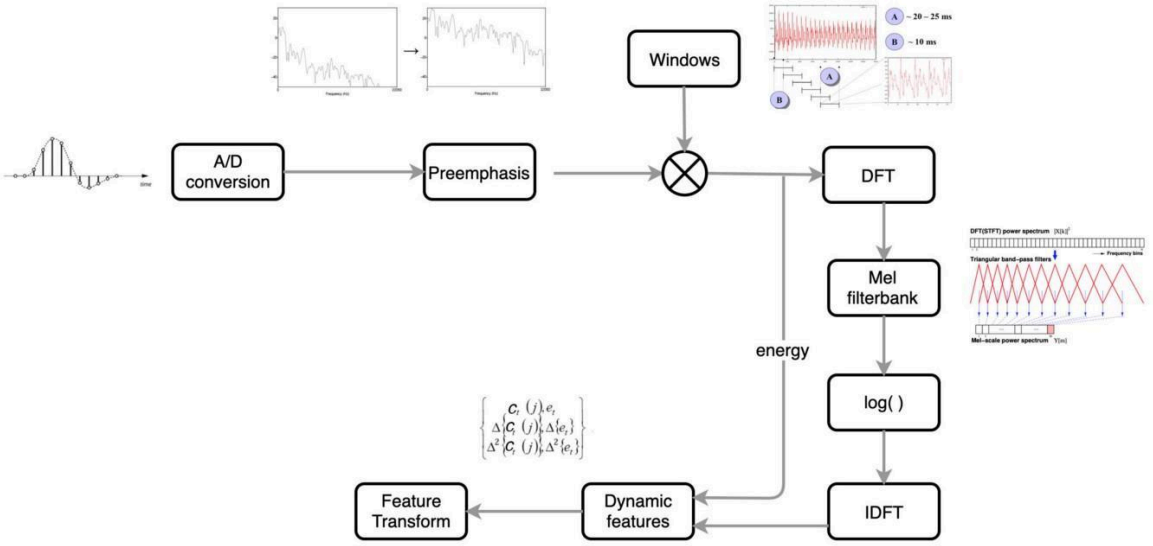
1. Nhận diện chính xác 4 từ khóa: Hệ thống phải có khả năng nhận diện chính xác và nhanh chóng 4 từ khóa đã định nghĩa: "bật đèn", "tắt đèn", "mở cửa", "đóng cửa".
2. Xử lý âm thanh và trích xuất đặc trưng: Xây dựng hệ thống xử lý tín hiệu âm thanh để loại bỏ nhiễu và chuẩn hóa dữ liệu âm thanh trước khi trích xuất đặc trưng với MFCC.
3. Huấn luyện mô hình học máy: Sử dụng các thuật toán học máy để huấn luyện mô hình nhận diện giọng nói. Mô hình này sẽ có thể chạy trực tiếp trên ESP32-S3 mà không cần sự hỗ trợ của các máy chủ bên ngoài.
4. Triển khai hệ thống trên phần cứng: Triển khai mô hình học máy đã huấn luyện lên phần cứng ESP32-S3 và đảm bảo hệ thống hoạt động ổn định trong các tình huống thực tế.
5. Tích hợp điều khiển thiết bị: Sau khi nhận diện được lệnh giọng nói, hệ thống sẽ thực hiện các hành động như bật đèn, tắt đèn, mở cửa, đóng cửa thông qua giao tiếp I/O của ESP32-S3.

Với mục tiêu này, nhóm sẽ phát triển hệ thống nhận diện giọng nói đơn giản nhưng mạnh mẽ, có thể ứng dụng trong nhiều tình huống thực tế, giúp người dùng điều khiển thiết bị dễ dàng và tiện lợi chỉ với giọng nói.

CHƯƠNG 2: XÂY DỰNG HỆ THỐNG

2.1. Kiến trúc model

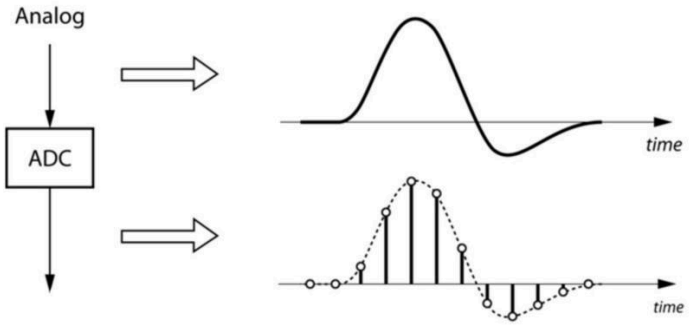
Hình dưới đây mô tả luồng xử lý từ âm thanh đầu vào ⟶ MFCC



2.1.1. A/D Conversion and Pre-emphasis

2.1.1.1. A/D Conversion

Âm thanh là dạng tín hiệu liên tục, trong khi đó máy tính làm việc với các con số rời rạc. Ta cần lấy mẫu tại các khoảng thời gian cách đều nhau với 1 tần số lấy mẫu xác định (sample rate) để chuyển từ dạng tín hiệu liên tục về dạng rời rạc. VD sample_rate = 8000 ⟶⟶ trong 1s lấy 8000 giá trị.



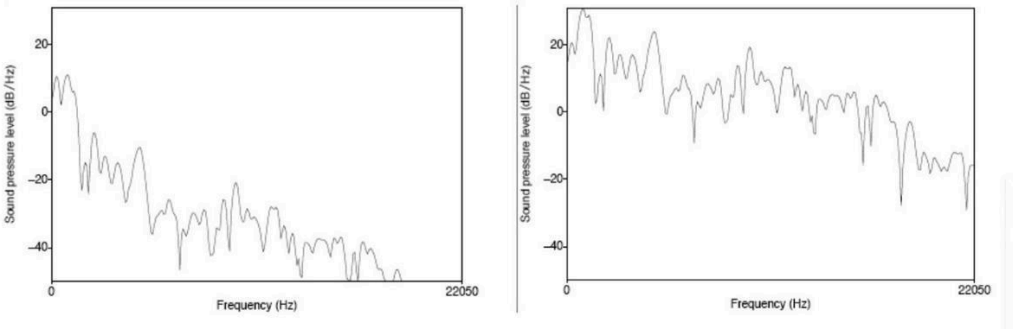
Tai người nghe được âm thanh trong khoảng 20Hz ⟶⟶ 20.000 Hz. Theo định lý lấy mẫu Nyquist–Shannon: với 1 tín hiệu có các tần số thành phần ≤fm≤*fm*, để đảm bảo việc lấy mẫu không làm mất mát thông tin (aliasing), tần số lấy mẫu fs*fs* phải đảm bảo fs≥2fm*fs*≥2*fm*.

Vậy để đảm bảo việc lấy mẫu không làm mất mát thông tin, tần số lấy mẫu fs=44100Hz*fs*=44100*Hz*. Tuy nhiên trong nhiều trường hợp, người ta chỉ cần lấy fs=8000Hz*fs*=8000*Hz* hoặc fs=16000Hz*fs*=16000*Hz*.

2.1.1.2. Pre-emphasis

Do đặc điểm cấu tạo thanh quản và các bộ phận phát âm nên tiếng nói của chúng ta có đặc điểm: các âm ở tần số thấp có mức năng lượng cao, các âm ở tần số cao lại có mức năng lượng khá thấp. Trong khi đó, các tần số cao này vẫn chứa nhiều thông tin về âm vị. Vì vậy chúng ta cần 1 bước pre-emphasis để kích các tín hiệu ở tần số cao này lên.

$$x'[t_d] = x[t_d] - \alpha x[t_d - 1] \qquad 0.95 < \alpha < 0.99$$



2.1.2. Spectrogram

2.1.2.1. Windowing

Thay vì biến đổi Fourier trên cả đoạn âm thanh dài, ta trượt 1 cửa sổ dọc theo tín hiệu để lấy ra các frame rồi mới áp dụng DFT trên từng frame này (DFT - Discrete Fourier Transform). Tốc độ nói của con người trung bình khoảng 3, 4 từ mỗi giây, mỗi từ khoảng 3-4 âm, mỗi âm chia thành 3-4 phần, như vậy 1 giây âm thanh được chia thành 36 - 40 phần, ta chọn độ rộng mỗi frame khoảng 20 - 25ms là vừa đủ rộng để bao 1 phần âm thanh. Các frame được overlap lên nhau khoảng 10ms để có thể *capture* lại sự thay đổi context.

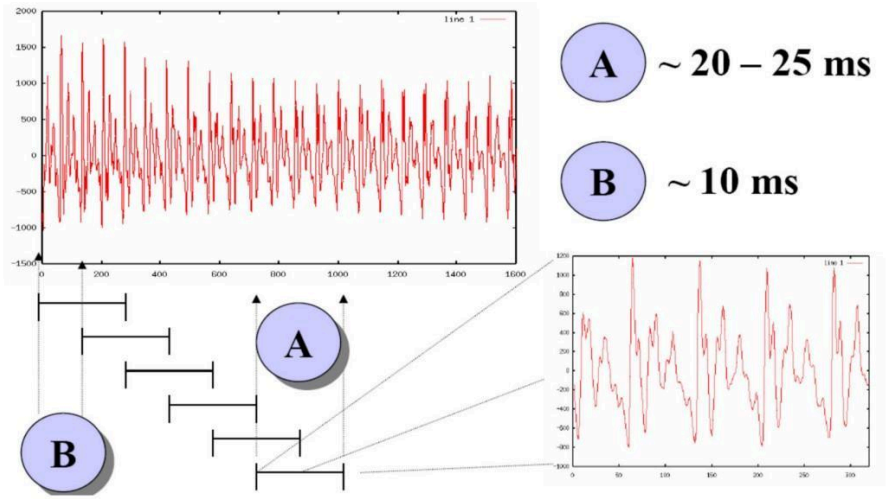
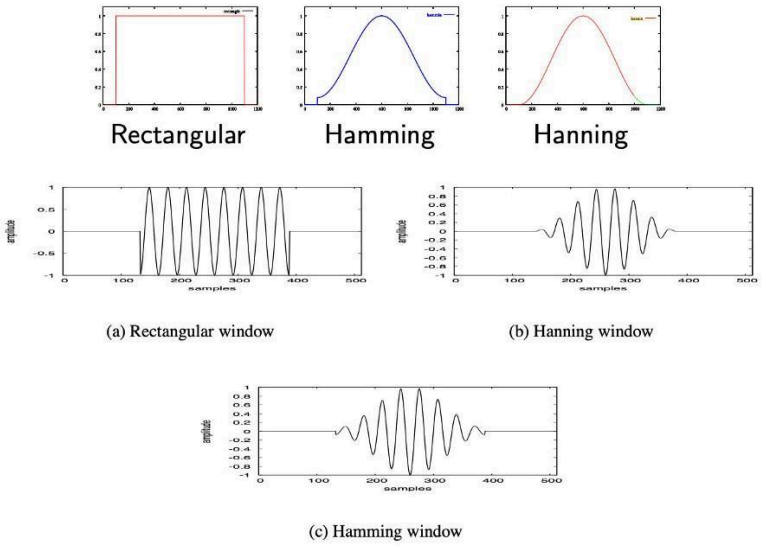


Image from Bryan Pellom

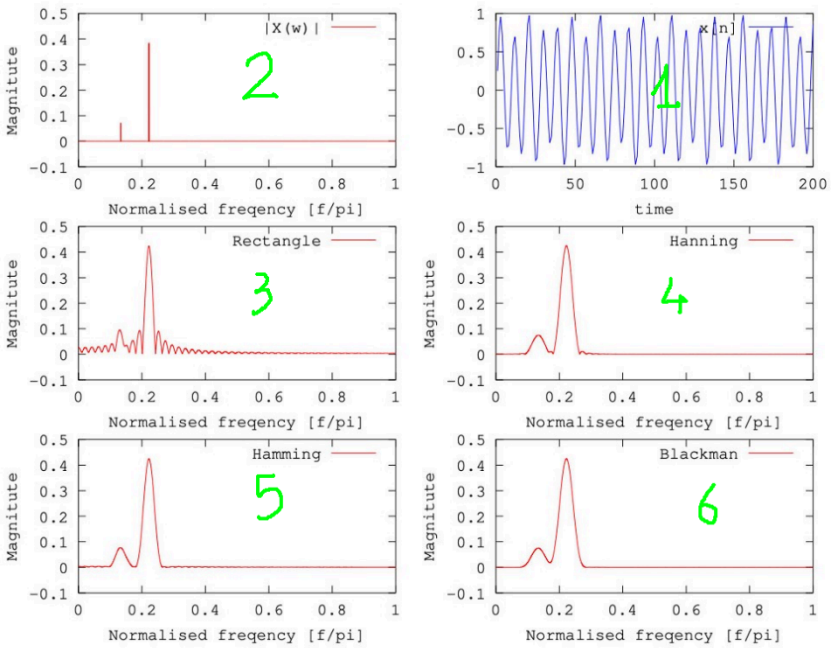
Tuy nhiên, việc cắt frame sẽ làm các giá trị ở 2 biên của frame bị giảm đột ngột (về giá trị 0), sẽ dẫn tới hiện tượng: khi DFT sang miền tần số sẽ có rất nhiều nhiễu ở tần số cao. Để khắc phục điều này, ta cần làm mượt bằng cách nhân chập frame với 1

vài loại window. Có 1 vài loại window phổ biến là Hamming window, Hanning window ... có tác dụng làm giá trị biên frame giảm xuống từ từ từ.



(Taylor, fig 12.1)

Hình dưới đây sẽ cho ta thấy rõ được tác dụng của các window này. Trong các hình nhỏ, hình 1 là frame được cắt ra từ âm thanh gốc, âm thanh gốc là sự kết hợp của 2 sóng hình 2. Nếu áp dụng rectangle window (tức là cắt trực tiếp), tín hiệu miền tần số tương ứng là hình 3, ta có thể thấy tín hiệu này chứa rất nhiều nhiễu. Nếu áp dụng các window như Hanning, Hamming, Blackman, tín hiệu miền tần số thu được khá mượt và sóng gốc ở hình2.

