ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC VÀ KỸ THUẬT MÁY TÍNH

Đồ án Thiết kế luận lý - CO3091

Báo cáo cuối kỳ

Đề tài: Phân loại âm thanh môi trường dùng ReSpeaker Mic Array v2.0

Giảng viên hướng dẫn:	Phạm Hoàng Anh	
Sinh viên thực hiện:	Trương Thiên Ân	2310190
	Nguyễn Thái Sơn	2312968
	Nguyễn Thanh Toàn	2313492
	Lê Công Vinh	2313912

THÀNH PHỐ HỒ CHÍ MINH, 12/2025

Mục lục

1 Giới thiệu **4**

2 Cơ sở lý thuyết **4**

2.1 Kỹ thuật xử lý tín hiệu số 4

2.1.1 Lý thuyết Bộ lọc Thông dải (Bandpass Filter) 4

2.1.2 Xấp xỉ Butterworth 5

2.1.3 Hiện thực hóa trong miền số (Digital Implementation) 6

2.2 Huấn luyện mô hình nhận diện âm thanh 7

2.2.1 Biểu diễn đặc trưng Log-Mel Spectrogram 7

2.2.2 Chuẩn hóa dữ liệu (Data Normalization) 8

2.2.3 Data Augmentation để tăng khả năng tổng quát hóa 8

2.2.4 Kiến trúc CNN cho phân loại âm thanh 10

2.2.5 Hàm mất mát Cross-Entropy và Label Smoothing 10

2.2.6 Tối ưu hóa và chiến lược học 11

2.2.7 Mất cân bằng lớp và Class Weight 12

2.2.8 Chiến lược huấn luyện và đánh giá 12

2.2.9 Ngưỡng tin cậy cho lớp Unknown 13

3 System Architecture **14**

3.1 Tổng quan kiến trúc hệ thống 14

3.2 Module phần cứng: SoundDetector (USB VAD/DOA) 14

3.2.1 Mô tả chức năng 14

3.2.2 Giao tiếp USB 15

3.3 Module DSP Processor: AudioProcessor 15

3.3.1 Mô tả chức năng 15

3.3.2 Pipeline xử lý 16

3.4 Module AI Classifier: AudioClassifier 16

3.4.1 Mô tả chức năng 16

3.4.2 Post-processing và ổn định theo thời gian 16

3.5 Module Integration Pipeline: SmartAudioSystem 17

3.5.1 Mô tả chức năng 17

3.5.2 Quản lý luồng xử lý 17

3.6 Module User Interface 17

3.6.1 GUI 17

3.6.2 CLI 18

4 Implementation **19**
4.1 Cấu trúc mã nguồn 19
4.1.1 Tổng quan tổ chức code 19
4.1.2 Chi tiết các module chính 19
4.2 Pipeline tích hợp trong `smart_audio_pipeline.py` 20
4.2.1 Luồng xử lý trong `process_and_predict()` 20
4.2.2 Pseudo-code triển khai 21
4.2.3 Định dạng *state* trả về 22
4.2.4 Cơ chế realtime trong GUI (thread + queue) 22
4.2.5 Dừng/chạy lại an toàn 23
4.3 Hiện thực bộ xử lý âm thanh 23
4.3.1 Phương thức khởi tạo:`__init__` 24
4.3.2 Phương thức `apply_bandpass(chunk)` 24
4.3.3 Phương thức `apply_spectral_gate(chunk)` 26
4.3.4 Phương thức `apply_agc(chunk)` 27
4.3.5 Phương thức `process(chunk)` 28
4.3.6 Phương thức `reset_states()` 29
4.4 Hiện thực mô hình học máy 29
4.4.1 Tiền xử lý dữ liệu 29
4.4.2 Huấn luyện mô hình 31
4.4.3 Suy luận thời gian thực 33
4.4.4 Hậu xử lý theo thời gian: 34

5 Kết quả kiểm thử & Đánh giá **35**
5.1 Kiểm thử bộ xử lý âm thanh 35
5.2 Kiểm thử mô hình học máy 36
5.2.1 Stability 36
5.2.2 Confidence 37

6 Kết luận **38**
6.1 Tóm tắt kết quả đạt được 38
6.2 Đánh giá hiệu suất và ưu điểm 38
6.3 Những hạn chế 38
6.4 Hướng phát triển tương lai 39
6.5 Kết luận cuối cùng 39

1 Giới thiệu

Phân loại âm thanh môi trường là một bài toán quan trọng trong IoT và các hệ thống thông minh hiện đại. Mục tiêu của đồ án là xây dựng một dịch vụ phân loại âm thanh (*Sound Detection Service*) hoạt động trong thời gian thực, kết hợp xử lý tín hiệu số (DSP) và trí tuệ nhân tạo (AI) để nhận diện các loại âm thanh môi trường từ dữ liệu thu thập bằng thiết bị ReSpeaker Mic Array v2.0.
Báo cáo được tổ chức theo các nội dung chính sau:

- Cơ sở lý thuyết: Trình bày lý thuyết DSP (bộ lọc Butterworth, AGC) và các phương pháp huấn luyện mô hình CNN cho phân loại âm thanh.
- Thiết kế hệ thống: Mô tả kiến trúc pipeline với bốn thành phần chính - thu thập âm thanh (VAD, DOA), xử lý tín hiệu (bandpass filter, spectral gating, AGC), phân loại AI (CNN), và giao diện (GUI/CLI).
- Triển khai và kiểm thử: Chi tiết quá trình thực thi các module, cấu hình tham số, và đánh giá kết quả trên dữ liệu test.
- Kết luận và hướng phát triển: Tóm tắt kết quả đạt được, những hạn chế hiện tại, và các đề xuất cải thiện trong tương lai.

2 Cơ sở lý thuyết

2.1 Kỹ thuật xử lý tín hiệu số

Trong xử lý tín hiệu số (DSP), việc thiết kế bộ lọc thường trải qua hai giai đoạn chính: xác định các thông số kỹ thuật trong miền tần số và lựa chọn hàm xấp xỉ phù hợp. Mục này trình bày cơ sở lý thuyết của bộ lọc thông dải (Bandpass Filter) sử dụng phương pháp xấp xỉ Butterworth và cách hiện thực hóa trong miền số dưới dạng bộ lọc đáp ứng xung vô hạn (IIR).

2.1.1 Lý thuyết Bộ lọc Thông dải (Bandpass Filter)

Bộ lọc thông dải là một hệ thống chọn lọc tần số, cho phép các thành phần tín hiệu nằm trong dải thông (passband) đi qua với độ suy giảm tối thiểu, đồng thời nén các thành phần tín hiệu nằm trong dải chắn (stopband).
Các tham số kỹ thuật quan trọng bao gồm:

- Dải thông (Passband): Khoảng tần số từ tần số cắt thấp (f_L) đến tần số cắt cao (f_H).

- Tần số trung tâm (f_0): Tần số cộng hưởng hình học của bộ lọc, được xác định bởi:

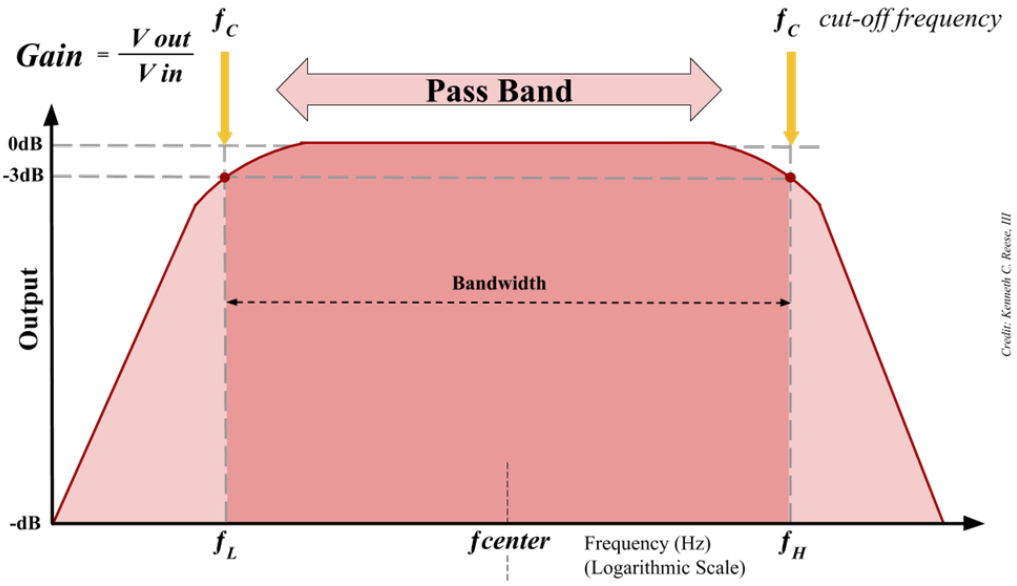
$$f_0 = \sqrt{f_L \cdot f_H} \tag{2.1}$$

- $\omega_0 = 2\pi f_0$ là tần số góc trung tâm.
- Băng thông (BW - Bandwidth): Độ rộng của dải tần số tại điểm suy giảm -3dB:

$$BW = f_H - f_L \tag{2.2}$$

- Hệ số phẩm chất (Q - Quality Factor): Đại lượng đặc trưng cho độ chọn lọc của bộ lọc. Giá trị Q càng cao, băng thông càng hẹp và bộ lọc càng chọn lọc:

$$Q = \frac{f_0}{BW} \tag{2.3}$$



Hình 2.1: Đáp ứng tần số của bộ lọc thông dải (Bandpass)

2.1.2 Xấp xỉ Butterworth

Để xác định các hệ số của đa thức trong hàm truyền đạt, phương pháp xấp xỉ Butterworth thường được sử dụng nhờ đặc tính *phẳng tối đa (maximally flat)* trong dải thông.
Đặc điểm của bộ lọc Butterworth bậc n:

- Đáp ứng biên độ: Không có gợn sóng (ripple) trong dải thông và suy giảm đơn điệu (monotonic) trong dải chắn.
- Bình phương biên độ: Được xác định bởi công thức:

$$|H(j\omega)|^2 = \frac{1}{1+\left(\frac{\omega}{\omega_c}\right)^{2n}} \tag{2.4}$$

- Độ dốc (Roll-off): Tại vùng chuyển tiếp, độ dốc suy giảm là $-20n$ dB/decade. Butterworth giúp bảo toàn dạng sóng của tín hiệu tốt hơn trong miền thời gian.
- Phân bố cực: Các cực (poles) của hàm truyền đạt $H(s)$ nằm đều nhau trên nửa mặt phẳng trái của vòng tròn đơn vị trong mặt phẳng phức s.

2.1.3 Hiện thực hóa trong miền số (Digital Implementation)

Trong các hệ thống nhúng và phần mềm, bộ lọc Butterworth được hiện thực hóa dưới dạng bộ lọc số *IIR (Infinite Impulse Response)*. Quá trình thiết kế thường sử dụng phương pháp *Biến đổi song tuyến (Bilinear Transform)* để chuyển đổi từ miền liên tục s sang miền rời rạc z:

$$s = \frac{2}{T}\frac{1-z^{-1}}{1+z^{-1}} \tag{2.5}$$

Trong đó T là chu kỳ lấy mẫu.
Sau khi biến đổi, hàm truyền đạt trong miền z có dạng phân thức hữu tỷ:

$$H(z) = \frac{B(z)}{A(z)} = \frac{b_0 + b_1 z^{-1} + \cdots + b_N z^{-N}}{1 + a_1 z^{-1} + \cdots + a_N z^{-N}} \tag{2.6}$$

Từ đó, ta có phương trình sai phân (Difference Equation) dùng để lập trình bộ lọc:

$$y[n] = \sum_{i=0}^{N} b_i x[n-i] - \sum_{j=1}^{N} a_j y[n-j] \tag{2.7}$$

Trong đó:

- $x[n], y[n]$: Tín hiệu đầu vào và đầu ra tại thời điểm n.
- b_i: Các hệ số feedforward (tử số).
- a_j: Các hệ số feedback (mẫu số), tạo nên đặc tính hồi quy của bộ lọc IIR.

