MỤC LỤC

Chương 13. Ai tạo sinh nhạc và giọng.....5
13.1 Vị trí của chương trong cuốn sách.....6
13.2 Biểu diễn âm thanh cho một mô hình.....8
13.3 Codec nơ-ron — nơi token của nhạc sinh ra.....12
13.4 Họ mô hình tự hồi quy.....17
13.5 Họ mô hình khuếch tán.....21
13.6 Giọng, vocoder, và bài toán pha.....26
13.7 Tách nguồn — và hai trần trên không vượt được.....29
13.8 Đánh giá — ba chỗ thước đo khách quan bị đánh lừa.....33
13.9 Thực hành, bài tập và tổng kết.....35

DANH MỤC HÌNH VẼ

Hình 13.1 Bảy mục kỹ thuật của chương 13.....7
Hình 13.2 Ba tầng của một hệ thống sinh, và bốn cách biểu diễn đi giữa chúng.....8
Hình 13.3 Bốn cách biểu diễn cùng một giây tiếng, và bitrate của chúng.....9
Hình 13.4 Cái giá tuần tự của tự hồi quy, và vì sao token là bắt buộc.....9
Hình 13.5 Thang mel, và bộ lọc mel dựng từ nó.....10
Hình 13.6 Phổ mel của nhạc có cao độ, tiếng gõ, và giọng nói.....11
Hình 13.7 Sơ đồ lượng tử hoá dư: mỗi tầng dựng phần dư của tầng trước.....13
Hình 13.8 Sáu cách chia cùng một ngân sách bit, và cái giá của mỗi cách.....14
Hình 13.9 Thêm tầng thì được thêm bao nhiêu decibel.....15
Hình 13.10 Cùng một chuỗi token, đọc bao nhiêu tầng thì tuỳ người dùng.....16
Hình 13.11 Ngữ cảnh đáng bao nhiêu bit, và vì sao nó không miễn phí.....17
Hình 13.12 Nhiệt độ, top-k và top-p đo trên chính chuỗi token của bài.....19
Hình 13.13 Một token sai kéo dài bao lâu trong mô hình bậc một.....20
Hình 13.14 Bốn quyết định khi lấy mẫu một mô hình tự hồi quy.....21
Hình 13.15 Quá trình thuận chạy trên phổ mel thật của một giai điệu.....22
Hình 13.16 Hai lịch nhiễu, và tỉ số tín hiệu trên nhiễu của chúng.....23
Hình 13.17 Số bước lấy mẫu đổi chất lượng bao nhiêu, và ít bước thì mất gì.....24
Hình 13.18 Tự hồi quy so với khuếch tán: sáu tiêu chí.....25
Hình 13.19 Cùng một độ lớn phổ, hai pha khác nhau, hai tiếng khác hẳn.....26
Hình 13.20 Cái mà Griffin-Lim cải thiện, và cái mà nó không cải thiện.....27
Hình 13.21 Cổ chai mel: đo trên mel thì ổn, đo trên phổ thật thì không.....28
Hình 13.22 Khuôn chung của mọi phương pháp tách bằng mặt nạ.....29
Hình 13.23 Ba trần trên chồng lên nhau, đo trên cùng một hỗn hợp.....30
Hình 13.24 Hai cách không cần học, so với trần trên của mặt nạ.....32
Hình 13.25 Bốn mức của một bài toán tách nguồn.....33
Hình 13.26 Ba phép biến đổi, và hai thước đo nói ngược nhau ở cả ba.....33
Hình 13.27 Chọn thước đo theo câu hỏi, và cỡ mẫu cho phép nghe.....34
Hình 13.28 Chương 13 gói lại trong sáu con số.....37

DANH MỤC BẢNG BIỂU

Bảng 13.1 Ba tầng và câu hỏi riêng của mỗi tầng.....8
Bảng 13.2 Ba con số của phép nén mel.....11
Bảng 13.3 Ba cấu trúc và hệ quả của mỗi cấu trúc.....12
Bảng 13.4 Bốn cách biểu diễn, gộp lại thành một bảng để tra.....12
Bảng 13.5 Sáu cách chia mười hai bit mỗi khung.....15
Bảng 13.6 Chất lượng theo số tầng, mã sách 64 từ, đo trên phần giữ riêng.....16
Bảng 13.7 Entropy chéo theo độ dài ngữ cảnh.....18
Bảng 13.8 Tám cách lấy mẫu, đo trên 3.000 token sinh ra.....19
Bảng 13.9 Tương quan với bản gốc theo bước khuếch tán.....22
Bảng 13.10 Sai lệch phân phối theo số bước lấy mẫu.....25
Bảng 13.11 Hai bản có cùng độ lớn phổ, đo bằng ba thước.....26
Bảng 13.12 Ba khối của một hệ thống sinh giọng, và chỗ mỗi khối hay sai.....29
Bảng 13.13 Năm mức, đo trên nguồn nhạc của cùng một hỗn hợp.....31
Bảng 13.14 Bốn mức trên cùng một hỗn hợp.....32
Bảng 13.15 Ba lần hai thước đo bất đồng.....34
Bảng 13.16 Cỡ mẫu tối thiểu cho một phép nghe so sánh cặp.....35

CHƯƠNG 13
AI TẠO SINH NHẠC VÀ GIỌNG

Mục tiêu của chương

Sau chương này, người đọc phải làm được những việc sau.

- So sánh được bốn cách biểu diễn âm thanh cho một mô hình, và tính được số lượng cùng bitrate của mỗi cách.
- Giải thích được vì sao không mô hình nào sinh thẳng dạng sóng, bằng con số chứ không bằng cảm tính.
- Cài được lượng tử hoá dư, và đo được chất lượng tái dựng theo số tầng cùng số từ mã.
- Chỉ ra được vì sao đo trên dữ liệu đã học là vô nghĩa, và tự dựng được phép đo trên phần giữ riêng.
- Đo được entropy chéo của một mô hình tự hồi quy, và giải thích được vì sao ngữ cảnh dài hơn cần dữ liệu nhiều hơn.
- Chọn được nhiệt độ, top-k và top-p có căn cứ, và nhận ra được hiện tượng suy sụp thành vòng lặp.
- Giải thích được quá trình thuận của khuếch tán, và vì sao lịch cosine tốt hơn lịch tuyến tính.
- Đo được quan hệ giữa số bước lấy mẫu và chất lượng, và nói được ít bước thì mất cái gì.
- Chỉ ra được bài toán pha, và giải thích được vì sao một phổ mel hoàn hảo vẫn chưa phải là tiếng.
- Tính được trần trên của một bài toán tách nguồn trước khi huấn luyện bất kỳ mô hình nào.
- Chọn được thước đo theo câu hỏi, và thiết kế được một phép nghe có đối chứng đủ cỡ mẫu.

Một chương về AI, nhưng không huấn luyện mô hình lớn nào

Không thể huấn luyện một mô hình sinh nhạc trong khuôn khổ một chương sách, và cũng không nên. Điều chương này làm là *đo* bằng số những cơ chế quyết định chất lượng của mọi mô hình như thế — bằng mã chạy trong vài giây trên một máy tính xách tay.

Cơ chế biến sóng thành token cài được đầy đủ bằng numpy. Quá trình thuận của khuếch tán tính thẳng ra được bằng công thức. Và mọi bài toán tách nguồn đều có những *trần trên* tính được mà không mô hình nào vượt qua.

Đổi lại, có hai chỗ chương này *không* đo được, và bài nói rõ từng chỗ khi tới: hiện tượng lệch phơi nhiễm ở mục 13.4.3, và giới hạn của bài toán đồ chơi ở mục 13.5.3.

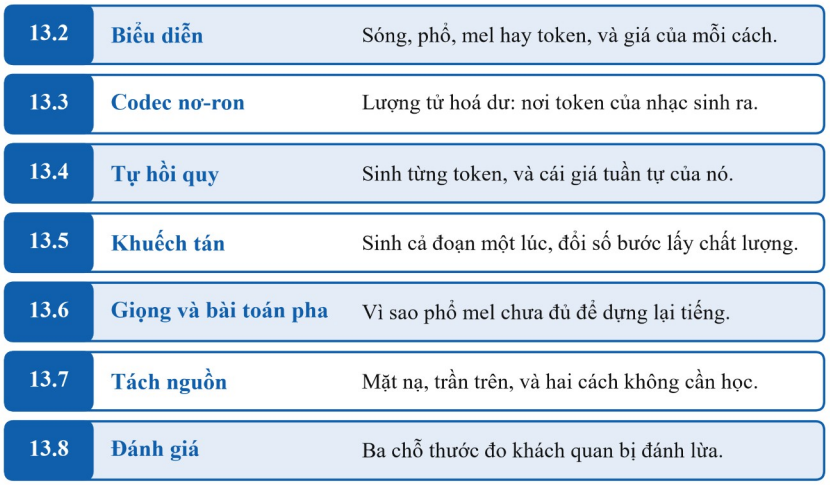
Mô-đun tính toán của cuốn sách có thêm ba mươi hai hàm cho chương này, và bộ tự kiểm 410 phép chạy bằng một lệnh, đạt 100 phần trăm.