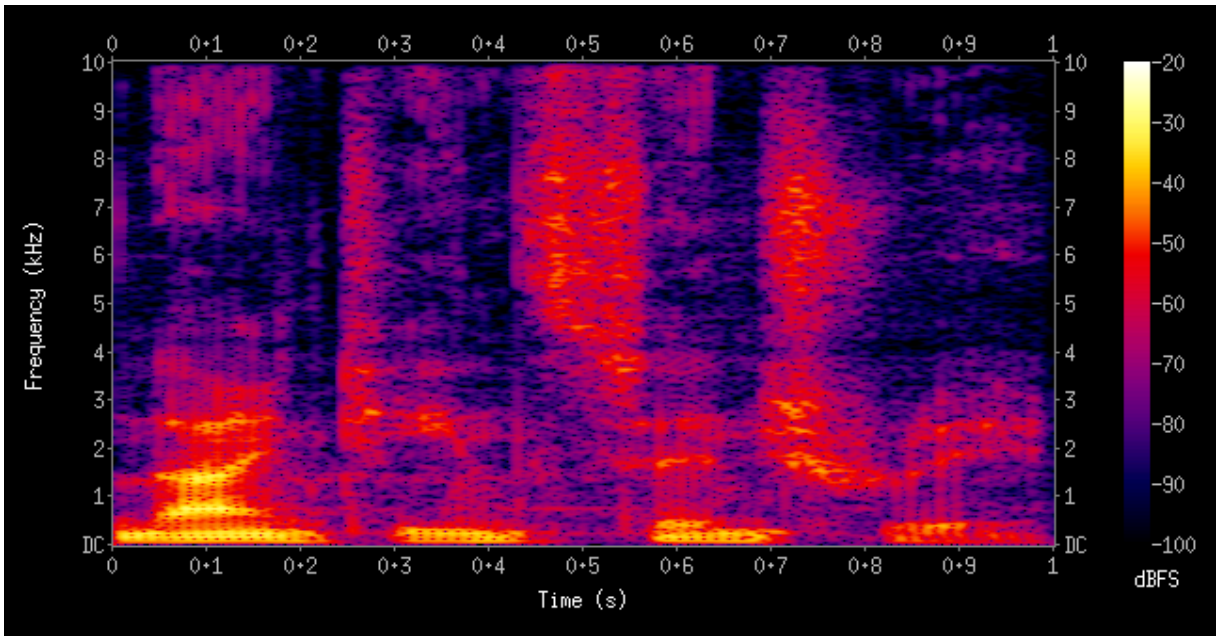


2.1.2.2. DFT

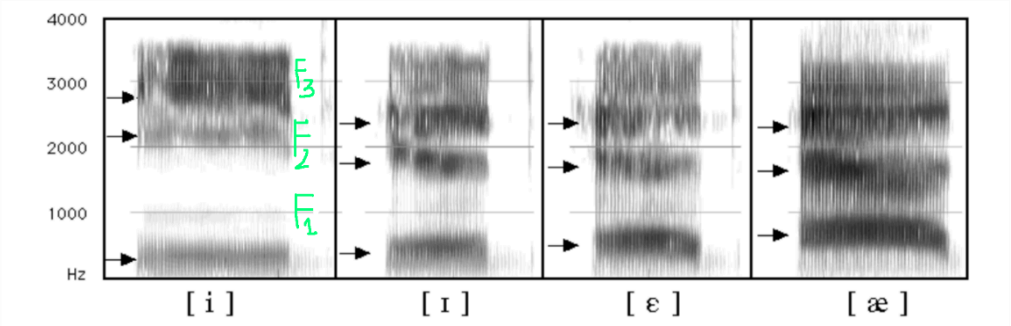
Trên từng frame, ta áp dụng DFT - Discrete Fourier Transform theo công thức:

$$X[k] = \sum_{n=0}^{N-1} x[n] \exp\left(-j\frac{2\pi}{N}kn\right)$$

Mỗi frame ta thu được 1 list các giá trị độ lớn (magnitude) tương ứng với từng tần số từ 0→N0→*N*. Áp dụng trên tất cả các frame, ta đã thu được 1 Spectrogram như hình dưới đây. Trục x*x* là trục thời gian (tương ứng với thứ tự các frame), trục y*y* thể hiện dải tần số từ 0→100000→10000 Hz, giá trị magnitude tại từng tần số được thể hiện bằng màu sắc. Qua quan sát spectrogram này, ta nhận thấy các tại các tần số thấp thường có magnitude cao, tần số cao thường có magnitude thấp.



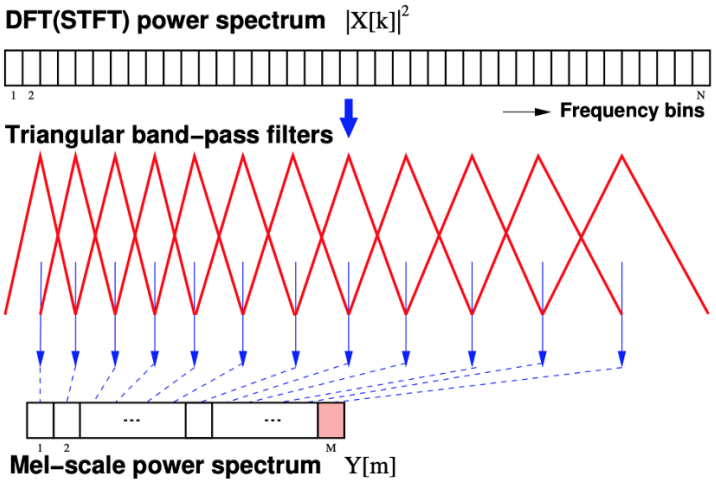
Hình dưới là các spectrogram của 4 nguyên âm. Quan sát spectrogram lần lượt từ dưới lên, người ta nhận thấy có 1 vài tần số đặc trưng gọi là các formant, gọi là các tần số F1, F2, F3 ... Các chuyên gia về ngữ âm học có thể dựa vào vị trí, thời gian, sự thay đổi các formant trên spectrogram để xác định đoạn âm thanh đó là của âm vị nào.



Như vậy ta đã biết cách tạo ra spectrogram. Tuy nhiên trong nhiều bài toán (đặc biệt là speech recognition), spectrogram không phải là sự lựa chọn hoàn hảo. Vì vậy ta cần thêm vài bước tính nữa để thu được dạng MFCC, tốt hơn, phổ biến hơn, hiệu quả hơn spectrogram.

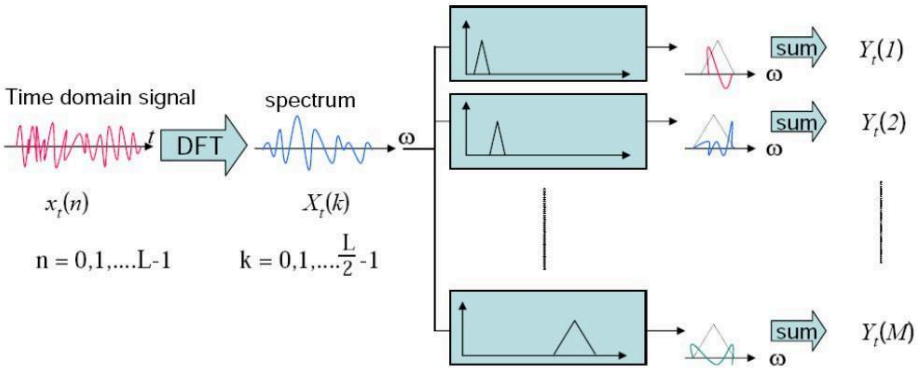
2.1.3. Mel filterbank

Như mình đã mô tả ở phần trước, cách cảm nhận của tai người là phi tuyến tính, không giống các thiết bị đo. Tai người cảm nhận tốt ở các tần số thấp, kém nhạy cảm với các tần số cao. Ta cần 1 cơ chế mapping tương tự như vậy.



Trước hết, ta bình phương các giá trị trong spectrogram thu được DFT power spectrum (phổ công suất). Sau đó, ta áp dụng 1 tập các bộ lọc thông dải Mel-scale filter trên từng khoảng tần số (mỗi filter áp dụng trên 1 dải tần xác định). Giá trị output của từng filter là năng lượng dải tần số mà filter đó cover (bao phủ) được. Ta thu được Mel-scale power spectrum. Ngoài ra, các filter dùng cho dải tần thấp thường hẹp hơn các filter dùng cho dải tần cao.

Quá trình này còn có thể mô tả bằng hình minh hoạ dưới đây:



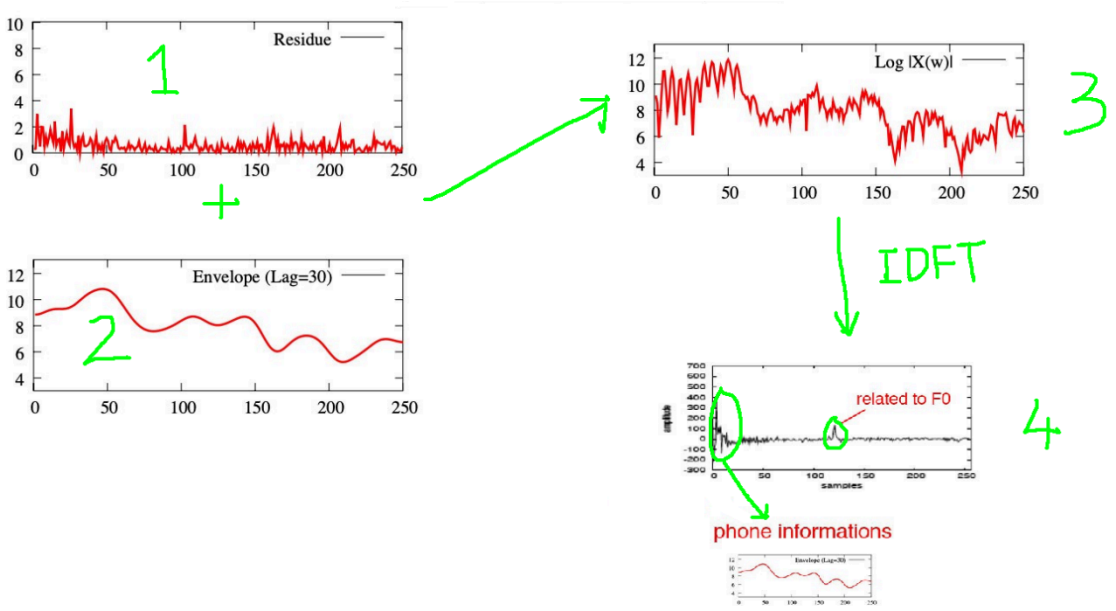
2.1.4. Cepstrum

2.1.4.1. Log

Mel filterbank trả về phổ công suất của âm thanh, hay còn gọi là phổ năng lượng. Thực tế rằng con người kém nhạy cảm trong sự thay đổi năng lượng ở các tần số cao, nhạy cảm hơn ở tần số thấp. Vì vậy ta sẽ tính log trên Mel-scale power spectrum. Điều này còn giúp giảm các biến thể âm thanh không đáng kể để nhận dạng giọng nói.

2.1.4.2. IDFT - Inverse DFT

Như đã mô tả ở phần trước, giọng nói của chúng ta có tần số F0 - tần số cơ bản và các formant F1, F2, F3 ... Tần số F0 ở nam giới khoảng 125 Hz, ở nữ là 210 Hz, đặc trưng cho cao độ giọng nói ở từng người. Thông tin về cao độ này không giúp ích trong nhận dạng giọng nói, nên ta cần tìm cách để loại thông tin về F0 đi, giúp các mô hình nhận dạng không bị phụ thuộc vào cao độ giọng từng người.



Trong hình này, tín hiệu chúng ta thu được là đồ thị 3, nhưng thông tin quan trọng chúng ta cần là phần 2, thông tin cần loại bỏ là phần 1. Để loại bỏ đi thông tin về F0, ta làm 1 bước biến đổi Fourier ngược (IDFT) về miền thời gian, ta thu được Cepstrum. Nếu để ý kỹ, ta sẽ nhận ra rằng tên gọi "cepstrum" thực ra là đảo ngược 4 chữ cái đầu của "spetrum".

Khi đó, với Cepstrum thu được, phần thông tin liên quan tới F0 và phần thông tin liên quan tới F1, F2, F3 ... nằm tách biệt nhau như 2 phần khoanh tròn trong hình 4. Ta chỉ đơn giản lấy thông tin trong đoạn đầu của cepstrum (phần được khoanh tròn to trong hình 4). Để tính MFCC, ta chỉ cần lấy 12 giá trị đầu tiên.

Phép biến đổi IDFT cũng tương đương với 1 phép biến đổi DCT discrete cosine transformation. DCT là 1 phép biến đổi trực giao. Về mặt toán học, phép biến đổi này tạo ra các uncorrelated features, có thể hiểu là các feature độc lập hoặc có độ tương quan kém với nhau. Trong các thuật toán Machine learning, uncorrelated features thường cho hiểu quả tốt hơn. Như vậy sau bước này, ta thu được 12 Cepstral features.