2.2 Huấn luyện mô hình nhận diện âm thanh

Mô hình phân loại âm thanh mà nhóm thực hiện được xây dựng dựa trên kiến trúc CNN (Convolutional Neural Network), xử lý log-mel spectrogram của âm thanh để trích xuất các đặc trưng thời gian - tần số cho mô hình học. Quá trình huấn luyện có kết hợp các kỹ thuật tăng cường dữ liệu (data augmentation), cơ chế regularization và chiến lược tối ưu hóa để đạt được độ chính xác cao và khả năng tổng quát hóa tốt trên dữ liệu thực tế.

2.2.1 Biểu diễn đặc trưng Log-Mel Spectrogram

Tín hiệu âm thanh dạng sóng được chuyển đổi qua miền thời gian - tần số nhờ biến đổi *Short-Time Fourier Transform (STFT)*. STFT chia tín hiệu thành các khung thời gian ngắn chồng lấn (với độ dài cửa sổ `n_fft` và bước nhảy `hop_length`), sau đó áp dụng biến đổi Fourier trên từng khung. Kết quả là một spectrogram biểu diễn năng lượng của tín hiệu trong miền thời gian - tần số.

Để mô phỏng cảm nhận của tai người, spectrogram được ánh xạ sang thang Mel thông qua *Mel filter bank*. Thang Mel là một thang tần số phi tuyến, phản ánh đặc tính nhận thức âm thanh của con người với độ phân giải cao ở vùng tần số thấp và độ phân giải thấp hơn ở vùng tần số cao:

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{2.8}$$

Trong đó m là tần số Mel và f là tần số Hertz.

Sau khi áp dụng Mel filter bank với `n_mels` bộ lọc (thường là 64), ta thu được Mel spectrogram. Để nén dải động và tăng khả năng phân biệt các thành phần âm thanh nhỏ, biến đổi logarit được áp dụng theo đơn vị decibel (dB):

$$S_{\text{dB}} = 10 \log_{10}\left(\frac{S}{S_{\text{ref}}}\right) \tag{2.9}$$

Trong đó S là năng lượng Mel spectrogram và S_{ref} là giá trị tham chiếu (thường là giá trị lớn nhất).

Đầu vào cuối cùng của mô hình là log-mel spectrogram với kích thước cố định $(128 \times 64 \times 1)$, tương ứng với 128 bước thời gian, 64 kênh Mel và 1 kênh màu (grayscale). Cấu trúc này phù hợp để áp dụng các lớp tích chập 2D (Conv2D) nhằm học các mẫu tần số và biến thiên theo thời gian.

2.2.2 Chuẩn hóa dữ liệu (Data Normalization)

Sau khi tính toán log-mel spectrogram, dữ liệu được chuẩn hóa để đưa tất cả mẫu vào cùng một không gian giá trị. Chuẩn hóa là bước quan trọng vì:

- Âm thanh thu từ các nguồn khác nhau có biên độ khác nhau (do mục tiêu ghi âm, vị trí microphone, ...).
- CNN hoạt động tốt hơn khi đầu vào nằm trong một khoảng giá trị nhất quán, thường là $[0,1]$ hoặc $[-1,1]$.

Chuẩn hóa min-max được áp dụng trên từng mẫu:

$$S_{\text{norm}} = \frac{S_{\text{dB}} - S_{\min}}{S_{\max} - S_{\min} + \varepsilon} \tag{2.10}$$

Trong đó $S_{\min}$ và $S_{\max}$ là giá trị nhỏ nhất và lớn nhất của log-mel spectrogram, ε là một hằng số nhỏ để tránh chia cho 0. Kết quả là spectrogram chuẩn hóa nằm trong khoảng $[0,1]$, sẵn sàng để đầu vào cho CNN.

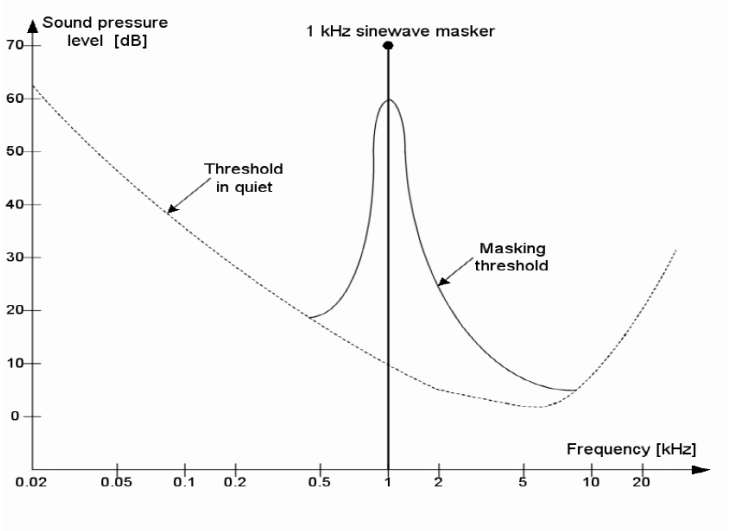
2.2.3 Data Augmentation để tăng khả năng tổng quát hóa

Để giảm hiện tượng overfitting và tăng độ bền vững (robustness) của mô hình với các điều kiện môi trường thực tế, dữ liệu huấn luyện được tăng cường thông qua nhiều kỹ thuật biến đổi. Mục tiêu là tạo ra các biến thể hợp lệ của cùng một nhãn, giúp mô hình học được các đặc trưng ổn định. Điều này đặc biệt quan trọng với bài toán nhận diện âm thanh môi trường, vì cùng một loại âm thanh có thể được ghi âm ở các tốc độ, âm lượng, cao độ khác nhau tùy theo ngữ cảnh.
Biến đổi trên waveform:

- Time Shift: Dịch chuyển tín hiệu theo trục thời gian một khoảng ngẫu nhiên, mô phỏng sự chênh lệch thời điểm bắt đầu của âm thanh.
- Gain: Thay đổi biên độ (âm lượng) của tín hiệu.
- Pitch Shift: Thay đổi cao độ (pitch) mà không làm thay đổi tốc độ âm thanh, giúp mô hình nhận diện các âm thanh tương tự ở các tần số khác nhau.
- Time Stretch: Thay đổi tốc độ phát mà không làm thay đổi cao độ, mô phỏng sự khác biệt về nhịp độ.

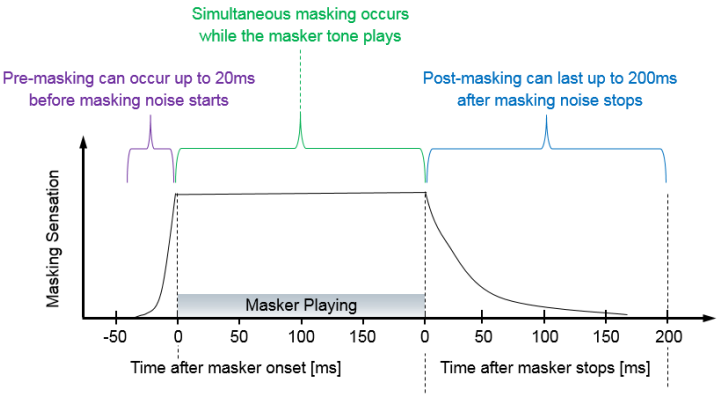
Trộn nhiễu môi trường: Thêm nhiễu nền (background noise) vào tín hiệu gốc với Signal-to-Noise Ratio (SNR) được kiểm soát. Kỹ thuật này giúp mô hình học cách phân biệt tín hiệu mục tiêu khỏi nhiễu môi trường, cải thiện khả năng hoạt động trong điều kiện thực tế.
SpecAugment trên spectrogram: SpecAugment là kỹ thuật augmentation trực tiếp trên biểu diễn log-mel spectrogram, bao gồm hai loại masking:

- Frequency Masking: Che (mask) các kênh tần số liên tiếp bằng cách đặt giá trị của chúng về 0, mô phỏng sự mất mát thông tin tần số. Số lượng kênh bị che tối đa là `freq_mask_max`.



Hình 2.2: Minh họa Frequency Masking: các kênh tần số liên tiếp bị che.

- Time Masking: Che các bước thời gian liên tiếp, mô phỏng sự gián đoạn tạm thời trong tín hiệu. Số lượng bước thời gian bị che tối đa là `time_mask_max`.



Hình 2.3: Minh họa Time Masking: các bước thời gian liên tiếp bị che.

SpecAugment được áp dụng với xác suất p (thường là 0.9) trong quá trình huấn luyện, giúp mô hình học được các đặc trưng bất biến với các biến đổi cục bộ trong miền thời gian và tần số. Kỹ thuật này tương tự như "che mắt"một phần của spectrogram, buộc mô hình phải học từ thông tin còn lại để nhận diện âm thanh. Điều này mô phỏng tình huống thực tế khi có nhiễu hoặc tín hiệu bị cắt đứt một phần.

2.2.4 Kiến trúc CNN cho phân loại âm thanh

Vì log-mel spectrogram có cấu trúc tương tự ảnh hai chiều, mô hình sử dụng các lớp tích chập 2D (`Conv2D`) để trích xuất đặc trưng không gian. Sự lựa chọn này là phù hợp vì spectrogram chứa các mẫu tần số (patterns) như ranh giới giữa vùng tần số cao/thấp, và CNN đã được chứng minh hiệu quả trong việc nhận diện các mẫu 2D từ ảnh và spectrogram. Kiến trúc mạng được thiết kế theo khối (block), với mỗi khối gồm:

- Hai lớp Conv2D: Sử dụng kernel kích thước (3×3) và padding `same` để giữ nguyên kích thước không gian. Số lượng bộ lọc tăng dần qua các khối ($32 \to 64 \to 128 \to 256$) để học các đặc trưng từ thô đến tinh vi.
- Batch Normalization: Chuẩn hóa đầu ra của mỗi lớp tích chập, giúp ổn định quá trình huấn luyện và tăng tốc độ hội tụ.
- MaxPooling2D: Giảm kích thước không gian với kernel (2×2), giúp giảm số tham số.
- Dropout: Tắt ngẫu nhiên một tỉ lệ neuron để giảm overfitting.

Sau các khối tích chập, mô hình sử dụng **Global Average Pooling (GAP)** thay cho fully connected layer. GAP tính trung bình của mỗi feature map trên toàn bộ không gian, giảm mạnh số tham số và giảm nguy cơ overfitting. Cuối cùng, một lớp `Dense` với hàm kích hoạt `softmax` tạo ra phân phối xác suất trên N lớp:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{N} e^{z_j}} \tag{2.11}$$

2.2.5 Hàm mất mát Cross-Entropy và Label Smoothing

Bài toán phân loại đa lớp sử dụng hàm mất mát *Categorical Cross-Entropy* để đo sai lệch giữa phân phối xác suất dự đoán $\hat{y}$ và phân phối nhãn thật y:

$$\mathscr{L}_{\text{CE}} = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{2.12}$$

Trong đó y_i là nhãn one-hot (1 cho lớp đúng, 0 cho các lớp khác) và $\hat{y}_i$ là xác suất dự đoán của lớp i.

Để giảm hiện tượng mô hình quá tự tin (overconfident) và cải thiện khả năng tổng quát hóa, *Label Smoothing* được áp dụng. Thay vì sử dụng nhãn one-hot cứng, label smoothing làm mềm nhãn bằng cách phân phối một lượng nhỏ xác suất cho các lớp khác:

$$y_i' = \begin{cases} 1 - \varepsilon + \frac{\varepsilon}{N} & \text{nếu } i = \text{lớp đúng} \\ \frac{\varepsilon}{N} & \text{ngược lại} \end{cases} \tag{2.13}$$

Trong đó ε là tham số smoothing (thường là 0.05), N là số lớp. Label smoothing giúp mô hình ít bị phụ thuộc vào các mẫu nhiễu và cải thiện hiệu suất trên dữ liệu validation/test.