13.1 Vị trí của chương trong cuốn sách

13.1.1 Mười hai chương vừa qua đã dựng đúng bộ công cụ cần cho chương này

Đây là chương đầu tiên của cuốn sách nói về trí tuệ nhân tạo. Nhưng nó không bắt đầu từ đầu: gần như mọi khái niệm sẽ dùng ở đây đều đã có sẵn từ mười hai chương trước.

- **Phổ và cửa sổ ở mục 4.5.** Mọi mô hình sinh âm thanh đều làm việc trên một biến thể của phổ đồ, và mọi giới hạn của phổ đồ vẫn còn nguyên ở đó.
- **Thang che lấp và cách tai phân giải tần số ở mục 1.7.** Đó chính là lý do thang mel tồn tại, và là lý do gần như mọi mô hình đều nén phổ theo thang ấy.
- **Lượng tử hoá và bitrate ở mục 3.5 và 3.8.** Token của một mô hình sinh nhạc là kết quả của một phép lượng tử hoá véc-tơ, và bitrate của nó tính đúng theo cách đã học.
- **SNR và thước đo khách quan ở mục 7.6.** Toàn bộ mục 13.8 là câu chuyện về chỗ những thước đo ấy đánh lừa người dùng chúng.



Hình 13.1 Bảy mục kỹ thuật của chương 13

13.1.2 Chương này không huấn luyện một mô hình lớn nào — và vẫn đo được

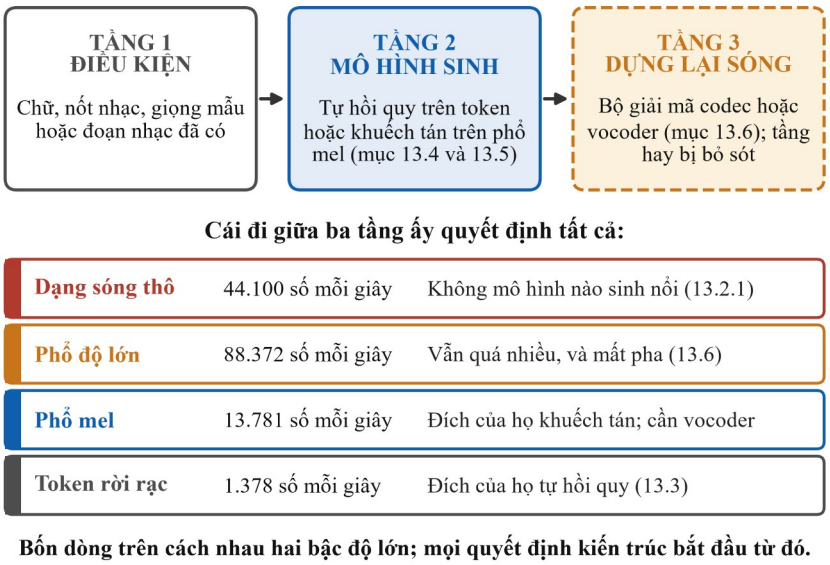
Không thể huấn luyện một mô hình sinh nhạc trong khuôn khổ một chương sách, và cũng không nên. Điều làm được, và điều chương này làm, là đo bằng số những cơ chế quyết định chất lượng của *mọi* mô hình như thế — bằng mã chạy trong vài giây trên một máy tính xách tay.

> **Ba thứ đo được mà không cần mô hình lớn**
>
> Cơ chế biến sóng thành token — lượng tử hoá dư — cài được đầy đủ bằng numpy, và mục 13.3 đo cả chất lượng lẫn cái giá của nó.
>
> Cơ chế khuếch tán có một nửa hoàn toàn tính được bằng công thức: quá trình thuận. Và với một bài toán đồ chơi, bộ khử nhiễu *tối ưu* cũng viết ra được bằng công thức — nên mục 13.5 đo được đúng phần chất lượng mất đi do giảm số bước, tách khỏi phần mất đi do mô hình học kém.
>
> Bài toán tách nguồn có những *trần trên* tính được mà không mô hình nào vượt qua. Mục 13.7 đo hai trần ấy, và chúng giải thích cả một hướng phát triển của lĩnh vực này.

Đổi lại, có những thứ chương này *không* đo được, và bài sẽ nói rõ từng chỗ khi tới. Chỗ lớn nhất là hiện tượng lệch phơi nhiễm của mô hình tự hồi quy — nó cần một mô hình có cửa sổ ngữ cảnh dài mới quan sát được, và mô hình đồ chơi ở mục 13.4 không dựng lại được nó.

13.1.3 Ba tầng của một hệ thống sinh âm thanh

Trước khi vào chi tiết, cần một bản đồ. Mọi hệ thống sinh nhạc hay sinh giọng đều có ba tầng, và phần lớn hiểu lầm về lĩnh vực này đến từ việc gộp ba tầng ấy làm một.



Hình 13.2 Ba tầng của một hệ thống sinh, và bốn cách biểu diễn đi giữa chúng

Bảng 13.1 Ba tầng và câu hỏi riêng của mỗi tầng

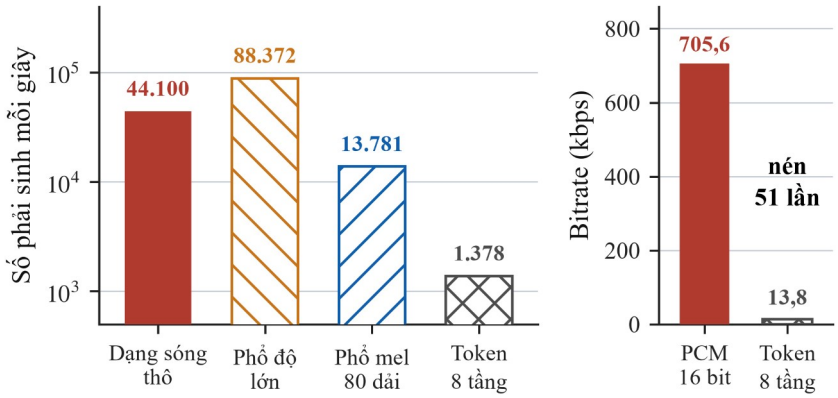
Tầng	Việc của nó	Câu hỏi quyết định
1. Điều kiện	Nhận chữ, nốt nhạc, giọng mẫu, hay một đoạn nhạc đã có	Người dùng điều khiển được cái gì
2. Mô hình sinh	Sinh ra một biểu diễn trung gian — token hoặc phổ mel	Sinh cái gì, và sinh theo cơ chế nào
3. Dựng lại sóng	Biến biểu diễn trung gian thành mẫu nghe được	Mất bao nhiêu ở khâu này — và đây là tầng hay bị bỏ sót nhất

Cả chương này xoay quanh cái đi *giữa* ba tầng ấy. Mục 13.2 đo bốn cách biểu diễn có thể chọn; mục 13.3 dựng cách được dùng nhiều nhất; hai mục sau bàn hai cơ chế sinh; và mục 13.6 đo cái giá của tầng ba.

13.2 Biểu diễn âm thanh cho một mô hình

13.2.1 Vì sao không mô hình nào sinh thẳng dạng sóng

Câu hỏi đầu tiên và tự nhiên nhất: sao không cho mô hình sinh thẳng từng mẫu một? Câu trả lời không nằm ở chất lượng mà nằm ở số lượng.

