Speaker Recognition using LSTM

I. Introduction to Speaker Recognition

The Key Differences: Speech recognition focuses on converting spoken language into written text, enabling transcription and text-based analysis. In contrast, **speaker recognition** aims to identify and authenticate individuals based on their unique vocal characteristics.

Speech recognition và Speaker recognition là hai bài toán lớn của lĩnh vực xử lý ngôn ngữ tự nhiên.

1. **Nhận dạng giọng nói (Speaker Recognition)**:
   - **Mục tiêu**: Bài toán này tập trung vào việc xác định và xác thực danh tính của cá nhân dựa trên các đặc điểm độc đáo của giọng nói của họ.
   - **Ứng dụng**: Speaker recognition được sử dụng rộng rãi trong các hệ thống nhận dạng người dùng, chẳng hạn như hệ thống mở khóa bằng giọng nói trên điện thoại di động hoặc trong các ứng dụng giám sát an ninh.
   - **Công nghệ liên quan**: Trong Speaker recognition, các thuật toán và mô hình học máy được sử dụng để phân tích và nhận dạng các đặc điểm của giọng nói như tần số, biên độ và đặc điểm cách phát âm của người nói.
2. **Nhận dạng tiếng nói (Speech Recognition)**:
   - **Mục tiêu**: Bài toán này nhằm chuyển đổi ngôn ngữ nói thành văn bản viết, cho phép transcribe và phân tích dữ liệu dựa trên văn bản và **ngữ cảnh.**
   - **Ứng dụng**: Speech recognition được áp dụng trong nhiều lĩnh vực như xử lý tài liệu, tự động chuyển văn bản thành giọng nói (text-to-speech), hỗ trợ người khuyết tật, và các ứng dụng trợ lý ảo.
   - **Công nghệ liên quan**: Trong speech recognition, các thuật toán như Hidden Markov Models (HMMs), Deep Neural Networks (DNNs), và Recurrent Neural Networks (RNNs), cùng với các kỹ thuật xử lý tín hiệu âm thanh được sử dụng để chuyển đổi âm thanh thành văn bản.

Cả hai bài toán đều sử dụng hai cơ chế nhận dạng giọng nói là **Text-dependent (TD)** và **Text-independent (ID).** Chúng đều đóng vai trò quan trọng trong xác thực và nhận dạng người nói. Dưới đây là sự phân biệt rõ ràng giữa hai cơ chế này:

1. **Text-dependent (TD)**:
   - **Đặc điểm chính**: TD giả định rằng đoạn văn bản được nói luôn giống nhau hoặc có biến thể nhỏ. Điều này giúp giảm bớt sự phức tạp trong quá trình nhận dạng.
   - **Ứng dụng thực tế**: Thường được sử dụng trong các ứng dụng hàng ngày như lệnh "Ok Google," "Hey Siri," "Alexa," hoặc trong việc xác thực mật khẩu thông qua giọng nói.
   - **Ưu điểm**: Dễ triển khai và đơn giản hóa quá trình nhận dạng, giảm thiểu biến thể âm vị và độ dài của đoạn văn bản.
   - **Nhược điểm**: Có nguy cơ bị tấn công bằng cách sử dụng bản ghi giọng nói của người dùng.
2. **Text-independent (ID)**:
   - **Đặc điểm chính**: ID không biết "Đã nói gì." Hệ thống không phụ thuộc vào bất kỳ đoạn văn bản cụ thể nào.
   - **Ứng dụng thực tế**: Thường được sử dụng trong các hệ thống xác thực an ninh, nơi yêu cầu một cách tiếp cận linh hoạt hơn và không giới hạn đối với đoạn văn bản.
   - **Ưu điểm**: Phù hợp cho các ứng dụng đòi hỏi mức độ linh hoạt cao và không cần phụ thuộc vào đoạn văn bản cụ thể.
   - **Nhược điểm**: Có thể đòi hỏi nhiều công sức hơn để triển khai do sự phức tạp cao hơn, đặc biệt là trong việc xác định và nhận dạng người nói dựa trên các đặc điểm không gian của giọng nói, thay vì dựa trên đoạn văn bản cụ thể.

Các hệ thống **text-independent** sẽ chú trọng vào việc phân tích và xác định các **đặc điểm đặc trưng của giọng nói**, chẳng hạn như **tần số, biên độ,** và các thuộc tính khác của âm thanh. Điều này cho phép hệ thống nhận dạng và xác minh người nói mà không cần biết trước văn bản cụ thể mà họ nói.

▼ Cần tính linh hoạt và bảo mật: ⇒ Sử dụng cơ chế **text-independent**

II. Motivation (Why not CNN? :)))):

- Phương pháp định danh người nói sử dụng CNN cho độ chính xác trên tập validation **~94%**, tương đối cao nhưng tồn đọng nhiều điểm bất cập chí mạng:
  - Mỗi khi người dùng có nhu cầu enroll thêm một speaker cần phải huấn luyện lại toàn bộ mô hình để lưu bộ tham số mới
  - Cần tương đối nhiều dữ liệu dạng audio để huấn luyện mô hình

  ⇒ Không thực tiễn trong bài toán định danh người nói ở dự án nhà thông minh, khi việc thêm thành viên trong nhà yêu cầu tốc độ xử lý nhanh với đầu vào dữ liệu để huấn luyện hợp lý.

▼ **Đề xuất giải pháp thay thế: LSTM**

- Mạng neuron LSTM sau khi được huấn luyện sẽ nhận đầu vào là một file audio, là đoạn ghi âm cần định danh người nói và cho output là một **embedding vector** biểu diễn số học của âm thanh được trích xuất, mang những thông tin quan trọng về đặc điểm tiếng nói của người trong file âm thanh đó.
- Embedding vector thường được thiết kế sao cho các vector tương tự nhau trong không gian embedding tương ứng với các file âm thanh của cùng một người nói sẽ gần nhau, trong khi các vector khác nhau tương ứng với các người nói khác nhau sẽ cách xa nhau. **(triplet loss)**
- Chúng ta có thể sử dụng linh hoạt embedding vector cùng các kỹ thuật Machine Learning khác chẳng hạn như KNNs, K-Means để quyết định người nói giúp cải thiện độ chính xác

⇒ **Linh hoạt** trong việc **enroll thêm người nói** và **hậu xử lý** để quyết định người nói!