2.2.6 Tối ưu hóa và chiến lược học

Mô hình được tối ưu hóa bằng thuật toán **Adam (Adaptive Moment Estimation)**, một biến thể của Stochastic Gradient Descent kết hợp momentum và adaptive learning rate. Adam điều chỉnh learning rate riêng cho từng tham số dựa trên ước lượng moment bậc nhất và bậc hai của gradient:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) g_t \tag{2.14}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) g_t^2 \tag{2.15}$$

$$\theta_t = \theta_{t-1} - \frac{\alpha}{\sqrt{\hat{v}_t} + \varepsilon} \hat{m}_t \tag{2.16}$$

Trong đó g_t là gradient tại bước t, m_t và v_t là moment bậc nhất và bậc hai, α là learning rate ban đầu (thường là 10^{-3}), β_1 và β_2 là hệ số decay (thường là 0.9 và 0.999).

Để tránh việc kẹt ở các plateau (vùng phẳng trong không gian loss), **ReduceLROnPlateau** được sử dụng để giảm learning rate khi validation loss không cải thiện sau một số epoch nhất định (patience). Learning rate được giảm theo hệ số (factor) và có ngưỡng tối thiểu (min_lr) để tránh learning rate quá nhỏ.

Early Stopping dừng quá trình huấn luyện khi validation loss không cải thiện sau một số epoch (patience lớn hơn, ví dụ 12), đồng thời khôi phục trọng số tốt nhất (restore_best_weights) để tránh overfitting.

Model Checkpoint lưu mô hình tốt nhất dựa trên validation loss, đảm bảo không mất mô hình có hiệu suất cao nhất trong quá trình huấn luyện.

2.2.7 Mất cân bằng lớp và Class Weight

Trong thực tế, tập dữ liệu thường có sự mất cân bằng giữa các lớp (class imbalance), với một số lớp có ít mẫu hơn hoặc khó học hơn. Để giải quyết vấn đề này, *class weight* được áp dụng để tăng trọng số loss của các lớp yếu/ít mẫu.
Class weight được tính dựa trên tần suất xuất hiện ngược (inverse frequency):

$$w_i = \frac{\bar{n}}{n_i + \varepsilon} \tag{2.17}$$

Trong đó w_i là trọng số của lớp i, n_i là số mẫu của lớp i, $\bar{n}$ là số mẫu trung bình và ε là hằng số nhỏ để tránh chia cho 0.
Đối với các lớp đặc biệt khó (ví dụ "engine", "rooster", "dog"), một hệ số boost bổ sung có thể được áp dụng để tăng thêm trọng số:

$$w_i' = w_i \times \text{boost}_i \tag{2.18}$$

Loss có trọng số được tính như sau:

$$\mathscr{L}_{\text{weighted}} = \sum_{i=1}^{N} w_i \cdot \mathscr{L}_i \tag{2.19}$$

Kỹ thuật này giúp mô hình chú ý nhiều hơn vào các lớp khó và cải thiện khả năng nhận diện đồng đều trên tất cả các lớp.

2.2.8 Chiến lược huấn luyện và đánh giá

Dữ liệu được chia thành ba tập: *train*, *validation* và *test* theo tỉ lệ 8:1:1. Tập train được sử dụng để cập nhật trọng số, tập validation để điều chỉnh hyperparameter và early stopping, và tập test để đánh giá hiệu suất cuối cùng trên dữ liệu chưa thấy.
Ngoài tập test sạch, một tập *test_noisy* chứa dữ liệu có nhiễu môi trường được sử dụng để đánh giá độ bền vững của mô hình trong điều kiện thực tế.
Các metric đánh giá bao gồm:

- Accuracy: Tỉ lệ dự đoán đúng trên tổng số mẫu.
- Precision: Tỉ lệ dự đoán đúng trong số các mẫu được dự đoán là dương tính cho mỗi lớp.
- Recall: Tỉ lệ mẫu dương tính thực sự được dự đoán đúng.

- F1-score: Trung bình điều hòa của Precision và Recall, cân bằng giữa hai metric này:

$$F1 = 2 \cdot \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{2.20}$$

- Confusion Matrix: Ma trận thể hiện số lượng dự đoán đúng/sai giữa các lớp, giúp phân tích các loại lỗi cụ thể.

Macro F1-score (trung bình F1-score của tất cả các lớp) thường được sử dụng làm metric chính để đánh giá hiệu suất tổng thể, đặc biệt khi dữ liệu có mất cân bằng lớp.

2.2.9 Ngưỡng tin cậy cho lớp Unknown

Đối với hệ thống thực tế, việc phát hiện các âm thanh ngoài tập huấn luyện (out-of-distribution) là rất quan trọng. Để giải quyết vấn đề này, một lớp đặc biệt "unknown" được thêm vào tập dữ liệu, chứa các âm thanh không thuộc các lớp mục tiêu.
Trong quá trình suy luận, một ngưỡng tin cậy (confidence threshold) τ được áp dụng: nếu xác suất dự đoán cao nhất nhỏ hơn τ, mẫu được gán nhãn "unknown". Giá trị τ tối ưu được xác định bằng cách tìm kiếm trên tập noisy để tối đa hóa macro F1-score:

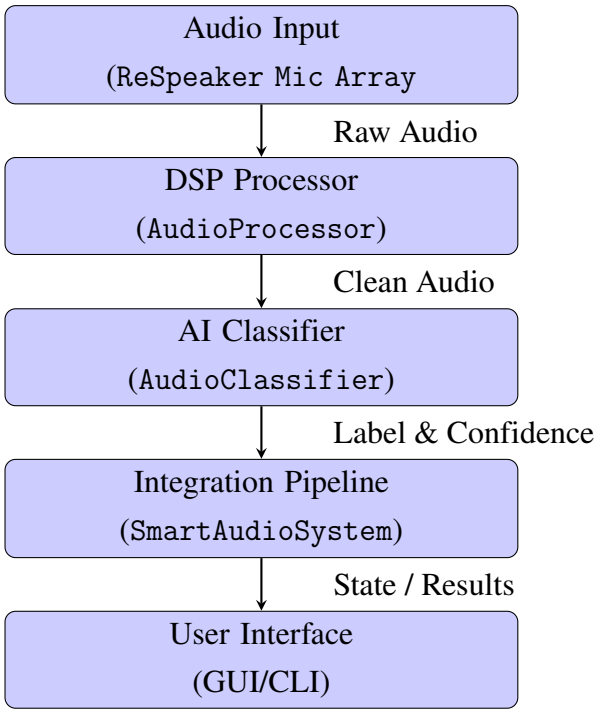
$$\tau^* = \arg\max_{\tau} \text{F1}_{\text{macro}}(\tau) \tag{2.21}$$

Kỹ thuật này giúp hệ thống tránh đưa ra dự đoán sai với độ tin cậy cao cho các âm thanh không mong đợi, tăng độ tin cậy và an toàn trong ứng dụng thực tế.

3 System Architecture

3.1 Tổng quan kiến trúc hệ thống

Hệ thống được thiết kế theo kiến trúc *pipeline* thời gian thực, trong đó âm thanh được xử lý theo từng khung (chunk) cố định và đi qua các khối chức năng theo một chiều. Mục tiêu của kiến trúc này là tách biệt rõ ràng trách nhiệm giữa các lớp: thu nhận tín hiệu, cải thiện chất lượng tín hiệu, suy luận AI và hiển thị kết quả. Nhờ đó, hệ thống vừa đảm bảo độ trễ thấp (phù hợp realtime) vừa dễ mở rộng/thay thế từng thành phần mà không ảnh hưởng toàn bộ.
Luồng xử lý tổng thể gồm 5 thành phần chính: (i) thu âm (Audio Input), (ii) tiền xử lý DSP, (iii) phân loại AI, (iv) pipeline tích hợp (*orchestrator*) và (v) tầng hiển thị (GUI/CLI). Sơ đồ tổng quan được trình bày trong Hình 3.1.



Hình 3.1: Luồng xử lý tổng quan của hệ thống theo pipeline

3.2 Module phần cứng: SoundDetector (USB VAD/DOA)

3.2.1 Mô tả chức năng

`SoundDetector` chịu trách nhiệm giao tiếp với ReSpeaker Mic Array v2.0 qua USB (control transfer) để truy xuất các tín hiệu/trạng thái đã được phần cứng xử lý sẵn. Các chức năng chính gồm:

- **VAD (Voice Activity Detection)**: trạng thái phát hiện giọng nói do phần cứng cung cấp; có thể dùng để đánh dấu đoạn có tiếng nói hoặc hỗ trợ hiển thị.
- **DOA (Direction of Arrival)**: ước lượng hướng nguồn âm trong miền 0–359°, phục vụ quan sát trực quan (radar/compass) và debug hành vi thu âm.
- **Control/Status**: đọc/ghi một số tham số vận hành (tùy firmware) để kiểm tra trạng thái thiết bị hoặc hiệu chỉnh khi cần.

3.2.2 Giao tiếp USB

Module sử dụng thư viện `pyusb` để thực hiện control transfer. Ví dụ minh họa thao tác đọc tham số DOA từ thiết bị:

```
response = dev.ctrl_transfer(
    bmRequestType=0xC0,  # CTRL_IN | VENDOR | DEVICE
    bRequest=0x00,
    wValue=0,
    wIndex=21,           # DOA parameter ID (depends on firmware)
    data_or_wLength=8
)
direction = int.from_bytes(response, byteorder='little')
```

Listing 1: Ví dụ đọc DOA bằng USB control transfer

3.3 Module DSP Processor: AudioProcessor

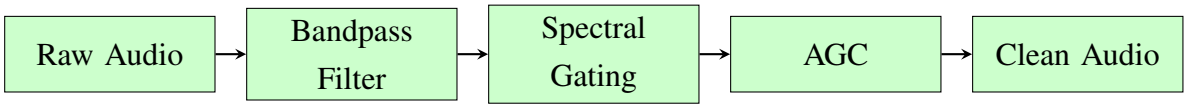
3.3.1 Mô tả chức năng

`AudioProcessor` thực hiện tiền xử lý tín hiệu nhằm cải thiện chất lượng đầu vào cho mô hình AI. Trong thực tế, tín hiệu thu được từ microphone có thể bị nhiễu nền, biến thiên biên độ hoặc chứa thành phần tần số không hữu ích. Do đó, DSP được áp dụng để: (i) loại bỏ phần nhiễu ngoài dải quan tâm, (ii) giảm nhiễu nền và (iii) ổn định mức âm lượng để mô hình AI hoạt động nhất quán hơn. Pipeline DSP trong hệ thống gồm ba bước chính:

1. **Bandpass Filter**: giới hạn dải tần hữu ích (100–7500 Hz) để giảm rumble tần thấp và hiss tần cao.
2. **Spectral Gating**: giảm nhiễu nền trong miền tần số, giúp tăng tỉ lệ tín hiệu/nhiễu trong bối cảnh môi trường có noise.
3. **AGC (Automatic Gain Control)**: ổn định biên độ (RMS) theo mục tiêu, đồng thời có ngưỡng gate để tránh khuếch đại nhiễu khi gần im lặng.

3.3.2 Pipeline xử lý

Hình 3.2 minh họa thứ tự xử lý của pipeline DSP.



Hình 3.2: Pipeline xử lý DSP trước khi đưa tín hiệu vào mô hình AI.

3.4 Module AI Classifier: AudioClassifier

3.4.1 Mô tả chức năng

`AudioClassifier` thực hiện phân loại âm thanh môi trường theo hướng *AI-only*. Về mặt luồng xử lý, module nhận *clean audio* từ DSP và tạo ra kết quả dự đoán dạng *label* và *confidence*. Để mô hình hoạt động ổn định trong thời gian thực, hệ thống sử dụng các cơ chế theo thời gian (temporal logic) thay vì suy luận độc lập trên từng chunk nhỏ.
Các nhiệm vụ chính của module gồm:

- **Feature Extraction**: chuyển tín hiệu audio đã qua DSP sang biểu diễn *Log-Mel Spectrogram*.
- **Buffering/Windowing**: tích lũy dữ liệu theo cửa sổ ngắn (ví dụ vài giây) để tạo input đủ "già" cho mô hình.
- **Inference**: chạy mô hình CNN để dự đoán phân phối xác suất theo lớp, sau đó lấy nhãn top-1 và độ tin cậy.
- **Temporal Post-processing**: làm mượt dự đoán theo thời gian và xử lý trường hợp im lặng nhằm giảm jitter của nhãn.