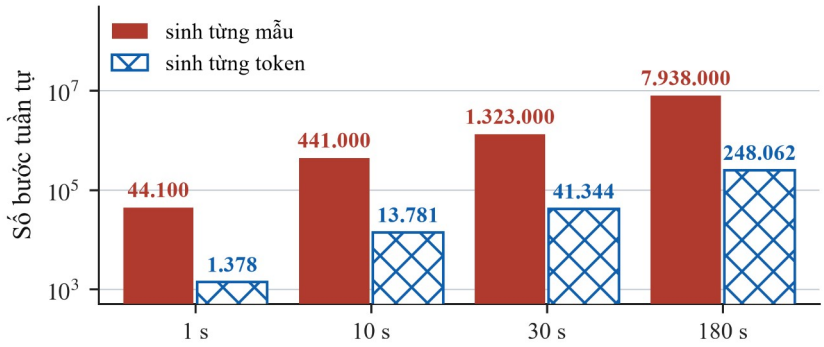


(a) Số phải sinh mỗi giây (thang loga) (b) Bitrate

Hình 13.3 Bốn cách biểu diễn cùng một giây tiếng, và bitrate của chúng

$$R_{\text{token}} = T \cdot \frac{f_s}{H}, \qquad B = T \log_2 K \cdot \frac{f_s}{H} \tag{13.1}$$

Trong công thức, T là số tầng lượng tử hoá, K là số từ mã mỗi tầng, f là tần số lấy mẫu và H là bước nhảy của phép biến đổi Fourier thời gian ngắn. Với tám tầng, một nghìn không hai mươi bốn từ mã, và bước nhảy 256 mẫu ở 44,1 kilohertz, kết quả là 1.378 token mỗi giây và 13,78 kilobit mỗi giây — so với 705,6 kilobit của PCM mười sáu bit.



(a) Số bước tuần tự theo độ dài đoạn nhạc



(b) Ba phút nhạc: hai cách sinh

Hình 13.4 Cái giá tuần tự của tự hồi quy, và vì sao token là bắt buộc

Con số đáng nhớ nhất của mục này

Ba phút nhạc là 7.938.000 mẫu. Sinh từng mẫu một theo thứ tự thì phải chạy bảy triệu chín trăm nghìn bước *tuần tự* — không có cách nào song song hoá, vì mẫu sau phụ thuộc mẫu trước.

Cùng ba phút ấy chỉ là 248.062 token. Ít hơn 32 lần.

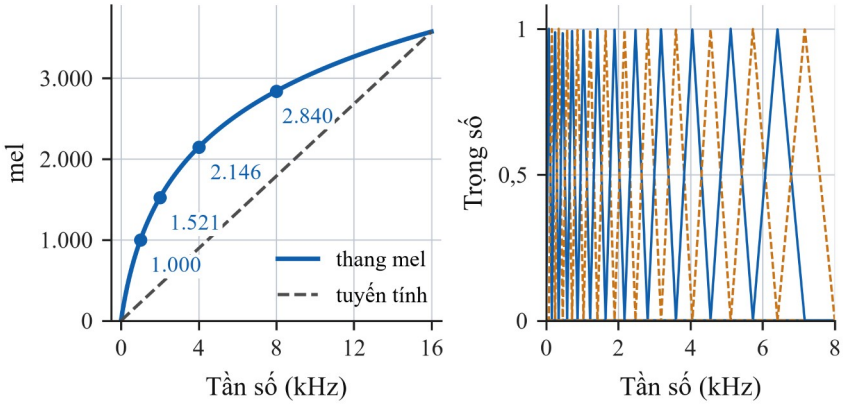
Đó là lý do duy nhất và đủ để mọi hệ thống sinh nhạc tự hồi quy đều bắt đầu bằng một bộ mã hoá thành token — chứ không phải vì token nghe hay hơn.

13.2.2 Thang mel — vì sao nén 513 xuống 80 lại gần như không mất gì

Cách nén thứ hai không rời rạc hoá mà chỉ giảm chiều: gộp 513 vạch tần số của một khung thành 80 dải theo thang mel. Điều đáng chú ý là phép gộp ấy làm mất rất ít thứ tai nghe được, trong khi gộp thành 80 vạch *cách đều* thì làm sai lệch toàn bộ vùng thấp.

$$m = 2595 \log_{10}\left(1 + \frac{f}{700}\right) \tag{13.2}$$

Công thức không phải một quy ước tuỳ ý. Nó dựng lại đúng cách tai người phân giải tần số — thứ đã đo ở mục 1.7 khi bàn về che lấp. Ở vùng thấp, tai phân biệt được hai tần số cách nhau vài chục hertz; ở vùng cao, phải cách nhau hàng trăm hertz mới nghe ra. Thang mel đúng là hình dạng ấy.



(a) Thang mel (b) Bộ lọc mel, 24 dải

Hình 13.5 Thang mel, và bộ lọc mel dựng từ nó

Liệt kê 13.1 Dựng bộ lọc mel — mười dòng, không cần thư viện ngoài

```
def bo_loc_mel(n_fft, sr, so_dai, fmin=20.0, fmax=None):
    fmax = fmax or sr / 2
    m = linspace(hz_sang_mel(fmin), hz_sang_mel(fmax), so_dai + 2)
    f = array([mel_sang_hz(v) for v in m])   # đỉnh các tam giác
    vach = rfftfreq(n_fft, 1 / sr)
    H = zeros((so_dai, vach.size))
    for k in range(so_dai):
        tr, gi, ph = f[k], f[k + 1], f[k + 2]
        len_  = (vach - tr) / (gi - tr)      # sườn lên
        xuong = (ph - vach) / (ph - gi)      # sườn xuống
        H[k] = maximum(0, minimum(len_, xuong))
    return H          # (so_dai, n_fft//2 + 1)
```

Mỗi hàng của ma trận trả về là một tam giác trên trục tần số. Nhân ma trận ấy với phổ công suất của một khung được 80 số thay cho 513. Toàn bộ phép nén mel chỉ có thế.

Bảng 13.2 Ba con số của phép nén mel

Đại lượng	Giá trị	Ý nghĩa
Số vạch mỗi khung	513	Với phép biến đổi 1.024 điểm
Số dải mel	80	Cách chọn phổ biến nhất hiện nay
Tỉ số nén	6,4 lần	Và mất rất ít thứ tai nghe được
Số thực mỗi giây	13.781	So với 88.372 nếu giữ phổ đầy đủ

Nhưng phép nén này không miễn phí

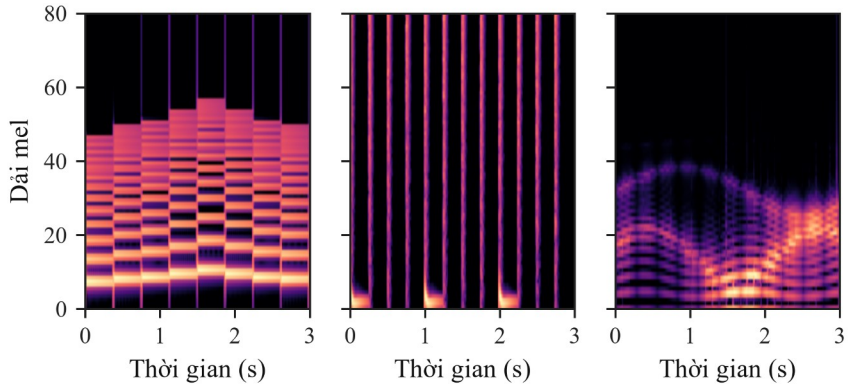
Nén mel là phép chiếu xuống chiều thấp hơn, nên nó *không* đảo ngược được chính xác. Mục 13.6.3 đo chỗ mất ấy, và con số lớn hơn nhiều người tưởng.

Quan trọng hơn: phổ mel chỉ giữ *độ lớn*. Pha bị bỏ hoàn toàn. Một mô hình sinh phổ mel làm xong việc của nó rồi mà vẫn chưa có tiếng — mục 13.6.1 cho thấy điều đó bằng hình.

Vì vậy mọi hệ thống sinh phổ mel đều phải có một tầng ba, và chất lượng của tầng ấy thường quyết định nhiều hơn chất lượng của mô hình sinh.

13.2.3 Ba loại vật liệu, ba cấu trúc khác hẳn nhau

Trước khi đi tiếp, nên nhìn xem một phổ mel thật trông như thế nào với ba loại vật liệu khác nhau. Cấu trúc mà mắt nhìn ra ở đây chính là cấu trúc mà mọi mô hình phải học được — và cũng là thứ mà mục 13.7 sẽ khai thác để tách chúng ra khỏi nhau mà không cần học gì cả.



