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


**CO3117 - Học máy**

---

**Báo cáo Học phần Mở rộng**

# Ứng dụng giải thuật Viterbi vào nhận diện hợp âm bài hát

---

Giảng viên hướng dẫn: Lê Thành Sách

Sinh viên thực hiện:

| | |
|---|---|
| Nguyễn Thái Học | 2311100 |
| Nguyễn Thái Sơn | 2312968 |
| Lương Minh Thuận | 2313348 |
| Phạm Trần Minh Trí | 2313622 |

THÀNH PHỐ HỒ CHÍ MINH, 11/2025

# Mục lục

# 1 Giới thiệu

Phần này tóm tắt nội dung và phương pháp được triển khai trong bộ mã tham khảo (notebook kèm theo) cho bài toán nhận diện hợp âm từ file âm thanh sử dụng Hidden Markov Model và thuật toán Viterbi.

## 1.1 Bối cảnh

Nhận diện hợp âm (chord recognition) là một bài toán quan trọng trong lĩnh vực Music Information Retrieval (MIR) và xử lý tín hiệu âm thanh. Mục tiêu là dự đoán chuỗi hợp âm xuất hiện theo thời gian trong một bản nhạc. Ứng dụng bao gồm phân tích âm nhạc, tạo chord chart tự động, hỗ trợ học nhạc, và tìm kiếm bài hát theo tiến trình hợp âm.

## 1.2 Dataset

Dữ liệu sử dụng trong project này bao gồm 50 bài hát của The Beatles, mỗi bài gồm:

- File âm thanh định dạng MP3
- File annotation (.lab) chứa thông tin chord label theo thời gian với định dạng: `start_time end_time chord_label`

Dataset được chia thành 40 bài cho training và 10 bài cho testing theo phương pháp random shuffle với seed cố định (seed=72) để đảm bảo tính tái lập.

## 1.3 Tổng quan phương pháp

Hệ thống nhận diện hợp âm thực hiện pipeline gồm các bước chính sau:

1. **Exploratory Data Analysis (EDA)**: Phân tích phân phối hợp âm, thống kê các transition phổ biến, và visualize đặc trưng âm thanh.
2. **Tiền xử lý (Preprocessing)**:
   - Đơn giản hóa không gian trạng thái về 25 trạng thái (24 hợp âm cơ bản: 12 nốt $\times$ {maj, min} + trạng thái "N"cho no-chord)
   - Ánh xạ enharmonic (Db $\rightarrow$ C#, Eb $\rightarrow$ D#, v.v.)
   - Chia dataset thành train/test sets
3. **Trích xuất đặc trưng (Feature Extraction)**:

- Tách thành phần harmonic bằng HPSS (Harmonic-Percussive Source Separation)
- Trích xuất đặc trưng: Chroma CQT (12 chiều), Tonnetz (6 chiều), Spectral Contrast (7 chiều)
- Tính đạo hàm bậc 1 và bậc 2 (delta features) để bắt động thái thay đổi
- Áp dụng median filter để giảm nhiễu
- Chuẩn hóa L2 normalization
- Tổng cộng: $(12+6+7)\times 3 = 75$ chiều đặc trưng cho mỗi frame

4. **Căn chỉnh nhãn**: Ánh xạ nhãn hợp âm từ file .lab sang từng frame thời gian dựa trên hop length.
5. **Huấn luyện HMM**:
   - Ước lượng phân phối ban đầu $\pi$ từ trạng thái đầu tiên của mỗi bài
   - Tính ma trận chuyển trạng thái $A$ với Laplace smoothing ($\varepsilon = 10^{-8}$)
   - Huấn luyện Gaussian Mixture Models (GMM) cho mỗi trạng thái làm mô hình phát xạ $B$
6. **Dự đoán bằng Viterbi**: Áp dụng thuật toán Viterbi trong miền log-probability để tìm chuỗi hợp âm có xác suất cao nhất, sau đó làm mịn bằng median filter.
7. **Đánh giá (Evaluation)**: Tính accuracy, vẽ confusion matrix để đánh giá hiệu năng trên test set.

## 1.4 Xử lý tín hiệu số (DSP) trong trích xuất đặc trưng

Phần này trình bày chi tiết cách các kỹ thuật xử lý tín hiệu số được áp dụng trong code để chuyển đổi tín hiệu âm thanh thô thành các đặc trưng phù hợp cho mô hình học máy.

### 1.4.1 Sample Rate (Tần số lấy mẫu)

**Sample Rate** là số lượng mẫu (samples) được ghi lại mỗi giây từ tín hiệu âm thanh liên tục. Trong code, giá trị được thiết lập là:

```
SAMPLE_RATE = 22050  # Hz (samples/second)
```

**Ý nghĩa và lựa chọn:**

- Theo định lý Nyquist-Shannon, để tái tạo hoàn hảo tín hiệu gốc, tần số lấy mẫu phải ít nhất gấp đôi tần số cao nhất trong tín hiệu. Với $f_s = 22050$ Hz, ta có thể biểu diễn chính xác các tần số lên đến $\frac{22050}{2} = 11025$ Hz.
- Tai người nghe được từ khoảng 20 Hz đến 20 kHz. Tuy nhiên, với phân tích âm nhạc, đặc biệt là nhận diện hợp âm, các thành phần tần số quan trọng thường nằm dưới 10 kHz. Do đó, 22.05 kHz là một lựa chọn phù hợp giúp giảm khối lượng dữ liệu (so với 44.1 kHz chuẩn CD) mà vẫn giữ được thông tin âm nhạc cần thiết.
- Khi gọi `librosa.load(audio_path, sr=22050)`, nếu file âm thanh gốc có sample rate khác (ví dụ 44.1 kHz), `librosa` sẽ tự động resample (lấy mẫu lại) về 22050 Hz.

**Ảnh hưởng đến pipeline:**

- Sample rate quyết định độ phân giải tần số của tín hiệu số.
- Nó ảnh hưởng trực tiếp đến số lượng mẫu trong tín hiệu: với một bài hát dài $T$ giây, số mẫu là $N = T \times f_s$.
- Ví dụ: một bài hát 3 phút (180 giây) sẽ có $180 \times 22050 = 3{,}969{,}000$ mẫu.

### 1.4.2 Hop Length (Độ dịch chuyển cửa sổ)

**Hop Length** xác định số mẫu mà cửa sổ phân tích dịch chuyển giữa các frame liên tiếp khi tính toán biến đổi thời gian-tần số (như STFT hoặc CQT). Trong code:

```
HOP_LENGTH = 512  # samples
```

**Ý nghĩa vật lý:**

- Hop length quyết định độ phân giải thời gian của các đặc trưng được trích xuất.
- Với $h = 512$ mẫu và $f_s = 22050$ Hz, khoảng thời gian giữa hai frame liên tiếp là:

$$\Delta t = \frac{h}{f_s} = \frac{512}{22050} \approx 0.0232 \text{ giây} \approx 23.2 \text{ ms} \tag{1.1}$$

- Do đó, ta thu được khoảng $\frac{1}{0.0232} \approx 43$ frame mỗi giây (như ghi chú trong code: `# (~43 frames/sec)`).