III. Data Preparation

▼ **Bộ train**

- Sử dụng Bộ dữ liệu LibriSpeech: một bộ sưu tập gần 1,000 giờ ghi âm sách nói, là một phần của dự án LibriVox.
- Dữ liệu huấn luyện được chia thành ba phần bao gồm bộ dữ liệu 100 giờ, 360 giờ và 500 giờ, trong khi dữ liệu phát triển và kiểm thử được chia thành các nhóm 'clean' và 'other', tùy thuộc vào khả năng của hệ thống nhận dạng giọng nói tự động. Mỗi tập phát triển và kiểm thử chứa khoảng 5 giờ âm thanh. Bộ dữ liệu này cũng cung cấp các mô hình ngôn ngữ n-gram và các văn bản tương ứng được trích từ sách Project Gutenberg, bao gồm 803 triệu token và 977 nghìn từ duy nhất.

```
FeaturesDict({
    'chapter_id': int64,
    'id': string,
    'speaker_id': int64,
    'speech': Audio(shape=(None,), dtype=int16),
    'text': Text(shape=(), dtype=string),
})
```

▼ **Bộ test**

- Nhóm tự xây dựng một bộ data nhỏ gồm 40 audio file, mỗi file dài 20s được các thành viên trong nhóm ghi lại hoạt động đọc báo, hát,... **Đây cũng là bộ data mà nhóm đã sử dụng để train mô hình CNN (**Tạm gọi là **"Data tiếng nói để train CNN**")
- Nhóm đồng thời cũng xây dựng một bộ data khác gồm 62 audio file, mỗi file dài ~3s mô tả các thành viên đang điều khiển ngôi nhà thông minh của mình **(**Tạm gọi là **"Data tiếng nói để điều khiển nhà**")

  Phòng khách:
  "{tên người nói}_mo_cua_phong_khach.wav": Tôi muốn Mở cửa trong phòng khách.
  "{tên người nói}_bat_den_phong_khach.wav": Tôi muốn Bật đèn trong phòng khách.
  "{tên người nói}_tat_den_phong_khach.wav": Tôi muốn Tắt đèn trong phòng khách.

  Phòng ngủ con cái:
  "{tên người nói}_bat_den_phong_ngu_con_cai.wav": Tôi muốn Bật đèn ở phòng ngủ con cái.
  "{tên người nói}_tat_den_phong_ngu_con_cai.wav": Tôi muốn Tắt đèn ở phòng ngủ con cái.
  "{tên người nói}_mo_cua_phong_ngu_con_cai.wav": Tôi muốn Mở cửa ở phòng ngủ con cái.

  Phòng ngủ ba mẹ:
  "{tên người nói}_bat_den_phong_ngu_ba_me.wav": Tôi muốn Bật đèn ở phòng ngủ ba mẹ.
  "{tên người nói}_tat_den_phong_ngu_ba_me.wav": Tôi muốn Tắt đèn ở phòng ngủ ba mẹ.
  "{tên người nói}_mo_cua_phong_ngu_ba_me.wav": Tôi muốn Mở cửa ở phòng ngủ ba mẹ.

  Phòng bếp:
  "{tên người nói}_xem_nhiet_do_do_am_phong_bep.wav": Tôi muốn Xem nhiệt độ độ ẩm trong phòng bếp.
  "{tên người nói}_bat_den_phong_bep.wav": Tôi muốn Bật đèn ở phòng ngủ ba mẹ.
  "{tên người nói}_tat_den_phong_bep.wav": Tôi muốn Tắt đèn ở phòng ngủ ba mẹ.

  Garage:
  "{tên người nói}_mo_cua_cuon_garage.wav": Tôi muốn Mở cửa cuốn trong garage.
  "{tên người nói}_dong_cua_cuon_garage.wav": Tôi muốn Đóng cửa cuốn trong garage.
  "{tên người nói}_bat_den_garage.wav": Tôi muốn Bật đèn ở garage.
  "{tên người nói}_tat_den_garage.wav": Tôi muốn Tắt đèn ở garage.

Nhóm train mô hình trên cả 3 tập: **100-clean, 360-clean và 500-other** và thực hiện đánh giá mô hình trên tập **Data tiếng nói để train CNN và Data tiếng nói để điều khiển nhà.**

IV. Data Processing: Feature Extraction and SpecAugment

▼ Trích xuất đặc trưng

- Trong quá trình Trích xuất đặc trưng, chúng ta muốn chuyển đổi các tệp âm thanh từ định dạng FLAC thành dạng đặc trưng mà mô hình của chúng ta có thể xử lý hiệu quả. Nhóm đã sử dụng **MFCC** làm vector đặc trưng vì sức mạnh của nó trong việc xử lý âm thanh, đặc biệt là trong các ứng dụng nhận dạng người nói.
- Điều quan trọng là MFCC là một **biểu diễn của âm thanh** dựa trên các **đặc trưng phổ** của nó, được thiết kế để **mô phỏng cách con người nghe tiếng nói**. Với MFCC, chúng ta có thể chuyển đổi tín hiệu âm thanh thành một loạt các vectơ đặc trưng, **mỗi vectơ đại diện cho một khung thời gian của tín hiệu**. ⇒ **MFCC (Mel-Frequency Cepstral Coefficients) là một biểu diễn đặc trưng được tạo ra từ tín hiệu âm thanh trên miền thời gian**
- Đoạn code trích xuất đặc trưng:

```
def extract_features(audio_file):
    """Extract MFCC features from an audio file, shape=(TIME, MFCC)."""
    waveform, sample_rate = sf.read(audio_file)
```

```
    # Convert stereo audio to mono
    if len(waveform.shape) == 2:
        waveform = librosa.to_mono(waveform.transpose())

    # Resample audio to 16kHz if not already
    if sample_rate != 16000:
        waveform = librosa.resample(waveform, sample_rate, 16000)

    # Compute Mel-frequency cepstral coefficients (MFCCs)
    features = librosa.feature.mfcc(y=waveform, sr=sample_rate, n_mfcc=myconfig.N_MFCC)
    # The shape of features will be (TIME, MFCC), e.g., (40, 441)

    return features.transpose()
```

- Nếu âm thanh có hai kênh (stereo), hàm `librosa.to_mono` được sử dụng để chuyển đổi nó thành dạng đơn kênh (mono). Điều này đảm bảo rằng chúng ta chỉ xử lý một kênh của âm thanh, giúp giảm bớt phức tạp và tiêu tốn bộ nhớ.
- Nếu tốc độ mẫu của âm thanh không phải là 16kHz, hàm `librosa.resample` được sử dụng để tái mẫu âm thanh, đảm bảo rằng tất cả các âm thanh đều có cùng một tốc độ mẫu (16kHz) để đồng bộ hóa quá trình trích xuất đặc trưng.
- `N_MFCC = 40`, điều này chỉ định rằng chúng ta muốn trích xuất 40 hệ số Mel-Frequency Cepstral Coefficients (MFCC) từ mỗi khung thời gian của tín hiệu âm thanh (vector 40 chiều cho mỗi khung thời gian)