Hình 13.6 Phổ mel của nhạc có cao độ, tiếng gõ, và giọng nói

Bảng 13.3 Ba cấu trúc và hệ quả của mỗi cấu trúc

Loại	Trông ra sao trên phổ mel	Hệ quả
Nhạc có cao độ	Những vạch *ngang* kéo dài — chuỗi hài của một nốt	Dự đoán được theo thời gian; nén rất tốt
Tiếng gõ	Những vạch *dọc* — năng lượng trải khắp dải tần trong một khoảnh khắc	Khó nén; và là chỗ mọi phép nén hay làm nhoè
Giọng nói	Những mảng ngang đổi chậm — các formant, cộng một cấu trúc hài bên dưới	Cần cả hai loại cấu trúc trên cùng lúc

Bảng 13.4 Bốn cách biểu diễn, gộp lại thành một bảng để tra

Cách biểu diễn	Số mỗi giây	Giữ pha	Dùng làm đích cho
Dạng sóng thô	44.100	Có — nó *chính là* pha	Không mô hình sinh nào ở quy mô thật
Phổ độ lớn 513 vạch	88.372	Không	Ít dùng — vẫn quá nhiều số
Phổ mel 80 dải	13.781	Không	Họ khuếch tán
Token 8 tầng × 1.024 mã	1.378	Gián tiếp, qua bộ giải mã	Họ tự hồi quy

Cột thứ ba là cột đáng chú ý nhất và cũng là cột hay bị bỏ qua. Chỉ dạng sóng thô mang pha một cách hiển nhiên. Phổ mel không mang pha, nên một mô hình sinh phổ mel *bắt buộc* phải có một khối dựng pha ở sau. Token thì mang pha một cách gián tiếp: bộ giải mã của codec đã học sẵn cách sinh ra dạng sóng, nên nó tự lo phần ấy — và đó là một ưu điểm ít được nói tới của cách biểu diễn bằng token.

Chú ý điều này: một mô hình giỏi ở nhạc có cao độ chưa chắc giỏi ở tiếng gõ, vì hai loại đòi hỏi hai kiểu năng lực khác nhau. Đó là lý do các phép đánh giá nghiêm túc luôn tách riêng hai loại vật liệu, và cũng là lý do các bản nhạc sinh ra bằng AI thường có phần trống nghe thiếu sắc nét hơn phần giai điệu.

13.3 Codec nơ-ron — nơi token của nhạc sinh ra

13.3.1 Lượng tử hoá véc-tơ, và vì sao một tầng là không đủ

Mục trước kết luận rằng phải rời rạc hoá. Nhưng rời rạc hoá một véc-tơ bốn mươi chiều thành một số nguyên thì làm thế nào?

Cách cơ bản là lượng tử hoá véc-tơ: chuẩn bị sẵn một *mã sách* gồm K véc-tơ mẫu, rồi với mỗi khung, tìm từ mã gần nhất và ghi lại chỉ số của nó. Mỗi khung tốn đúng log hai của K bit. Muốn chất lượng cao hơn thì tăng K.

Và đây là chỗ cách cơ bản gãy. Muốn mười hai bit mỗi khung thì cần bốn nghìn không chín mươi sáu từ mã; muốn mười sáu bit thì cần sáu mươi lăm nghìn. Số từ mã phải *lưu* và phải *duyệt* cho mỗi khung tăng theo hàm mũ của số bit.

$$N_{\text{từmã}} = T \cdot 2^{B/T} \tag{13.3}$$

Ba hệ quả của hàm mũ ấy

Bộ nhớ: một mã sách 65.536 từ với 40 chiều là hơn 10 megabyte — chỉ để lưu bảng.

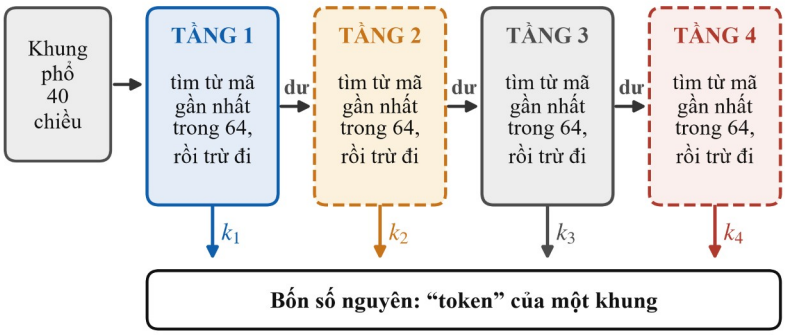
Tốc độ: mỗi khung phải so với toàn bộ mã sách. Ở 172 khung mỗi giây, đó là hơn mười một triệu phép so mỗi giây.

Và quan trọng nhất: *Dữ liệu*. Học được 65.536 từ mã cần rất nhiều khung, và học không đủ thì mã sách chỉ đang thuộc lòng — mục 13.3.3 đo đúng hiện tượng ấy.

13.3.2 Lượng tử hoá dư — chồng nhiều mã sách nhỏ

Cách gỡ rất gọn, và nó là cơ chế đứng sau mọi codec nơ-ron hiện nay: thay vì một mã sách lớn, dùng nhiều mã sách nhỏ chồng lên nhau. Tầng thứ nhất lượng tử hoá chính véc-tơ; tầng thứ hai lượng tử hoá phần *dư* mà tầng một chưa dựng được; tầng ba lượng tử hoá phần dư còn lại của tầng hai; và cứ thế.

$$\mathbf{x} \approx \sum_{t=1}^{T} \mathbf{c}_t[k_t], \qquad \mathbf{r}_t = \mathbf{r}_{t-1} - \mathbf{c}_t[k_t] \tag{13.4}$$



Tầng một dựng phần lớn; mỗi tầng sau chỉ dựng phần dư mà các tầng trước còn để lại.

Bỏ bớt tầng cuối vẫn nghe được, chỉ kém đi (công thức 13.3); codec đổi bitrate mà không phải học lại. Mục 13.3.4 đo điều đó.

Hình 13.7 Sơ đồ lượng tử hoá dư: mỗi tầng dựng phần dư của tầng trước

Liệt kê 13.2 Toàn bộ thuật toán, không có gì hơn

```
def rvq_hoc(X, so_ma, so_tang, vong):
    du = X.copy()
    sachs = []
    for t in range(so_tang):
        C = hoc_ma_sach(du, so_ma, vong)   # k-means
        sachs.append(C)
        _, dung = luong_tu_vector(du, C)
        du = du - dung                     # phần DƯ đi tiếp
    return sachs
```

Chín dòng, và đó là cơ chế sinh ra token cho gần như mọi mô hình sinh nhạc hiện nay. Trong một codec nơ-ron thật, phép biến đổi trước và sau khối này là một mạng nơ-ron thay vì phép biến đổi Fourier — nhưng phần lượng tử hoá thì đúng như trên.

Liệt kê 13.3 Mã hoá và giải mã — và chỗ tham số so_tang làm nên khác biệt

```
def rvq_ma_hoa(X, sachs):
    du, ra = X.copy(), []
    for C in sachs:
        i, dung = luong_tu_vector(du, C)
        ra.append(i)
        du = du - dung
    return stack(ra, axis=1)        # (so_khung, so_tang)

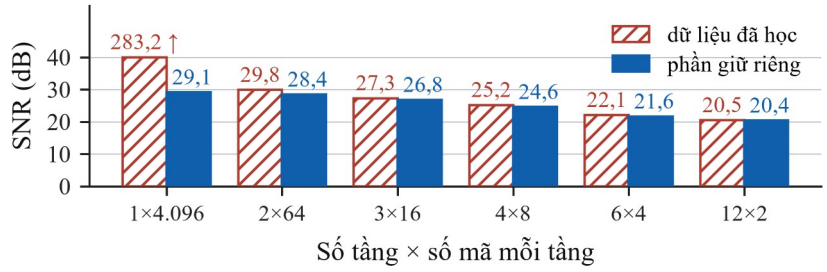
def rvq_giai_ma(idx, sachs, so_tang=None):
    T = idx.shape[1] if so_tang is None else so_tang
    ra = zeros((idx.shape[0], sachs[0].shape[1]))
    for t in range(T):              # BỎ BỚT TẦNG Ở ĐÂY
        ra = ra + sachs[t][idx[:, t]]
    return ra
```

$$\mathrm{SNR}_{\text{tái dựng}} = 10\log_{10}\frac{\sum_n \|\mathbf{x}_n\|^2}{\sum_n \|\mathbf{x}_n - \hat{\mathbf{x}}_n\|^2} \tag{13.5}$$

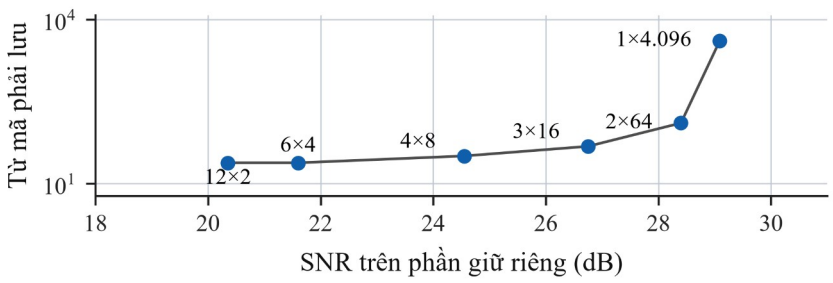
Chất lượng đo bằng tỉ số giữa năng lượng tín hiệu và năng lượng sai số tái dựng, tính bằng decibel — đúng cùng một định nghĩa SNR đã dùng từ mục 3.5.