2.1.5. MFCC

Như vậy, mỗi frame ta đã extract ra được 12 Cepstral features làm 12 feature đầu tiên của MFCC. feature thứ 13 là năng lượng của frame đó, tính theo công thức:

$$Energy = \sum_{t=t_1}^{t_2} x^2[t]$$

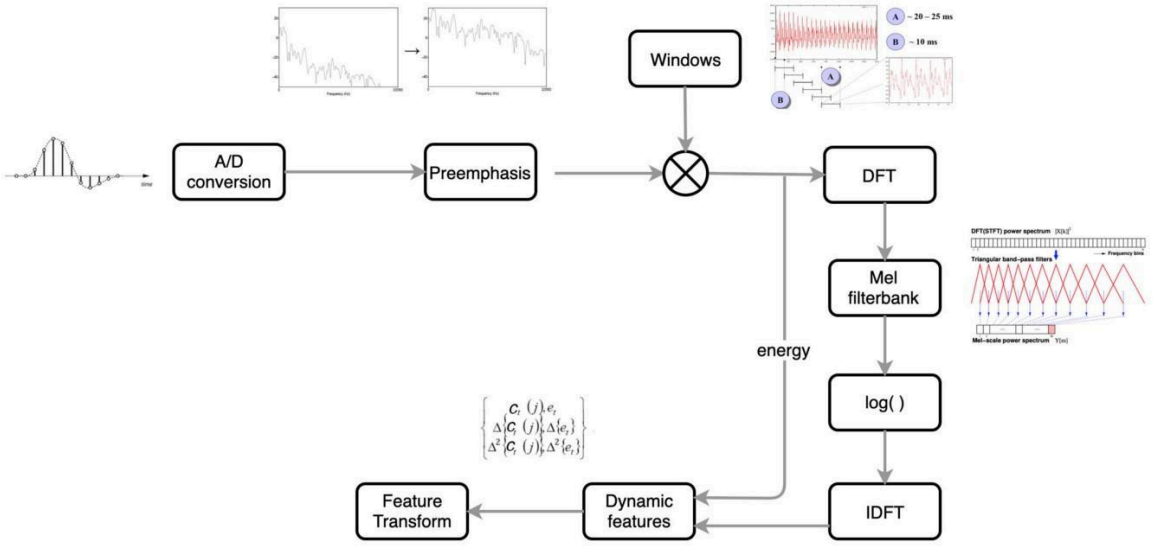
Trong nhận dạng tiếng nói, thông tin về bối cảnh và sự thay đổi rất quan trọng. VD tại những điểm mở đầu hoặc kết thúc ở nhiều phụ âm, sự thay đổi này rất rõ rệt, có thể nhận dạng các âm vị dựa vào sự thay đổi này. 13 hệ số tiếp theo chính là đạo hàm bậc 1 (theo thời gian) của 13 feature đầu tiên. Nó chứa thông tin về sự thay đổi từ frame thứ tt đến frame t+1t+1. Công thức:

$$d(t)=[c(t+1)-c(t-1)]/2d(t)$$

Tương tự như vậy, 13 giá trị cuối của MFCC là sự thay đổi d(t)$d(t)$ theo thời gian - đạo hàm của d(t)$d(t)$, đồng thời là đạo hàm bậc 2 của c(t)$c(t)$. Công thức:

$$b(t)=d(t+1)-d(t-1)/2b(t)$$

Vậy, từ 12 cepstral feature và power feature thứ 13, ta đạo hàm 2 lần và thu được 39 feature. Đây chính là MFCC feature. Cùng nhìn lại toàn bộ quá trình để tạo ra MFCC:



2.2. Thu thập và xử lý dữ liệu âm thanh

Dữ liệu âm thanh trong đồ án này được thu thập qua micro máy tính, sử dụng nền tảng Edge Impulse để ghi lại các lệnh giọng nói. Edge Impulse là một nền tảng phát triển AI và TinyML cho các thiết bị nhúng, giúp thu thập, xử lý và huấn luyện mô hình học máy một cách hiệu quả ngay trên thiết bị đầu cuối.

Các lệnh giọng nói cần thu thập bao gồm "bật đèn", "tắt đèn", "mở cửa", và "đóng cửa", và mỗi lệnh sẽ được thu âm trong các điều kiện môi trường khác nhau, đảm bảo rằng mô hình học được cách nhận diện giọng nói trong nhiều tình huống khác nhau. Các mẫu giọng nói này được thu âm với micro máy tính chất lượng cao, được

tích hợp vào Edge Impulse, cho phép quá trình thu thập và lưu trữ dữ liệu diễn ra dễ dàng và trực tiếp.

Mỗi lệnh giọng nói sẽ được ghi nhận với các tham số cố định như:

- Tốc độ nói: Người dùng được yêu cầu nói các lệnh một cách rõ ràng và với tốc độ bình thường để mô hình học được cách nhận diện giọng nói tự nhiên.
- Thời gian thu âm: Mỗi lệnh giọng nói sẽ được ghi âm trong vòng 1 giây, giúp dữ liệu thu được có độ dài phù hợp để trích xuất các đặc trưng âm thanh.
- Sự đa dạng về giọng nói: Để đảm bảo mô hình học được các biến thể của lệnh giọng nói, dữ liệu được thu thập từ nhiều người với các giọng nói và tông giọng khác nhau.

Edge Impulse cho phép thu thập dữ liệu âm thanh trực tiếp qua các thiết bị cảm biến, đồng thời hỗ trợ việc tạo bộ dữ liệu và gắn nhãn cho các mẫu giọng nói một cách dễ dàng. Các bước này đảm bảo rằng hệ thống có thể học từ dữ liệu thực tế và không bị giới hạn bởi điều kiện môi trường hay tông giọng của người nói.

Khi dữ liệu âm thanh được thu thập, bước tiếp theo là tiền xử lý dữ liệu để chuẩn hóa và làm sạch tín hiệu âm thanh. Trong trường hợp này, quá trình tiền xử lý đã được thực hiện thông qua thư viện MFCC (Mel Frequency Cepstral Coefficients). Các bước tiền xử lý này bao gồm:

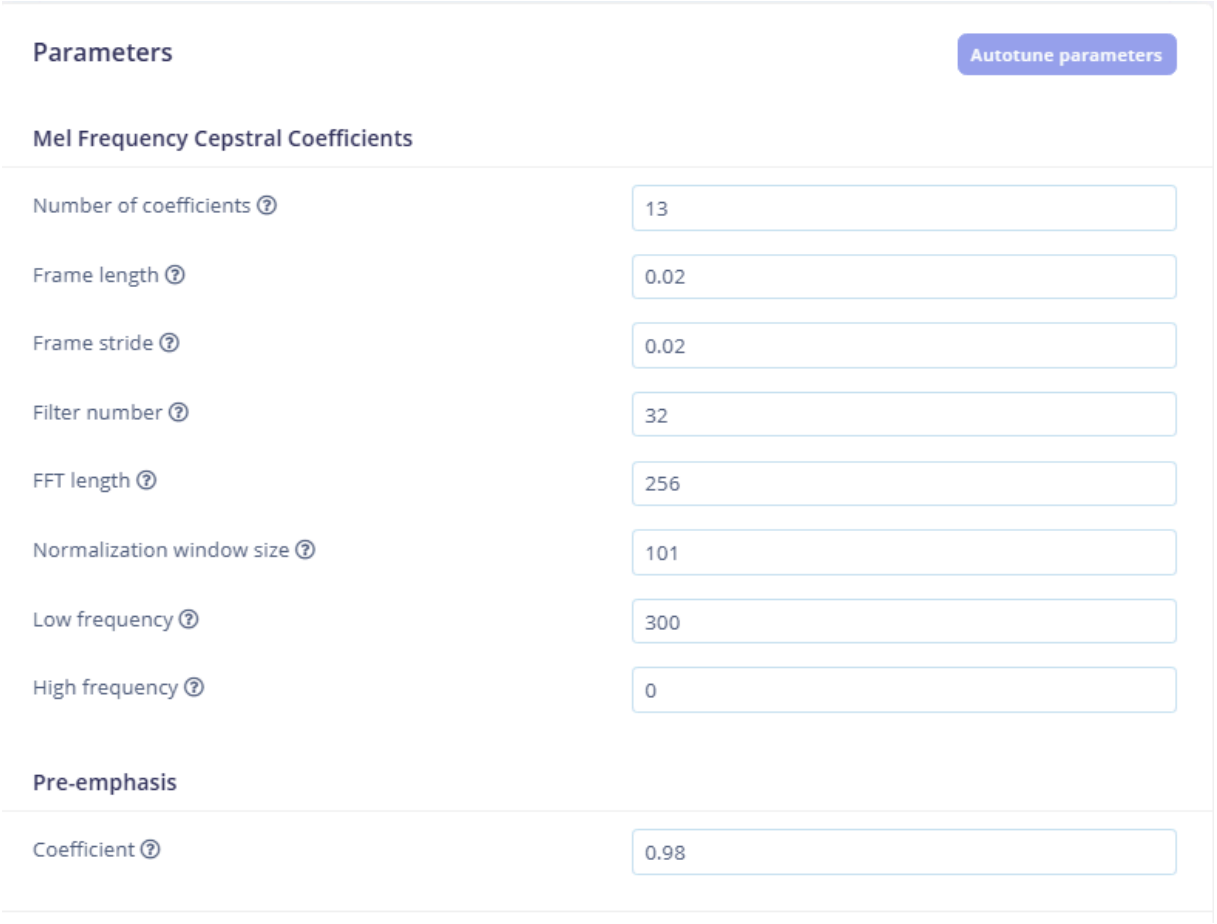
- Lọc nhiễu: Các tạp âm và nhiễu nền trong tín hiệu âm thanh sẽ được loại bỏ. Việc này đảm bảo rằng các lệnh giọng nói được nhận diện một cách chính xác, không bị ảnh hưởng bởi các âm thanh không mong muốn xung quanh.
- Cắt tín hiệu thành các khung nhỏ: Để mô hình có thể xử lý tín hiệu âm thanh dễ dàng hơn, dữ liệu âm thanh sẽ được chia thành các khung (frames) nhỏ, mỗi khung có độ dài khoảng 20ms. Các khung này sẽ giúp mô hình phân tích âm thanh theo từng phần nhỏ, từ đó học được các đặc trưng âm thanh quan trọng.
- Chuẩn hóa tín hiệu âm thanh: Tín hiệu âm thanh sẽ được chuẩn hóa để đảm bảo rằng mức độ âm lượng và tốc độ nói không ảnh hưởng đến quá trình trích xuất đặc trưng.
- Trích xuất đặc trưng MFCC: Bằng cách sử dụng thư viện MFCC, dữ liệu âm thanh sẽ được chuyển đổi thành các đặc trưng có thể sử dụng trong mô hình học máy. Các hệ số MFCC là đại diện cho các đặc trưng tần số của tín hiệu âm thanh, giúp mô hình nhận diện được các yếu tố cơ bản của âm thanh như âm sắc và nhịp điệu.

Edge Impulse đóng vai trò quan trọng trong việc xử lý và trích xuất đặc trưng MFCC từ dữ liệu thu thập được. Các đặc trưng này sẽ được sử dụng làm đầu vào cho mô hình học máy. Sau khi thu thập dữ liệu âm thanh, Edge Impulse hỗ trợ việc đánh

giá chất lượng của bộ dữ liệu, xác nhận độ chính xác của các nhãn gắn cho mỗi mẫu âm thanh, và tối ưu hóa quá trình tiền xử lý trước khi đưa dữ liệu vào huấn luyện.

2.3. Thiết kế và huấn luyện mô hình TinyML

Kiến trúc mô hình nhận diện giọng nói trong đồ án này được xây dựng trên nền tảng TinyML, với mục tiêu nhận diện 4 từ khóa: "bật đèn", "tắt đèn", "mở cửa" và "đóng cửa". Mô hình sử dụng các đặc trưng âm thanh đã được trích xuất bằng MFCC, giúp hệ thống nhận diện và phân loại giọng nói một cách chính xác.



Dưới đây là các thông số quan trọng được cấu hình khi trích xuất đặc trưng MFCC:

- Số lượng hệ số (Number of coefficients): 13
- Độ dài khung (Frame length): 0.02 giây
- Độ trượt khung (Frame stride): 0.02 giây
- Số lượng bộ lọc (Filter number): 32
- Chiều dài FFT (FFT length): 256
- Kích thước cửa sổ chuẩn hóa (Normalization window size): 101
- Tần số thấp (Low frequency): 300 Hz

- Tần số cao (High frequency): 0 Hz (chỉ sử dụng tần số thấp và trung bình)

Giải thích về các thông số:

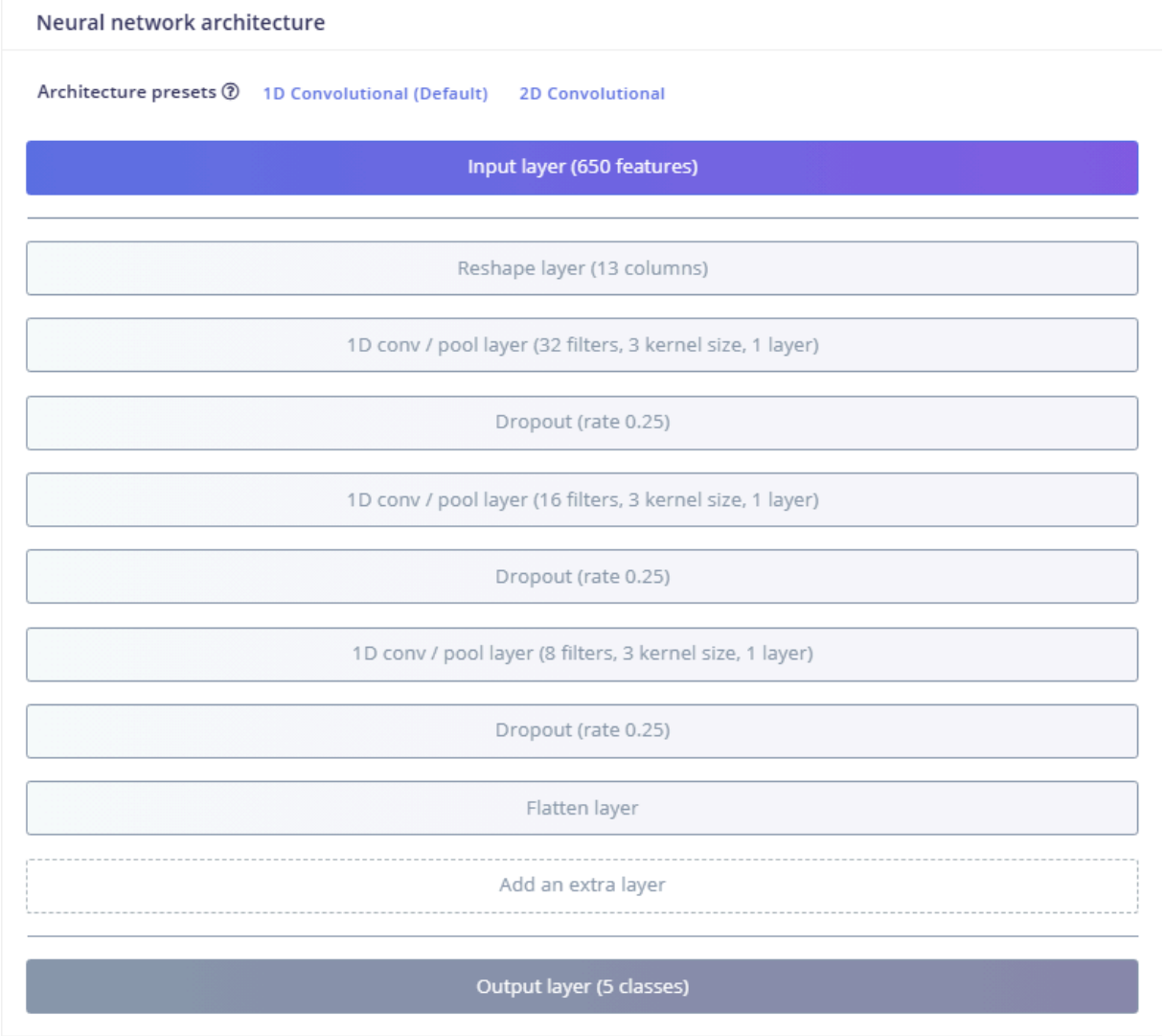
- Số lượng hệ số (Number of coefficients): Số lượng hệ số MFCC được sử dụng để mô tả đặc trưng của âm thanh. 13 hệ số MFCC là một giá trị phổ biến và hiệu quả trong nhận diện giọng nói.
- Độ dài khung và độ trượt khung: Các giá trị này xác định cách thức tín hiệu âm thanh được chia thành các đoạn ngắn. Mỗi khung có độ dài 20ms (0.02 giây) và độ trượt giữa các khung cũng là 0.02 giây, giúp mô hình phân tích các đặc trưng theo thời gian.
- Số lượng bộ lọc và chiều dài FFT: Các bộ lọc được sử dụng để phân tích tần số của tín hiệu âm thanh, trong khi chiều dài FFT giúp xác định độ phân giải của phổ tần số.
- Tần số thấp và cao: Các giá trị này chỉ định giới hạn tần số cho tín hiệu âm thanh, giúp tập trung vào các phần tần số có liên quan đến giọng nói.