3.4.2 Post-processing và ổn định theo thời gian

Để tăng độ ổn định dự đoán trong thời gian thực, hệ thống áp dụng các kỹ thuật sau:

- **Smoothing**: trung bình hóa/xử lý tích lũy dự đoán trên K bước gần nhất nhằm giảm nhảy nhãn do nhiễu hoặc dao động nhỏ của tín hiệu.
- **Confidence threshold**: nếu confidence dưới ngưỡng, hệ thống trả về `unknown` để hạn chế dự đoán ngẫu nhiên.

- **Fast-switch**: cho phép chuyển nhãn nhanh khi lớp mới có confidence cao và xuất hiện liên tiếp, giúp phản ứng tốt với sự kiện âm thanh rõ ràng.
- **Reset on silence**: khi RMS thấp liên tục vượt quá một khoảng thời gian, buffer/lịch sử sẽ được reset để tránh "dính" nhãn cũ.

3.5 Module Integration Pipeline: SmartAudioSystem

3.5.1 Mô tả chức năng

`SmartAudioSystem` đóng vai trò *orchestrator*, kết nối các module xử lý (DSP) và suy luận (AI) thành một pipeline end-to-end. Module này chuẩn hóa đầu ra dưới dạng một *state* thống nhất (ví dụ: RMS, gain, label, confidence), từ đó GUI và CLI chỉ cần đọc state để hiển thị mà không phụ thuộc vào chi tiết nội bộ của từng khối.

3.5.2 Quản lý luồng xử lý

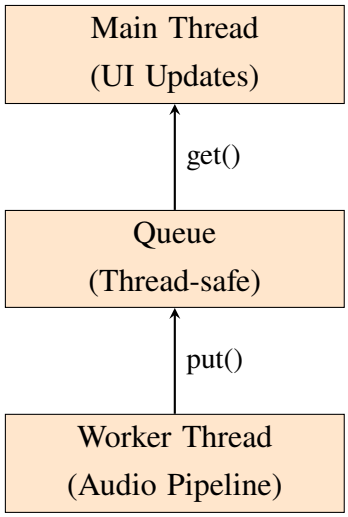
Pipeline hoạt động theo vòng lặp liên tục:

1. Thu audio theo chunk (ví dụ 1024 samples $\approx$ 64ms ở 16kHz).
2. Tiền xử lý DSP (Bandpass $\rightarrow$ Spectral Gate $\rightarrow$ AGC).
3. Cập nhật buffer/window và suy luận AI theo cơ chế *hop* (không nhất thiết suy luận mỗi chunk).
4. Đóng gói kết quả thành state kèm metadata (RMS, gain, label/confidence).
5. Chuyển state đến tầng hiển thị (GUI/CLI).

3.6 Module User Interface

3.6.1 GUI

GUI được xây dựng bằng Tkinter theo mô hình đa luồng để đảm bảo giao diện luôn responsive. Do Tkinter không thread-safe, toàn bộ cập nhật UI chỉ thực hiện trên main thread. Worker thread đảm nhiệm chạy pipeline và đẩy state sang main thread thông qua queue an toàn luồng (Hình 3.3).



Hình 3.3: Mô hình threading cho GUI

Các thành phần giao diện có thể bao gồm:

- **DSP Panel**: hiển thị RMS đầu vào/đầu ra và hệ số gain của AGC.
- **AI Panel**: hiển thị nhãn dự đoán và confidence.
- **Radar/Direction Panel (tuỳ chọn)**: hiển thị DOA nếu bật `SoundDetector`.
- **Event Logs**: lưu các sự kiện theo thời gian để quan sát hành vi hệ thống.

3.6.2 CLI

CLI được xây dựng bằng thư viện Rich nhằm hỗ trợ kiểm thử nhanh và theo dõi realtime. CLI được thiết kế như một tầng mỏng (thin layer), tái sử dụng cùng pipeline với GUI để đảm bảo đầu ra nhất quán, đồng thời hỗ trợ quan sát nhanh các chỉ số quan trọng (RMS/gain/label/confidence) mà không cần chạy giao diện đồ họa.

4 Implementation

4.1 Cấu trúc mã nguồn

4.1.1 Tổng quan tổ chức code

Hệ thống được tổ chức theo cấu trúc modular, trong đó mỗi module đảm nhận một trách nhiệm cụ thể. Cấu trúc thư mục được thiết kế để tối ưu hóa khả năng bảo trì, mở rộng và tái sử dụng code.

```
Sound-detect-Service/
|
+-- Core Components:
|   +-- audio_classifier.py
|   +-- audio_processor.py
|   +-- sound_detector.py
|   +-- smart_audio_pipeline.py
|
+-- User Interface:
|   +-- gui_app.py
|   +-- cli.py
|
+-- Configuration:
|   +-- config.py
|
+-- AI Model:
    +-- audio_cnn_best.h5 (11 classes)
    +-- audio_classifier_model.ipynb
```

Listing 2: Cấu trúc thư mục dự án

4.1.2 Chi tiết các module chính

Bảng 4.1 mô tả chi tiết chức năng của từng file trong hệ thống.

File	Chức năng
`sound_detector.py`	Giao tiếp USB với ReSpeaker, đọc VAD/DOA, quản lý kết nối hardware
`audio_processor.py`	DSP pipeline: Bandpass Filter, Spectral Gating, AGC với stateful processing
`audio_classifier.py`	AI classifier: Feature extraction, CNN inference, temporal smoothing, buffering
`smart_audio_pipeline.py`	Integration orchestrator: kết nối DSP + AI, chuẩn hóa output, lifecycle management
`gui_app.py`	Tkinter GUI với threading model, queue-based communication, real-time visualization
`cli.py`	Rich CLI với commands (start, status, test), terminal-based monitoring
`config.py`	Configuration constants: thresholds, paths, hyperparameters

Bảng 4.1: Chi tiết các module chính trong hệ thống

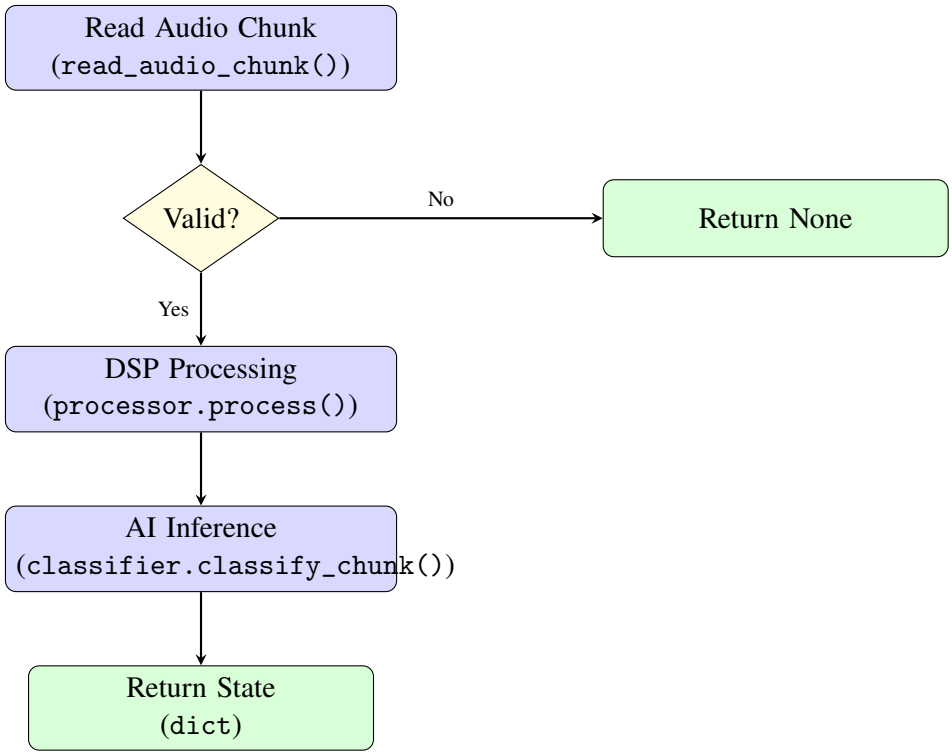
4.2 Pipeline tích hợp trong `smart_audio_pipeline.py`

File `smart_audio_pipeline.py` định nghĩa lớp `SmartAudioSystem` như một lớp *orchestrator*, có nhiệm vụ ghép chuỗi xử lý thời gian thực thành một API thống nhất.
Mỗi lần gọi, hệ thống đọc một chunk âm thanh, chạy DSP, suy luận AI và trả về một *state* chuẩn hóa.
Nhờ đó, GUI/CLI chỉ làm nhiệm vụ hiển thị và điều khiển, không cần phụ thuộc vào chi tiết nội bộ của DSP/AI.

4.2.1 Luồng xử lý trong `process_and_predict()`

Hình 4.1 minh họa luồng xử lý chính của hàm `process_and_predict()`.
Luồng xử lý có thể tóm tắt như sau:

1. **Đọc chunk**: lấy `raw_chunk` từ audio stream. Nếu không có dữ liệu hợp lệ, trả về `None`.
2. **DSP**: xử lý `raw_chunk` để tạo `clean_chunk` và
   cập nhật một số chỉ số giám sát (RMS/gain).



Hình 4.1: Luồng xử lý trong hàm `process_and_predict()`.

3. **AI**: suy luận trên `clean_chunk` và tạo kết quả `label/confidence`

   (có thể chạy theo cơ chế hop/window).

4. **Đóng gói**: chuẩn hóa đầu ra thành state cho GUI/CLI.

4.2.2 Pseudo-code triển khai

Listing 3 minh họa pseudo-code của `process_and_predict()` (các chi tiết DSP/AI được trình bày ở các phần sau của báo cáo).

```
def process_and_predict(self):
    raw_chunk = classifier.read_audio_chunk()
    if raw_chunk is None:
        return None

    clean_chunk = processor.process(raw_chunk)
    env_label, env_conf, rms_clean = classifier.classify_chunk(clean_chunk)

    return {
        "env_label": env_label,
        "env_conf": float(env_conf),
```

```
        "rms_raw": float(rms(raw_chunk)),
        "rms_clean": float(rms_clean),
        "gain_applied": float(processor.current_gain),
        # optional: timestamp / frame_index
    }
```