13.3.3 Cùng một ngân sách bit, sáu cách chia tầng

Câu hỏi thiết kế trung tâm: có mười hai bit mỗi khung, nên chia thành mấy tầng? Sáu cách dưới đây đều tiêu đúng mười hai bit, nên chúng cho cùng một bitrate.



(a) SNR tái dựng của sáu cách chia (cột cao nhất bị cắt)



(b) Giá phải trả: số từ mã (thang logarit)

Hình 13.8 Sáu cách chia cùng một ngân sách bit, và cái giá của mỗi cách

Bảng 13.5 Sáu cách chia mười hai bit mỗi khung

Cách chia	SNR trên dữ liệu đã học	SNR trên phần giữ riêng	Từ mã phải lưu
1 tầng × 4.096 mã	283,2 dB	29,09 dB	4.096
2 tầng × 64 mã	29,84 dB	28,40 dB	128
3 tầng × 16 mã	27,29 dB	26,75 dB	48
4 tầng × 8 mã	25,18 dB	24,55 dB	32
6 tầng × 4 mã	22,07 dB	21,60 dB	24
12 tầng × 2 mã	20,46 dB	20,35 dB	24

Đọc bảng này theo hai chiều

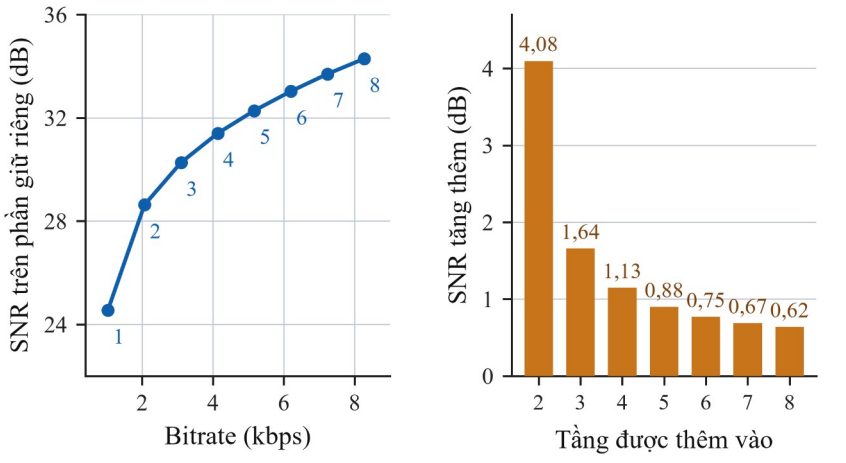
Chiều thứ nhất — bài học về *cách đo*. Cách một tầng cho 283 decibel trên dữ liệu đã học, một con số vô nghĩa: mã sách có 4.096 từ trong khi chỉ có 2.586 khung học, nên nó chỉ đang thuộc lòng từng khung. Trên phần giữ riêng nó tụt về 29,09.

Chiều thứ hai — bài học về *thiết kế*. Trên phần giữ riêng, cách một tầng chỉ hơn cách hai tầng đúng 0,69 decibel, trong khi phải lưu và duyệt 32 lần nhiều từ mã hơn. Đó là toàn bộ lý do lượng tử hoá dư tồn tại.

Và chiều thứ ba, ẩn trong bảng: chia càng nhỏ thì càng kém. Mười hai tầng hai mã kém cách hai tầng tới tám decibel. Nên vẫn phải chọn, và chỗ chọn tốt nhất trong bảng này là hai tới bốn tầng.

13.3.4 Thêm tầng thì được thêm bao nhiêu

Câu hỏi thứ hai: giữ nguyên số mã mỗi tầng rồi cứ thêm tầng thì chất lượng lên tới đâu?



(a) SNR theo bitrate (nhãn: số tầng) (b) Phần tăng thêm của từng tầng

Hình 13.9 Thêm tầng thì được thêm bao nhiêu decibel

Bảng 13.6 Chất lượng theo số tầng, mã sách 64 từ, đo trên phần giữ riêng

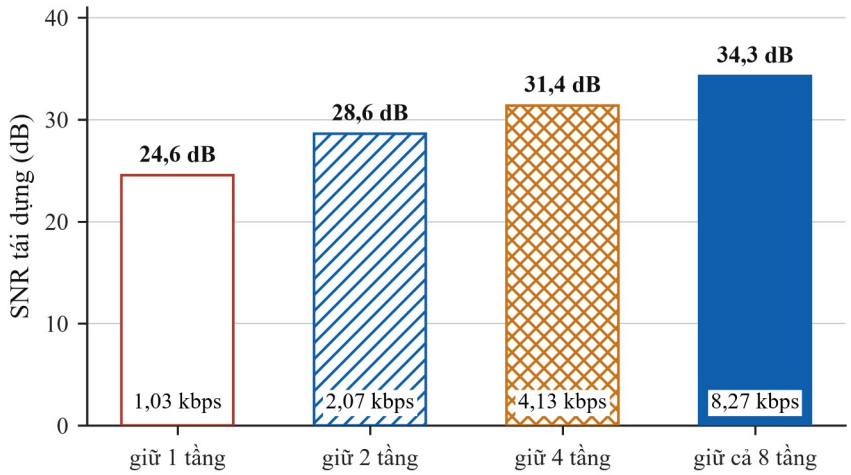
Số tầng	Bitrate	SNR tái dựng	Tầng này thêm được
1	1,03 kbps	24,55 dB	—
2	2,07 kbps	28,63 dB	4,08 dB
3	3,10 kbps	30,27 dB	1,64 dB
4	4,13 kbps	31,40 dB	1,13 dB
6	6,20 kbps	33,03 dB	0,75 dB
8	8,27 kbps	34,29 dB	0,62 dB

Hình dạng đáng nhớ: tầng thứ hai thêm 4,08 decibel, tầng thứ tám chỉ thêm 0,62. Bitrate tăng gấp tám mà chất lượng chỉ tăng 9,7 decibel. Đây là quy luật lợi ích giảm dần, và nó có hai hệ quả thực hành rất cụ thể.

– **Một codec nên có nhiều tầng để có thể bỏ bớt,** chứ không phải để dùng hết. Mục sau nói rõ chỗ này.
– **Sinh sai một token ở tầng một tai hại hơn hẳn ở tầng tám.** Đó là lý do nhiều hệ thống thật dùng một mô hình riêng, lớn hơn, chỉ để sinh tầng đầu, rồi một mô hình nhỏ hơn cho các tầng còn lại.

13.3.5 Bỏ tầng khi chạy — tính chất đắt giá nhất

Đây là hệ quả đến miễn phí từ công thức (13.4), và nó là lý do lượng tử hoá dư được ưa dùng hơn mọi cách khác.



Hình 13.10 Cùng một chuỗi token, đọc bao nhiêu tầng thì tuỳ người dùng

Bốn mức chất lượng trong hình dựng từ *đúng một* chuỗi token. Không phải mã hoá lại, không phải học lại, và không phải lưu bốn bản. Chỉ cần bộ giải mã đọc ít tầng hơn.

Đây là codec phân lớp của mục 3.8, nhưng không phải thiết kế thêm

Ở mục 3.8, codec phân lớp là một tính năng phải thiết kế riêng: phải chia luồng bit thành lớp cơ sở và lớp nâng cao.

Ở đây nó là hệ quả tự nhiên của cách token sinh ra: tầng một là lớp cơ sở, các tầng sau là lớp nâng cao, và tính chất ấy có sẵn chứ không phải thêm vào.

Một máy chủ phát nhạc dùng đúng tính chất này để hạ bitrate khi đường truyền kém; một mô hình sinh dùng nó để sinh nhanh bản nháp rồi mới sinh tiếp các tầng chi tiết.