Cấu trúc của mô hình học máy:

1. Lớp đầu vào (Input layer):

Mô hình sẽ nhận 650 đặc trưng từ MFCC làm đầu vào. Các đặc trưng này được tổ chức thành các cột (columns) với số lượng là 13.

2. Các lớp mạng (Network layers):



Lớp Reshape: Được sử dụng để chuyển đổi các đặc trưng từ dạng vector thành dạng phù hợp cho các lớp tiếp theo.

Lớp Conv / Pooling (Convolutional and Pooling layers):

- Lớp đầu tiên là một lớp Conv1D với 32 bộ lọc, mỗi bộ lọc có kích thước 3x1. Lớp này giúp mô hình học các đặc trưng tần số từ dữ liệu âm thanh.
- Các lớp tiếp theo có các bộ lọc giảm dần: 16 bộ lọc và 8 bộ lọc, giúp mô hình nhận diện các đặc trưng âm thanh phức tạp hơn.

Lớp Dropout: Mỗi lớp Conv đều có một lớp Dropout với tỷ lệ 0.25 để giảm overfitting trong quá trình huấn luyện.

Lớp Flatten: Sau khi áp dụng các lớp Conv và Pooling, đầu ra sẽ được "làm phẳng" để đưa vào lớp output.

3. Lớp đầu ra (Output layer):

Lớp đầu ra có 5 lớp (để phân loại 5 loại lệnh giọng nói, bao gồm 4 từ khóa và 1 lớp cho các âm thanh không nhận diện được).

CHƯƠNG 3: TRIỂN KHAI HỆ THỐNG

3.1. Tạo bộ xử lý logic điều khiển giọng nói

Bộ xử lý logic điều khiển giọng nói trong hệ thống này là một thành phần rất quan trọng, chịu trách nhiệm xử lý dữ liệu âm thanh thu được từ micro, phân loại tín hiệu âm thanh và điều khiển các thiết bị ngoại vi (như đèn hoặc cửa) dựa trên kết quả phân loại từ mô hình học máy.

Quá trình xử lý logic điều khiển giọng nói:

1. Thu thập và xử lý âm thanh:

Dữ liệu âm thanh được thu thập từ micro máy tính qua nền tảng Edge Impulse. Trong code, phần này được thực hiện bởi các hàm như microphone_inference_start và microphone_inference_record.

2. Phân loại giọng nói:

Sau khi thu thập và xử lý tín hiệu âm thanh thành các đặc trưng MFCC, mô hình Edge Impulse sẽ thực hiện phân loại giọng nói. Phần quan trọng nhất trong việc phân loại là sử dụng các hàm như run_classifier_continuous và microphone_audio_signal_get_data.

3. Điều khiển thiết bị:

Sau khi mô hình nhận diện được lệnh giọng nói (ví dụ: "bật đèn", "tắt đèn", "mở cửa", "đóng cửa"), hệ thống sẽ điều khiển các thiết bị thông qua các chân GPIO. Các hàm như xEventGroupSetBits, xEventGroupClearBits và digitalWrite sẽ giúp điều khiển các thiết bị này.

Giải thích về một số hàm quan trọng:

- xEventGroupSetBits và xEventGroupClearBits:

  Đây là các hàm của FreeRTOS để quản lý sự kiện trong hệ thống đa nhiệm. Các hàm này được sử dụng để thiết lập hoặc xóa các bit trong Event Group, từ đó điều khiển trạng thái của các thiết bị (như đèn hoặc cửa).

  Ví dụ, khi nhận diện được lệnh "bật đèn", xEventGroupSetBits(ledEventGroup, LED_ON_BIT); sẽ kích hoạt bit LED_ON_BIT, báo hiệu cho hệ thống rằng đèn cần được bật.

- digitalWrite:

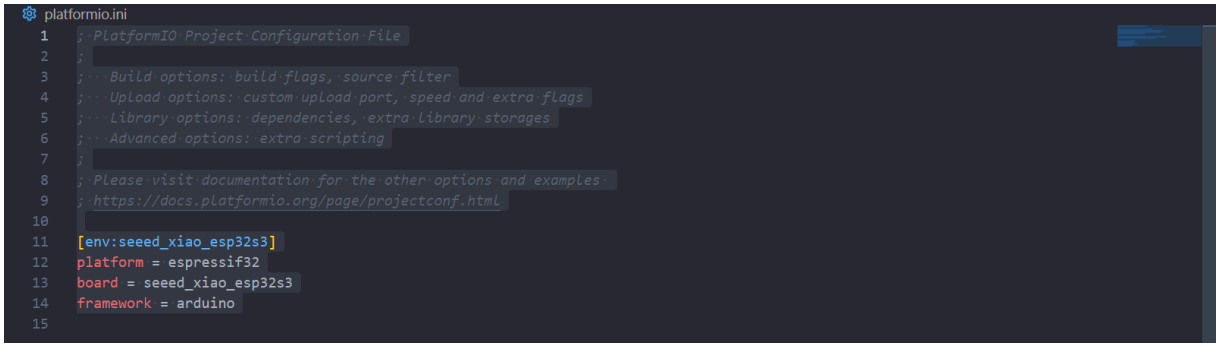
  Hàm này dùng để điều khiển trạng thái của các chân GPIO. Chẳng hạn, khi digitalWrite(led_on, HIGH); được gọi, nó sẽ bật đèn (kết nối với chân GPIO led_on).

3.2. Thiết lập và nạp model vào phần cứng

Sau khi mô hình đã được huấn luyện trên nền tảng Edge Impulse, bước tiếp theo là nạp mô hình vào vi điều khiển ESP32-S3 để mô hình có thể chạy trực tiếp trên phần cứng. Các mô hình này được nạp dưới dạng TensorFlow Lite (TFLite) để sử dụng tài nguyên hệ thống hiệu quả hơn.

Quá trình thiết lập và nạp mô hình:

1. **Cấu hình môi trường và phần cứng:**



Trong tệp platformio.ini, môi trường phát triển cho ESP32-S3 được cấu hình. Cụ thể, bạn sử dụng framework Arduino và platform espressif32 để hỗ trợ việc biên dịch mã nguồn và nạp mô hình vào phần cứng.

Đoạn mã trong main.cpp khởi tạo thư viện Edge Impulse SDK để sử dụng mô hình huấn luyện sẵn. Điều này giúp dễ dàng nạp mô hình phân loại giọng nói vào bộ nhớ của ESP32-S3.

2. **Nạp mô hình phân loại giọng nói:**

run_classifier_init: Hàm này giúp khởi tạo môi trường phân loại giọng nói, chuẩn bị bộ nhớ và tài nguyên để mô hình có thể chạy trên ESP32-S3.

Mô hình phân loại giọng nói đã huấn luyện được sử dụng trong hàm run_classifier_continuous. Hàm này cho phép mô hình phân tích tín hiệu âm thanh liên tục và đưa ra kết quả phân loại.

Giải thích về một số hàm quan trọng:

- run_classifier_init:

Hàm này được gọi trong quá trình thiết lập để khởi tạo mọi thứ cần thiết cho mô hình phân loại. Nó chuẩn bị bộ nhớ và các đối tượng cần thiết để mô hình có thể chạy mượt mà trên ESP32-S3.

- run_classifier_continuous:

Hàm này chạy mô hình phân loại giọng nói trong một vòng lặp liên tục. Mỗi khi có tín hiệu âm thanh mới được thu thập từ micro, hàm này sẽ xử lý tín hiệu và phân loại nó thành một trong các lệnh giọng nói đã được huấn luyện.

3.3. Tích hợp để hệ thống có thể điều khiển thiết bị bằng giọng nói

Một khi mô hình phân loại giọng nói đã được nạp vào phần cứng, hệ thống cần được tích hợp với các thiết bị ngoại vi như **đèn**, **cửa**, và **bất kỳ thiết bị nào khác** cần được điều khiển thông qua giọng nói.

Quá trình tích hợp điều khiển thiết bị:

1. **Điều khiển đèn và cửa thông qua GPIO**:

   Các thiết bị ngoại vi được điều khiển thông qua các chân GPIO trên ESP32-S3. Ví dụ, khi mô hình nhận diện được lệnh "bật đèn", hệ thống sẽ kích hoạt LED_ON_BIT và bật đèn thông qua việc điều khiển chân GPIO.

2. **Cơ chế Event Group**:

   Hệ thống sử dụng Event Group để quản lý các trạng thái điều khiển. Các bit trong Event Group đại diện cho các trạng thái của thiết bị (như bật/tắt đèn, mở/đóng cửa). Việc thiết lập hoặc xóa các bit này sẽ giúp thay đổi trạng thái của các thiết bị.

Giải thích về một số hàm quan trọng:

- xEventGroupSetBits và xEventGroupClearBits:

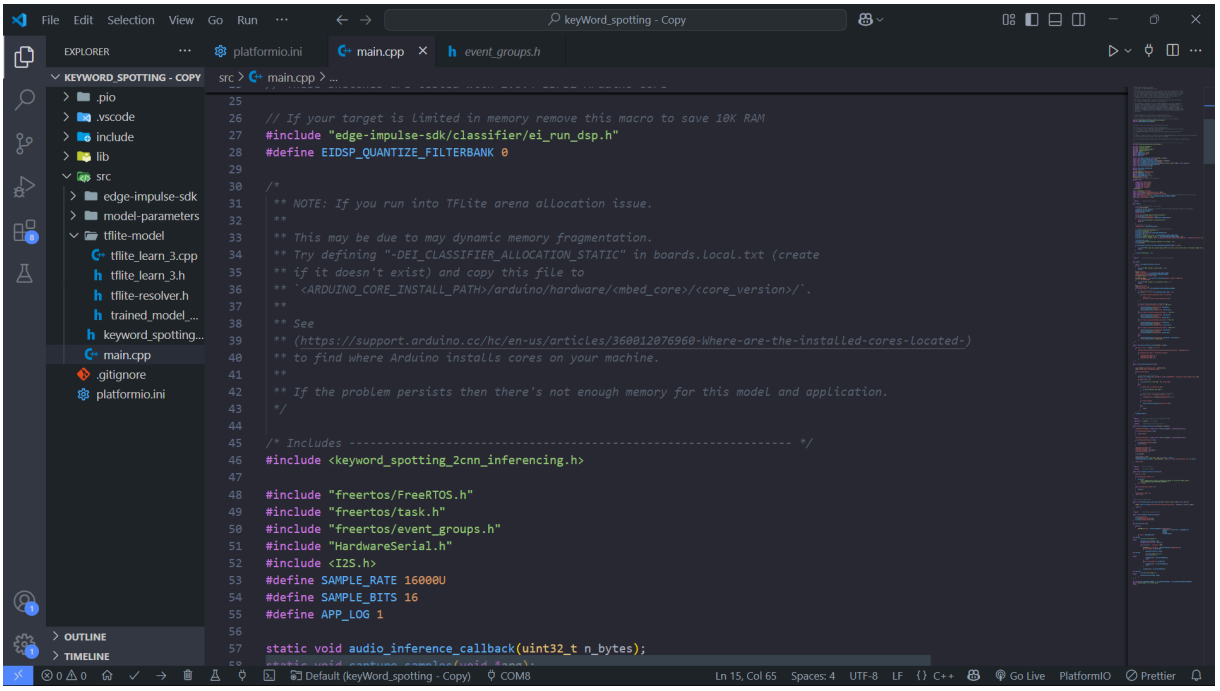
  Các hàm này được sử dụng để quản lý các sự kiện liên quan đến việc điều khiển thiết bị. Khi mô hình nhận diện được một lệnh giọng nói như "bật đèn", xEventGroupSetBits(ledEventGroup, LED_ON_BIT); sẽ được gọi để bật đèn. Tương tự, khi nhận diện lệnh "tắt đèn", xEventGroupClearBits(ledEventGroup, LED_ON_BIT); sẽ được gọi để tắt đèn.