**Trade-off giữa độ phân giải thời gian và tính toán:**

- *Hop length nhỏ* (ví dụ 128): Nhiều frame hơn mỗi giây $\Rightarrow$ độ phân giải thời gian cao (theo dõi biến đổi nhanh), nhưng tăng khối lượng tính toán và dữ liệu.
- *Hop length lớn* (ví dụ 2048): Ít frame hơn $\Rightarrow$ giảm tính toán, nhưng mất mát thông tin biến đổi nhanh (độ phân giải thời gian thấp).
- Giá trị 512 là một cân bằng phổ biến: đủ chi tiết để bắt được sự thay đổi hợp âm (thường diễn ra trong vài trăm ms) mà không quá nặng về tính toán.

### 1.4.3 Quá trình trích xuất đặc trưng từ tín hiệu âm thanh

Trong hàm `extract_features` của module `feature_extraction.py`, các bước DSP chính được thực hiện như sau:

**Bước 1: Nạp và resample tín hiệu**

```
y, sr = librosa.load(audio_path, sr=SAMPLE_RATE)
```

- y: mảng numpy 1D chứa các giá trị biên độ của tín hiệu âm thanh (thường được chuẩn hoá trong khoảng $[-1, 1]$).
- Chiều dài: $N = \text{len}(y)$ mẫu.

**Bước 2: Harmonic-Percussive Source Separation (HPSS)**

```
y_harm, _ = librosa.effects.hpss(y, margin=2.0)
```

- HPSS tách tín hiệu thành hai thành phần: harmonic (giai điệu, hợp âm) và percussive (nhịp, trống).
- Chỉ sử dụng thành phần harmonic `y_harm` cho việc trích xuất đặc trưng hợp âm, giúp loại bỏ nhiễu từ nhạc cụ gõ.
- Tham số `margin=2.0` kiểm soát mức độ tách biệt giữa hai thành phần.

**Bước 3: Trích xuất Chroma CQT**

```
chroma = librosa.feature.chroma_cqt(
    y=y_harm, sr=sr,
    hop_length=hop_length,
    bins_per_octave=48
).T
```

- **CQT** (Constant-Q Transform) là một biến đổi thời gian-tần số có độ phân giải tần số theo thang logarit, phù hợp với âm nhạc vì các nốt nhạc cách nhau theo tỷ lệ logarit.
- Chromagram tập hợp năng lượng của tất cả các octave về 12 lớp âm (pitch class): C, C#, D, ..., B.
- `bins_per_octave=48` cho độ phân giải cao (4 bins cho mỗi semitone).
- Đầu ra: ma trận $(12 \times M)$ với $M$ là số frame, sau đó transpose thành $(M \times 12)$.

**Bước 4: Trích xuất Tonnetz**

```
tonnetz = librosa.feature.tonnetz(
    y=y_harm, sr=sr, hop_length=hop_length
).T
```

- Tonnetz (Tonal Centroid Features) biểu diễn mối quan hệ hài hòa giữa các nốt nhạc dựa trên không gian tonal.
- Đầu ra: ma trận $(6 \times M)$, transpose thành $(M \times 6)$.
- Cung cấp thông tin bổ sung về cấu trúc hòa âm.

**Bước 5: Trích xuất Spectral Contrast**

```
spectral_contrast = librosa.feature.spectral_contrast(
    y=y_harm, sr=sr, hop_length=hop_length, n_bands=6
).T
```

- Spectral contrast đo sự chênh lệch giữa đỉnh và đáy trong mỗi băng tần số.
- Giúp phân biệt các texture âm thanh khác nhau.
- Đầu ra: ma trận $(7 \times M)$ (6 bands + 1 for low frequencies), transpose thành $(M \times 7)$.

**Bước 6: Median Filtering**

```
chroma = scipy_median_filter(chroma, size=(3, 1))
tonnetz = scipy_median_filter(tonnetz, size=(3, 1))
spectral_contrast = scipy_median_filter(spectral_contrast, size=(3, 1))
```

- Áp dụng median filter với cửa sổ kích thước 3 theo chiều thời gian để giảm nhiễu ngắn hạn.

- Giữ được các biên (edges) tốt hơn so với mean filter.

**Bước 7: Chuẩn hóa L2**

```
chroma = librosa.util.normalize(chroma, norm=2, axis=1)
tonnetz = librosa.util.normalize(tonnetz, norm=2, axis=1)
spectral_contrast = librosa.util.normalize(spectral_contrast, norm=2, axis=1)
```

- Mỗi vector đặc trưng (hàng) được chuẩn hóa L2: $\|\mathbf{x}\|_2 = 1$.
- Giúp đặc trưng không bị ảnh hưởng bởi độ lớn tổng thể của tín hiệu (loudness).

**Bước 8: Kết hợp đặc trưng cơ sở**

```
base = np.hstack([chroma, tonnetz, spectral_contrast])
```

- Ghép nối 3 loại đặc trưng theo chiều cột: $(M \times 12) + (M \times 6) + (M \times 7) = (M \times 25)$.

**Bước 9: Tính đạo hàm (Delta Features)**

```
delta = librosa.feature.delta(base.T)
delta2 = librosa.feature.delta(base.T, order=2)
```

- **Delta features** (đạo hàm bậc 1): bắt sự thay đổi của đặc trưng theo thời gian.
- **Delta-delta features** (đạo hàm bậc 2): bắt gia tốc của sự thay đổi.
- Được tính bằng convolution với kernel cố định, thường dùng trong speech/audio recognition.
- Cả hai đều có shape $(25 \times M)$, sau đó transpose về $(M \times 25)$.

**Bước 10: Ghép nối cuối cùng**

```
features = np.hstack([base, delta, delta2])
```

- Kết hợp đặc trưng cơ sở và delta features: $(M \times 25) + (M \times 25) + (M \times 25) = (M \times 75)$.
- Mỗi frame được biểu diễn bởi vector 75 chiều chứa thông tin về spectral content, tonal harmony, và động thái thay đổi.