▼ Tăng cường dữ liệu

```
def apply_specaug(features):
    """Apply SpecAugment to features."""
    # Get the shape of the input features
    seq_len, n_mfcc = features.shape
    # Copy the input features to the outputs
    outputs = features
    # Compute the mean feature value
    mean_feature = np.mean(features)

    # Apply frequency masking with probability SPECAUG_FREQ_MASK_PROB
    if random.random() < myconfig.SPECAUG_FREQ_MASK_PROB:
        # Randomly select the width of the mask
        width = random.randint(1, myconfig.SPECAUG_FREQ_MASK_MAX_WIDTH)
        # Randomly select the starting index for the mask
        start = random.randint(0, n_mfcc - width)
        # Apply the mask to the frequency dimension by setting masked values to the mean feat
        outputs[:, start: start + width] = mean_feature

    # Apply time masking with probability SPECAUG_TIME_MASK_PROB
    if random.random() < myconfig.SPECAUG_TIME_MASK_PROB:
        # Randomly select the width of the mask
        width = random.randint(1, myconfig.SPECAUG_TIME_MASK_MAX_WIDTH)
        # Randomly select the starting index for the mask
        start = random.randint(0, seq_len - width)
        # Apply the mask to the time dimension by setting masked values to the mean feature v
        outputs[start: start + width, :] = mean_feature
```

```
    return outputs
```

- SpecAugment là một phương pháp tăng cường dữ liệu được thiết kế đặc biệt cho việc xử lý dữ liệu âm thanh, đặc biệt là trong các tác vụ liên quan đến xử lý giọng nói hoặc âm thanh. Tương tự như các kỹ thuật tăng cường dữ liệu được sử dụng trong xử lý hình ảnh, SpecAugment giúp mô hình học được các đặc điểm quan trọng và trở nên robust hơn đối với các biến đổi và nhiễu trong dữ liệu âm thanh.
- SpecAugment được áp dụng trực tiếp vào đầu vào đặc trưng của mạng nơ-ron, chẳng hạn như các hệ số filter bank. Phương pháp tăng cường này bao gồm việc biến đổi đặc trưng bằng cách uốn cong các đặc trưng, che phủ các khối kênh tần số và các khối bước thời gian. Mục tiêu của SpecAugment là xây dựng một chính sách tăng cường tác động trực tiếp lên log mel spectrogram, giúp mạng học được các đặc trưng hữu ích.

1. Frequency Masking (Che phủ tần số): Áp dụng sao cho f kênh tần số mel liên tiếp $[f_0, f_0+f)$ bị che phủ, trong đó f được chọn từ phân phối đồng đều từ 0 đến tham số che phủ tần số F, và f_0 được chọn từ $[0, \nu - f)$. Ở đây, ν là số kênh tần số mel.

2. Time Masking (Che phủ thời gian): Áp dụng sao cho t bước thời gian liên tiếp $[t_0, t_0+t)$ bị che phủ, trong đó t được chọn từ phân phối đồng đều từ 0 đến tham số che phủ thời gian T, và t_0 được chọn từ $[0, \tau - t)$. Ở đây, τ là số bước thời gian.

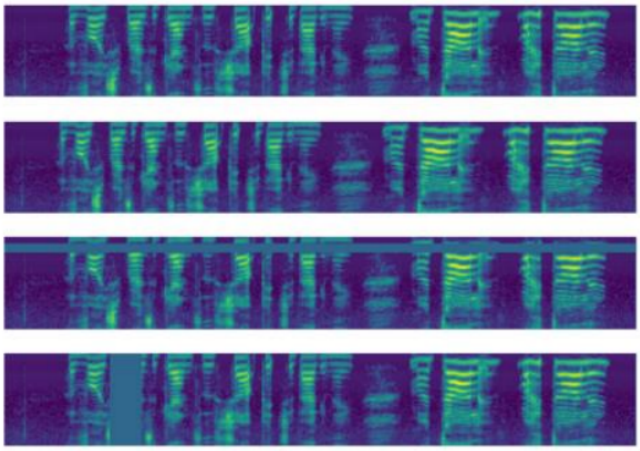


Figure 2.14. Augmentations applied to the base input, given at the top. From top to bottom, the figures depict the log mel spectrogram of the base input with no augmentation, time warp, frequency masking and time masking applied

V. Loss Function: Triplet Loss

- Triplet loss là một kỹ thuật phổ biến được sử dụng trong nhận dạng người nói, trong đó mô hình được huấn luyện để học cách phân biệt giữa các người nói dựa trên các đặc trưng của họ. Trong triplet loss, đầu vào của chúng ta bao gồm ba phân đoạn: một anchor (gốc), một positive (dương), và một negative (âm). Hai phân đoạn anchor và positive thuộc về cùng một người nói, trong khi phân đoạn negative thuộc về một người nói khác. Mục tiêu của triplet loss là đảm bảo rằng khoảng cách giữa anchor và positive là nhỏ nhất có thể, trong khi khoảng cách giữa anchor và negative là lớn nhất có thể. Điều này đảm bảo rằng mô hình học được cách phân biệt giữa các người nói dựa trên các đặc trưng của họ.

$$\sum_{i}^{N}[||f(x_i^a) - f(x_i^p)||_2^2 - ||f(x_i^a) - f(x_i^n)||_2^2 + \propto]$$

Figure 2.15. Triplet loss formula

- **f(x):** Là hàm ánh xạ đầu vào x thành một vector n chiều (n-dimensional).
- **i:** Đại diện cho dữ liệu thứ i.
- **a (anchor), p (positive), n (negative):** Các chỉ số cho anchor, positive và negative samples, tương ứng.
- **w:** Là vector nhúng được sinh ra bởi hàm ánh xạ.

- Mục tiêu chính của Triplet Loss là đảm bảo rằng các cặp không giống nhau (dissimilar) cách xa nhau ít nhất một giá trị biên (margin) so với các cặp giống nhau (similar). Nói cách khác, tối thiểu hóa khoảng cách giữa Anchor và Positive, đồng thời tối đa hóa khoảng cách giữa Anchor và Negative. Điều này đồng nghĩa với việc mô hình cố gắng giảm sự khác biệt giữa các mẫu tương đồng (Positive) trong không gian nhúng, trong khi tăng cường sự khác biệt giữa các mẫu không tương đồng (Negative).
-