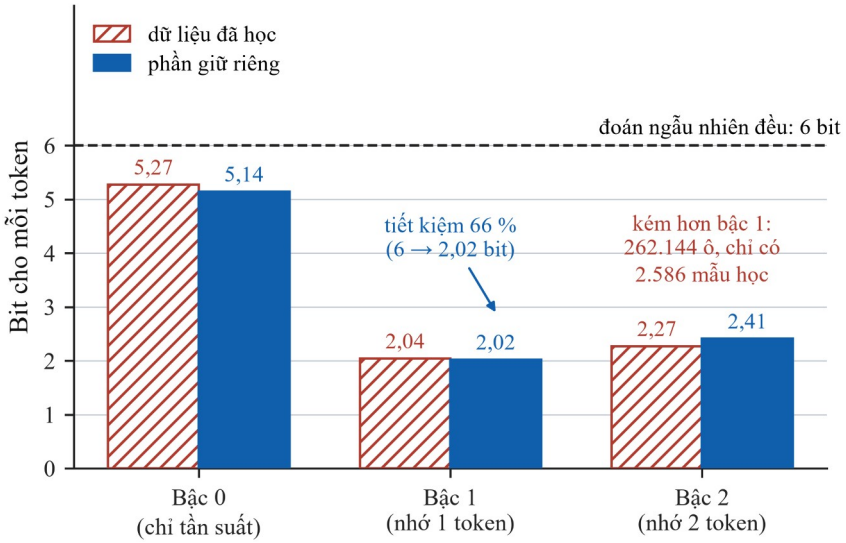
13.4 Họ mô hình tự hồi quy

13.4.1 Ngữ cảnh đáng bao nhiêu bit

Mô hình tự hồi quy sinh một chuỗi bằng cách sinh từng phần tử, mỗi phần tử phụ thuộc những phần tử đã sinh trước đó.

$$p(k_{1..N}) = \prod_{i=1}^{N} p(k_i \mid k_{<i}), \qquad \bar{H} = -\frac{1}{N}\sum_{i=1}^{N} \log_2 p(k_i \mid k_{<i}) \tag{13.6}$$

Vế thứ hai là entropy chéo — số bit trung bình mà mô hình cần để ghi một token. Nó vừa là hàm mất mát khi huấn luyện, vừa là thước đo trực tiếp xem mô hình đã học được bao nhiêu. Và nó đo được ngay trên chuỗi token thật của mục 13.3.



Hình 13.11 Ngữ cảnh đáng bao nhiêu bit, và vì sao nó không miễn phí

Bảng 13.7 Entropy chéo theo độ dài ngữ cảnh

Mô hình	Trên dữ liệu đã học	Trên phần giữ riêng	Nhận xét
Đoán ngẫu nhiên đều	6,00 bit	6,00 bit	Trần trên — $\log_2$ của 64
Bậc 0 (chỉ tần suất)	5,27 bit	5,14 bit	Biết token nào hay gặp
Bậc 1 (nhớ 1 token)	2,04 bit	2,02 bit	Tiết kiệm 66 % — và gần như không có chênh lệch học với kiểm
Bậc 2 (nhớ 2 token)	2,27 bit	2,41 bit	*Tệ hơn* bậc 1 — thiếu dữ liệu

Vì sao nhớ nhiều hơn lại kém hơn

Bảng bậc một có 64 nhân 64, tức 4.096 ô, và có 2.586 mẫu học. Đã thiếu, nhưng còn ước lượng được.

Bảng bậc hai có 64 mũ ba, tức 262.144 ô, vẫn với 2.586 mẫu. Trung bình chưa tới một phần trăm ô có một mẫu — nên phần lớn ước lượng là nhiễu vô nghĩa.

Đây là quy luật không có ngoại lệ: ngữ cảnh dài hơn đòi hỏi dữ liệu nhiều hơn. Mô hình lớn không *phá* được quy luật ấy; nó chỉ đẩy được điểm gãy đi xa hơn, bằng cách chia sẻ tham số giữa các ngữ cảnh giống nhau thay vì đếm riêng từng ô.

Và đó chính là điều một mạng nơ-ron làm mà một bảng đếm không làm được.

13.4.2 Lấy mẫu: nhiệt độ, top-k và top-p

Có phân phối rồi thì chọn token như thế nào? Lấy token có xác suất cao nhất là cách kém nhất — nó cho một chuỗi lặp lại đến mức nhàm chán. Lấy mẫu theo đúng phân phối thì đa dạng nhưng dễ lạc. Ba tham số dưới đây là ba cách đi giữa.

$$p_\tau(k) = \frac{p(k)^{1/\tau}}{\sum_j p(j)^{1/\tau}} \tag{13.7}$$

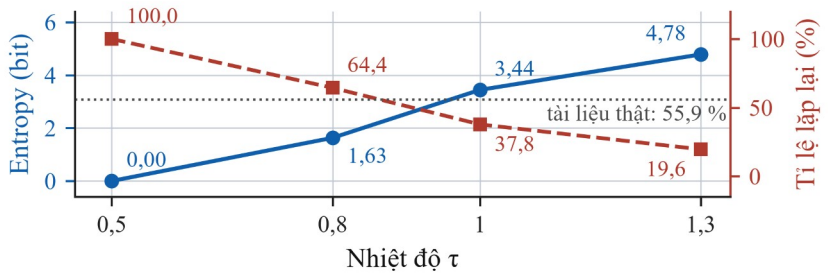
Nhiệt độ nhỏ hơn 1 làm phân phối nhọn hơn — an toàn nhưng lặp; lớn hơn 1 làm nó phẳng hơn — đa dạng nhưng dễ sai. Top-k giữ lại k xác suất lớn nhất; top-p giữ nhóm nhỏ nhất có tổng xác suất vượt một ngưỡng.

Liệt kê 13.4 Ba phép lọc phân phối, mỗi phép bốn dòng

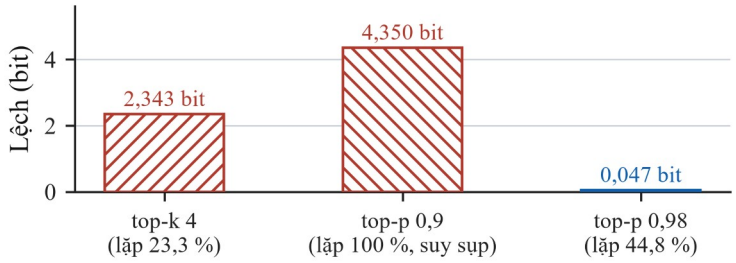
```
def ap_nhiet(p, nhiet):
    q = log(maximum(p, 1e-30)) / nhiet
    q = exp(q - q.max())
    return q / q.sum()

def loc_top_k(p, k):
    nguong = sort(p)[-k]
    q = where(p >= nguong, p, 0.0)
    return q / q.sum()
```

```
def loc_top_p(p, nguong):
    thu = argsort(p)[::-1]
    giu = searchsorted(cumsum(p[thu]), nguong) + 1
    q = zeros_like(p); q[thu[:giu]] = p[thu[:giu]]
    return q / q.sum()
```



(a) Nhiệt độ: entropy (liền) và tỉ lệ lặp lại (đứt)



(b) Top-k và top-p: độ lệch so với tài liệu thật

Hình 13.12 Nhiệt độ, top-k và top-p đo trên chính chuỗi token của bài

Bảng 13.8 Tám cách lấy mẫu, đo trên 3.000 token sinh ra

Cách lấy mẫu	Entropy	Tỉ lệ lặp lại	Lệch tài liệu thật
Tài liệu thật	5,20 bit	55,9 %	—
Nhiệt 0,5	0,00 bit	100,0 %	4,350 bit
Nhiệt 0,8	1,63 bit	64,4 %	0,179 bit
Nhiệt 1,0	3,44 bit	37,8 %	0,108 bit
Nhiệt 1,3	4,78 bit	19,6 %	0,354 bit
Top-k 4	1,85 bit	23,3 %	2,343 bit
Top-p 0,9	0,00 bit	100,0 %	4,350 bit
Top-p 0,98	2,93 bit	44,8 %	0,047 bit

Ba điều rút ra từ bảng này

Nhiệt 0,5 *suy sụp hoàn toàn*: mô hình lặp đúng một token, tỉ lệ lặp 100 phần trăm. Đây chính là cái vòng lặp mà ai từng nghịch một mô hình sinh nhạc đều đã gặp, và nguyên nhân là tham số chứ không phải mô hình.

Top-p 0,9 *cũng suy sụp*, vì phân phối chuyển tiếp ở đây quá nhọn nên nhân chỉ còn đúng một token. Nghĩa là top-p không tự động an toàn hơn top-k — nó chỉ đổi cách chọn ngưỡng, và ngưỡng vẫn phải chọn bằng cách *đo*.

Con số lệch thấp nhất trong bảng là top-p 0,98, và nó thấp hơn cả lấy mẫu nguyên bản. Đó là lý do các hệ thống thật hay đặt ngưỡng rất cao chứ không đặt vừa phải.

13.4.3 Một token sai kéo dài bao lâu

Điểm yếu hay được nhắc nhất của mô hình tự hồi quy là lệch phơi nhiễm: khi huấn luyện, mô hình luôn nhìn thấy ngữ cảnh *đúng*; khi sinh, nó phải nhìn ngữ cảnh do chính nó vừa sinh ra, kể cả khi ngữ cảnh ấy đã sai.



Hình 13.13 Một token sai kéo dài bao lâu trong mô hình bậc một

Phép đo cho mô hình bậc một sinh sai đúng một token rồi chạy tiếp: sau tám token còn lệch 0,25 bit, sau ba mươi hai token chỉ còn 0,024 — tức khoảng hai phần mười giây là mất hẳn.