### 1.4.4 Căn chỉnh nhãn với frame

Sau khi có ma trận đặc trưng $(M \times 75)$, cần ánh xạ nhãn hợp âm (từ file `.lab`) sang từng frame. Quá trình này được thực hiện trong hàm `load_labels`:

```
def load_labels(lab_file, n_frames, hop=512, sr=22050):
    labels = ["N"] * n_frames
    with open(lab_file, "r") as f:
        for line in f:
            start, end, chord_raw = ...
            chord = simplify_chord(chord_raw)
            s = int(start * sr / hop)
            e = int(end * sr / hop)
            for i in range(s, min(e, n_frames)):
                labels[i] = chord
    return labels
```

- File `.lab` chứa các dòng dạng: `start_time end_time chord_label`.
- Với mỗi khoảng thời gian $[\text{start}, \text{end})$ trong file nhãn, tính chỉ số frame tương ứng:

$$i_{\text{start}} = \left\lfloor \frac{\text{start} \times f_s}{h} \right\rfloor \tag{1.2}$$

$$i_{\text{end}} = \left\lfloor \frac{\text{end} \times f_s}{h} \right\rfloor \tag{1.3}$$

- Gán nhãn hợp âm (đã được đơn giản hóa) cho tất cả các frame trong khoảng $[i_{\text{start}}, i_{\text{end}})$.

**Ví dụ minh hoạ:**

- Giả sử một đoạn hợp âm C:maj kéo dài từ giây thứ 0.5 đến 2.0.
- Với hop length 512 và sample rate 22050:

$$i_{\text{start}} = \left\lfloor \frac{0.5 \times 22050}{512} \right\rfloor = \lfloor 21.53 \rfloor = 21 \tag{1.4}$$

$$i_{\text{end}} = \left\lfloor \frac{2.0 \times 22050}{512} \right\rfloor = \lfloor 86.13 \rfloor = 86 \tag{1.5}$$

- Các frame có index từ 21 đến 85 (65 frames) sẽ được gán nhãn `C:maj`.

## 1.5 Triển khai chi tiết và các lựa chọn kỹ thuật

### 1.5.1 Đơn giản hoá không gian trạng thái

- Nhãn gốc trong dataset có thể rất đa dạng (ví dụ: `C:maj7`, `Db:min`, `G:sus4`, `F#:dim`).
- Hàm `simplify_chord` trong `preprocessing.py` ánh xạ về 25 trạng thái:
  - 24 hợp âm cơ bản: 12 nốt $\times$ {major, minor}
  - Trạng thái `N`: không có hợp âm (silence, noise, no-chord)
- Ánh xạ enharmonic: Db $\rightarrow$ C#, Eb $\rightarrow$ D#, Gb $\rightarrow$ F#, Ab $\rightarrow$ G#, Bb $\rightarrow$ A# để tránh trùng lặp.
- Các hợp âm phức tạp (7th, 9th, sus, dim, aug) được đơn giản hóa về major hoặc minor tùy theo chord quality.
- Lý do: giảm độ phức tạp mô hình (25 trạng thái thay vì hàng trăm), giúp GMM huấn luyện tốt hơn với dữ liệu hạn chế (40 bài training).

### 1.5.2 Tính toán tham số HMM

Trong module `training.py`, hàm `calc_hmm_parameters` ước lượng các tham số:
**1. Phân phối ban đầu $\pi$:**

```
pi = np.zeros(n_states)
for label_seq in label_list:
    first = CHORD_TO_ID[simplify_chord(label_seq[0])]
    pi[first] += 1
pi = (pi + epsilon) / (pi + epsilon).sum()
```

- Đếm trạng thái đầu tiên của mỗi bài hát trong training set.
- Thêm smoothing $\varepsilon = 10^{-8}$ để tránh xác suất bằng 0.
- Chuẩn hóa: $\sum_{i=1}^{N} \pi_i = 1$.

**2. Ma trận chuyển trạng thái $A$:**

```
A = np.zeros((n_states, n_states))
for label_seq in label_list:
    for current, nxt in zip(label_seq[:-1], label_seq[1:]):
```

```
            cur_id = CHORD_TO_ID[simplify_chord(current)]
            nxt_id = CHORD_TO_ID[simplify_chord(nxt)]
            A[cur_id, nxt_id] += 1
A = (A + epsilon) / (A + epsilon).sum(axis=1, keepdims=True)
```

- Đếm số lần chuyển từ trạng thái $i$ sang trạng thái $j$ trong tất cả các bài hát.
- Thêm Laplace smoothing $\varepsilon = 10^{-8}$ cho mỗi phần tử.
- Chuẩn hóa theo hàng: $\sum_{j=1}^{N} A_{ij} = 1, \forall i$.
- Smoothing giúp tránh log(0) trong Viterbi và cho phép các transition hiếm vẫn có khả năng xảy ra.

**3. Mô hình phát xạ $B$ (GMM):** Hàm `train_GMM` huấn luyện một GMM cho mỗi trạng thái:

```
gmm = GaussianMixture(
    n_components=current_n_components,
    covariance_type='full',
    max_iter=100,
    random_state=72,
    n_init=3
)
gmm.fit(X_state)
```

- Tập hợp tất cả các vector đặc trưng 75 chiều thuộc cùng một trạng thái.
- Số components: 1-3 (được điều chỉnh tự động nếu trạng thái có ít samples).
- `covariance_type='full'`: ma trận hiệp phương sai đầy đủ $(75 \times 75)$ để bắt được tương quan giữa các chiều đặc trưng.
- GMM cho phép mô hình hóa phân phối multimodal: cùng một hợp âm có thể được chơi theo nhiều cách khác nhau (voicing, inversion, instrumentation).
- Nếu một trạng thái không có đủ samples, số components được giảm xuống hoặc GMM không được huấn luyện (gán `None`).

### 1.5.3 Thuật toán Viterbi trong miền log

Module `evaluation.py` implement hàm `viterbi_log` để tìm chuỗi trạng thái tối ưu:

```
def viterbi_log(pi, A, log_B):
    n_states = A.shape[0]
    T = log_B.shape[1]
    delta = np.zeros((T, n_states))
    phi = np.zeros((T, n_states), dtype=int)

    log_pi = np.log(pi + 1e-10)
    log_A = np.log(A + 1e-10)

    # Khởi tạo
    delta[0, :] = log_pi + log_B[:, 0]

    # Đệ quy
    for t in range(1, T):
        for j in range(n_states):
            temp = delta[t-1, :] + log_A[:, j]
            delta[t, j] = np.max(temp) + log_B[j, t]
            phi[t, j] = np.argmax(temp)

    # Backtracking
    q_star = np.zeros(T, dtype=int)
    q_star[T-1] = np.argmax(delta[T-1, :])
    for t in range(T-2, -1, -1):
        q_star[t] = phi[t+1, q_star[t+1]]

    return q_star
```

- Viterbi tìm $\arg\max_{s_1,\ldots,s_T} P(s_1,\ldots,s_T \mid x_1,\ldots,x_T)$.
- Tất cả tính toán trong miền log để tránh underflow khi nhân nhiều xác suất nhỏ:

$$\log P(s_1,\ldots,s_T,x_1,\ldots,x_T) = \log \pi_{s_1} + \sum_{t=2}^{T} \log A_{s_{t-1},s_t} + \sum_{t=1}^{T} \log B_{s_t}(x_t) \tag{1.6}$$

- GMM cung cấp `score_samples()` trả về log-likelihood trực tiếp.
- Ma trận $\delta[t, j]$ lưu log-xác suất cao nhất để đạt trạng thái $j$ tại thời điểm $t$.
- Ma trận $\phi[t, j]$ lưu trạng thái trước đó trên đường đi tối ưu (backpointer).
- Độ phức tạp: $O(T \times N^2)$ với $T$ là số frame, $N = 25$ trạng thái.