- digitalWrite:

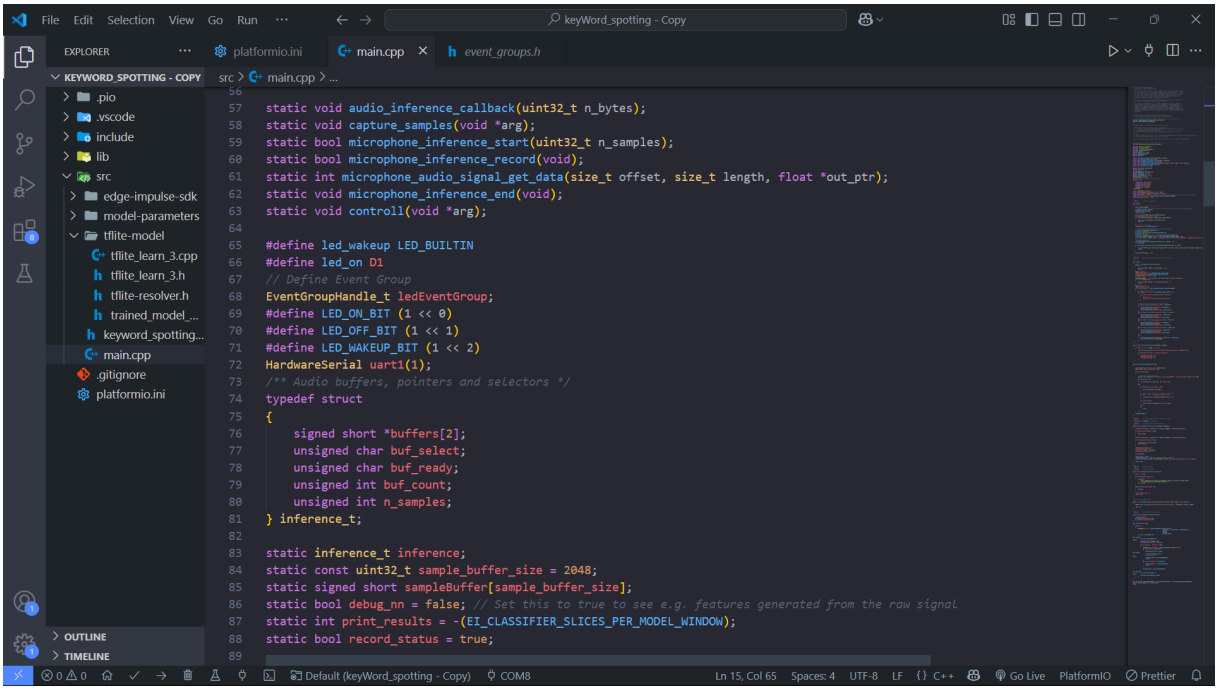
  Hàm digitalWrite được sử dụng để thay đổi trạng thái của các chân GPIO. Ví dụ, digitalWrite(led_on, HIGH); sẽ bật đèn và digitalWrite(led_on, LOW); sẽ tắt đèn.

- vTaskDelay:

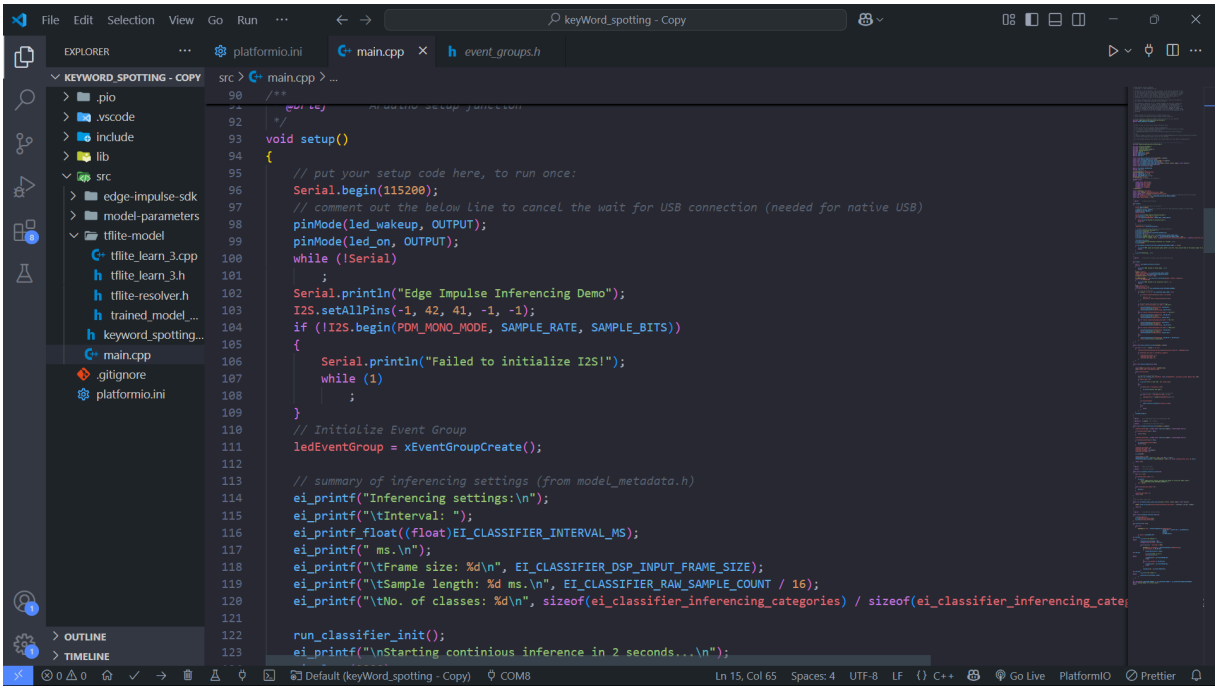
  Hàm này được sử dụng để tạo độ trễ giữa các hành động, giúp hệ thống điều khiển các thiết bị một cách chính xác và trơn tru. Ví dụ, sau khi kích hoạt đèn, hệ thống có thể sử dụng vTaskDelay(50 / portTICK_PERIOD_MS); để tạo độ trễ ngắn trước khi thực hiện hành động tiếp theo.

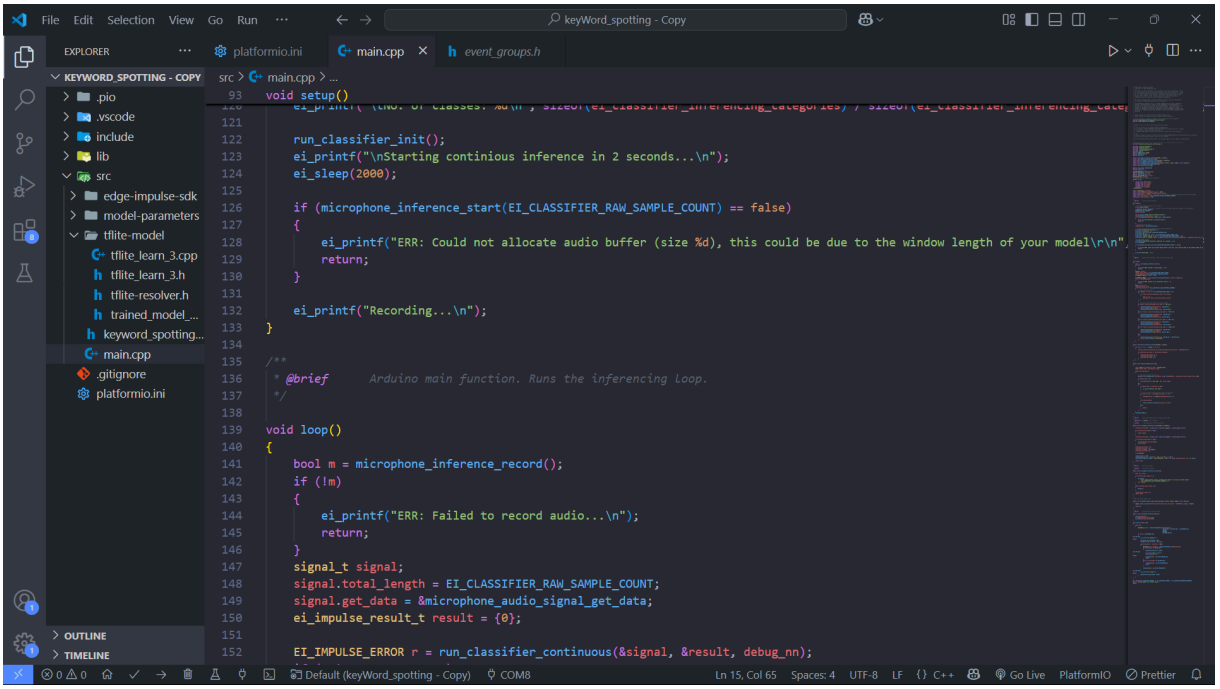


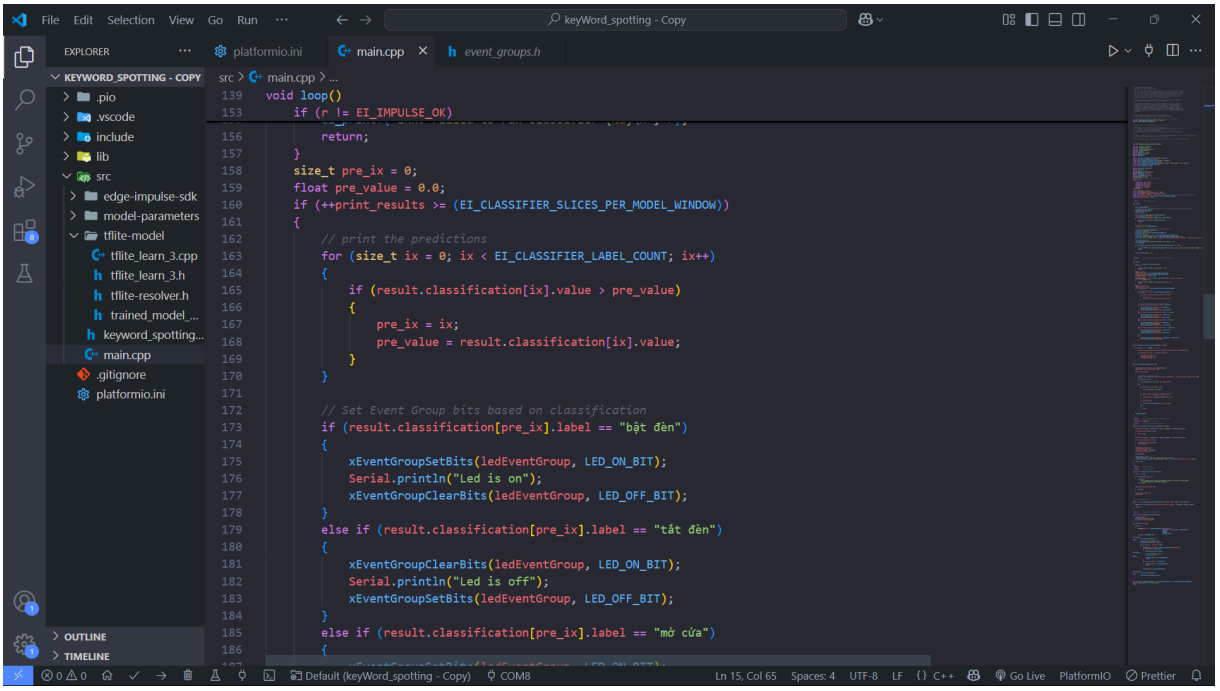
```
// If your target is limited in memory remove this macro to save 10K RAM
#include "edge-impulse-sdk/classifier/ei_run_dsp.h"
#define EIDSP_QUANTIZE_FILTERBANK 0

/*
 ** NOTE: If you run into TFLite arena allocation issue.
 **
 ** This may be due to may dynamic memory fragmentation.
 ** Try defining "-DEI_CLASSIFIER_ALLOCATION_STATIC" in boards.local.txt (create
 ** if it doesn't exist) and copy this file to
 ** `<ARDUINO_CORE_INSTALL_PATH>/arduino/hardware/<mbed_core>/<core_version>/`.
 **
 ** See
 ** (https://support.arduino.cc/hc/en-us/articles/360012076960-Where-are-the-installed-cores-located-)
 ** to find where Arduino installs cores on your machine.
 **
 ** If the problem persists then there's not enough memory for this model and application.
 */

/* Includes ---------------------------------------------------------------- */
#include <keyword_spotting_2cnn_inferencing.h>

#include "freertos/FreeRTOS.h"
#include "freertos/task.h"
#include "freertos/event_groups.h"
#include "HardwareSerial.h"
#include <I2S.h>
#define SAMPLE_RATE 16000U
#define SAMPLE_BITS 16
#define APP_LOG 1

static void audio_inference_callback(uint32_t n_bytes);
```