Nhưng phải đọc con số này cho đúng

Con số ấy *đúng* với một mô hình chỉ nhớ một bước. Sai lệch bị loại bỏ nhanh chính vì mô hình không có trí nhớ dài.

Một mô hình có cửa sổ ngữ cảnh dài thì token sai *nằm nguyên* trong ngữ cảnh và tiếp tục ảnh hưởng tới mọi token sinh sau — cho tới khi nó trôi ra khỏi cửa sổ. Đó mới là lệch phơi nhiễm thật.

Mô hình đồ chơi trong bài *không* dựng lại được hiện tượng ấy, và bài này không có cách nào đo nó bằng số. Đây là một trong hai chỗ mà chương 13 phải dừng ở mức mô tả — và nói rõ như vậy tốt hơn là đưa ra một con số không đứng vững.

Quyết định	Đặt thấp thì	Đặt cao thì	Chỗ thường dùng
Nhiệt độ τ	Suy sụp thành vòng lặp (đo ở τ = 0,5)	Lạc điệu, sai hoà thanh	0,8 – 1,0 cho nhạc 0,6 – 0,8 cho giọng
Top-k	Mất mọi thứ bất ngờ	Gần như không lọc gì	k khoảng 5 – 10 % số mã
Top-p	Cũng suy sụp (đo ở p = 0,9)	Gần như không lọc gì	0,95 – 0,99
Độ dài ngữ cảnh	Không giữ được cấu trúc dài	Cần nhiều dữ liệu hơn hẳn	Tuỳ dữ liệu, và luôn phải đo

Điều không nằm trong bảng nhưng quan trọng hơn cả bốn dòng:

Mọi con số ở đây chỉ nói về chuỗi token. Chất lượng âm thanh ra còn phụ thuộc bộ giải mã ở tầng ba: một chuỗi token hoàn hảo qua một vocoder kém vẫn cho âm thanh kém. Mục 13.6 đo đúng điều đó.

Hình 13.14 Bốn quyết định khi lấy mẫu một mô hình tự hồi quy

13.5 Họ mô hình khuếch tán

13.5.1 Ý tưởng: học đi ngược một quá trình đã biết

Mô hình tự hồi quy sinh theo thứ tự thời gian. Mô hình khuếch tán sinh theo một trục hoàn toàn khác: nó bắt đầu từ nhiễu thuần và gọt dần thành tín hiệu, sinh *cả đoạn* cùng một lúc ở mỗi bước.

Điều làm cho ý tưởng ấy khả thi là một quan sát đơn giản: quá trình *thuận* — từ tín hiệu sạch đi tới nhiễu thuần — thì biết trước hoàn toàn, không cần học gì cả.

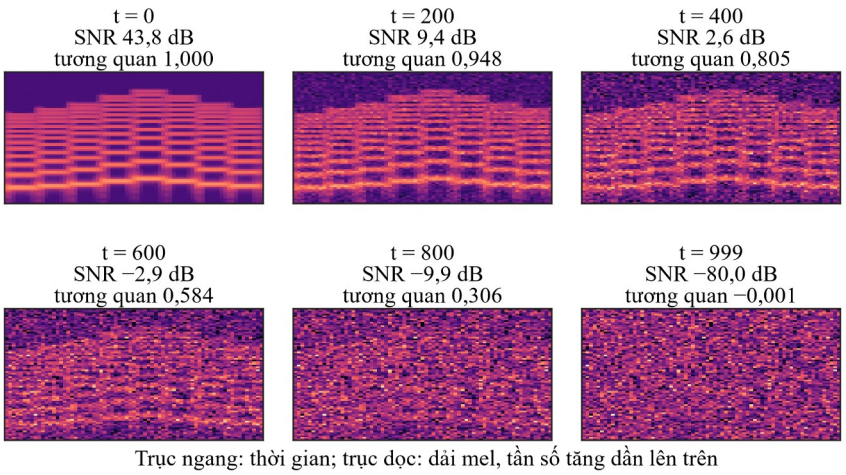
$$\mathbf{x}_t = \sqrt{\bar{\alpha}_t}\,\mathbf{x}_0 + \sqrt{1-\bar{\alpha}_t}\,\boldsymbol{\varepsilon}, \qquad \boldsymbol{\varepsilon} \sim \mathcal{N}(0, \mathbf{I}) \tag{13.8}$$

Trong công thức, alpha ngang là phần tín hiệu còn lại ở bước t, và epsilon là nhiễu Gauss chuẩn. Chỉ một phép trộn có tỉ lệ đã biết. Vì thế sinh ra dữ liệu huấn luyện cho bài toán ngược là chuyện miễn phí: lấy một đoạn nhạc thật, chọn một bước t bất kỳ, trộn nhiễu vào theo đúng tỉ lệ ấy, rồi bảo mô hình đoán lại phần nhiễu đã thêm.

Liệt kê 13.5 Quá trình thuận — ba dòng, và không có gì phải học

```
def them_nhieu(x0, abar_t):
    e = standard_normal(x0.shape)
    return sqrt(abar_t) * x0 + sqrt(1 - abar_t) * e, e

# Sinh một mẫu huấn luyện cho bài toán ngược:
t  = randint(0, T)                 # chọn một bước bất kỳ
xt, e = them_nhieu(x0, abar[t])    # trộn nhiễu vào
# rồi bảo mô hình đoán lại chính e từ (xt, t).
```



Hình 13.15 Quá trình thuận chạy trên phổ mel thật của một giai điệu

Bảng 13.9 Tương quan với bản gốc theo bước khuếch tán

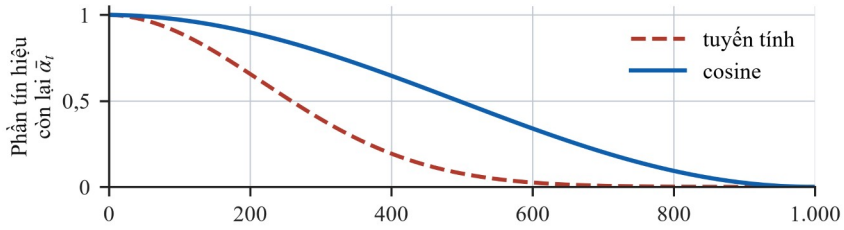
Bước t	SNR	Tương quan với phổ gốc	Mắt còn nhận ra gì
0	43,8 dB	1,000	Nguyên vẹn
200	9,4 dB	0,948	Cấu trúc còn rõ
400	2,6 dB	0,805	Vẫn thấy các nốt
600	−2,9 dB	0,584	Chỉ còn vệt mờ
800	−9,9 dB	0,306	Gần như không
999	−80,0 dB	−0,001	Không gì cả

Việc *học* là học đi ngược lại từng bước một, và mỗi bước chỉ phải khử một lượng nhiễu nhỏ. Đó là lý do bài toán khó ấy trở nên dễ hơn hẳn việc sinh thẳng ra một phổ mel hoàn chỉnh: nó được chia thành một nghìn bài toán con, mỗi bài dễ.

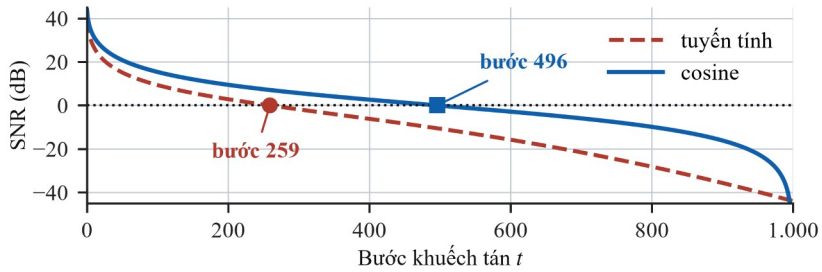
13.5.2 Lịch nhiễu — và vì sao lịch tuyến tính lãng phí

Alpha ngang đi từ gần một xuống gần không theo một *lịch* do người thiết kế chọn. Lịch tuyến tính là lựa chọn đầu tiên trong lịch sử, và nó có một vấn đề chỉ lộ ra khi nhìn bằng decibel.

$$\mathrm{SNR}(t) = 10 \log_{10} \frac{\bar{\alpha}_t}{1 - \bar{\alpha}_t} \tag{13.9}$$



(a) Hai lịch nhiễu: phần tín hiệu còn lại theo bước



(b) Tỉ số tín hiệu trên nhiễu của hai lịch

Hình 13.16 Hai lịch nhiễu, và tỉ số tín hiệu trên nhiễu của chúng

Lịch tuyến tính tụt xuống dưới không decibel ngay ở bước 259 trên một nghìn. Nghĩa là hơn bảy mươi tư phần trăm số bước còn lại được tiêu vào việc khử một thứ mà tín hiệu đã yếu hơn nhiễu — gần như vô ích, vì ở đó không còn gì để học.