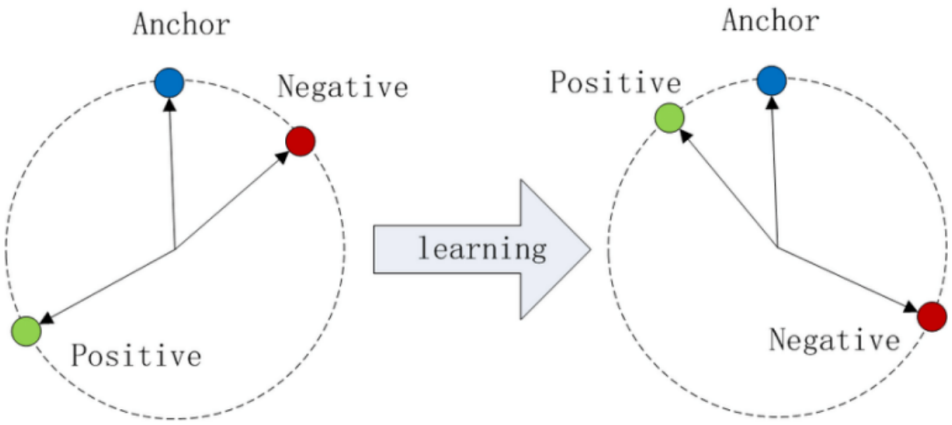
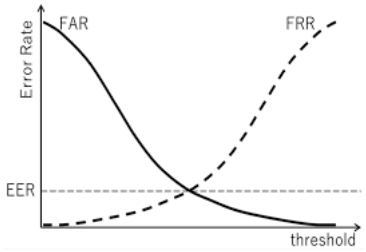


Figure 2.16 The objective of triplet loss

Trong Triplet Loss, quá trình sử dụng các nhóm ba mục gọi là triplets, mỗi triplet bao gồm:

1. Anchor (mẫu gốc): Là một mẫu cố định với một danh tính nhất định.
2. Positive (mẫu tương đồng): Là một mẫu gần với anchor, thường là một mẫu cùng lớp hoặc cùng phân loại.
3. Negative (mẫu không tương đồng): Là một mẫu xa với anchor, không cùng lớp hoặc phân loại với anchor.

VI. Evaluation Metrics



- False Accept Rate (FAR) is the probability that a natural fingerprint is wrongly identified as a fake fingerprint
- False Reject Rate(FRR) is the probability of a fake fingerprint being misplaced as a natural fingerprint.

Trong **bài toán nhận diện người nói**:

Giả sử chúng ta có một hệ thống nhận dạng người nói được đào tạo để phân biệt giữa hai người nói: A và B.

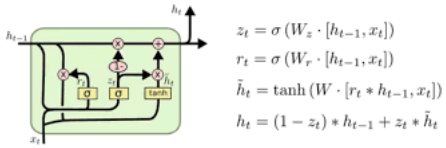
1. **False Acceptance Rate (FAR)**: Đây là tỷ lệ của số lượng mẫu không chính xác của người B được chấp nhận như là người A so với tổng số mẫu của người B trong tập dữ liệu kiểm tra.
2. **False Rejection Rate (FRR)**: Đây là tỷ lệ của số lượng mẫu của người A bị từ chối nhưng thực tế là của người A so với tổng số mẫu của người A trong tập dữ liệu kiểm tra.
3. **Equal Error Rate (EER)**: Đây là điểm trên đồ thị ROC curve nơi tỷ lệ FAR và tỷ lệ FRR là bằng nhau. Nó là một thước đo quan trọng của hiệu suất của hệ thống nhận dạng. EER thường được hiểu là điểm mà khi bạn điều chỉnh ngưỡng quyết định, tỷ lệ lỗi chấp nhận và từ chối sẽ là bằng nhau.
4. **Equal Error Rate (EER)**: Đây là tỷ lệ lỗi giữa tỷ lệ False Acceptance Rate (FAR) và False Rejection Rate (FRR) khi chúng bằng nhau. Nó thường được hiểu là điểm trên đồ thị ROC curve nơi tỷ lệ FAR và tỷ lệ FRR là bằng nhau. EER là một thước đo quan trọng của hiệu suất của hệ thống nhận dạng. Nó được đo lường dưới dạng phần trăm và thấp nhất là tốt nhất, vì nó chỉ ra rằng tỷ lệ lỗi chấp nhận và từ chối là gần như bằng nhau.
5. **EER Threshold**: Đây là ngưỡng quyết định tương ứng với EER, nơi mà tỷ lệ FAR và FRR là bằng nhau. Nó là giá trị ngưỡng mà khi áp dụng cho quyết định của hệ thống nhận dạng, tỷ lệ lỗi chấp nhận và từ chối là gần như bằng nhau. EER Threshold là giá trị quyết định mà khi bạn vượt qua nó, mẫu sẽ được chấp nhận, và khi bạn dưới nó, mẫu sẽ bị từ chối.

```
def recognize_speaker(audio_file):
    similarities_dict = {}
    encoder = my_neural_network.MyEncoder().encoder

    features = features_extraction.extract_mfcc(audio_file)
    recognizing_embedding = inference.my_inference(features, encoder)

    if recognizing_embedding is None:
        return None

    enrolled_embeddings = crud.getAllEmbeddings()
    for emb in enrolled_embeddings:
        emb_arr = crud.stringToArray(emb.embedding)
        similarity_score = cosine_similarity(recognizing_embedding, emb_arr)

        if emb.speaker_ssn not in similarities_dict:
            similarities_dict[emb.speaker_ssn] = similarity_score
        elif similarities_dict[emb.speaker_ssn] < similarity_score:
            similarities_dict[emb.speaker_ssn] = similarity_score

    print(similarities_dict)
```

```
    max_similarity = max(similarities_dict.items(), key=operator.itemgetter(1))[1]
    keys = [k for k,v in similarities_dict.items() if v==max_similarity]
    print(keys[0], max_similarity)

    if max_similarity < EER_THRESHOLD:
        return None
    return keys[0]
```

Dựa vào code trên có thể thấy, sau khi duyệt qua toàn bộ speaker có trong database, nếu góc cosine lớn nhất vẫn có giá trị nhỏ hơn **EER_THRESHOLD (0.577)** thì sẽ nhận diện là người nói chưa tồn tại trong database, còn không sẽ trả về danh tính của người nói có độ tương đồng cao nhất.