### 1.5.4 Post-processing với Median Filter

Sau khi Viterbi cho ra chuỗi trạng thái dự đoán, áp dụng median filter để làm mịn:

```
predicted_state_ids = viterbi_log(pi, A, log_B)
predicted_state_ids = medfilt(predicted_state_ids, kernel_size=5)
```

- Median filter với kernel size 5 loại bỏ các outlier (trạng thái nhảy đột ngột do nhiễu).
- Giữ được các biên hợp âm thực sự tốt hơn so với smoothing filter khác.
- Cải thiện tính mượt mà của chuỗi hợp âm dự đoán.

## 1.6 Tóm tắt workflow

Quy trình hoàn chỉnh từ raw audio đến chord prediction:

1. **EDA**: Phân tích dataset (50 bài Beatles) → hiểu phân phối chord, transition patterns
2. **Split**: 40 bài train, 10 bài test (random seed=72)
3. **Feature Extraction**: Audio → HPSS → Chroma + Tonnetz + Spectral Contrast → Median Filter → L2 Norm → Delta Features → 75-dim vectors
4. **Label Alignment**: .lab files → simplify chords → map to frame indices → 25 state IDs
5. **HMM Training**: Count transitions → $\pi$, $A$ with smoothing; Train GMMs → $B$
6. **Prediction**: Test audio → Extract features → Viterbi (log-domain) → Median filter → Chord sequence
7. **Evaluation**: Compare predictions vs ground truth → Accuracy, Confusion Matrix

Các phần tiếp theo sẽ trình bày chi tiết về cơ sở lý thuyết HMM, thuật toán Viterbi, kết quả thực nghiệm và đánh giá.

# 2 Cơ sở lý thuyết

## 2.1 Mô hình Markov ẩn (Hidden Markov Model - HMM)

### 2.1.1 Định nghĩa và thành phần

Hidden Markov Model là một mô hình xác suất thống kê được sử dụng để mô hình hoá các chuỗi có cấu trúc tuần tự, trong đó trạng thái thực sự (hidden state) không quan sát được trực tiếp, mà chỉ có thể suy luận thông qua các quan sát (observations).
Một HMM được định nghĩa bởi bộ năm thành phần $\lambda = (S, O, A, B, \pi)$:

- **Tập trạng thái ẩn** $S$: $S = \{s_1, s_2, \ldots, s_N\}$ với $N$ là số trạng thái. Trong bài toán của chúng ta, $N = 25$ trạng thái (24 hợp âm cơ bản: 12 nốt $\times$ {maj, min} + trạng thái `N` cho no-chord).
- **Tập quan sát** $O$: Không gian các giá trị quan sát có thể. Trong trường hợp này, mỗi quan sát là một vector đặc trưng 75 chiều: $o_t \in \mathbb{R}^{75}$ (bao gồm chroma, tonnetz, spectral contrast và các delta features).
- **Ma trận chuyển trạng thái** $A$: Ma trận $N \times N$ với $A_{ij} = P(s_{t+1} = j \mid s_t = i)$ biểu diễn xác suất chuyển từ trạng thái $i$ sang trạng thái $j$. Thoả mãn:

$$\sum_{j=1}^{N} A_{ij} = 1, \quad \forall i \in \{1, \ldots, N\} \tag{2.1}$$

- **Mô hình phát xạ** $B$: Tập hợp các phân phối xác suất $B = \{b_1, b_2, \ldots, b_N\}$, trong đó $b_i(o)$ là xác suất (hoặc mật độ xác suất) quan sát được $o$ khi đang ở trạng thái $i$. Trong implementation của chúng ta:

$$b_i(o) = P(o_t = o \mid s_t = i) \text{ được mô hình hoá bởi } \text{GMM}_i(o) \tag{2.2}$$

- **Phân phối trạng thái ban đầu** $\pi$: Vector $N$ chiều với $\pi_i = P(s_1 = i)$ là xác suất trạng thái ban đầu là $i$. Thoả mãn:

$$\sum_{i=1}^{N} \pi_i = 1 \tag{2.3}$$

### 2.1.2 Giả định Markov

HMM dựa trên hai giả định quan trọng:

**1. Tính Markov bậc nhất (First-order Markov Property):**

$$P(s_t \mid s_1, s_2, \ldots, s_{t-1}) = P(s_t \mid s_{t-1}) \tag{2.4}$$

Trạng thái hiện tại chỉ phụ thuộc vào trạng thái trước đó, không phụ thuộc vào các trạng thái xa hơn trong quá khứ.

**Ý nghĩa trong nhận diện hợp âm:** Hợp âm hiện tại có xu hướng liên quan mật thiết đến hợp âm ngay trước đó (ví dụ: trong tiến trình I-IV-V-I). Mặc dù giả định này là đơn giản hoá (trong thực tế, cấu trúc âm nhạc có thể phụ thuộc vào ngữ cảnh dài hơn), nhưng nó đủ hiệu quả cho nhiều ứng dụng và giúp giảm độ phức tạp tính toán.

**2. Tính độc lập quan sát (Output Independence):**

$$P(o_t \mid s_1, \ldots, s_T, o_1, \ldots, o_{t-1}, o_{t+1}, \ldots, o_T) = P(o_t \mid s_t) \tag{2.5}$$

Quan sát tại thời điểm $t$ chỉ phụ thuộc vào trạng thái tại $t$, độc lập với các quan sát và trạng thái khác.

**Ý nghĩa:** Vector chroma tại một frame chỉ được sinh ra từ hợp âm đang diễn ra tại frame đó, không chịu ảnh hưởng trực tiếp từ chroma của frame trước/sau.