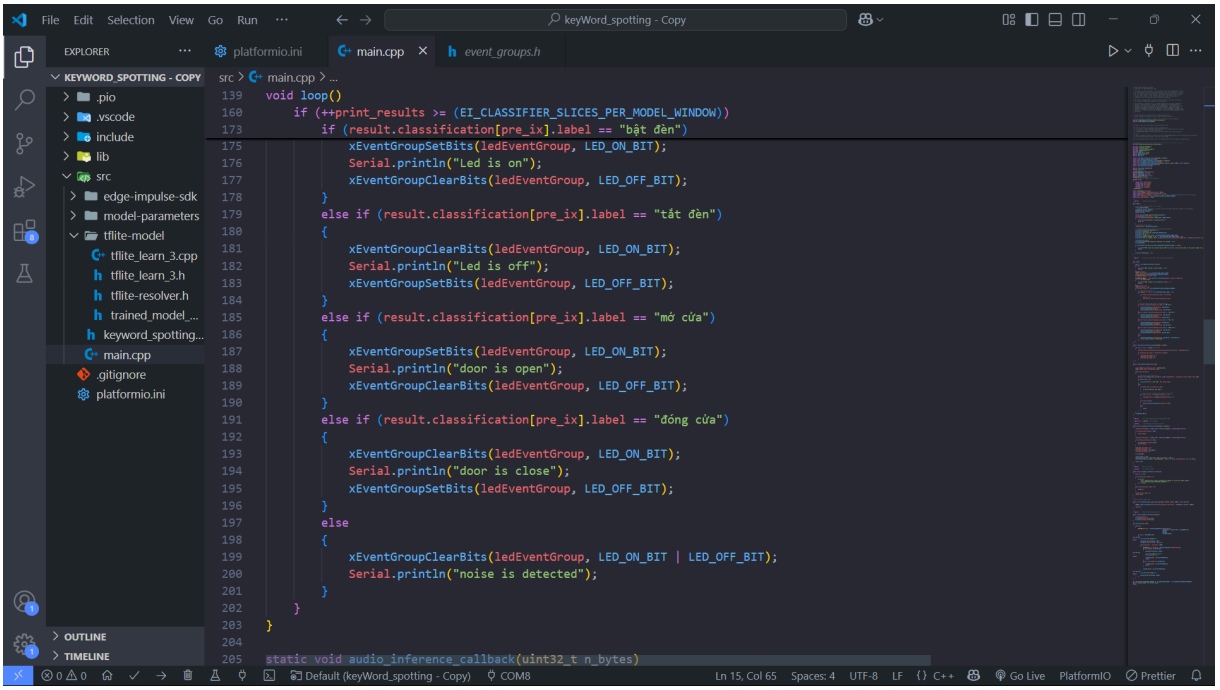


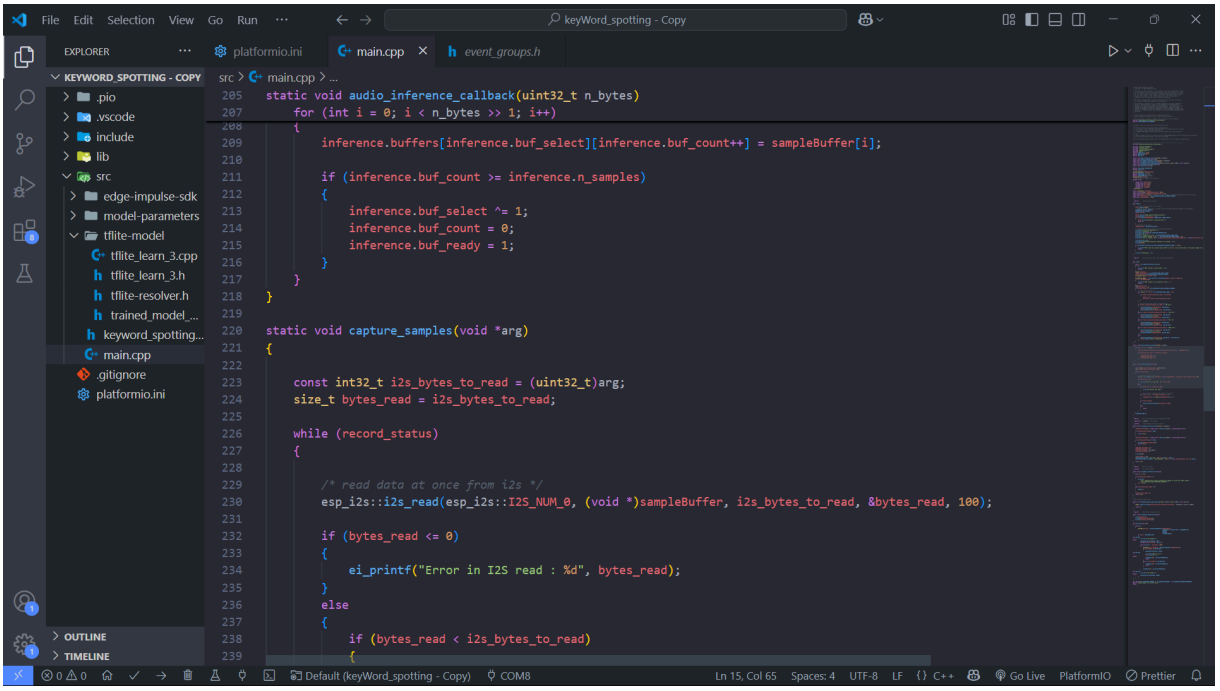
```
static void audio_inference_callback(uint32_t n_bytes);
static void capture_samples(void *arg);
static bool microphone_inference_start(uint32_t n_samples);
static bool microphone_inference_record(void);
static int microphone_audio_signal_get_data(size_t offset, size_t length, float *out_ptr);
static void microphone_inference_end(void);
static void controll(void *arg);

#define led_wakeup LED_BUILTIN
#define led_on D1
// Define Event Group
EventGroupHandle_t ledEventGroup;
#define LED_ON_BIT (1 << 0)
#define LED_OFF_BIT (1 << 1)
#define LED_WAKEUP_BIT (1 << 2)
HardwareSerial uart1(1);
/** Audio buffers, pointers and selectors */
typedef struct
{
    signed short *buffers[2];
    unsigned char buf_select;
    unsigned char buf_ready;
    unsigned int buf_count;
    unsigned int n_samples;
} inference_t;

static inference_t inference;
static const uint32_t sample_buffer_size = 2048;
static signed short sampleBuffer[sample_buffer_size];
static bool debug_nn = false; // Set this to true to see e.g. features generated from the raw signal
static int print_results = -(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW);
static bool record_status = true;
```

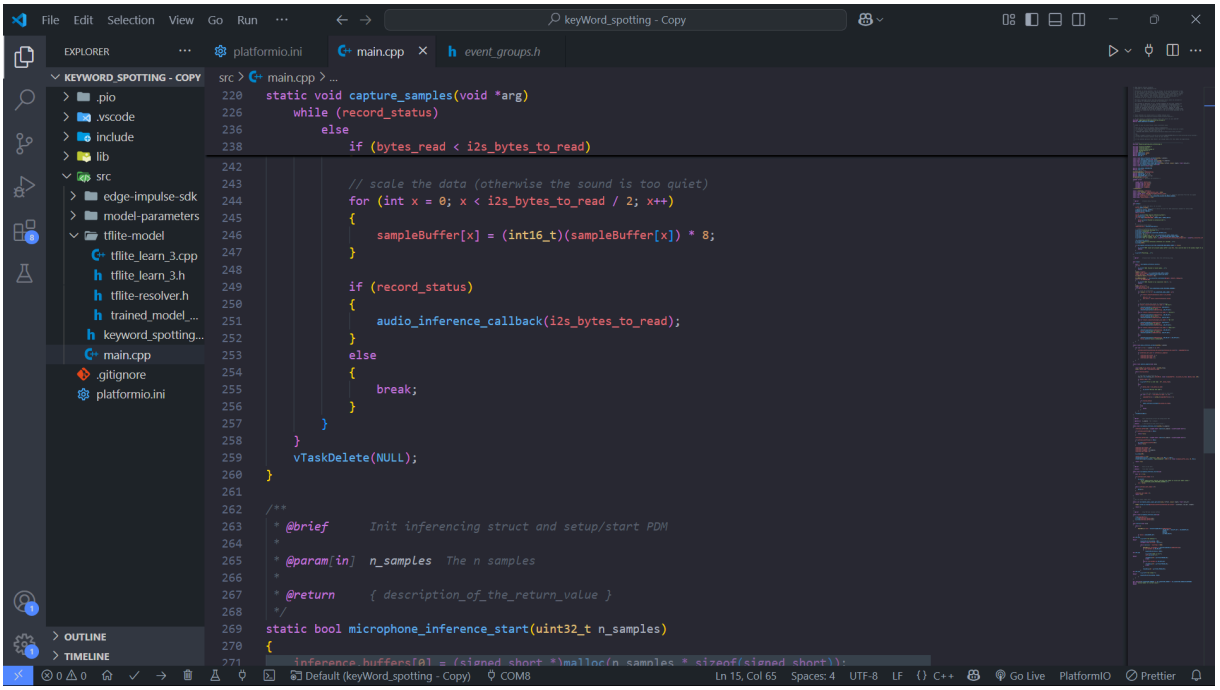

File Edit Selection View Go Run
keyWord_spotting - Copy
EXPLORER
platformio.ini
main.cpp
event_groups.h
KEYWORD_SPOTTING - COPY
.pio
.vscode
include
lib
src
edge-impulse-sdk
model-parameters
tflite-model
tflite_learn_3.cpp
tflite_learn_3.h
tflite-resolver.h
trained_model_...
keyword_spotting...
main.cpp
.gitignore
platformio.ini
OUTLINE
TIMELINE
src > main.cpp > ...
/**
*/
void setup()
{
// put your setup code here, to run once:
Serial.begin(115200);
// comment out the below line to cancel the wait for USB connection (needed for native USB)
pinMode(led_wakeup, OUTPUT);
pinMode(led_on, OUTPUT);
while (!Serial)
;
Serial.println("Edge Impulse Inferencing Demo");
I2S.setAllPins(-1, 42, 41, -1, -1);
if (!I2S.begin(PDM_MONO_MODE, SAMPLE_RATE, SAMPLE_BITS))
{
Serial.println("Failed to initialize I2S!");
while (1)
;
}
// Initialize Event Group
ledEventGroup = xEventGroupCreate();
// summary of inferencing settings (from model_metadata.h)
ei_printf("Inferencing settings:\n");
ei_printf("\tInterval: ");
ei_printf_float((float)EI_CLASSIFIER_INTERVAL_MS);
ei_printf(" ms.\n");
ei_printf("\tFrame size: %d\n", EI_CLASSIFIER_DSP_INPUT_FRAME_SIZE);
ei_printf("\tSample length: %d ms.\n", EI_CLASSIFIER_RAW_SAMPLE_COUNT / 16);
ei_printf("\tNo. of classes: %d\n", sizeof(ei_classifier_inferencing_categories) / sizeof(ei_classifier_inferencing_cate
run_classifier_init();
ei_printf("\nStarting continious inference in 2 seconds...\n");
Default (keyWord_spotting - Copy)
COM8
Ln 15, Col 65
Spaces: 4
UTF-8
LF
C++
Go Live
PlatformIO
Prettier

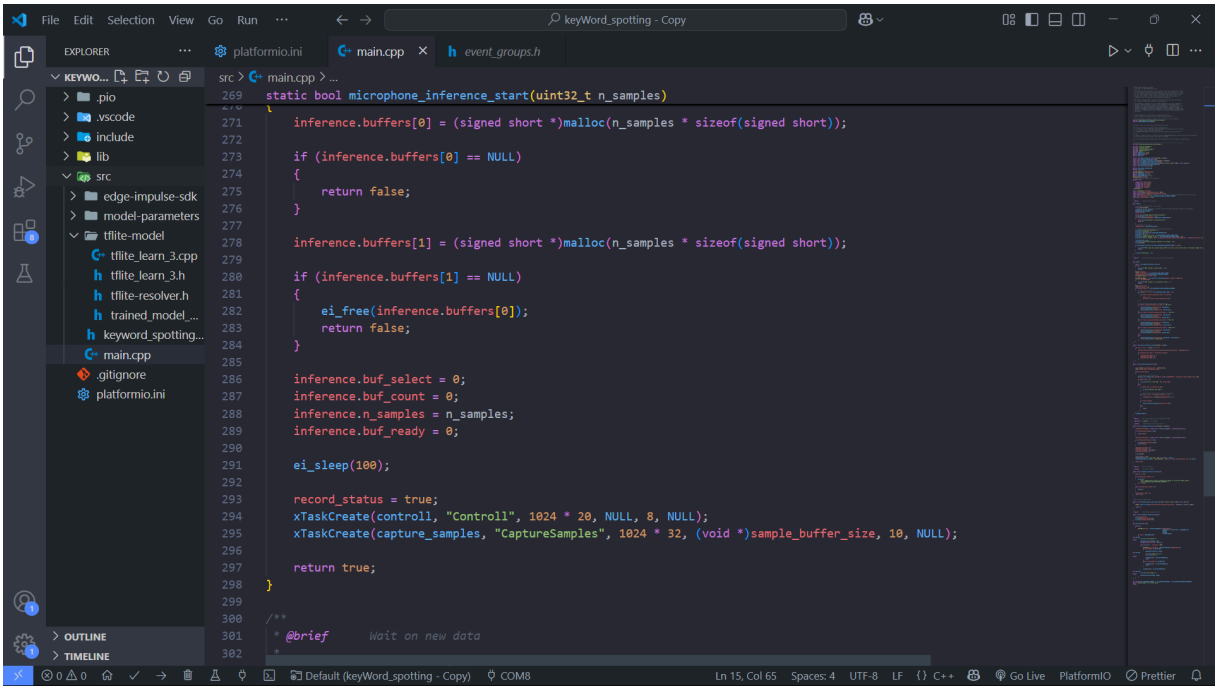

File Edit Selection View Go Run
keyWord_spotting - Copy
EXPLORER
platformio.ini
main.cpp
event_groups.h
KEYWORD_SPOTTING - COPY
.pio
.vscode
include
lib
src
edge-impulse-sdk
model-parameters
tflite-model
tflite_learn_3.cpp
tflite_learn_3.h
tflite-resolver.h
trained_model_...
keyword_spotting...
main.cpp
.gitignore
platformio.ini
OUTLINE
TIMELINE
src > main.cpp > ...
void setup()
run_classifier_init();
ei_printf("\nStarting continious inference in 2 seconds...\n");
ei_sleep(2000);
if (microphone_inference_start(EI_CLASSIFIER_RAW_SAMPLE_COUNT) == false)
{
ei_printf("ERR: Could not allocate audio buffer (size %d), this could be due to the window length of your model\r\n",
return;
}
ei_printf("Recording...\n");
}
/**
* @brief Arduino main function. Runs the inferencing loop.
*/
void loop()
{
bool m = microphone_inference_record();
if (!m)
{
ei_printf("ERR: Failed to record audio...\n");
return;
}
signal_t signal;
signal.total_length = EI_CLASSIFIER_RAW_SAMPLE_COUNT;
signal.get_data = &microphone_audio_signal_get_data;
ei_impulse_result_t result = {0};
EI_IMPULSE_ERROR r = run_classifier_continuous(&signal, &result, debug_nn);
Default (keyWord_spotting - Copy)
COM8
Ln 15, Col 65
Spaces: 4
UTF-8
LF
C++
Go Live
PlatformIO
Prettier