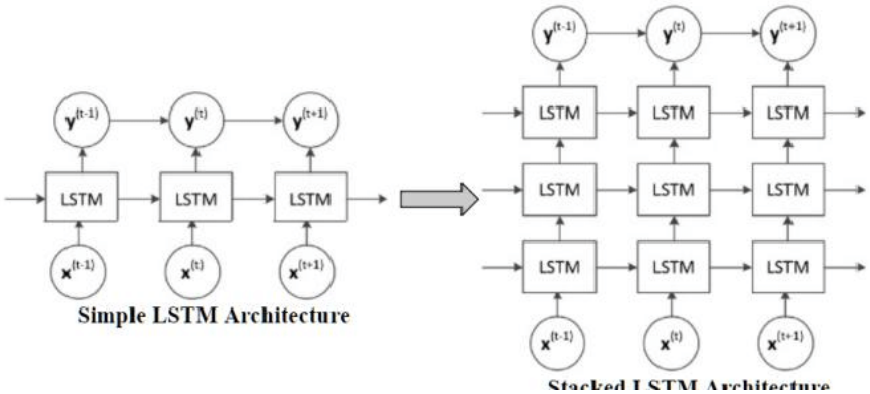
VII. Model Architecture



https://towardsdatascience.com/grus-and-lstm-s-741709a9b9b1

```
class LstmSpeakerEncoder(BaseSpeakerEncoder):

    def __init__(self, saved_model=""):
        super(LstmSpeakerEncoder, self).__init__()
        self.lstm = nn.LSTM(
            input_size=myconfig.N_MFCC,
            hidden_size=myconfig.LSTM_HIDDEN_SIZE,
            num_layers=myconfig.LSTM_NUM_LAYERS,
            batch_first=True,
            bidirectional=myconfig.BI_LSTM)
        if saved_model:
            self._load_from(saved_model)

    def _aggregate_frames(self, batch_output):
        if myconfig.FRAME_AGGREGATION_MEAN:
            return torch.mean(batch_output, dim=1, keepdim=False)
        else:
            return batch_output[:, -1, :]

    def forward(self, x):
        D = 2 if myconfig.BI_LSTM else 1
        h0 = torch.zeros(D * myconfig.LSTM_NUM_LAYERS, x.shape[0], myconfig.LSTM_HIDDEN_SIZE).to
        c0 = torch.zeros(D * myconfig.LSTM_NUM_LAYERS, x.shape[0], myconfig.LSTM_HIDDEN_SIZE).to
        y, (hn, cn) = self.lstm(x, (h0, c0))
        return self._aggregate_frames(y)
```

Kiến trúc mạng LSTM: Mô hình sử dụng một kiến trúc LSTM (Long Short-Term Memory) để xử lý hiệu quả tín hiệu âm thanh liên tục. LSTM là lựa chọn phù hợp để bắt các mối quan hệ dài hạn trong dữ liệu tuần tự như âm thanh. Trong kiến trúc này, sử dụng một LSTM bidirectional với 3 lớp LSTM được xếp chồng lên nhau để cải thiện hiệu suất của mô hình trong việc bắt các mẫu thời gian.



Lớp `LstmSpeakerEncoder` : Lớp này là lớp triển khai của mô hình. Trong hàm khởi tạo, một mô-đun LSTM được tạo ra với các tham số được cấu hình như số lượng hệ số MFCC (myconfig.N_MFCC), số lượng đơn vị ẩn trong mỗi lớp LSTM (myconfig.LSTM_HIDDEN_SIZE), số lượng lớp LSTM được xếp chồng (myconfig.LSTM_NUM_LAYERS), và cấu hình có sử dụng LSTM song hướng hay không (myconfig.BI_LSTM).

Truyền tiến (forward pass): Trong quá trình truyền tiến, đầu vào x được truyền qua lớp LSTM. Trạng thái ẩn ban đầu h0 và trạng thái tế bào ban đầu c0 được khởi tạo như các tensor chứa giá trị 0 có kích thước phù hợp. Đầu ra y và trạng thái ẩn cuối cùng hn và trạng thái tế bào cuối cùng cn được thu được từ lớp LSTM. Hàm _aggregate_frames được sử dụng để tổng hợp các khung đầu ra của lớp LSTM thành một biểu diễn cố định có độ dài, phụ thuộc vào giá trị của myconfig.FRAME_AGGREGATION_MEAN. Đầu ra được tổng hợp đại diện cho vectơ nhận dạng người nói cho chuỗi âm thanh đầu vào.

VIII. Model Training

```
def train_network(speaker_to_utterance, num_steps, saved_model="", pool=None):
    losses = []
    start_time = time.time()
    encoder = get_speaker_encoder()

    #Train
    optimizer = torch.optim.Adam(encoder.parameters(), lr=myconfig.LEARNING_RATE)
    start_time = time.time()
    print("Start training at:", time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(start_tim
e)))  # Thay đổi ở đây

    for step in range(num_steps):
        optimizer.zero_grad()

        #build batch input
```

```
        batch_input = feature_extraction.get_batched_triplet_input(speaker_to_utterance, mycon
fig.BATCH_SIZE, pool)
        batch_input = batch_input.to(myconfig.DEVICE)  # Chuyển đầu vào lên GPU
        batch_output = encoder(batch_input)     #batch_output.shape=[24,64*2]
        loss = get_triplet_loss_from_batch_output(batch_output, myconfig.BATCH_SIZE)
        loss.backward()
        optimizer.step()
        losses.append(loss.item())
        print(f"step: {step}/{num_steps} loss: {loss.item()}")

    #     saving model
        if saved_model is not None and (step + 1) % myconfig.SAVE_MODEL_FREQUENCY == 0:
            checkpoint = saved_model
            if checkpoint.endswith(".pt"):
                checkpoint = checkpoint[:-3]
            checkpoint += ".ckpt-" + str(step + 1) + ".pt"
            save_model(checkpoint,encoder, losses, start_time)

    training_time = time.time() - start_time
    print("End training at:", time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(time.time())))
# Thay đổi ở đây
    print("Finished training in", training_time, "seconds")

    if saved_model is not None:
        save_model(saved_model, encoder, losses, start_time)
    return losses


def run_training():
    print("Training data:", myconfig.TRAIN_DATA_DIR)
    speaker_to_utterance = dataset.get_librispeech_speaker_to_utterance(myconfig.TRAIN_DATA_DI
R)

    with multiprocessing.Pool(myconfig.NUM_PROCESSES) as pool:
        losses = train_network(speaker_to_utterance,
                               myconfig.TRAINING_STEPS,
                               myconfig.SAVED_MODEL_PATH,
                               pool)
    plt.plot(losses)
    plt.xlabel("step")
    plt.ylabel("loss")
    plt.show()
```

Các bước huấn luyện mô hình:

1. Khởi tạo: Bộ mã hóa người nói, quan trọng cho mô hình, được khởi tạo bằng hàm `get_speaker_encoder_LSTM()`, lấy ra bộ mã hóa người nói dựa trên LSTM đã được định nghĩa trước đó.

2. Định nghĩa Bộ Tối Ưu Hóa: Bộ tối ưu hóa Adam được chọn để huấn luyện mô hình. Nó được cấu hình với các tham số của bộ mã hóa, và tỷ lệ học là `myconfig.LEARNING_RATE`.