Listing 3: Pseudo-code: `process_and_predict()`

4.2.3 Định dạng *state* trả về

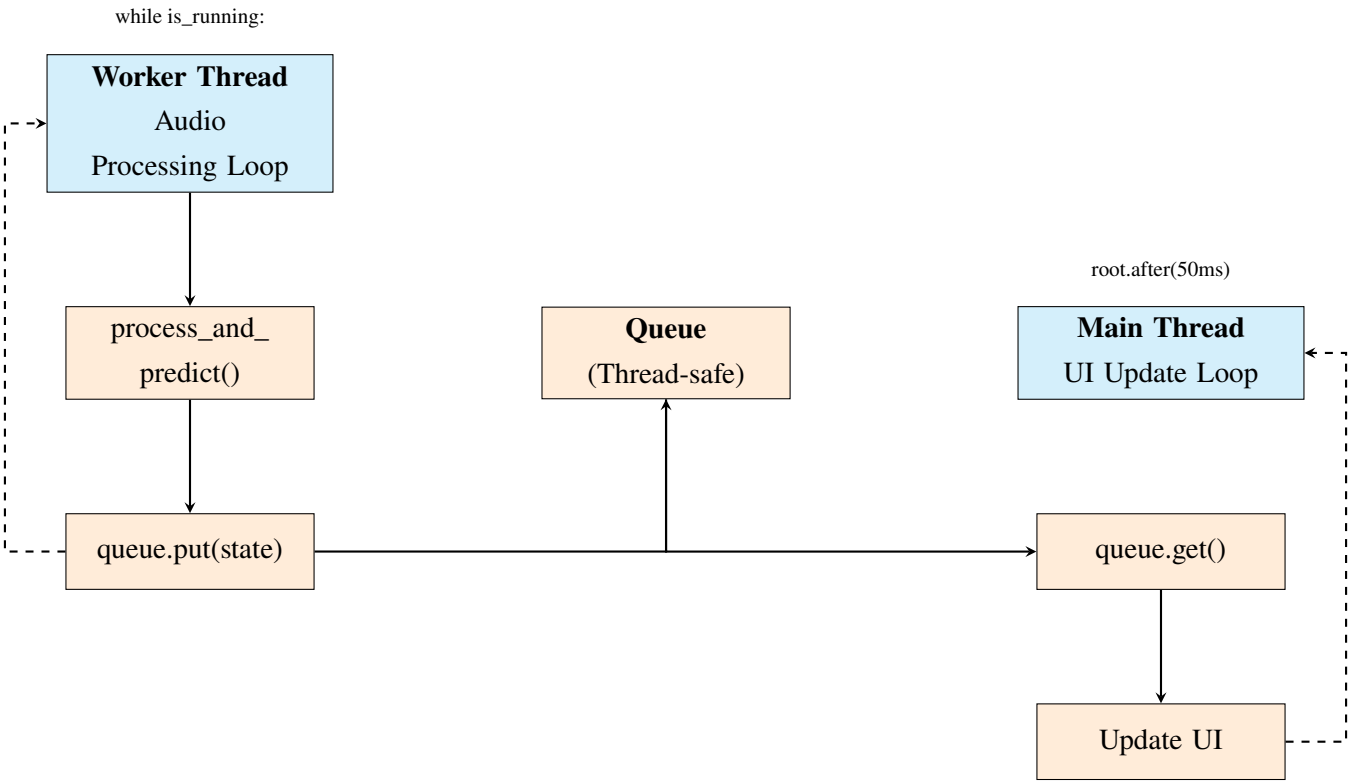
Đầu ra của pipeline được chuẩn hóa thành dictionary với các trường chính như Bảng 4.2.

Field	**Type**	**Mô tả**
`env_label`	str/None	Nhãn dự đoán; có thể `None`/`unknown` tùy logic realtime
`env_conf`	float	Độ tin cậy (0–1)
`rms_raw`	float	RMS của audio đầu vào
`rms_clean`	float	RMS sau tiền xử lý DSP
`gain_applied`	float	Gain hiện tại do AGC áp dụng

Bảng 4.2: Định dạng state dictionary trả về từ pipeline.

4.2.4 Cơ chế realtime trong GUI (thread + queue)

Hình 4.2 mô tả cơ chế chạy realtime: worker thread liên tục tạo state và đẩy vào queue; main thread định kỳ lấy state (ưu tiên state mới nhất) để cập nhật UI an toàn.



Hình 4.2: Cơ chế realtime với worker thread và queue

4.2.5 Dừng/chạy lại an toàn

Để tránh rò rỉ tài nguyên và đảm bảo có thể Start/Stop nhiều lần, hệ thống sử dụng cơ chế *stop flag/stop event* cho worker loop và thực hiện cleanup theo thứ tự. Khi Stop, main thread gửi tín hiệu dừng; worker thread thoát vòng lặp một cách graceful và đóng audio stream/giải phóng tài nguyên. Khi Start lại, hệ thống khởi tạo lại các thành phần cần thiết và khởi chạy worker thread mới với trạng thái sạch.

4.3 Hiện thực bộ xử lý âm thanh

Trong đồ án này file audio_processor.py đóng vai trò như một bộ xử lý âm thanh đầu vào. Tại đây, âm thanh thô được nhận từ respeaker sẽ được sẽ được lọc và làm sạch nhằm đảm bảo chất lượng âm thanh tốt nhất trước khi đưa vào phân loại. Dưới đây là mô tả chi tiết chức năng và giải thuật của từng phương thức thành phần.

4.3.1 Phương thức khởi tạo: `__init__`

Đây là nơi thiết lập các tham số vận hành cho hệ thống với tần số lấy mẫu $f_s = 16000$ Hz với chunk size là 1024 mẫu. Dưới đây là các tham số cho các tầng xử lý phía sau:

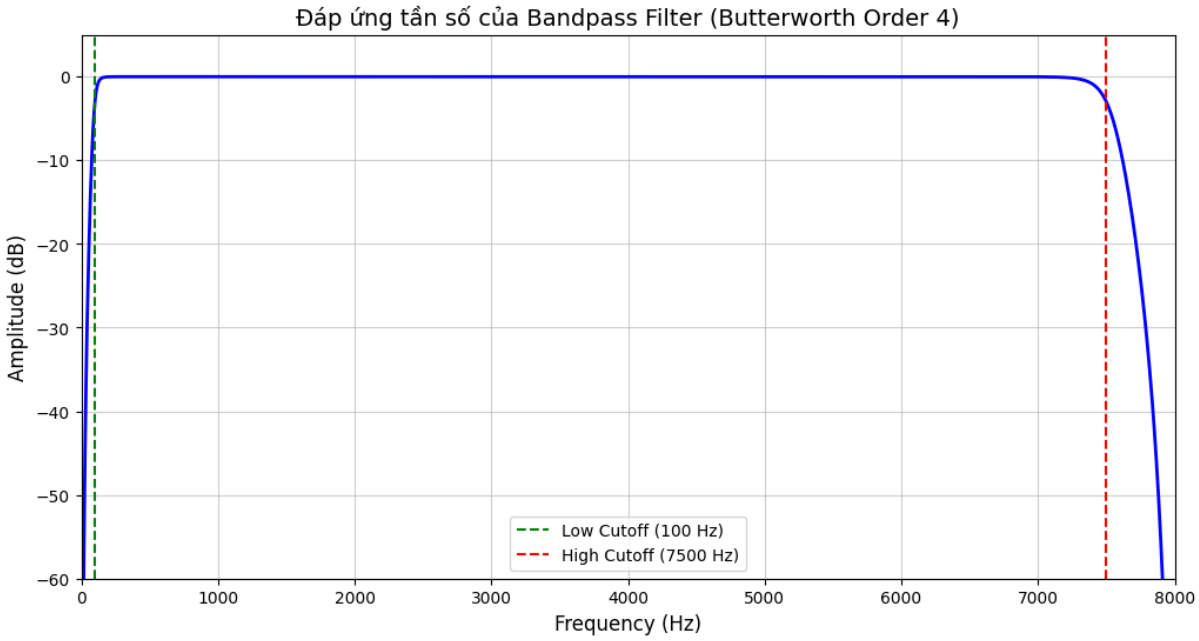
- **Bandpass Filter:** Hệ thống sử dụng bộ lọc IIR Butterworth bậc 4. Ngoài ra với mục tiêu tối ưu hóa quá trình tính toán, bộ lọc này được tách thành chuỗi các bộ lọc con bậc hai nối tiếp nhau theo cấu trúc SOS. Từ đó hạn chế tối đa các sai số làm tròn của máy tính, đảm bảo kết quả lọc chính xác và hệ thống vận hành ổn định hơn.
  - Tần số cắt thấp (f_{low}): 100 Hz.
  - Tần số cắt cao (f_{high}): 7500 Hz.
- **Spectral Gating:**
  - `spectral_threshold` $= 0.6$: Ngưỡng cắt phổ. Tham số này xác định mức độ mạnh tay khi loại bỏ các thành phần tần số yếu hơn mức sàn nhiễu ước tính.
- **AGC & Noise Gate:**
  - `agc_max_gain` $= 15.0$ dB: Giới hạn mức khuếch đại tối đa.
  - `gate_threshold` $= 0.012$: Ngưỡng biên độ RMS xác định ranh giới phân biệt giữa Tín hiệu và Nhiễu.
  - `gate_ratio` $= 0.10$: Hệ số suy giảm áp dụng cho vùng nhiễu.
  - `agc_target_rms` $= 0.12$: Mức năng lượng mục tiêu mà bộ AGC sẽ cố gắng đưa tín hiệu giọng nói đạt tới.
  - `agc_smooth` $= 0.1$: Hệ số làm mượt cho việc thay đổi Gain trong vùng tiếng nói, giúp âm lượng thay đổi tự nhiên, không bị giật cục.

4.3.2 Phương thức `apply_bandpass(chunk)`

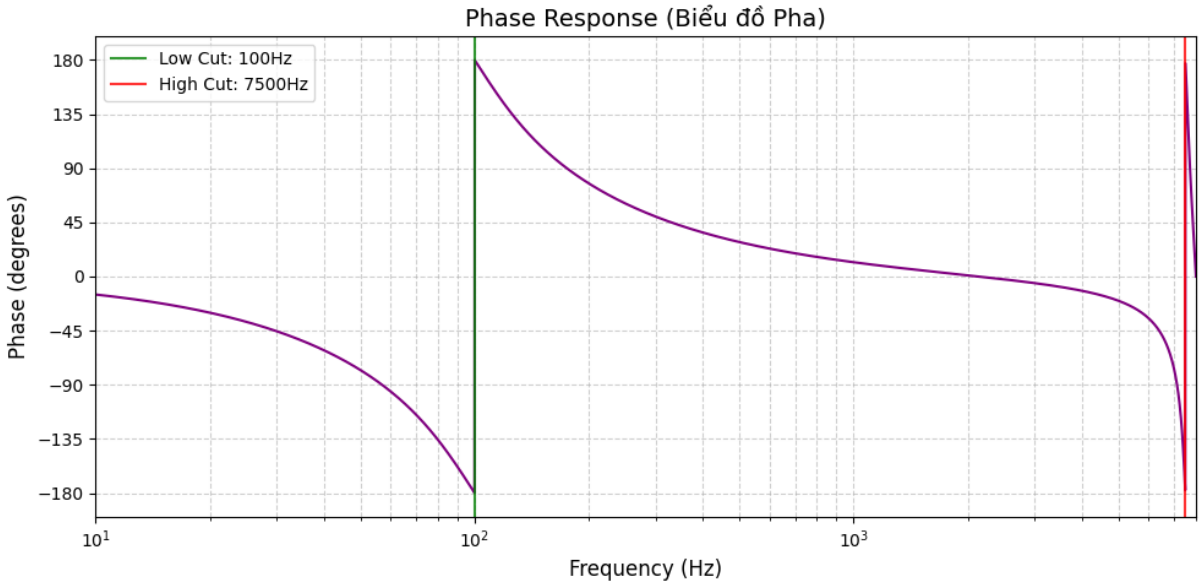
Phương thức này chịu trách nhiệm loại bỏ các thành phần tần số không mong muốn nằm ngoài dải giọng nói.

- **Input:** Mảng numpy chứa khung tín hiệu âm thanh thô.
- **Xử lý:** Sử dụng hàm `signal.sosfilt` để áp dụng bộ lọc Butterworth lên dữ liệu.
- **Lưu ý kỹ thuật:** Quan trọng nhất ở bước này là việc truyền và cập nhật tham số

- **Output:** Tín hiệu đã được lọc dải thông.



Hình 4.3: Đáp ứng tần số của bộ lọc Bandpass -Butterworth bậc 4.



Hình 4.4: Đáp ứng pha của bộ lọc.