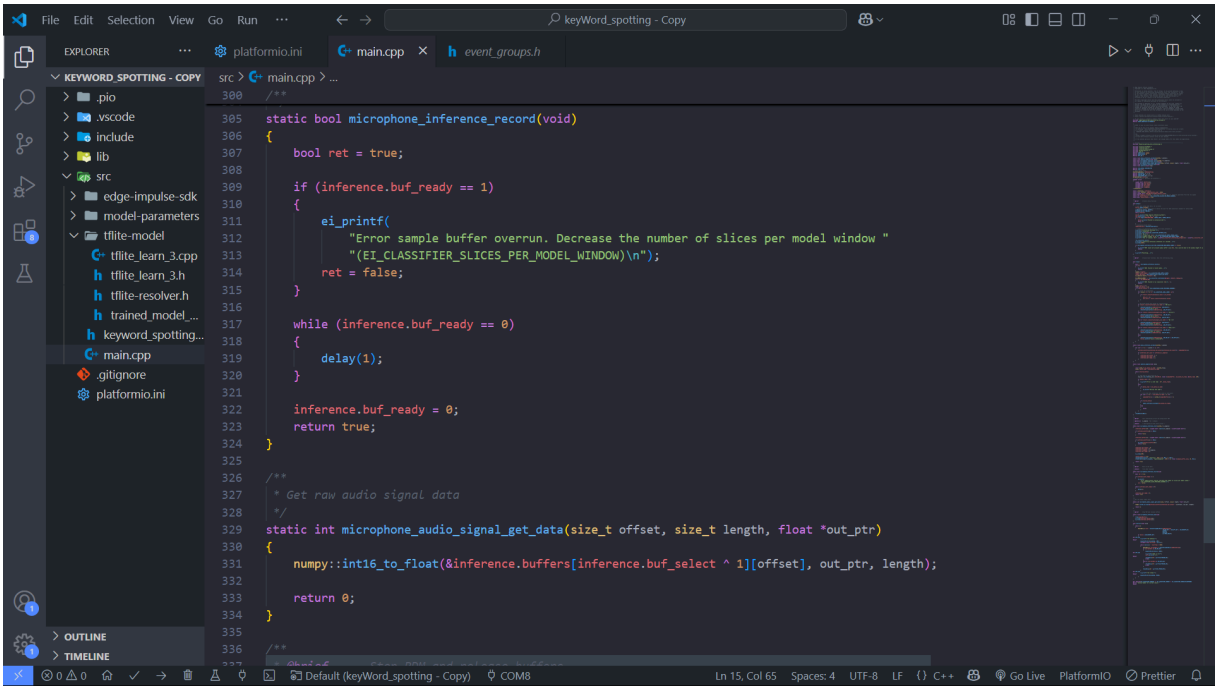

File Edit Selection View Go Run
keyWord_spotting - Copy
EXPLORER
platformio.ini
main.cpp
event_groups.h
KEYWORD_SPOTTING - COPY
.pio
.vscode
include
lib
src
edge-impulse-sdk
model-parameters
tflite-model
tflite_learn_3.cpp
tflite_learn_3.h
tflite-resolver.h
trained_model_...
keyword_spotting...
main.cpp
.gitignore
platformio.ini
OUTLINE
TIMELINE
src > main.cpp > ...
void loop()
if (r != EI_IMPULSE_OK)
return;
size_t pre_ix = 0;
float pre_value = 0.0;
if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW))
// print the predictions
for (size_t ix = 0; ix < EI_CLASSIFIER_LABEL_COUNT; ix++)
if (result.classification[ix].value > pre_value)
pre_ix = ix;
pre_value = result.classification[ix].value;
// Set Event Group bits based on classification
if (result.classification[pre_ix].label == "bật đèn")
xEventGroupSetBits(ledEventGroup, LED_ON_BIT);
Serial.println("Led is on");
xEventGroupClearBits(ledEventGroup, LED_OFF_BIT);
else if (result.classification[pre_ix].label == "tắt đèn")
xEventGroupClearBits(ledEventGroup, LED_ON_BIT);
Serial.println("Led is off");
xEventGroupSetBits(ledEventGroup, LED_OFF_BIT);
else if (result.classification[pre_ix].label == "mở cửa")
Default (keyWord_spotting - Copy)
COM8
Ln 15, Col 65
Spaces: 4
UTF-8
LF
C++
Go Live
PlatformIO
Prettier

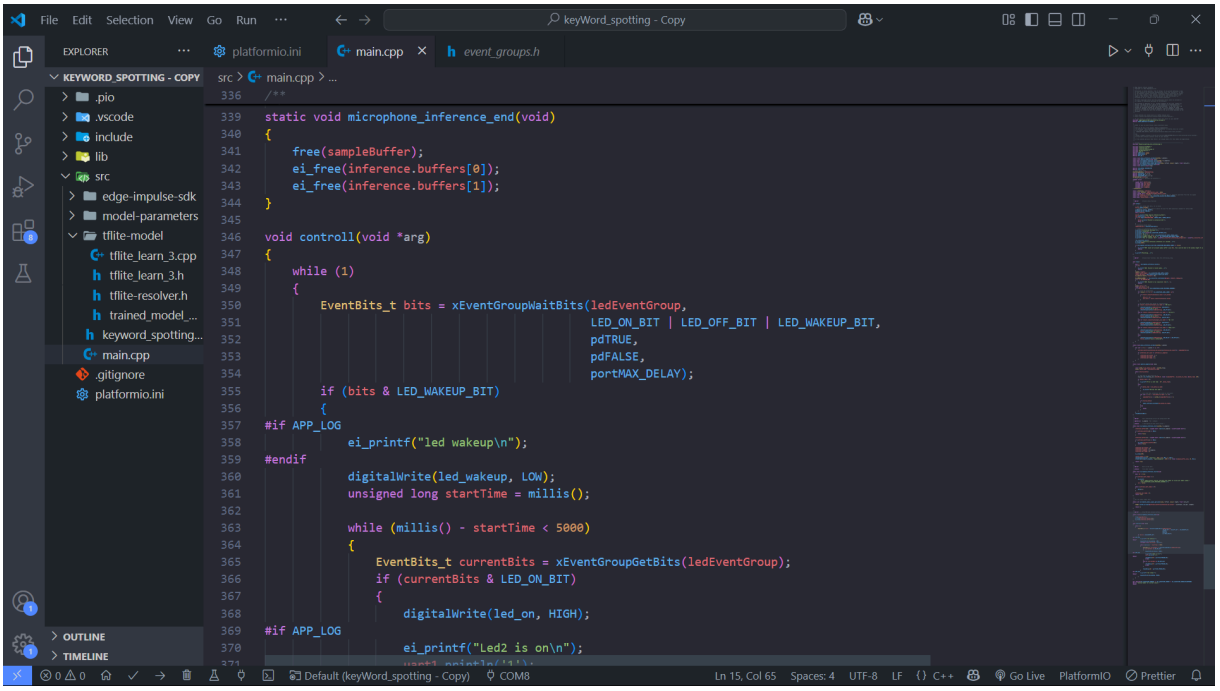

File Edit Selection View Go Run
keyWord_spotting - Copy
EXPLORER
platformio.ini
main.cpp
event_groups.h
KEYWORD_SPOTTING - COPY
.pio
.vscode
include
lib
src
edge-impulse-sdk
model-parameters
tflite-model
tflite_learn_3.cpp
tflite_learn_3.h
tflite-resolver.h
trained_model_...
keyword_spotting...
main.cpp
.gitignore
platformio.ini
OUTLINE
TIMELINE
src > main.cpp > ...
void loop()
if (++print_results >= (EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW))
if (result.classification[pre_ix].label == "bật đèn")
xEventGroupSetBits(ledEventGroup, LED_ON_BIT);
Serial.println("Led is on");
xEventGroupClearBits(ledEventGroup, LED_OFF_BIT);
else if (result.classification[pre_ix].label == "tắt đèn")
xEventGroupClearBits(ledEventGroup, LED_ON_BIT);
Serial.println("Led is off");
xEventGroupSetBits(ledEventGroup, LED_OFF_BIT);
else if (result.classification[pre_ix].label == "mở cửa")
xEventGroupSetBits(ledEventGroup, LED_ON_BIT);
Serial.println("door is open");
xEventGroupClearBits(ledEventGroup, LED_OFF_BIT);
else if (result.classification[pre_ix].label == "đóng cửa")
xEventGroupClearBits(ledEventGroup, LED_ON_BIT);
Serial.println("door is close");
xEventGroupSetBits(ledEventGroup, LED_OFF_BIT);
else
xEventGroupClearBits(ledEventGroup, LED_ON_BIT | LED_OFF_BIT);
Serial.println("noise is detected");
static void audio_inference_callback(uint32_t n_bytes)
Default (keyWord_spotting - Copy)
COM8
Ln 15, Col 65
Spaces: 4
UTF-8
LF
C++
Go Live
PlatformIO
Prettier


File Edit Selection View Go Run
keyWord_spotting - Copy
EXPLORER
platformio.ini
main.cpp
event_groups.h
KEYWORD_SPOTTING - COPY
.pio
.vscode
include
lib
src
edge-impulse-sdk
model-parameters
tflite-model
tflite_learn_3.cpp
tflite_learn_3.h
tflite-resolver.h
trained_model_...
keyword_spotting...
main.cpp
.gitignore
platformio.ini
OUTLINE
TIMELINE
src > main.cpp > ...
static void audio_inference_callback(uint32_t n_bytes)
for (int i = 0; i < n_bytes >> 1; i++)
{
inference.buffers[inference.buf_select][inference.buf_count++] = sampleBuffer[i];
if (inference.buf_count >= inference.n_samples)
{
inference.buf_select ^= 1;
inference.buf_count = 0;
inference.buf_ready = 1;
}
}
}
static void capture_samples(void *arg)
{
const int32_t i2s_bytes_to_read = (uint32_t)arg;
size_t bytes_read = i2s_bytes_to_read;
while (record_status)
{
/* read data at once from i2s */
esp_i2s::i2s_read(esp_i2s::I2S_NUM_0, (void *)sampleBuffer, i2s_bytes_to_read, &bytes_read, 100);
if (bytes_read <= 0)
{
ei_printf("Error in I2S read : %d", bytes_read);
}
else
{
if (bytes_read < i2s_bytes_to_read)
{
Default (keyWord_spotting - Copy)
COM8
Ln 15, Col 65
Spaces: 4
UTF-8
LF
C++
Go Live
PlatformIO
Prettier

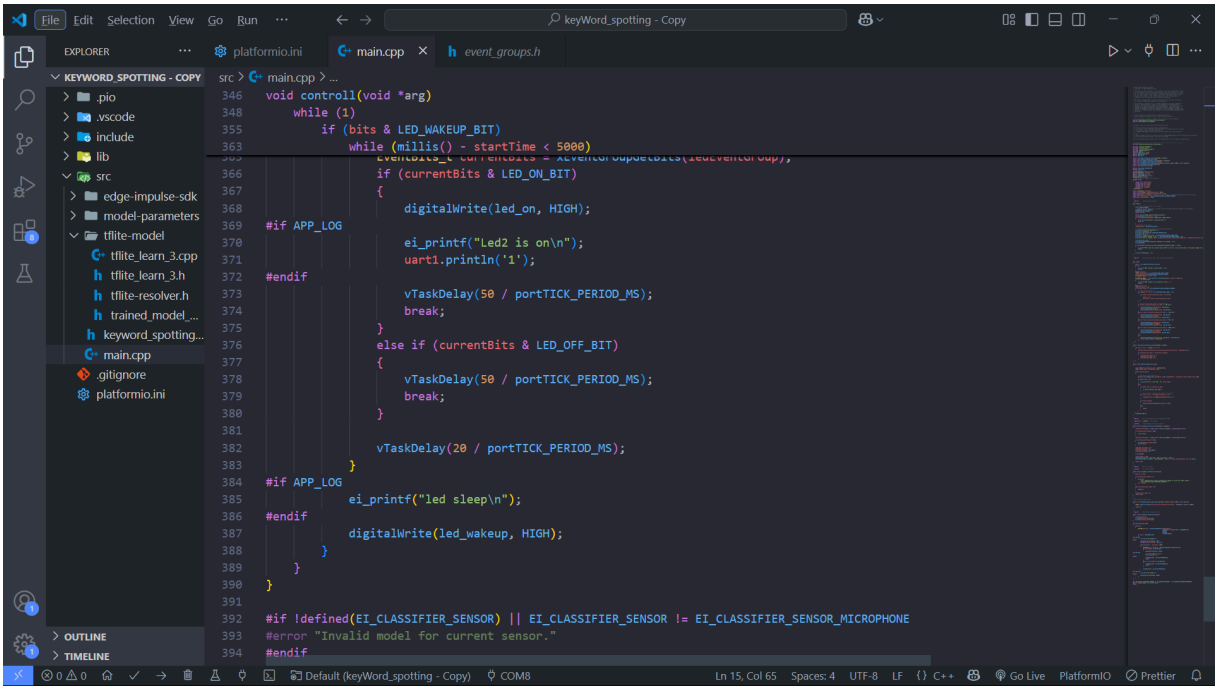

File Edit Selection View Go Run
keyWord_spotting - Copy
EXPLORER
platformio.ini
main.cpp
event_groups.h
KEYWORD_SPOTTING - COPY
src > main.cpp > ...
static void capture_samples(void *arg)
while (record_status)
else
if (bytes_read < i2s_bytes_to_read)
// scale the data (otherwise the sound is too quiet)
for (int x = 0; x < i2s_bytes_to_read / 2; x++)
{
sampleBuffer[x] = (int16_t)(sampleBuffer[x]) * 8;
}
if (record_status)
{
audio_inference_callback(i2s_bytes_to_read);
}
else
{
break;
}
}
}
vTaskDelete(NULL);
}
/**
* @brief Init inferencing struct and setup/start PDM
*
* @param[in] n_samples The n samples
*
* @return { description_of_the_return_value }
*/
static bool microphone_inference_start(uint32_t n_samples)
{
Default (keyWord_spotting - Copy)
COM8
Ln 15, Col 65
Spaces: 4
UTF-8
LF
C++
Go Live
PlatformIO
Prettier


File Edit Selection View Go Run
keyWord_spotting - Copy
EXPLORER
platformio.ini
main.cpp
event_groups.h
src > main.cpp > ...
KEYWO...
.pio
.vscode
include
lib
src
edge-impulse-sdk
model-parameters
tflite-model
tflite_learn_3.cpp
tflite_learn_3.h
tflite-resolver.h
trained_model_...
keyword_spotting...
main.cpp
.gitignore
platformio.ini
OUTLINE
TIMELINE
269 static bool microphone_inference_start(uint32_t n_samples)
271 inference.buffers[0] = (signed short *)malloc(n_samples * sizeof(signed short));
273 if (inference.buffers[0] == NULL)
274 {
275 return false;
276 }
278 inference.buffers[1] = (signed short *)malloc(n_samples * sizeof(signed short));
280 if (inference.buffers[1] == NULL)
281 {
282 ei_free(inference.buffers[0]);
283 return false;
284 }
286 inference.buf_select = 0;
287 inference.buf_count = 0;
288 inference.n_samples = n_samples;
289 inference.buf_ready = 0;
291 ei_sleep(100);
293 record_status = true;
294 xTaskCreate(controll, "Controll", 1024 * 20, NULL, 8, NULL);
295 xTaskCreate(capture_samples, "CaptureSamples", 1024 * 32, (void *)sample_buffer_size, 10, NULL);
297 return true;
298 }
300 /**
301 * @brief Wait on new data
Default (keyWord_spotting - Copy)
COM8
Ln 15, Col 65
Spaces: 4
UTF-8
LF
C++
Go Live
PlatformIO
Prettier