### 2.1.3 Xác suất kết hợp

Với một chuỗi quan sát $O = (o_1, o_2, \ldots, o_T)$ và một chuỗi trạng thái ẩn $S = (s_1, s_2, \ldots, s_T)$, xác suất kết hợp được tính như sau:

$$P(O, S \mid \lambda) = \pi_{s_1} \cdot b_{s_1}(o_1) \cdot \prod_{t=2}^{T} A_{s_{t-1}, s_t} \cdot b_{s_t}(o_t) \tag{2.6}$$

Trong miền log (để tránh underflow):

$$\log P(O, S \mid \lambda) = \log \pi_{s_1} + \log b_{s_1}(o_1) + \sum_{t=2}^{T} \left[ \log A_{s_{t-1}, s_t} + \log b_{s_t}(o_t) \right] \tag{2.7}$$

## 2.2 Ba bài toán cơ bản của HMM

### 2.2.1 Bài toán 1: Đánh giá (Evaluation)

**Vấn đề:** Cho mô hình $\lambda$ và chuỗi quan sát $O$, tính $P(O \mid \lambda)$.

**Giải pháp:** Thuật toán Forward-Backward (hoặc chỉ Forward).

**Ứng dụng:** Đánh giá mức độ phù hợp của một chuỗi quan sát với mô hình.

### 2.2.2 Bài toán 2: Giải mã (Decoding)

**Vấn đề:** Cho mô hình $\lambda$ và chuỗi quan sát $O$, tìm chuỗi trạng thái $S^* = \arg\max_S P(S \mid O, \lambda)$ có xác suất cao nhất.
**Giải pháp:** Thuật toán Viterbi (sẽ trình bày chi tiết ở phần sau).
**Ứng dụng:** Đây chính là bài toán chúng ta giải quyết trong nhận diện hợp âm.

### 2.2.3 Bài toán 3: Học (Learning)

**Vấn đề:** Cho tập chuỗi quan sát, ước lượng tham số $\lambda$ sao cho $P(O \mid \lambda)$ đạt cực đại.
**Giải pháp:** Thuật toán Baum-Welch (Expectation-Maximization cho HMM).
**Trong project này:** Chúng ta không dùng Baum-Welch vì có sẵn nhãn (supervised learning). Thay vào đó, tham số được ước lượng trực tiếp từ dữ liệu có nhãn:

- $\hat{\pi}_i = \frac{\text{số bài hát bắt đầu bằng hợp âm } i}{\text{tổng số bài hát}}$
- $\hat{A}_{ij} = \frac{\text{số lần chuyển từ } i \text{ sang } j}{\text{số lần xuất hiện trạng thái } i}$
- $\hat{b}_i(o)$: huấn luyện GMM trên tất cả các vector chroma tương ứng với trạng thái $i$

## 2.3 Gaussian Mixture Model (GMM) cho mô hình phát xạ

### 2.3.1 Lý do sử dụng GMM

Vector đặc trưng 75 chiều của cùng một hợp âm có thể có phân phối phức tạp do:

- Nhiều cách voicing khác nhau (inversion, doubling notes)
- Âm sắc nhạc cụ đa dạng
- Nhiễu, harmonics phụ
- Biến thiên trong spectral contrast và tonnetz tùy theo instrumentation
- Động thái thay đổi được bắt bởi delta features

GMM cho phép mô hình hoá phân phối multimodal (nhiều đỉnh) bằng cách kết hợp nhiều Gaussian thành phần. Với không gian đặc trưng 75 chiều, việc sử dụng số lượng components nhỏ (1-3) giúp tránh overfitting trong khi vẫn bắt được sự đa dạng của dữ liệu.

### 2.3.2 Định nghĩa GMM

Một GMM với $K$ thành phần có mật độ xác suất:

$$b_i(o) = \sum_{k=1}^{K} w_{ik} \cdot \mathcal{N}(o \mid \mu_{ik}, \Sigma_{ik}) \tag{2.8}$$

trong đó:

- $w_{ik}$: trọng số (mixing coefficient) của thành phần $k$ trong GMM của trạng thái $i$, với $\sum_{k=1}^{K} w_{ik} = 1$
- $\mu_{ik} \in \mathbb{R}^{75}$: vector trung bình của thành phần $k$
- $\Sigma_{ik} \in \mathbb{R}^{75\times 75}$: ma trận hiệp phương sai (covariance matrix) của thành phần $k$
- $\mathcal{N}(o \mid \mu, \Sigma)$: phân phối Gaussian đa biến:

$$\mathcal{N}(o \mid \mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} \exp\left(-\frac{1}{2}(o-\mu)^T \Sigma^{-1}(o-\mu)\right) \tag{2.9}$$

### 2.3.3 Huấn luyện GMM

Trong code, mỗi GMM được huấn luyện bằng thuật toán Expectation-Maximization (EM) từ `sklearn.mixture.GaussianMixture`:

```
gmm = GaussianMixture(
    n_components=current_n_components,
    covariance_type='full',
    max_iter=100,
    random_state=72,
    n_init=3
)
gmm.fit(X_state)
```

- `n_components`: số thành phần Gaussian, được điều chỉnh tự động từ 1-3 tùy theo số lượng samples của mỗi trạng thái. Nếu một trạng thái có ít samples, số components sẽ được giảm xuống để tránh overfitting.
- `covariance_type='full'`: ma trận hiệp phương sai đầy đủ $(75 \times 75)$ để bắt được tương quan giữa các chiều đặc trưng (không giả định độc lập giữa các chiều)

- `random_state=72`: đảm bảo tính tái lập của kết quả
- `n_init=3`: số lần khởi tạo ngẫu nhiên khác nhau, chọn kết quả tốt nhất
- Phương thức `gmm.score_samples(o)`: trả về $\log b_i(o)$ cho vector quan sát $o$

## 2.4 Thuật toán Viterbi

### 2.4.1 Bài toán

Cho:

- Mô hình HMM $\lambda = (S, O, A, B, \pi)$
- Chuỗi quan sát $O = (o_1, o_2, \ldots, o_T)$

Tìm chuỗi trạng thái $S^* = (s_1^*, s_2^*, \ldots, s_T^*)$ sao cho:

$$S^* = \arg\max_S P(S \mid O, \lambda) = \arg\max_S P(S, O \mid \lambda) \tag{2.10}$$

### 2.4.2 Ý tưởng chính

Viterbi sử dụng quy hoạch động (dynamic programming) để tránh liệt kê tất cả $N^T$ chuỗi trạng thái có thể.
Định nghĩa biến trung gian:

$$V_t(j) = \max_{s_1,\ldots,s_{t-1}} P(s_1, \ldots, s_{t-1}, s_t = j, o_1, \ldots, o_t \mid \lambda) \tag{2.11}$$

$V_t(j)$ là xác suất cao nhất của đường đi từ thời điểm 1 đến $t$ kết thúc tại trạng thái $j$ và sinh ra các quan sát $o_1, \ldots, o_t$.