4.3.3 Phương thức `apply_spectral_gate(chunk)`

Phương thức này thực hiện giảm nhiễu dựa trên miền tần số. Các bước thực hiện được miêu tả như dưới đây:

- **Biến đổi thuận (FFT):** Chuyển tín hiệu $x[n]$ sang miền tần số bằng hàm `np.fft.rfft`:

$$X[k] = \mathscr{F}\{x[n]\}$$

- **Tính toán ngưỡng:** Xác định mức sàn nhiễu trung bình (`noise_floor`). Ngưỡng cắt được thiết lập là:

$$T = \text{noise\_floor} \times 0.6$$

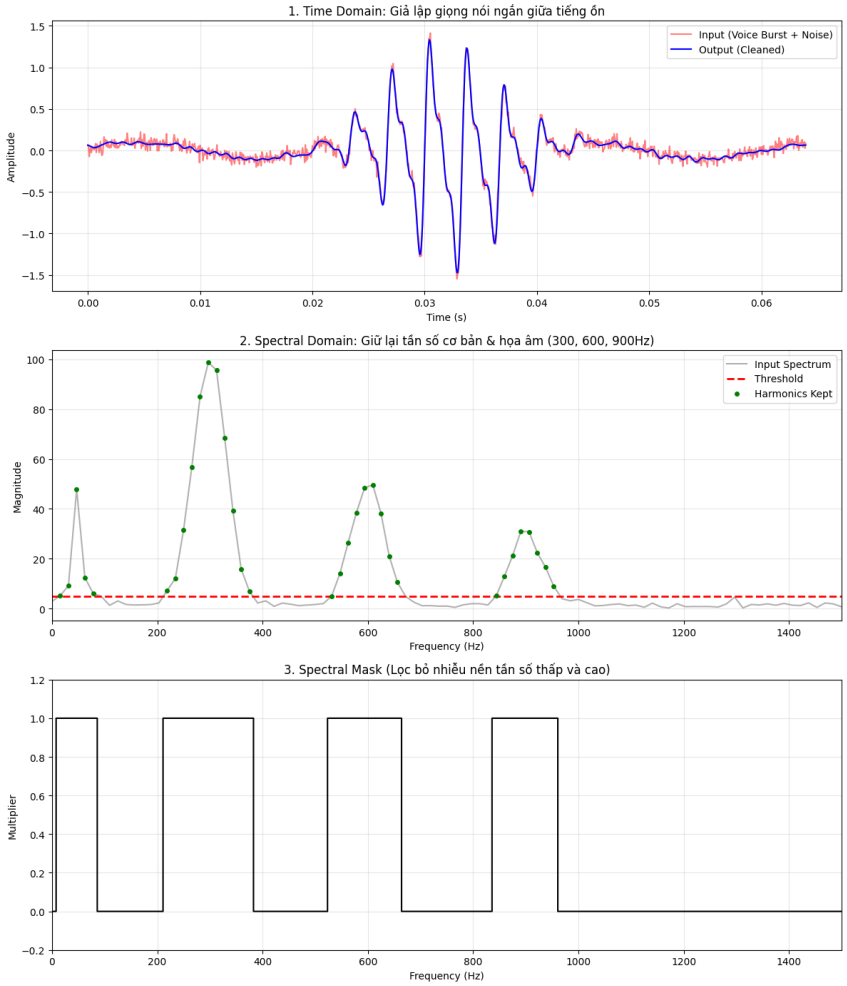
- **Masking:** Tạo mặt nạ nhị phân $M[k]$ để giữ lại hoặc loại bỏ các thành phần tần số:

$$M[k] = \begin{cases} 1 & \text{nếu } |X[k]| \geq T \\ 0 & \text{nếu } |X[k]| < T \end{cases}$$

  Tín hiệu sau lọc: $\hat{X}[k] = X[k] \cdot M[k]$.

- **Biến đổi ngược (IFFT):** Khôi phục tín hiệu về miền thời gian:

$$\hat{x}[n] = \mathscr{F}^{-1}\{\hat{X}[k]\}$$



Hình 4.5: Quy trình Spectral Gating: (1) Tín hiệu thời gian, (2) Phân tích phổ và chọn lọc họa âm, (3) Tạo mặt nạ nhị phân để loại bỏ nhiễu nền.

4.3.4 Phương thức `apply_agc(chunk)`

Đây là sự kết hợp giữa Noise Gate và AGC.

- **Tính RMS:** Đo năng lượng trung bình của khung tín hiệu.
- **Phân loại tín hiệu:**
  - Nếu $RMS <$ `gate_threshold`: Hệ thống nhận định là vùng nền. Gain mục tiêu được đặt về `gate_ratio` = 0.1 để dìm nhiễu nền xuống mức thấp nhưng không tắt hẳn.
  - Nếu $RMS \geq$ `gate_threshold`: Hệ thống nhận định là vùng âm thanh mục tiêu. Gain mục tiêu được tính toán để đưa tín hiệu về mức chuẩn 0.12 RMS. Giá trị này bị kìm hãm bởi `agc_max_gain` để tránh phóng đại tiếng ồn đột ngột.

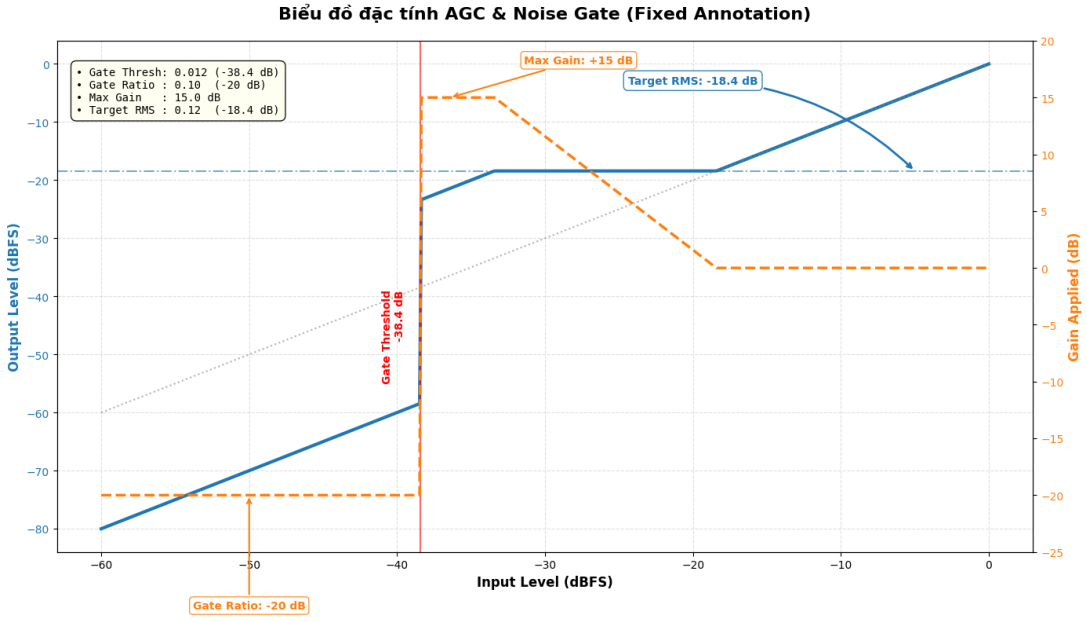
- **Làm mượt:** Để tránh tình trạng xuất hiện các âm thanh nhiễu không đáng có khi cường độ âm thanh bị thay đổi đột ngột giữa vùng nền và vùng âm thanh mục tiêu, hệ số Gain thực tế (G_{curr}) sẽ được cập nhật theo công thức Trung bình động lũy thừa:

$$G_{curr}[n] = G_{curr}[n-1] \cdot (1-S) + G_{target} \tag{4.1}$$

- **Cắt giới hạn biên độ:** Sau khi tín hiệu được nhân với hệ số Gain đã làm mượt, kết quả có thể vượt quá giới hạn biểu diễn của âm thanh kỹ thuật số. Từ đó xuất hiện rủi ro tạo ra các hài âm bậc cao rất chói tai.
  - **Giải pháp:** Hệ thống sử dụng hàm `np.clip` để ép dữ liệu vào khoảng an toàn:

$$y[i] = \max(-1.0, \min(1.0, x[i] \cdot G_{curr})) \tag{4.2}$$

  - **Ý nghĩa:** Đây là một tầng bảo vệ cuối cùng, đảm bảo rằng dù người dùng có nói sát mic hay Gain được đẩy lên cao, tín hiệu đầu ra vẫn luôn nằm trong chuẩn chuẩn hóa của hệ thống.



Hình 4.6: Đặc tuyến truyền đạt Input/Output của hệ thống AGC và Noise Gate.

4.3.5 Phương thức `process(chunk)`

Hàm điều phối chính, kết nối các bước xử lý thành một luồng tuần tự:

$$X_{raw} \xrightarrow{\text{Bandpass Filter}} X_{filt} \xrightarrow{\text{Spectral Gating}} X_{clean} \xrightarrow{\text{AGC \& Gate}} Y_{out} \tag{4.3}$$

Trong đó:

- **Tầng 1 (Bandpass):** Định hình dải tần sơ bộ, loại bỏ nhiễu hạ âm và siêu âm.
- **Tầng 2 (Spectral Gating):** Thực hiện lọc nhiễu nền trong miền tần số dựa trên ngưỡng $\lambda = 0.6$.
- **Tầng 3 (AGC):** Thực hiện chuẩn hóa biên độ cuối cùng và áp dụng Noise Gate để làm sạch vùng im lặng.

4.3.6 Phương thức `reset_states()`

Hàm tiện ích dùng để đặt lại toàn bộ trạng thái của bộ xử lý về mặc định. Phương thức này được dùng khi bắt đầu một phiên ghi âm mới.

4.4 Hiện thực mô hình học máy

4.4.1 Tiền xử lý dữ liệu

Mục tiêu của tầng tiền xử lý là biến đổi dữ liệu âm thanh thô (file *.wav*) thành tensor đặc trưng có kích thước cố định để đưa vào mô hình CNN. Quy trình được triển khai theo chuỗi script: *make_subset.py* → *augment.py* → *preprocess.py*. **Các lớp được chọn để tạo subset:** Tập dữ liệu con được tạo bằng cách giữ nguyên các lớp mục tiêu trong *TARGET_CLASSES*, bao gồm: {*dog, cat, water_drops, laughing, coughing, door_wood_knock, clock_alarm, crying_baby, engine, frog, mouse_click, rooster, car_horn*}. Các lớp còn lại trong ESC-50 được gom về lớp *unknown* (giới hạn số mẫu bằng *UNKNOWN_MAX_SAMPLES*) để mô hình có khả năng phản hồi "không chắc chắn" khi gặp âm thanh ngoài tập mục tiêu.

- **Bước 1 – Tạo dataset con + gán lớp *unknown* (*make_subset.py*):**
  - **Mục đích:** Chọn ra các lớp mục tiêu để huấn luyện và gom phần còn lại về một lớp "không xác định" (*unknown*) nhằm tăng khả năng tổng quát hoá khi chạy thực tế.
  - **Thiết lập:**
    - *TARGET_CLASSES*: tập lớp cần giữ nguyên (ví dụ: *dog, cat, laughing, ...*).
    - *UNKNOWN_MAX_SAMPLES*: giới hạn số mẫu của lớp *unknown* để tránh lệch phân phối dữ liệu.

    - ∗ *TRAIN_RATIO/VAL_RATIO/TEST_RATIO*: tỉ lệ chia dữ liệu (train/val/test) theo kiểu **stratified** (mỗi lớp đều có mẫu ở cả 3 tập).
    - ∗ *RANDOM_STATE*: cố định seed để tái lập thí nghiệm.
  - **Output:** Tạo thư mục dataset con (ví dụ *ESC50_subset_with_unknown/*) và các file CSV split: *...train.csv*, *...val.csv*, *...test.csv*.
  - **Lưu ý kỹ thuật:** Chia stratified theo từng lớp giúp tránh tình trạng lớp hiếm bị "mất" ở val/test (đặc biệt khi mỗi lớp có ít mẫu).