File Edit Selection View Go Run
keyWord_spotting - Copy
EXPLORER
platformio.ini
main.cpp
event_groups.h
src > main.cpp > ...
KEYWORD_SPOTTING - COPY
.pio
.vscode
include
lib
src
edge-impulse-sdk
model-parameters
tflite-model
tflite_learn_3.cpp
tflite_learn_3.h
tflite-resolver.h
trained_model_...
keyword_spotting...
main.cpp
.gitignore
platformio.ini
OUTLINE
TIMELINE
305 static bool microphone_inference_record(void)
306 {
307 bool ret = true;
309 if (inference.buf_ready == 1)
310 {
311 ei_printf(
312 "Error sample buffer overrun. Decrease the number of slices per model window "
313 "(EI_CLASSIFIER_SLICES_PER_MODEL_WINDOW)\n");
314 ret = false;
315 }
317 while (inference.buf_ready == 0)
318 {
319 delay(1);
320 }
322 inference.buf_ready = 0;
323 return true;
324 }
326 /**
327 * Get raw audio signal data
328 */
329 static int microphone_audio_signal_get_data(size_t offset, size_t length, float *out_ptr)
330 {
331 numpy::int16_to_float(&inference.buffers[inference.buf_select ^ 1][offset], out_ptr, length);
333 return 0;
334 }
336 /**
Default (keyWord_spotting - Copy)
COM8
Ln 15, Col 65
Spaces: 4
UTF-8
LF
C++
Go Live
PlatformIO
Prettier



```
/**
static void microphone_inference_end(void)
{
    free(sampleBuffer);
    ei_free(inference.buffers[0]);
    ei_free(inference.buffers[1]);
}

void controll(void *arg)
{
    while (1)
    {
        EventBits_t bits = xEventGroupWaitBits(ledEventGroup,
                                               LED_ON_BIT | LED_OFF_BIT | LED_WAKEUP_BIT,
                                               pdTRUE,
                                               pdFALSE,
                                               portMAX_DELAY);
        if (bits & LED_WAKEUP_BIT)
        {
#if APP_LOG
            ei_printf("led wakeup\n");
#endif
            digitalWrite(led_wakeup, LOW);
            unsigned long startTime = millis();

            while (millis() - startTime < 5000)
            {
                EventBits_t currentBits = xEventGroupGetBits(ledEventGroup);
                if (currentBits & LED_ON_BIT)
                {
                    digitalWrite(led_on, HIGH);
#if APP_LOG
                    ei_printf("Led2 is on\n");
```



```
void controll(void *arg)
    while (1)
        if (bits & LED_WAKEUP_BIT)
            while (millis() - startTime < 5000)
                if (currentBits & LED_ON_BIT)
                {
                    digitalWrite(led_on, HIGH);
#if APP_LOG
                    ei_printf("Led2 is on\n");
                    uart1.println('1');
#endif
                    vTaskDelay(50 / portTICK_PERIOD_MS);
                    break;
                }
                else if (currentBits & LED_OFF_BIT)
                {
                    vTaskDelay(50 / portTICK_PERIOD_MS);
                    break;
                }

                vTaskDelay(20 / portTICK_PERIOD_MS);
            }
#if APP_LOG
            ei_printf("led sleep\n");
#endif
            digitalWrite(led_wakeup, HIGH);
        }
    }
}

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

CHƯƠNG 4: THỬ NGHIỆM VÀ ĐÁNH GIÁ

Kết quả thử nghiệm được đánh giá thông qua các chỉ số như accuracy, loss, và các chỉ số trong ma trận nhầm lẫn (Confusion Matrix). Dưới đây là bảng kết quả từ ma trận nhầm lẫn và các chỉ số quan trọng.

- Độ chính xác (Accuracy): 97.9%
  Đây là tỷ lệ tổng thể các dự đoán chính xác so với tổng số thử nghiệm.
- Hàm mất mát (Loss): 0.11
  Loss thấp cho thấy mô hình đang học khá tốt và không gặp phải vấn đề lớn về việc tối ưu hóa trong quá trình huấn luyện.
- Ma trận nhầm lẫn (Confusion Matrix):
  Ma trận nhầm lẫn giúp đánh giá hiệu suất của mô hình trong việc phân loại từng lệnh giọng nói. Dưới đây là kết quả chi tiết:

	BẬT ĐÈN	MỞ CỬA	NOISE	TẮT ĐÈN	ĐÓNG CỬA
BẬT ĐÈN	97.7%	0%	1.5%	0.8%	0%
MỞ CỬA	0%	55.6%	11.1%	0%	33.3%
NOISE	0.2%	0%	99.5%	0.2%	0%
TẮT ĐÈN	3.0%	0%	2.0%	94.9%	0%
ĐÓNG CỬA	0%	0%	0%	0%	100%
F1 SCORE	0.97	0.71	0.99	0.96	0.87

- F1 Score:
  - Bật đèn: 0.97
  - Mở cửa: 0.71
  - Noise: 0.99
  - Tắt đèn: 0.96
  - Đóng cửa: 0.87

Giải thích:

- Bật đèn: Mô hình đạt độ chính xác 97.7% trong việc nhận diện lệnh "bật đèn", với rất ít nhầm lẫn sang các lệnh khác. Tuy nhiên, có khoảng 1.5% tín hiệu được phân loại là "Noise".

- Mở cửa: Độ chính xác thấp hơn một chút so với "bật đèn" với 55.6% khi nhận diện lệnh "mở cửa". Một phần lớn của các dự đoán sai là do nhầm với "Noise" (11.1%) và "Đóng cửa" (33.3%).
- Noise: Lệnh "Noise" được nhận diện với độ chính xác rất cao, lên tới 99.5%, với phần lớn các nhầm lẫn xảy ra khi tín hiệu được phân loại thành "Tắt đèn" (2.0%).
- Tắt đèn: Mô hình có độ chính xác 94.9% trong việc nhận diện lệnh "tắt đèn". Các nhầm lẫn chủ yếu là do lệnh "Bật đèn" (3.0%) và "Noise" (2.0%).
- Đóng cửa: Đây là lệnh nhận diện tốt nhất với độ chính xác 100%, không có sự nhầm lẫn.

Đánh giá:

- Hệ thống nhận diện giọng nói hoạt động tốt trong việc phân biệt các lệnh giọng nói như "bật đèn", "tắt đèn", và "đóng cửa", với độ chính xác cao.
- Tuy nhiên, việc nhận diện lệnh "mở cửa" gặp một số khó khăn, đặc biệt là khi có sự can thiệp của tạp âm (noise). Phần lớn nhầm lẫn xảy ra giữa "mở cửa" và "đóng cửa".
- Mô hình có khả năng xử lý rất tốt các tín hiệu nhiễu (noise), với độ chính xác cao trong việc nhận diện "noise" là 99.5%.

CHƯƠNG 5: KẾT LUẬN VÀ KIẾN NGHỊ

5.1 Kết luận

Hệ thống nhận diện giọng nói trong đồ án đã đạt được nhiều kết quả tích cực, đặc biệt là trong các thử nghiệm đánh giá độ chính xác của việc nhận diện các lệnh giọng nói cơ bản như "bật đèn", "tắt đèn", "mở cửa", và "đóng cửa". Hệ thống sử dụng TinyML và mô hình phân loại giọng nói huấn luyện từ Edge Impulse, triển khai trên phần cứng ESP32-S3 đã cho thấy khả năng hoạt động hiệu quả mà không cần kết nối mạng, giúp tối ưu hóa tài nguyên và giảm độ trễ.

Các kết quả chính:

- Độ chính xác tổng thể (Accuracy) của hệ thống đạt 97.9% trong môi trường thử nghiệm. Đây là một kết quả rất ấn tượng, đặc biệt là trong các thử nghiệm với môi trường ít tạp âm.
- F1 Score cho thấy hệ thống có hiệu suất tốt, với điểm số cao trong việc nhận diện các lệnh như "Bật đèn" (0.97), "Noise" (0.99), và "Tắt đèn" (0.96). Tuy nhiên, lệnh "Mở cửa" có F1 Score thấp hơn (0.71), cho thấy một số vấn đề trong việc phân biệt giữa "Mở cửa" và "Đóng cửa", đặc biệt trong môi trường có tạp âm.
- Khả năng xử lý tạp âm của mô hình là một điểm mạnh, với 99.5% độ chính xác khi nhận diện "Noise". Tuy nhiên, khi môi trường có tạp âm mạnh, sự chính xác của các lệnh giọng nói khác giảm đi một chút.

Mô hình đã cho thấy khả năng xử lý tín hiệu âm thanh hiệu quả, đồng thời có thể được triển khai trực tiếp trên vi điều khiển ESP32-S3, giảm độ trễ và phụ thuộc vào kết nối mạng. Điều này chứng tỏ rằng hệ thống không chỉ hoạt động ổn định mà còn có tiềm năng lớn trong việc ứng dụng vào các thiết bị thông minh IoT trong tương lai.

Ứng dụng thực tế: Hệ thống nhận diện giọng nói này có thể được ứng dụng trong nhiều lĩnh vực khác nhau, đặc biệt trong các hệ thống gia đình thông minh và tự động hóa. Ví dụ, người dùng có thể dễ dàng điều khiển các thiết bị như đèn, quạt, máy lạnh, hoặc cửa ra vào thông qua lệnh giọng nói mà không cần sử dụng đến các thiết bị điều khiển phức tạp. Hệ thống này cũng mở ra cơ hội ứng dụng trong các khu vực công cộng, như các công viên thông minh, văn phòng tự động hóa, hay trong các thiết bị hỗ trợ người khuyết tật.

5.2 Kiến nghị

Mặc dù hệ thống đã hoạt động khá hiệu quả, vẫn còn một số điểm cần cải thiện để nâng cao khả năng nhận diện giọng nói và ứng dụng trong môi trường thực tế. Sau đây là các kiến nghị giúp nâng cao chất lượng và khả năng ứng dụng của hệ thống:

1. Cải thiện khả năng phân biệt các lệnh tương tự:

- Trong thử nghiệm, hệ thống gặp khó khăn trong việc phân biệt giữa "Mở cửa" và "Đóng cửa", với một phần lớn sự nhầm lẫn giữa các lệnh này. Để cải thiện, có thể huấn luyện lại mô hình với dữ liệu đa dạng hơn, bao gồm các trường hợp giọng nói khác nhau và âm vực khác nhau của người nói, từ đó giúp mô hình nhận diện chính xác hơn.
- Một giải pháp khác là tăng cường dữ liệu bằng cách thu thập thêm các mẫu giọng nói từ nhiều nguồn và trong nhiều môi trường khác nhau. Điều này sẽ giúp mô hình học được các biến thể khác nhau của giọng nói và phân biệt các lệnh dễ dàng hơn.

2. Tăng cường khả năng xử lý tạp âm:
    - Mặc dù hệ thống hoạt động tốt trong môi trường ít tạp âm, nhưng khi có nhiều tiếng ồn nền, độ chính xác giảm đi một chút. Do đó, xử lý tín hiệu âm thanh nâng cao như lọc tạp âm và loại bỏ nhiễu trong thời gian thực có thể giúp tăng cường độ chính xác của hệ thống.
    - Cũng có thể thử sử dụng các thuật toán phân tích phổ âm thanh như Spectrogram hoặc Wavelet Transform để cải thiện khả năng nhận diện giọng nói trong môi trường nhiều tạp âm.
3. Mở rộng khả năng nhận diện lệnh giọng nói:
    - Hệ thống hiện tại chỉ có thể nhận diện 4 lệnh giọng nói cơ bản. Để hệ thống trở nên linh hoạt hơn và có thể điều khiển nhiều thiết bị khác nhau, cần mở rộng mô hình để nhận diện thêm các lệnh giọng nói như "bật quạt", "mở cửa sổ", hoặc "tắt điều hòa".
    - Việc mở rộng hệ thống có thể áp dụng cho các ứng dụng trong môi trường công nghiệp, y tế, hoặc các khu vực công cộng, nơi cần phải điều khiển nhiều thiết bị thông qua giọng nói.
4. Tối ưu hóa mô hình để giảm độ trễ:
    - Mặc dù hệ thống có độ trễ khá thấp, khoảng 500ms, nhưng trong một số ứng dụng yêu cầu phản hồi nhanh hơn, có thể tối ưu hóa mô hình thêm nữa bằng cách giảm kích thước mô hình hoặc sử dụng các thuật toán phân loại hiệu quả hơn.
    - Tăng cường khả năng parallel processing và sử dụng các kỹ thuật giảm tải tính toán như quantization hoặc pruning để giảm kích thước của mô hình mà không làm giảm độ chính xác.
5. Phát triển giao diện người dùng và ứng dụng di động:
    - Để tăng cường tính tiện lợi và khả năng sử dụng, có thể phát triển một ứng dụng di động hoặc giao diện người dùng (UI) cho phép người dùng điều khiển hệ thống từ xa, nhận thông báo khi có sự thay đổi trạng thái của các thiết bị (như khi đèn bật/tắt hoặc cửa mở/đóng).

- Giao diện người dùng cũng có thể cung cấp các tính năng tùy chỉnh, chẳng hạn như cho phép người dùng thêm các lệnh giọng nói tùy chỉnh hoặc điều chỉnh các thiết lập hệ thống như độ nhạy và tốc độ phản hồi.

6. Tích hợp với các hệ thống IoT khác:
    - Hệ thống có thể dễ dàng được tích hợp với các hệ thống IoT khác để tạo thành một mạng lưới các thiết bị thông minh. Ví dụ, có thể tích hợp với hệ thống Home Automation, các thiết bị smart home, hoặc các hệ thống an ninh thông minh để điều khiển các thiết bị trong nhà hoặc trong các khu vực công cộng.
    - Việc tích hợp với các hệ thống như vậy sẽ giúp mở rộng khả năng của hệ thống và làm cho nó trở thành một phần của hệ sinh thái IoT rộng lớn hơn.

TÀI LIỆU THAM KHẢO

[1] P. W. &. D. Situnayake, TinyML Machine Learning with TensorFlow Lite on Arduino and Ultra-Low-Power Microcontrollers.

[2] R. D. R. Richard L. Klevans, Voice Recognition.

[3] S. F. Barrett, Arduino V: Machine Learning.