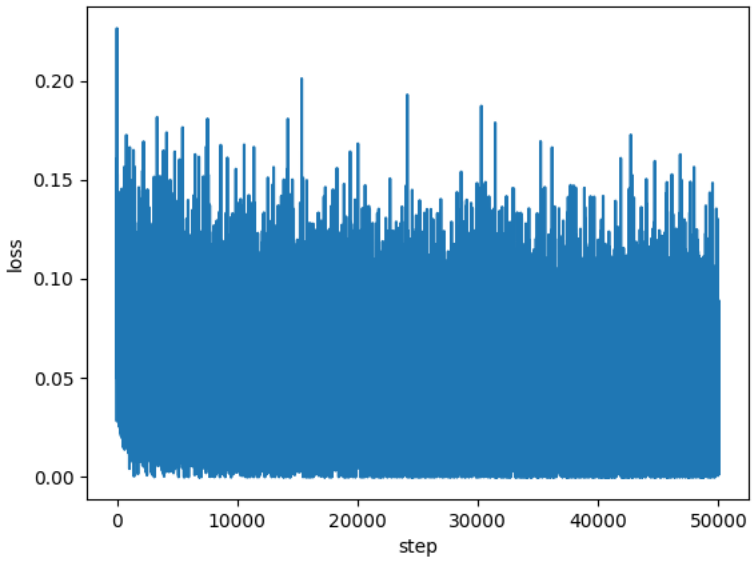
3. Vòng Lặp Huấn Luyện: Vòng lặp huấn luyện lặp qua `num_steps`, đại diện cho tổng số bước huấn luyện. Trong mỗi lần lặp:

- Độ dốc của bộ tối ưu hóa được thiết lập lại về 0 (`optimizer.zero_grad()`) để xóa bỏ bất kỳ độ dốc đã tích luỹ từ các bước trước.
- Chuẩn bị Đầu Vào Batches: Một batch của các đầu vào triplet được xây dựng từ từ điển `speaker_to_utterance` bằng cách sử dụng `feature_extraction.get_batched_triplet_input()`. Hàm này ngẫu nhiên chọn các đoạn âm thanh anchor, positive và negative từ các người nói khác nhau và hình thành một batch đầu vào, với kích thước batch được xác định bởi `myconfig.BATCH_SIZE`. **Mỗi step sẽ lấy 8 triplets để tạo thành một batch để huấn luyện mô hình.**
- Truyền Xuôi và Tính Toán Mất Mát: Đầu vào batch được truyền qua bộ mã hóa để thu được đầu ra batch, đại diện cho việc nhúng người nói. Mất mát triplet được tính toán bằng `get_triplet_loss_from_batch_output()`, dựa trên đầu ra batch và kích thước batch (`myconfig.BATCH_SIZE`).
- Lan Truyền Ngược và Cập Nhật Tham Số: Mất mát được lan truyền ngược qua mô hình, và bộ tối ưu hóa thực hiện cập nhật tham số dựa trên các độ dốc này.
- Theo Dõi Mất Mát: Giá trị mất mát hiện tại được thêm vào danh sách `losses` để theo dõi tiến trình huấn luyện.

4. Lưu Mô Hình: Nếu đường dẫn `saved_model` được cung cấp và bước hiện tại là bội số của `myconfig.SAVE_MODEL_FREQUENCY`, mô hình được lưu vào tệp checkpoint.

5. Hoàn Thành Huấn Luyện: Sau khi hoàn thành vòng lặp huấn luyện, tổng thời gian huấn luyện được tính toán, và giá trị mất mát cuối cùng được hiển thị. Nếu đường dẫn `saved_model` được cung cấp, mô hình đã được huấn luyện, cùng với các mất mát và thời gian bắt đầu, sẽ được lưu.

Quá trình huấn luyện mô hình nhận dạng người nói giúp **tối ưu hóa các tham số của mô hình** sao cho nó **cực tiểu hóa khoảng cách giữa các embedding của các mẫu âm thanh cùng một người nó**i và **tối đa hóa khoảng cách giữa các embedding của các người nói khác nhau**. Mục tiêu của mất mát triplet trong quá trình này là đảm bảo rằng các embedding của các mẫu âm thanh từ **cùng một người nói được gần nhau** trong không gian nhúng, trong khi các embedding từ các **người nói khác nhau được phân tách xa nhau**. Điều này giúp mô hình học được các đặc trưng của từng người nói và từ đó có khả năng phân biệt giữa chúng.