- **Bước 2 – Augment dữ liệu + trộn noise theo SNR (*augment.py*):**
  - **Mục đích:** Tăng độ đa dạng của dữ liệu train để mô hình bền vững hơn trước thay đổi tốc độ nói, cao độ, gain và nhiễu môi trường.
  - **Phạm vi áp dụng:**
    - ∗ **Chỉ augment cho train** (tạo bộ *train_aug*).
    - ∗ **Test-noisy** được tạo riêng chỉ bằng trộn noise (không augment hình học) để đánh giá worst-case hợp lý.
  - **Các phép biến đổi (ví dụ cấu hình):**
    - ∗ *augment_speed*: time-stretch với *rate* ngẫu nhiên trong $[0.9, 1.1]$.
    - ∗ *augment_pitch*: pitch shift với *n_steps* ngẫu nhiên trong $[-2, 2]$ semitone.
    - ∗ *augment_gain*: thay đổi gain trong khoảng $[-6, 6]$ dB.
    - ∗ *augment_shift*: dịch thời gian (không wrap-around) với *shift_max* mặc định 0.2 (tức $\pm 20\%$ độ dài).
    - ∗ *random_augment_combo_train*: mỗi mẫu chọn ngẫu nhiên tối đa 2 phép biến đổi, giúp đa dạng nhưng vẫn tránh phá vỡ đặc trưng lớp.
  - **Trộn noise theo SNR:**
    - ∗ *TRAIN_NOISE_PROB*: xác suất trộn noise (ví dụ 0.7).
    - ∗ *TRAIN_NOISE_SNR_RANGE_DB*: khoảng SNR (ví dụ $(10, 30)$ dB).
    - ∗ Với test-noisy: *TEST_NOISE_PROB* thường đặt 1.0 để đánh giá "luôn có nhiễu".
  - **Output:** Sinh ra thư mục train augmented và CSV tương ứng.

- **Bước 3 – Trích đặc trưng log-mel và đóng gói *.npz* (*preprocess.py*):**
  - **Input:** File CSV (train_aug/val/test/test_noisy) + thư mục audio tương ứng.
  - **Chuẩn hoá waveform:**

* Resample về $f_s = 16$ kHz.
* Chuẩn hoá độ dài về *DEFAULT_DURATION = 5.0s* $\Rightarrow$ *DEFAULT_SAMPLES = 80000* mẫu (pad/cắt).

– **Tính Mel-spectrogram và log-scale:**

* *N_FFT = 1024*, *HOP_LENGTH = 512*, *N_MELS = 64*.
* Mel-spectrogram được chuyển sang dB bằng *power_to_db* (log-mel).

– **Cố định số frame theo thời gian:**

* *TIME_STEPS = 128*.
* Nếu số frame < 128 thì pad, nếu > 128 thì cắt bớt.
* Sau bước này, tensor có dạng $(T, M) = (128, 64)$.

– **Chuẩn hoá miền giá trị:**

* Dùng Min-Max normalization đưa về $[0, 1]$:

$$\hat{x} = \frac{x - x_{\min}}{x_{\max} - x_{\min} + 10^{-6}}$$

– **Đóng gói dữ liệu:**

* X: $(N, 128, 64, 1)$ (thêm kênh 1 để phù hợp CNN 2D).
* y: nhãn dạng index.
* *classes*: danh sách lớp theo thứ tự mapping.

– **Lưu ý kỹ thuật quan trọng (mapping lớp nhất quán):**

* Nếu mỗi split tự sort *classes* theo CSV của riêng nó, mapping có thể lệch.
* Vì vậy *preprocess.py* hỗ trợ tham số *–classes_from* để lấy mapping chuẩn (thường từ *train_aug.npz*) và áp cho val/test/noisy.

4.4.2 Huấn luyện mô hình

Mô hình phân loại được huấn luyện trên tensor log-mel $(128, 64, 1)$, đầu ra là phân phối xác suất trên các lớp (bao gồm cả *unknown*). Pipeline huấn luyện sử dụng *tf.data* để tối ưu tốc độ đọc dữ liệu và có bổ sung **SpecAugment** nhằm tăng tính bền vững với biến thiên thời gian/tần số.

- **Input:** *train_aug.npz* (X,y,classes), *val.npz*, *test.npz*, và (tuỳ chọn) *test_noisy.npz*.
- **Tiền xử lý nhãn:** Nhãn *y* được chuyển sang one-hot với độ sâu *num_classes*.
- **Thiết lập training:**

  - *BATCH = 64*.
  - **Loss:** Categorical Crossentropy với *label_smoothing = 0.05*.
  - **Optimizer:** Adam với learning rate 1×10^{-3}.
  - **Epochs:** tối đa *80*.
  - **Callbacks:**
    * *ModelCheckpoint*: lưu model tốt nhất theo *val_loss*.
    * *ReduceLROnPlateau*: giảm LR khi *val_loss* plateau (factor 0.5, patience 3, min_lr 10^{-6}).
    * *EarlyStopping*: dừng sớm khi không cải thiện (patience 12), phục hồi weight tốt nhất.

- **SpecAugment (augment trên spectrogram):**
  - Xác suất áp dụng: $p = 0.9$.
  - Frequency masking: tối đa *freq_mask_max = 14* bins.
  - Time masking: tối đa *time_mask_max = 40* frames.
  - Mục đích: giúp mô hình ít phụ thuộc vào một dải tần hay một đoạn thời gian cố định, tăng khả năng chịu nhiễu và biến thiên môi trường.

- **Kiến trúc CNN (tóm tắt):**
  - 3 block tích chập với số kênh *[32, 64, 128]*.
  - Mỗi block gồm 2 lớp *Conv2D(3x3)* + *BatchNorm* + *ReLU*, sau đó *MaxPool(2x2)* và *Dropout(0.25)*.
  - Sau cùng: *Conv2D(256)* $\rightarrow$ *BatchNorm* $\rightarrow$ *ReLU* $\rightarrow$ *GlobalAveragePooling* $\rightarrow$ *Dropout(0.5)* $\rightarrow$ *Dense(num_classes, softmax)*.

- **Xử lý mất cân bằng lớp (Class weight):**
  - Trọng số lớp được tính theo tần suất xuất hiện (lớp hiếm được tăng trọng số).
  - Ngoài ra có thể "boost" thêm một số lớp khó/quan trọng (*engine*, *rooster*, *dog*) để cải thiện macro-F1.

- **Xuất mô hình triển khai:**
  - Mô hình tốt nhất được lưu dạng *.keras* (checkpoint), sau đó chuyển sang file *textbfh5* để dễ load trong ứng dụng realtime.

– (Tuỳ chọn) Có thể tinh chỉnh ngưỡng *unknown* bằng cách quét threshold trên tập *test_noisy* và chọn ngưỡng tối ưu theo macro-F1.

4.4.3 Suy luận thời gian thực

Trong triển khai realtime, hệ thống đọc dữ liệu từ microphone theo từng *chunk* và thực hiện phân loại online. Điểm quan trọng của thiết kế là **không dự đoán ở mọi chunk**, mà dùng buffer + hop + smoothing để tăng ổn định đầu ra.

- **Thiết lập audio stream:**
  - *DEFAULT_RATE = 16000* Hz.
  - *DEFAULT_CHUNK = 1024* mẫu mỗi lần đọc.
  - Dữ liệu đọc từ mic dạng *int16* và được chuẩn hoá về float32 trong miền $[-1, 1]$ bằng phép chia */32768.0*.
- **Cơ chế buffer cửa sổ:**
  - Hệ thống duy trì *env_buffer* tối đa *ENV_DURATION_SEC = 5.0s* (tương ứng 80000 mẫu).
  - Mỗi lần có chunk mới, chunk được nối vào buffer; nếu vượt quá 5s thì cắt giữ phần cuối.
  - Khi suy luận, hệ thống lấy cửa sổ gần nhất *ENV_WINDOW_DURATION_SEC = 2.0s* (tương ứng 32000 mẫu) để phản ánh trạng thái môi trường "hiện tại".
- **Gating theo năng lượng (RMS):**
  - RMS được tính nhanh:
$$\text{RMS} = \sqrt{\mathbb{E}[x^2] + 10^{-9}}$$
  - Nếu *rms* < *ENV_MIN_RMS* (ví dụ 0.005), hệ thống tránh đưa ra dự đoán mới để giảm false trigger trong vùng im lặng/nhiễu nhỏ.
- **Tiền xử lý khi inference:**
  - Dù cửa sổ lấy là 2s, trước khi trích đặc trưng hệ thống vẫn pad/cắt về chuẩn 5s để thống nhất với mô hình đã train.
  - Log-mel sử dụng đúng tham số như training: *n_fft=1024*, *hop=512*, *n_mels=64*, cắt/pad thời gian về *128* frame, và Min-Max về $[0, 1]$.

- **Áp ngưỡng confidence cho *unknown*:**
  - Sau khi có nhãn dự đoán và xác suất tương ứng, nếu điểm dự đoán **bé hơn** *ENV_CONF_THRESHOLD* thì output bị ép thành *unknown*.
  - Cách này giúp hệ thống "thừa nhận không chắc chắn" thay vì trả về nhãn sai một cách tự tin.

4.4.4 Hậu xử lý theo thời gian:

Do dự đoán theo từng khung ngắn dễ bị rung (label flicker) và nhảy nhãn khi môi trường nhiễu, hệ thống thêm một tầng hậu xử lý theo thời gian gồm 4 cơ chế: **hop**, **smoothing**, **silence reset**, và **fast switch**.

1. **Hop (giảm tần suất predict để ổn định):**
   - Tham số: *ENV_PRED_HOP_SEC* (ví dụ 0.5s).
   - Ý tưởng: chỉ gọi model mỗi *0.5s*. Nếu chưa tới thời điểm hop, hệ thống giữ kết quả gần nhất (*last_label/last_conf*) để UI/logic không bị thay đổi liên tục.
   - Lợi ích: giảm nhạy với nhiễu tức thời, giảm tải tính toán khi chạy realtime.
2. **Temporal smoothing (trung bình xác suất theo cửa sổ):**
   - Tham số: *ENV_SMOOTH_K* (ví dụ $K = 3$, thường 3–7 là hợp lý).
   - Cách làm: mỗi lần predict thu được vector xác suất *probs*. Lưu các vector này trong hàng đợi độ dài K:
     $$\bar{\mathbf{p}} = \frac{1}{K}\sum_{i=1}^{K}\mathbf{p}_i$$
     Sau đó chọn nhãn bằng $\arg\max(\bar{\mathbf{p}})$ và confidence là giá trị lớn nhất của $\bar{\mathbf{p}}$.
   - Lợi ích: làm mượt đầu ra, chống nhảy nhãn do một dự đoán bất thường.
   - Đánh đổi: tăng độ trễ nhỏ (vì lấy trung bình trên nhiều lần dự đoán).
3. **Reset khi im lặng kéo dài (silence reset):**
   - Tham số: *ENV_RESET_ON_SILENCE_SEC* (ví dụ 0.8s).
   - Nếu *rms* liên tục thấp hơn *ENV_MIN_RMS* trong thời gian $> 0.8s$, hệ thống:
     - reset *env_buffer*,
     - xoá lịch sử *env_prob_hist*,

  - trả hệ thống về trạng thái "chờ tín hiệu".
- Mục đích: tránh trường hợp label cũ bị "kéo dài" qua vùng im lặng (giữ sai ngữ cảnh).

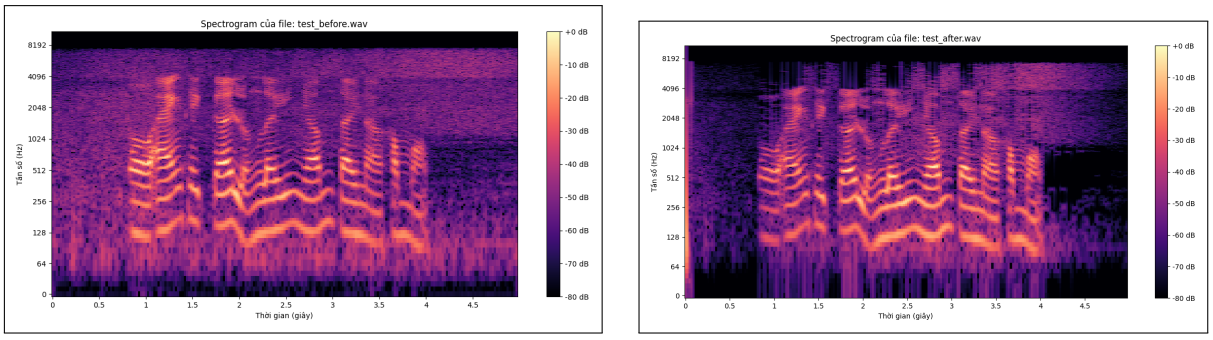
4. **Fast switch (chuyển nhanh khi nhãn mới đủ chắc):**
   - Tham số: *ENV_SWITCH_CONF* (0.60) và *ENV_SWITCH_STREAK* (2).
   - Khi nhãn raw (chưa smoothing) đổi sang lớp khác và có confidence đủ cao, nếu hiện tượng này lặp lại liên tiếp đủ *STREAK* lần thì hệ thống **xóa history smoothing** để chuyển nhanh sang nhãn mới.
   - Lợi ích: giảm độ trễ khi môi trường thật sự đổi trạng thái (ví dụ từ *engine* sang *car_horn*), nhưng vẫn chống nhảy nhãn do một lần dự đoán sai.