### 2.4.3 Công thức đệ quy

**Khởi tạo (t = 1):**

$$V_1(j) = \pi_j \cdot b_j(o_1), \quad \forall j \in \{1, \ldots, N\} \tag{2.12}$$

Trong miền log:

$$\log V_1(j) = \log \pi_j + \log b_j(o_1) \tag{2.13}$$

**Đệ quy (t = 2, ..., T):**

$$V_t(j) = \max_i \left[ V_{t-1}(i) \cdot A_{ij} \right] \cdot b_j(o_t) \tag{2.14}$$

Trong miền log:

$$\log V_t(j) = \max_i \left[\log V_{t-1}(i) + \log A_{ij}\right] + \log b_j(o_t) \tag{2.15}$$

Đồng thời, lưu lại con trỏ (backpointer) để truy vết ngược:

$$\text{Ptr}_t(j) = \arg\max_i \left[\log V_{t-1}(i) + \log A_{ij}\right] \tag{2.16}$$

**Kết thúc:** Trạng thái cuối cùng có xác suất cao nhất:

$$s_T^* = \arg\max_j \log V_T(j) \tag{2.17}$$

**Truy vết ngược (Backtracking):**

$$s_t^* = \text{Ptr}_{t+1}(s_{t+1}^*), \quad t = T-1, T-2, \ldots, 1 \tag{2.18}$$

### 2.4.4 Độ phức tạp tính toán

- Thời gian: $O(T \times N^2)$ (với $T$ là số frame, $N$ là số trạng thái)
- Không gian: $O(T \times N)$

So với việc liệt kê tất cả đường đi ($O(N^T)$), Viterbi hiệu quả hơn rất nhiều.

### 2.4.5 Ví dụ minh hoạ nhỏ

Giả sử:

- 2 trạng thái: C:maj (state 0), G:maj (state 1)
- 3 quan sát: $o_1, o_2, o_3$
- $\pi = [0.7, 0.3]$
- $A = \begin{bmatrix} 0.8 & 0.2 \\ 0.4 & 0.6 \end{bmatrix}$
- Emission probabilities đã biết: $b_0(o_1), b_1(o_1), \ldots$

**Bước 1 (t=1):**

$$\log V_1(0) = \log 0.7 + \log b_0(o_1) \tag{2.19}$$

$$\log V_1(1) = \log 0.3 + \log b_1(o_1) \tag{2.20}$$

**Bước 2 (t=2):**

$$\log V_2(0) = \max \begin{cases} \log V_1(0) + \log 0.8 \\ \log V_1(1) + \log 0.4 \end{cases} + \log b_0(o_2) \tag{2.21}$$

$$\log V_2(1) = \max \begin{cases} \log V_1(0) + \log 0.2 \\ \log V_1(1) + \log 0.6 \end{cases} + \log b_1(o_2) \tag{2.22}$$

(Tương tự cho $t = 3$, sau đó backtrack để tìm chuỗi trạng thái tối ưu)

# 3 Kết quả và đánh giá

## 3.1 Cấu hình thực nghiệm

### 3.1.1 Dataset

- Dữ liệu huấn luyện: Tập các file âm thanh (.mp3/.wav) kèm file annotation (.lab)
- Mỗi file .lab chứa thông tin: `start_time end_time chord_label`
- Sau khi trích xuất đặc trưng và căn chỉnh nhãn, dữ liệu được tổ chức thành:
  - `X_train_list`, `X_test_list`: Danh sách ma trận đặc trưng $(M_i \times 75)$ cho mỗi bài hát, với $M_i$ là số frame của bài thứ $i$. Mỗi frame là vector 75 chiều (chroma + tonnetz + spectral contrast + delta features)
  - `y_train_list`, `y_test_list`: Danh sách vector nhãn tương ứng, mỗi phần tử là chord label (string) cho từng frame
  - Tổng cộng: 40 bài train, 10 bài test (random seed=72)

### 3.1.2 Tham số mô hình

- Số trạng thái HMM: $N = 25$ (24 hợp âm cơ bản: 12 nốt $\times$ {maj, min} + trạng thái `N` cho no-chord)
- Sample rate: 22050 Hz
- Hop length: 512 samples ($\approx$ 23.2 ms, $\approx$ 43 frames/giây)
- Chiều vector đặc trưng: 75 (chroma 12 + tonnetz 6 + spectral contrast 7, mỗi loại có base + delta + delta$^2$)
- Số thành phần GMM mỗi trạng thái: 1-3 (adaptive, điều chỉnh tự động dựa trên số samples của mỗi trạng thái)
- Covariance type: `full` (ma trận $75 \times 75$)
- Random seed: 72 (cho tính tái lập)

## 3.2 Quy trình huấn luyện

### 3.2.1 Bước 1: Trích xuất đặc trưng và căn chỉnh nhãn

```
for audio_file in train_songs:
    features = extract_features(audio_file, hop_length=512)
    labels = load_labels(lab_file, n_frames, hop=512, sr=22050)
    X_train_list.append(features)  # shape: (M_i, 75)
    y_train_list.append(labels)    # length: M_i
```

Kết quả: Danh sách các cặp (75-dim features, aligned labels) cho mỗi bài hát trong tập train và test.

### 3.2.2 Bước 2: Tính toán tham số HMM

- **Ma trận** $A$: Đếm số lần chuyển trạng thái từ `y_train_list`, thêm Laplace smoothing $\varepsilon = 10^{-8}$, chuẩn hoá theo hàng
- **Vector** $\pi$: Đếm trạng thái đầu tiên của mỗi bài trong `y_train_list`, thêm smoothing, chuẩn hoá
- **GMM cho mỗi trạng thái**: Tập hợp tất cả vector 75 chiều thuộc trạng thái đó từ `X_train_list`, huấn luyện GMM với 1-3 components (adaptive)

Output:

```
A, pi = calc_hmm_parameters(y_train_list, n_states=25, epsilon=1e-8)
pi shape: (25,)
A shape: (25, 25)

GMM_Models = train_GMM(X_train_list, y_train_list,
                       n_states=25, n_components=3)
EMISSION_MODELS: 25 GMM models (một số có thể là None nếu thiếu data)
```

## 3.3 Quy trình dự đoán

Với một file âm thanh mới từ test set:

### 3.3.1 Bước 1: Trích xuất đặc trưng

```
features = extract_features(audio_file, hop_length=512)
```

Output: Ma trận đặc trưng $(M \times 75)$ với $M$ là số frame.

### 3.3.2 Bước 2: Tính log-likelihood

```
log_B = compute_log_likelihoods(GMM_Models, features)
```

Output: Ma trận log-likelihood $(25 \times M)$, trong đó `log_B[i, t]` $= \log P(x_t|s_t = i)$.

### 3.3.3 Bước 3: Áp dụng Viterbi

```
predicted_state_ids = viterbi_log(pi, A, log_B)
predicted_state_ids = medfilt(predicted_state_ids, kernel_size=5)
```

Output: Mảng state ID $(M,)$ sau khi làm mịn bằng median filter.

### 3.3.4 Bước 4: Chuyển đổi về nhãn hợp âm

```
predicted_chords = [ID_TO_CHORD.get(sid, 'N')
                    for sid in predicted_state_ids]
```

Output: Danh sách nhãn hợp âm (ví dụ: `['C:maj', 'C:maj', 'F:maj', ...]`).