`Loss sau khi train 50000 steps`

IX. Model Evaluation Statistics

Model	Num Eval Triplets	Average EER	Best EER	Average Thresho
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt (best chị Nhi)	**1000**	**0.1733**	**0.1605**	**0.5760**
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	1000	0.2978	0.2865	0.9134
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	1000	0.3021	0.2890	0.9042
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	**1000**	**0.1878**	**0.1745**	**0.7316**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**1000**	**0.1398**	**0.1285**	**0.8934**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	1000	0.1363	0.1280	0.8951
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	1000	0.1604	0.1425	0.9379
mfcc_lstm_model_360h_50000epochs_specaug_8batch_4stacks_cpu.pt	1000	0.1440	0.1335	0.9111

```
import numpy as np
import torch
import torch.nn as nn
import time
import os
import matplotlib.pyplot as plt
import torch.optim as optim
import csv
from collections import defaultdict

from inference import my_inference
import multiprocessing
import multiprocessing.pool
import myconfig
import dataset
from evaluation import TripletScoreFetcher
from neural_net import get_speaker_encoder

def FRR(labels, scores, thres):
    fn = 0
    tp = 0
    for i in range(len(labels)):
        if scores[i] >= thres:
            if labels[i] == 0:
                fn += 1
            else:
                tp += 1
    if (tp+fn) == 0:
        return 0
    return fn / (tp+fn)

def FAR(labels, scores, thres):
    tn = 0
    fp = 0
    for i in range(len(labels)):
        if scores[i] < thres:
            if labels[i] == 0:
                tn += 1
            else:
```

```
                fp += 1
    if (tn+fp) == 0:
        return 0
    return fp / (fp+tn)

def compute_equal_error_rate2(labels, scores):
    min_delta = 1
    eer = 1
    thres = 0
    while thres < 1:
        FAR(labels, scores, thres)
        far_ = FAR(labels, scores, thres)
        frr_ = FRR(labels, scores, thres)
        delta_ = abs(far_ - frr_)
        if delta_ < min_delta:
            min_delta = delta_
            eer = (far_ + frr_) / 2
            thres += myconfig.EVAL_THRESHOLD_STEP
    return eer, thres

def compute_equal_error_rate(labels, scores):
    if len(labels) != len(scores):
        raise ValueError("Length of labels and scored must match")
    eer_threshold = None
    eer = None
    min_delta = 1
    threshold = 0.0
    while threshold < 1.0:
        accept = [score >= threshold for score in scores]
        fa = [a and (1-l) for a, l in zip(accept, labels)]
        fr = [(1-a) and l for a, l in zip(accept, labels)]
        far = sum(fa) / (len(labels) - sum(labels))
        frr = sum(fr) / sum(labels)
        delta = abs(far - frr)
        if delta < min_delta:
            min_delta = delta
            eer = (far + frr) / 2
            eer_threshold = threshold
        threshold += myconfig.EVAL_THRESHOLD_STEP

    return eer, eer_threshold

def compute_scores(encoder, dict_speaker, num_eval_triplets=myconfig.NUM_EVAL_TRIPLETS):
    labels = []
    scores = []
    fetcher = TripletScoreFetcher(dict_speaker, encoder, num_eval_triplets)
    with multiprocessing.pool.ThreadPool(myconfig.NUM_PROCESSES) as pool:
        while num_eval_triplets > len(labels)//2:
            label_score_pairs = pool.map(fetcher, range(len(labels)//2, num_eval_triplets))
            for triplet_labels, triplet_scores in label_score_pairs:
                labels.extend(triplet_labels)
                scores.extend(triplet_scores)
        pool.close()
```

```
    print("Evaluated", len(labels)//2, "triplets in total")
    return (labels, scores)

def run_eval(model_path):
    start_time = time.time()
    spk_to_utts = dataset.get_librispeech_speaker_to_utterance(myconfig.TEST_DATA_DIR)
    print("Evaluation data:", myconfig.TEST_DATA_DIR)

    encoder = get_speaker_encoder(model_path)
    labels, scores = compute_scores(encoder, spk_to_utts, myconfig.NUM_EVAL_TRIPLETS)
    eer, eer_threshold = compute_equal_error_rate(labels, scores)
    eval_time = time.time() - start_time
    print("Finished evaluation in", eval_time, "seconds")
    print("eer_threshold =", eer_threshold, "eer =", eer)
    return eer, eer_threshold

MODEL_PATHS = [
    # r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Sp
eaker Recognition\LSTM\saved_model\nhi's model\mfcc_2lstm_model_100k_specaug_batch_8_saved_mod
el.pt",
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spea
ker Recognition\LSTM\saved_model\nhi's model\mfcc_lstm_model_100k_specaug_batch_8_saved_model.
pt",
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spea
ker Recognition\LSTM\saved_model\train-clean-100-hours-100-epochs-specaug-8-batch-3-stacks-cpu
\mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt",
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spea
ker Recognition\LSTM\saved_model\train-clean-360-hours-100-epochs-specaug-8-batch-3-stacks-cpu
\mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt",
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spea
ker Recognition\LSTM\saved_model\train-clean-360-hours-20000-epochs-specaug-8-batch-3-stacks-c
pu\mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt",
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spea
ker Recognition\LSTM\saved_model\train-clean-360-hours-50000-epochs-specaug-8-batch-3-stacks-c
pu\mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt",
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spea
ker Recognition\LSTM\saved_model\train-clean-360-hours-50000-epochs-specaug-8-batch-3-stacks-g
pu\mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt",
    # r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Sp
eaker Recognition\LSTM\saved_model\train-clean-360-hours-50000-epochs-specaug-8-batch-4-stacks
-cpu\mfcc_lstm_model_360h_50000epochs_specaug_8batch_4stacks_cpu.pt",
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spea
ker Recognition\LSTM\saved_model\train-other-500-hours-50000-epochs-specaug-8-batch-3-stacks-c
pu\mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt",
]

MODEL_NAMES = [
    # "mfcc_2lstm_model_100k_specaug_batch_8_saved_model.pt",
    "mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt",
    "mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt",
    "mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt",
    "mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt",
    "mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt",
    "mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt",
```

```
    #"mfcc_lstm_model_360h_50000epochs_specaug_8batch_4stacks_cpu.pt",
    "mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt",

]

start_time = time.time()

# Chạy thống kê cho từng model
eer_results = defaultdict(list)
eer_threshold_results = defaultdict(list)

for i, model_path in enumerate(MODEL_PATHS):
    for j in range(10):  # Chạy 10 lần để lấy giá trị trung bình
        print(f"Running evaluation for model {model_path}, run {j+1}/10")
        eer, eer_threshold = run_eval(model_path)
        eer_results[MODEL_NAMES[i]].append(eer)
        eer_threshold_results[MODEL_NAMES[i]].append(eer_threshold)


end_time = time.time()
execution_time = end_time - start_time
print("Execution time:", execution_time, "seconds")


# Khai báo đường dẫn của file CSV
csv_file_path = "statistic-train-models.csv"

# Khởi tạo dữ liệu kết quả
data = [("Model", "Num Eval Triplets", "Average EER", "Best EER", "Average EER-Threshold", "Be
st EER-Threshold")]

# Thêm dữ liệu từ kết quả thống kê vào danh sách data
for model in MODEL_NAMES:
    num_eval_triplets = myconfig.NUM_EVAL_TRIPLETS
    avg_eer = sum(eer_results[model]) / len(eer_results[model])
    best_eer = min(eer_results[model])
    avg_eer_threshold = sum(eer_threshold_results[model]) / len(eer_threshold_results[model])
    best_eer_threshold = min(eer_threshold_results[model])

    data.append((model, num_eval_triplets, avg_eer, best_eer, avg_eer_threshold, best_eer_thre
shold))


# Ghi dữ liệu vào file CSV
with open(csv_file_path, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)

print("CSV file 'statistic-train-models.csv' has been created successfully.")
```

Trong ví dụ trên, EER =
0.1285 nghĩa là tỷ lệ lỗi là 12.85% trong cả hai tỷ lệ chấp nhận sai và tỷ lệ từ chối sai. EER_THRESHOLD = **0.73** nghĩa là nếu độ tương đồng cosine dưới 0.73 nghĩa là dữ liệu người nói chưa tồn tại trong database.

X. Model Testing Statistics

▼ **Sử dụng Model thuần túy**

Model	Test Dataset	EER-Threshold	Accurate Prediction	Tota Pred
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	**Data Tiếng nói tổng hợp**	**0**	**74**	**102**
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	**Data Tiếng nói để train**	**0**	**36**	**40**
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	**Data Tiếng nói để điều khiển nhà**	**0**	**38**	**62**
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	60	102
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	30	40
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	30	62
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	83	102
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	36	40
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	47	62
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	81	102
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	38	40
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	43	62
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói tổng hợp**	**0**	**85**	**102**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói để train**	**0**	**39**	**40**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói để điều khiển nhà**	**0**	**46**	**62**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói tổng hợp	0	76	102
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói để train	0	38	40
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói để điều khiển nhà	0	38	62
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	74	102

mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	37	40
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	37	62

```
import os
import torch
import neural_net
import inference
import myconfig
from collections import defaultdict
import csv

MODEL_PATHS = [
    # r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\S
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
    # r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\S
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
]

MODEL_NAMES = [
    # "mfcc_2lstm_model_100k_specaug_batch_8_saved_model.pt",
    "mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt",
    "mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt",
    "mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt",
    "mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt",
    "mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt",
    "mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt",
    #"mfcc_lstm_model_360h_50000epochs_specaug_8batch_4stacks_cpu.pt",
    "mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt",
]

DATASET_TEST_PATH = [
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
    r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_ai\AI Module\Spe
]

DATASET_NAMES = [
    "Data Tiếng nói tổng hợp",
    "Data Tiếng nói để train",
    "Data Tiếng nói để điều khiển nhà",
]

DATASET_BASE_PATH = r"D:\Code\BachKhoa\PBL 5\PBL05_smart_home_with_voice_print_and_antifraud_
```

```
SPEAKERS = [
    "Trí",
    "Đạt",
    "Tuấn",
    "Phát"
]

csv_file_path = "statistic-test-models-single-audio-file.csv"

data = [("Model", "Test Dataset", "EER-Threshold", "Accurate Prediction", "Total Prediction",

for i, model_path in enumerate(MODEL_PATHS):
    # Load pre-trained encoder
    encoder = neural_net.get_speaker_encoder(model_path)

    embedding_vector = defaultdict(list)
    for speaker in SPEAKERS:
        speaker_folder_path = os.path.join(DATASET_BASE_PATH, speaker)
        if os.path.exists(speaker_folder_path):
            audio_files = [f for f in os.listdir(speaker_folder_path) if f.endswith('.wav')]
            if audio_files:
                audio_file_path = os.path.join(speaker_folder_path, audio_files[0])
                embedding_vector[speaker] = inference.get_embedding(audio_file_path, encoder)
            else:
                print(f"No audio files found in folder {speaker_folder_path}")
        else:
            print(f"Folder {speaker_folder_path} does not exist")


    for j, dataset_test_path in enumerate(DATASET_TEST_PATH):
        total_prediction = 0
        accurate_prediction = 0

        for speaker in os.listdir(dataset_test_path):
            speaker_folder_path = os.path.join(dataset_test_path, speaker)
            for audio_file in os.listdir(speaker_folder_path):
                audio_file_path = os.path.join(speaker_folder_path, audio_file)
                audio_file_embedding = inference.get_embedding(audio_file_path, encoder)

                speaker_distance = defaultdict(lambda:0)
                for user in SPEAKERS:
                    speaker_distance[user] = inference.compute_distance(embedding_vector[user

                for user in SPEAKERS:
                    if speaker_distance[user] == min(speaker_distance.values()):
                        prediction = user
                        break

                accurate_prediction += 1 if prediction == speaker else 0
                total_prediction += 1

        data.append((MODEL_NAMES[i], DATASET_NAMES[j], 0, accurate_prediction, total_predicti

        print(f"Model: {MODEL_NAMES[i]}")
```

```
        # print(f"Seq length: {myconfig.SEQ_LEN}")
        print(f"Accurate prediction: {accurate_prediction}")
        print(f"Total prediction: {total_prediction}")
        print(f"Accuracy: {accurate_prediction / total_prediction*100}%  ")


# Ghi dữ liệu vào file CSV
with open(csv_file_path, mode='w', newline='') as file:
    writer = csv.writer(file)
    writer.writerows(data)

print("CSV file 'statistic-train-models-single-audio-file.csv' has been created successfully.
```

▼ **Sử dụng KNN**

Model	Test Dataset	EER-Threshold	N_Taken Audio	K-N Neig
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	**Data Tiếng nói tổng hợp**	**0**	**5**	**3**
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	**Data Tiếng nói để train**	**0**	**5**	**3**
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	**Data Tiếng nói để điều khiển nhà**	**0**	**5**	**3**
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	7	5
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	7	5
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	7	5
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	10	7
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	10	7
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	10	7
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	5	3
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	5	3
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	5	3
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	7	5
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	7	5
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	7	5

mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	10	7
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	10	7
mfcc_lstm_model_100h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	10	7
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	5	3
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	5	3
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	5	3
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	7	5
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	7	5
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	7	5
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	10	7
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	10	7
mfcc_lstm_model_360h_100epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	10	7
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	5	3
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	5	3
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	5	3
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	7	5
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	7	5
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	7	5
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	10	7
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	10	7
mfcc_lstm_model_360h_20000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	10	7
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói tổng hợp**	**0**	**5**	**3**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói**	**0**	**5**	**3**

	để train			
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói để điều khiển nhà**	**0**	**5**	**3**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	7	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	7	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	7	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	10	7
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	10	7
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	10	7
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói tổng hợp	0	5	3
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói để train	0	5	3
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói để điều khiển nhà	0	5	3
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói tổng hợp	0	7	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói để train	0	7	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói để điều khiển nhà	0	7	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói tổng hợp	0	10	7
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói để train	0	10	7
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_gpu.pt	Data Tiếng nói để điều khiển nhà	0	10	7
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	5	3
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	5	3
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	5	3
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	7	5
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	7	5
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	7	5

mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	10	7
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	10	7
mfcc_lstm_model_500h_10000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	10	7

▼ **Sử dụng KMean**

Model	Test Dataset	EER-Threshold	N_Taken Audio	K-C
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	**Data Tiếng nói tổng hợp**	**0**	**1**	**1**
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	**Data Tiếng nói để train**	**0**	**1**	**1**
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	**Data Tiếng nói để điều khiển nhà**	**0**	**1**	**1**
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	2	1
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	2	1
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	2	1
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	2	2
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	2	2
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	2	2
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	5	1
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	5	1
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	5	1
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	5	2
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	5	2
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	5	2
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	5	5
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	5	5
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	5	5

mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	10	1
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	10	1
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	10	1
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	10	2
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	10	2
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	10	2
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói tổng hợp	0	10	5
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để train	0	10	5
mfcc_lstm_model_100k_specaug_batch_8_saved_model.pt	Data Tiếng nói để điều khiển nhà	0	10	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	1	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	1	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	1	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	2	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	2	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	2	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói tổng hợp**	**0**	**2**	**2**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói để train**	**0**	**2**	**2**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói để điều khiển nhà**	**0**	**2**	**2**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	5	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	5	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	5	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	5	2
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói	0	5	2

	để train			
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	5	2
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	5	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	5	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	5	5
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	10	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	10	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	10	1
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói tổng hợp	0	10	2
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để train	0	10	2
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	Data Tiếng nói để điều khiển nhà	0	10	2
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói tổng hợp**	**0**	**10**	**5**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói để train**	**0**	**10**	**5**
mfcc_lstm_model_360h_50000epochs_specaug_8batch_3stacks_cpu.pt	**Data Tiếng nói để điều khiển nhà**	**0**	**10**	**5**

Thống kê kết quả tốt nhất của các phương pháp:

	Sử dụng Model thuần túy	Sử dụng KNN	Sử dụng KMean
Data Tổng hợp	**83.333**	**93.137**	**92.16**
Data Tiếng nói để train (20 giây)	**97.5**	**97.5**	**100.0**
Data Tiếng nói điều khiển nhà (3 giây)	**74.193**	**90.323**	**87.10**