Tóm tắt luồng realtime (mỗi chunk):

1. Đọc *chunk* → chuẩn hoá float32 → tính RMS.
2. Nếu RMS thấp kéo dài: reset trạng thái.
3. Nếu đủ điều kiện (đủ buffer + tới hop): preprocess → predict *probs*.
4. Cập nhật smoothing history → lấy trung bình → ra nhãn.
5. Nếu confidence thấp: ép nhãn thành *unknown*.

5 Kết quả kiểm thử & Đánh giá

5.1 Kiểm thử bộ xử lý âm thanh



(a) Tín hiệu thô

(b) Sau khi xử lý

Hình 5.1: So sánh dạng sóng tín hiệu trước và sau khi đi qua luồng xử lý của AudioProcessor.

Dựa trên kết quả quan sát từ Hình 5.1, ta có thể đưa ra các đánh giá chi tiết về hiệu quả của bộ xử lý `AudioProcessor` như sau:

- **Loại bỏ nhiễu nền:** Ở Hình 5.1a, các vùng không có những âm thanh mục tiêu (khoảng thời gian 0s - 0.8s và sau 4.2s) xuất hiện nhiều dải màu tím và hồng nhạt trên khắp các dải tần, cho thấy sự hiện diện của white noise và nhiễu môi trường. Sau khi qua bộ lọc (Hình 5.1b), các vùng này gần như chuyển sang màu đen hoàn toàn, chứng tỏ mức nhiễu đã được triệt tiêu đáng kể.
- **Lọc dải tần:** Các thành phần nhiễu ở dải tần cực thấp (dưới 64Hz) và dải tần cao (trên 8192Hz) đã được suy giảm mạnh. Điều này cho thấy các bộ lọc thông dải Butterworth đã hoạt động hiệu quả trong việc tập trung vào dải tần số của các âm thanh dùng để phân loại trong đồ án này.
- **Bảo toàn đặc trưng tín hiệu :** Mặc dù nhiễu nền bị loại bỏ, các cấu trúc của âm thanh mục tiêu (các vạch màu vàng, cam sáng trong khoảng tần số 128Hz - 4096Hz) vẫn giữ được độ sắc nét và cường độ cần thiết.

Kết quả kiểm nghiệm cho thấy bộ xử lý âm thanh đã hoàn thành tốt vai trò tiền xử lý, tạo ra một tín hiệu sạch hơn nhưng vẫn đảm bảo không làm biến dạng các đặc trưng âm học quan trọng của nguồn âm chính.

5.2 Kiểm thử mô hình học máy

Âm thanh được đọc với $f_s = 16$ kHz và `chunk`=1024 mẫu; buffer giữ tối đa 5 giây, và mô hình phân loại trên đoạn 2 giây gần nhất trong buffer. Dự đoán chỉ được cập nhật mỗi 0.5s giữa các lần cập nhật, hệ thống giữ kết quả gần nhất để UI không bị thay đổi liên tục.

5.2.1 Stability

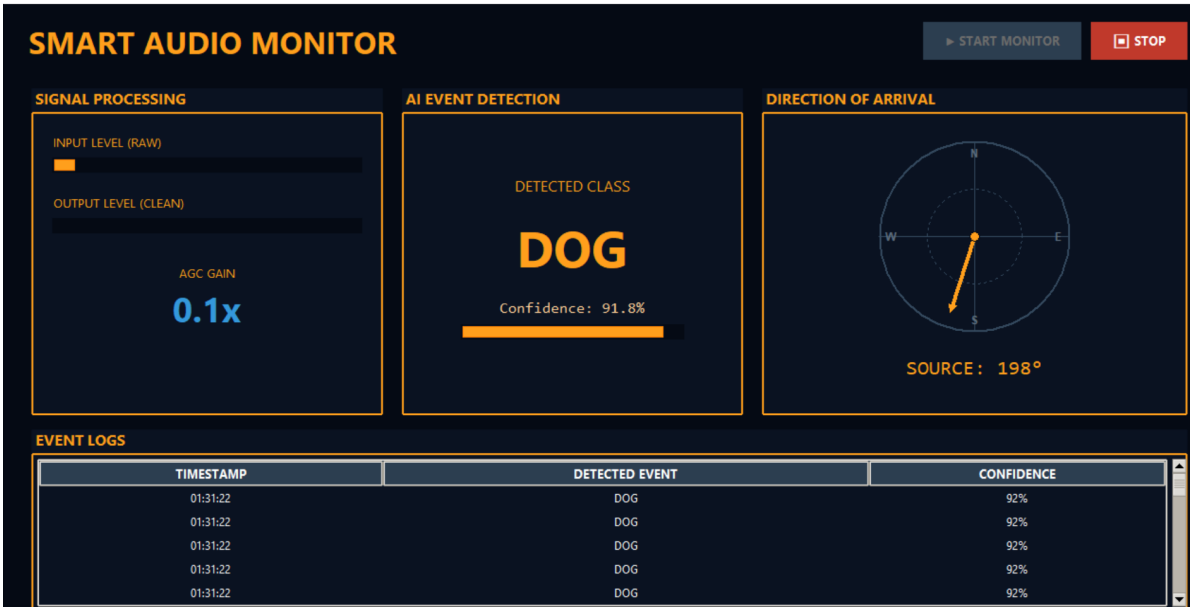
Độ ổn định được đánh giá theo mức độ nhảy nhãn theo thời gian (label switches), trong đó hệ thống đã triển khai: (i) smoothing bằng trung bình xác suất trên cửa sổ `ENV_SMOOTH_K=3` và (ii) fast-switch để giảm trễ khi môi trường thật sự đổi lớp. Quan sát thực nghiệm cho thấy:

- Ở trường hợp âm thanh ổn định (Hình 5.2), hệ thống giữ nhãn `DOG` với confidence cao (91.8%), và log hiển thị lặp lại cùng nhãn — cho thấy đầu ra ổn định trong đoạn quan sát.
- Ở tình huống môi trường thay đổi (Hình 5.3), log ghi nhận chuyển nhãn từ `LAUGHING` (65%) tại 01:44:12 sang `DOG` (61%) tại 01:44:13, tức có xảy ra label switch tương ứng với thay đổi sự kiện âm thanh.

5.2.2 Confidence

Confidence của hệ thống là xác suất lớn nhất sau smoothing,
và có ngưỡng ENV_CONF_THRESHOLD = 0.5: nếu conf < 0.5 thì ép nhãn về unknown.

- Trường hợp DOG cho confidence 91.8% (rất cao), vượt xa ngưỡng nên dự đoán nằm trong vùng "chắc chắn".
- Trường hợp chuyển LAUGHING → DOG có confidence 61–65% (trung bình nhưng vẫn lớn hơn 0.5), do đó không kích hoạt unknown. Đây là vùng confidence cần theo dõi thêm khi môi trường nhiễu để hạn chế false positive.



Hình 5.2: Kết quả realtime trên Smart Audio Monitor: hệ thống phát hiện lớp DOG với confidence 91.8%, thể hiện dự đoán nằm trong vùng tin cậy cao và ổn định trong đoạn quan sát.

Hình 5.3: Minh hoạ label switch trong Event Logs: LAUGHING (65%) tại 01:44:12 chuyển sang DOG (61%) tại 01:44:13; cả hai vượt ngưỡng ENV_CONF_THRESHOLD=0.5 nên không bị ép về unknown.

6 Kết luận

6.1 Tóm tắt kết quả đạt được

Đồ án đã xây dựng thành công một hệ thống phân loại âm thanh môi trường hoạt động trong thời gian thực với kiến trúc pipeline tích hợp từ phần cứng đến phần mềm. Hệ thống bao gồm các module thu thập âm thanh từ ReSpeaker Mic Array v2.0, xử lý tín hiệu số, phân loại bằng mô hình học sâu và giao diện tương tác cho người dùng.
Pipeline xử lý tín hiệu kết hợp các kỹ thuật DSP truyền thống nhằm làm sạch và chuẩn hóa tín hiệu đầu vào, trong khi mô hình CNN được huấn luyện trên đặc trưng log-mel spectrogram để học các đặc trưng âm thanh có ý nghĩa. Thông qua việc áp dụng các kỹ thuật regularization và data augmentation, mô hình đạt độ chính xác tốt trên tập kiểm tra và duy trì hiệu suất ổn định khi làm việc với dữ liệu có nhiễu.
Hệ thống cuối cùng có khả năng nhận diện 14 lớp âm thanh môi trường và đáp ứng yêu cầu vận hành trong thời gian thực.

6.2 Đánh giá hiệu suất và ưu điểm

Hệ thống thể hiện một số ưu điểm nổi bật:

- Độ trễ thấp và khả năng hoạt động thời gian thực: Việc xử lý theo từng khung âm thanh cố định giúp hệ thống phản hồi nhanh, phù hợp cho các ứng dụng tương tác.
- Thiết kế mô-đun: Các thành phần của hệ thống được tách biệt rõ ràng, giúp việc bảo trì, mở rộng hoặc thay thế từng module trở nên thuận tiện.
- Cách tiếp cận hybrid hiệu quả: Sự kết hợp giữa DSP và học sâu giúp giảm nhiễu đầu vào, cải thiện tính ổn định của đặc trưng và nâng cao khả năng tổng quát hóa của mô hình.

6.3 Những hạn chế

Bên cạnh các kết quả đạt được, hệ thống vẫn tồn tại một số hạn chế:

- Mô hình phụ thuộc vào chất lượng và phạm vi của tập dữ liệu huấn luyện, do đó hiệu suất có thể suy giảm khi gặp các âm thanh ngoài phân bố dữ liệu đã học.
- Hệ thống hiện được thiết kế cho bài toán phân loại đơn nhãn, nên chưa xử lý hiệu quả các tình huống có nhiều nguồn âm thanh đồng thời.

- Việc triển khai trên các thiết bị nhúng có tài nguyên hạn chế vẫn còn gặp khó khăn do yêu cầu về bộ nhớ và năng lực tính toán của mô hình.
- Các tham số xử lý và ngưỡng quyết định được thiết lập cố định, chưa có cơ chế tự thích ứng theo sự thay đổi của môi trường theo thời gian.

6.4 Hướng phát triển tương lai

Trong tương lai, hệ thống có thể được cải thiện theo các hướng sau:

- Mở rộng và đa dạng hóa tập dữ liệu huấn luyện để nâng cao khả năng tổng quát hóa trong môi trường thực tế.
- Nghiên cứu các kiến trúc mô hình nâng cao như attention hoặc transformer nhằm khai thác tốt hơn mối quan hệ theo thời gian của tín hiệu âm thanh.
- Bổ sung cơ chế phát hiện âm thanh bất thường để tăng độ tin cậy khi vận hành ngoài thực tế.
- Áp dụng các kỹ thuật tối ưu hóa mô hình như quantization hoặc pruning để phục vụ triển khai trên thiết bị edge.
- Mở rộng bài toán sang phân loại đa nhãn nhằm xử lý các tình huống âm thanh chồng lấp.

6.5 Kết luận cuối cùng

Đồ án đã chứng minh tính khả thi của việc xây dựng một hệ thống phân loại âm thanh môi trường thời gian thực theo cách tiếp cận kết hợp giữa xử lý tín hiệu số và học sâu. Mặc dù còn tồn tại một số hạn chế, hệ thống hiện tại cung cấp một nền tảng vững chắc cho các nghiên cứu và phát triển tiếp theo, hướng tới các ứng dụng thực tế như nhà thông minh, giám sát an ninh và hỗ trợ con người trong môi trường sống.