### 3.3.5 Bước 5: Đánh giá

So sánh chuỗi dự đoán với ground truth:

```
acc, y_true_flat, y_pred_flat = calculate_accuracy(
    all_predicted_chords, all_true_labels
)
print(f"Overall Accuracy: {acc * 100:.2f}%")
```

Vẽ confusion matrix để phân tích chi tiết:

```
plot_confusion_matrix(y_true_flat, y_pred_flat, CHORD_STATES)
```

Confusion matrix cho thấy ma trận nhầm lẫn giữa 25 trạng thái, giúp xác định các cặp hợp âm thường bị nhầm lẫn (ví dụ: C:maj vs C:min, hoặc các hợp âm enharmonic).

## 3.4 Phân tích kết quả

### 3.4.1 Thống kê hợp âm dự đoán

Sau khi merge tất cả file dự đoán, notebook thực hiện phân tích đơn giản:

- Đếm tần suất xuất hiện của mỗi hợp âm
- Hiển thị top 10 hợp âm phổ biến nhất
- Tổng số frame được phân loại

**Nhận xét chung:**

- Các hợp âm phổ biến trong nhạc pop/rock (C:maj, G:maj, F:maj, A:min, D:min) thường xuất hiện nhiều hơn và có accuracy cao hơn do có nhiều training samples
- Trạng thái "N"(no-chord) có thể chiếm tỷ lệ đáng kể nếu có nhiều đoạn im lặng, intro/outro, hoặc đoạn chỉ có percussion
- Các hợp âm ít phổ biến (ví dụ: D#:maj, G#:min) có thể có accuracy thấp hơn do thiếu training data
- Vector đặc trưng 75 chiều (với tonnetz và spectral contrast) giúp phân biệt tốt hơn so với chỉ dùng chroma 12 chiều

### 3.4.2 Ưu điểm của phương pháp

- **Tận dụng thông tin chuỗi**: HMM mô hình hoá tiến trình hợp âm tự nhiên (ví dụ: I-IV-V-I), giúp dự đoán mượt mà hơn so với phân loại độc lập từng frame. Ma trận chuyển trạng thái $A$ học được các transition phổ biến từ dữ liệu.
- **Đặc trưng phong phú**: Vector 75 chiều kết hợp chroma (pitch content), tonnetz (harmonic relationships), spectral contrast (timbre), và delta features (temporal dynamics), cung cấp thông tin toàn diện hơn chỉ dùng chroma.
- **Xử lý nhiễu tốt**: GMM với 1-3 components (adaptive) và median filtering giúp giảm nhiễu ngắn hạn và xử lý được sự đa dạng của cùng một hợp âm.
- **Hiệu quả tính toán**: Viterbi trong miền log với $O(T \times N^2)$ khả thi cho real-time với $N = 25$ và $T$ vài nghìn frame. Median filter post-processing có độ phức tạp $O(T)$.
- **Supervised learning đơn giản**: Không cần Baum-Welch, chỉ cần đếm tần suất từ dữ liệu có nhãn. Dễ debug và interpret kết quả.

### 3.4.3 Hạn chế và hướng cải thiện

- **Không gian trạng thái rút gọn**: 25 trạng thái (chỉ maj/min) không thể biểu diễn các hợp âm phức tạp (7th, 9th, sus4, dim, aug, add9). Mở rộng không gian trạng thái cần dataset lớn hơn để tránh overfitting.
- **Giả định Markov bậc 1**: Chỉ xét trạng thái trước đó, không bắt được cấu trúc dài hạn (verse-chorus, bridge patterns). Có thể dùng higher-order HMM hoặc hierarchical models.
- **Phụ thuộc annotation quality**: Nếu file .lab có lỗi, không đồng nhất về chord notation, hoặc timing không chính xác, tham số HMM sẽ bị ảnh hưởng. Cần pre-processing và validation dữ liệu nhãn.
- **GMM components cố định**: Dù adaptive (1-3), vẫn chưa tối ưu cho từng trạng thái. Có thể dùng BIC/AIC để chọn số components riêng hoặc thử non-parametric models (e.g., DPGMM).
- **Dataset nhỏ**: Chỉ 50 bài Beatles (40 train, 10 test) chưa đủ đa dạng về thể loại, phong cách, nhạc cụ. Mô hình có thể không generalize tốt cho nhạc khác.
- **Evaluation metrics hạn chế**: Chỉ có frame-level accuracy. Nên thêm segment-level metrics, weighted accuracy (theo chord frequency), và perceptual evaluation.

## 3.5 Đề xuất cải tiến

1. **Mở rộng dataset**: Thu thập thêm dữ liệu từ nhiều thể loại (jazz, classical, pop, rock) và nguồn khác nhau (McGill Billboard, Isophonics) để tăng tính generalization.
2. **Mở rộng tập trạng thái**: Thêm các hợp âm phổ biến (7th, sus4, dim, aug) với vocab hierarchy (major/minor as base classes, extensions as variants).
3. **Duration modeling**: Dùng Hidden Semi-Markov Model (HSMM) hoặc thêm duration features để mô hình hoá explicit chord duration, tránh chuyển đổi quá nhanh.
4. **Advanced features**: Thêm beat-synchronous features (aggregate theo beat thay vì fixed hop), bass chroma (để detect inversions), hoặc learned features từ pre-trained models.
5. **Deep learning hybrid**: Dùng CNN/Transformer để học embeddings từ spectrogram, sau đó feed vào HMM hoặc CRF để capture temporal structure. Hoặc end-to-end với Bi-LSTM + CTC.

6. **Ensemble methods**: Kết hợp nhiều models (HMM với features khác nhau, GMM vs Deep models) bằng voting hoặc stacking để tăng robustness.

7. **Post-processing cải tiến**: Thay median filter bằng Conditional Random Field (CRF) hoặc Viterbi với language model constraints để enforce musical grammar.

8. **Evaluation toàn diện**: Thêm cross-validation, segment-level metrics (MIREX standards), ablation study (từng loại feature), và human evaluation (perceptual quality).

# 4 Kết luận

## 4.1 Tóm tắt công việc

Báo cáo này trình bày một hệ thống hoàn chỉnh cho bài toán nhận diện hợp âm từ tín hiệu âm thanh sử dụng Hidden Markov Model và thuật toán Viterbi. Các đóng góp chính bao gồm:

1. **Pipeline xử lý tín hiệu số phong phú**: Từ tín hiệu âm thanh thô đến vector đặc trưng 75 chiều kết hợp nhiều loại features (chroma CQT, tonnetz, spectral contrast) cùng delta features để bắt động thái thay đổi. Áp dụng HPSS để tách harmonic, median filtering để giảm nhiễu, và L2 normalization. Tham số DSP: sample rate 22050 Hz, hop length 512.

2. **Mô hình hoá HMM supervised với 25 trạng thái**: Ước lượng tham số HMM $(\pi, A)$ từ dữ liệu có nhãn với Laplace smoothing ($\varepsilon = 10^{-8}$). Sử dụng GMM adaptive (1-3 components tùy số samples) với full covariance để mô hình hoá emission probabilities.

3. **Giải mã Viterbi hiệu quả**: Implementation thuật toán Viterbi trong miền log-probability để tránh underflow, kết hợp median filter (kernel size 5) post-processing để làm mịn chuỗi dự đoán.

4. **Thiết kế modular functions**: Các hàm `extract_features()`, `load_labels()`, `calc_hmm_paramet` `train_GMM()`, `viterbi_log()` được tổ chức thành modules độc lập, dễ tái sử dụng và test.

5. **Đánh giá định lượng**: Thiết lập train/test split (40/10 bài, seed=72), tính frame-level accuracy, và visualize confusion matrix để phân tích chi tiết hiệu năng trên 25 trạng thái.

## 4.2 Ý nghĩa và ứng dụng

Hệ thống này có thể được ứng dụng trong:

- **Phân tích âm nhạc tự động**: Hỗ trợ nhạc sĩ, nhà nghiên cứu âm nhạc học phân tích cấu trúc hợp âm của bản nhạc.
- **Hệ thống học nhạc**: Tự động tạo chord chart cho người học guitar, piano.
- **Tạo nhạc tự động (music generation)**: Cung cấp thông tin hợp âm làm điều kiện cho mô hình sinh nhạc.
- **Music Information Retrieval (MIR)**: Làm nền tảng cho các tác vụ như tìm kiếm bài hát theo progression, phân loại thể loại dựa trên harmony.

## 4.3 Bài học kinh nghiệm

- **Tầm quan trọng của tham số DSP**: Lựa chọn sample rate và hop length ảnh hưởng trực tiếp đến chất lượng đặc trưng và hiệu suất tính toán. Cần cân bằng giữa độ phân giải thời gian (43 frames/giây) và chi phí tính toán.
- **Multi-feature fusion hiệu quả**: Kết hợp chroma (pitch content), tonnetz (harmonic relationships), spectral contrast (timbre) cùng delta features (temporal dynamics) cho accuracy tốt hơn đáng kể so với chỉ dùng chroma. Vector 75 chiều capture được nhiều khía cạnh của âm nhạc.
- **Trade-off giữa độ phức tạp mô hình và dữ liệu**: Rút gọn không gian trạng thái về 25 (chỉ maj/min) giúp tránh overfitting với dataset nhỏ (50 bài). Mở rộng thêm chord types cần dataset lớn hơn.
- **Smoothing quan trọng**: Laplace smoothing với $\varepsilon = 10^{-8}$ (rất nhỏ) vào ma trận $A$ và $\pi$ giúp tránh $\log(0)$ trong Viterbi mà không làm bias model quá nhiều. Giá trị quá lớn sẽ làm mất đi thông tin từ data.
- **GMM adaptive hiệu quả**: Điều chỉnh số components (1-3) theo số samples của mỗi trạng thái tránh overfitting cho rare chords. Full covariance matrix (75×75) bắt được correlation giữa các features nhưng tốn memory.
- **Post-processing quan trọng**: Median filter sau Viterbi loại bỏ outliers (chord nhảy đột ngột) hiệu quả hơn mean filter vì giữ được biên. Kernel size 5 là trade-off tốt giữa smoothing và preserving transitions.

## 4.4 Hướng phát triển tương lai

Các hướng nghiên cứu tiếp theo có thể bao gồm:

1. **Mở rộng không gian trạng thái**: Thêm các chord types phổ biến (7th, sus4, dim, aug) với hierarchical vocabulary hoặc factorized representations (root × quality). Cần augment dataset hoặc transfer learning để tránh overfitting.
2. **Kết hợp deep learning end-to-end**: Sử dụng Transformer hoặc Bi-LSTM trực tiếp trên spectrogram/CQT, với CTC loss hoặc attention mechanism. Hoặc hybrid approach: CNN extract features → HMM/CRF temporal modeling.

3. **Transfer learning cross-dataset**: Pre-train trên dataset lớn đa dạng (McGill Billboard, Isophonics, JAAH) để học robust representations, sau đó fine-tune hoặc few-shot learning cho domain cụ thể (genre-specific, artist-specific).

4. **Duration modeling explicit**: Dùng Hidden Semi-Markov Model (HSMM) hoặc add duration features để mô hình hoá chord duration distribution, tránh chuyển đổi quá nhanh (minimum duration constraints).

5. **Beat-synchronous analysis**: Aggregate features theo beats (từ beat tracker) thay vì fixed hop length. Chord changes thường align với beats, giúp reduce noise và tăng musical consistency.

6. **Segment-level metrics**: Thêm đánh giá segment-level (oversegmentation, undersegmentation, boundary precision/recall) theo MIREX standards, không chỉ frame-level accuracy. Weighted metrics theo chord frequency.

7. **Real-time streaming**: Optimize với Cython/numba, use incremental Viterbi (online decoding), và look-ahead buffer nhỏ. Target latency < 100ms cho live performance applications.

8. **Uncertainty quantification**: Output confidence scores cho mỗi prediction (từ Viterbi probabilities hoặc GMM posteriors), giúp downstream applications handle uncertain regions.

## 4.5 Lời kết

Báo cáo này đã trình bày chi tiết một hệ thống nhận diện hợp âm hoàn chỉnh sử dụng Hidden Markov Model, từ lý thuyết nền tảng (DSP, HMM, Viterbi) đến implementation cụ thể với Python và thư viện mã nguồn mở (librosa, scikit-learn, scipy). Các điểm nổi bật:

- **Feature engineering comprehensive**: Kết hợp 3 loại features (chroma, tonnetz, spectral contrast) với delta/delta-delta, tạo vector 75 chiều rich hơn nhiều so với chỉ dùng chroma 12 chiều.

- **Modeling careful**: HMM 25 states với GMM adaptive (1-3 components), Laplace smoothing chính xác ($10^{-8}$), và median filter post-processing cho kết quả stable.

- **Evaluation rigorous**: Train/test split có reproducibility (seed=72), frame-level accuracy, confusion matrix analysis trên Beatles dataset (50 songs).

Mặc dù phương pháp HMM/GMM là "classical"so với deep learning hiện đại, nó vẫn là nền tảng quan trọng giúp hiểu sâu về:

- Bản chất sequential của âm nhạc (temporal dependencies)
- Trade-offs trong model design (state space size, feature dimensionality, model complexity vs. data size)
- Probabilistic reasoning và dynamic programming

Code implementation modular và well-documented, dễ dàng extend cho các tác vụ tương tự (key detection, structure analysis). Hy vọng tài liệu này hữu ích cho người đọc quan tâm đến Music Information Retrieval, signal processing, và probabilistic graphical models.