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \qquad f(t) = \cos^2\left(\frac{t/T + s}{1+s} \cdot \frac{\pi}{2}\right) \tag{13.10}$$

Lịch cosine đẩy mốc ấy tới bước 496, đúng giữa hành trình. Nó không thêm một tham số học nào, chỉ đổi công thức, và đó là toàn bộ nội dung của một cải tiến đã trở thành mặc định của cả lĩnh vực.

> **Một cách đọc khác về cùng con số**
>
> Có thể nghĩ về lịch nhiễu như phân bổ *ngân sách*: một nghìn bước là ngân sách, và mỗi bước nên được tiêu vào một mức SNR khác nhau.
>
> Lịch tuyến tính tiêu ba phần tư ngân sách vào vùng dưới không decibel — nơi mọi thứ trông như nhau.
>
> Lịch cosine chia đôi ngân sách. Và các lịch mới hơn còn đi xa hơn: chúng lấy mẫu bước t theo phân phối của chính SNR chứ không theo phân phối đều của t.

13.5.3 Số bước lấy mẫu đổi chất lượng bao nhiêu

Huấn luyện với một nghìn bước rồi khi sinh lại nhảy bước để chạy nhanh — đó là điều mọi hệ thống thật đều làm. Câu hỏi là nhảy bao nhiêu thì mất bao nhiêu.

$$\mathbf{x}_s = \sqrt{\bar{\alpha}_s}\,\hat{\mathbf{x}}_0 + \sqrt{1-\bar{\alpha}_s}\,\hat{\boldsymbol{\varepsilon}}_t \tag{13.11}$$

Công thức trên là một bước lấy mẫu tất định kiểu *ddim*: từ trạng thái ở bước t, ước lượng bản sạch, rồi nhảy thẳng tới bước s bất kỳ nhỏ hơn t. Số bước dùng thật vì thế tách rời khỏi số bước đã huấn luyện.

Liệt kê 13.6 Lấy mẫu nhảy bước — và so_buoc là tham số quyết định

```
def lay_mau_ddim(xT, abar, so_buoc, khu):
    moc = linspace(len(abar) - 1, 0, so_buoc + 1).round()
    x = xT.copy()
    for k in range(so_buoc):
        t, s = int(moc[k]), int(moc[k + 1])
        at = abar[t]
        x0 = khu(x, at)                    # ước lượng bản sạch
        e  = (x - sqrt(at) * x0) / sqrt(1 - at)
        a_s = abar[s] if s > 0 else 1.0
        x = sqrt(a_s) * x0 + sqrt(1 - a_s) * e
    return x
```

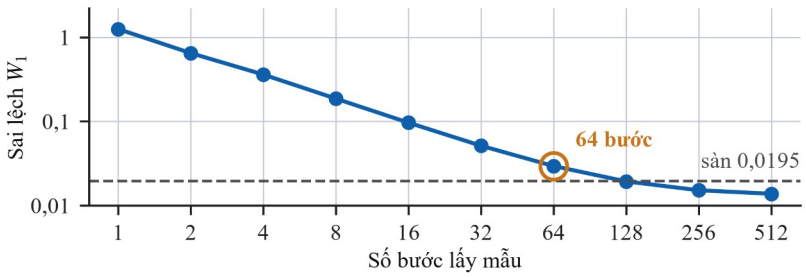
Phép đo này đo được đúng phần cần đo

Nếu đo trên một mô hình thật thì không tách được: chất lượng kém có thể do ít bước, mà cũng có thể do mô hình học chưa tốt.

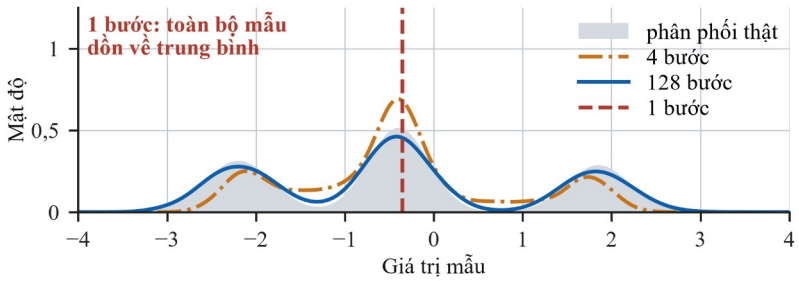
Vì thế phép đo ở đây dùng một bài toán đồ chơi — phân phối gốc là hỗn hợp ba Gauss một chiều — nơi bộ khử nhiễu *tối ưu* viết ra được bằng công thức chính xác.

Kết quả vì thế là phần chất lượng mất đi *hoàn toàn* do số bước, với một mô hình hoàn hảo. Mọi mô hình thật chỉ có thể kém hơn thế.

Giới hạn của cách làm này cũng phải nói rõ: một hỗn hợp Gauss một chiều không phải âm thanh. Thứ chuyển được sang bài toán thật là *hình dạng* của đường cong, không phải con số tuyệt đối.



(a) Số bước đổi chất lượng bao nhiêu



(b) Ít bước thì mất gì: phân phối sinh ra

Hình 13.17 Số bước lấy mẫu đổi chất lượng bao nhiêu, và ít bước thì mất gì

Bảng 13.10 Sai lệch phân phối theo số bước lấy mẫu

Số bước	Sai lệch W_1	So với sàn	Nhận xét
1	1,270	65 lần	Mọi mẫu dồn về trung bình — mất cả ba đỉnh
4	0,344	18 lần	Dựng lại được ba đỉnh, sai hẳn tỉ lệ
16	0,090	4,6 lần	Đã gần đúng
64	0,024	1,2 lần	Đủ dùng
128	0,020	1,0 lần	Chạm sàn
512	0,020	1,0 lần	Không hơn gì 128 bước

Quy luật rõ ràng: mỗi lần gấp đôi số bước thì sai lệch giảm gần một nửa, cho tới khi chạm sàn ở khoảng 64 tới 128 bước. Đó là lý do các hệ thống thật thường dùng vài chục bước chứ không dùng cả nghìn, và cũng là lý do các hệ thống *một* bước phải được chưng cất riêng chứ không thể chỉ đơn giản giảm số bước xuống một.

Điều đáng chú ý ở cột cuối: một bước không cho một mẫu kém — nó cho *trung bình* của phân phối. Chuyển sang âm thanh, đó chính là lý do tiếng sinh ra từ quá ít bước nghe *mờ* và thiếu chi tiết, chứ không nghe méo hay nhiễu.

13.5.4 Hai họ mô hình, hai cách trả giá

Tiêu chí	Tự hồi quy trên token	Khuếch tán trên phổ mel
Số bước tuần tự cho 30 giây	41.344 bước, tăng theo độ dài (gấp 827 lần)	50 bước, không phụ thuộc độ dài
Sinh được đoạn dài không	Được, cứ nối tiếp mãi	Khó: cửa sổ cố định, phải ghép
Sửa giữa chừng được không	Không: đã sinh là xong	Được: sửa mặt nạ rồi khử nhiễu lại
Đích sinh ra là gì	Chuỗi token rời rạc	Phổ mel liên tục
Cần gì ở tầng ba	Bộ giải mã của codec	Một vocoder, chỗ mất mát (mục 13.6)
Cái sai điển hình	Suy sụp thành vòng lặp, hoặc lạc điệu	Mờ, mất chi tiết khi ít bước

Hai họ đang hội tụ: nhiều hệ thống mới sinh token bằng khuếch tán, hoặc sinh phổ mel bằng mô hình tự hồi quy trên các mảnh phổ.

Bảng này so sánh hai cơ chế, không phải hai nhóm sản phẩm.

Hình 13.18 Tự hồi quy so với khuếch tán: sáu tiêu chí

Con số quyết định nhất trong bảng nằm ở dòng đầu: ba mươi giây nhạc cần 41.344 bước tuần tự với mô hình tự hồi quy tám tầng, và khoảng năm mươi bước với mô hình khuếch tán — bất kể đoạn dài bao nhiêu. Tỉ số ấy là 827 lần, và nó tăng tuyến tính theo độ dài.

Nhưng bảng này là bảng về hai cơ chế, không phải hai sản phẩm

Nhiều hệ thống mới sinh *token* bằng khuếch tán — tức lấy đích của họ này và cơ chế của họ kia.

Nhiều hệ thống khác sinh phổ mel bằng mô hình tự hồi quy trên các mảnh phổ.

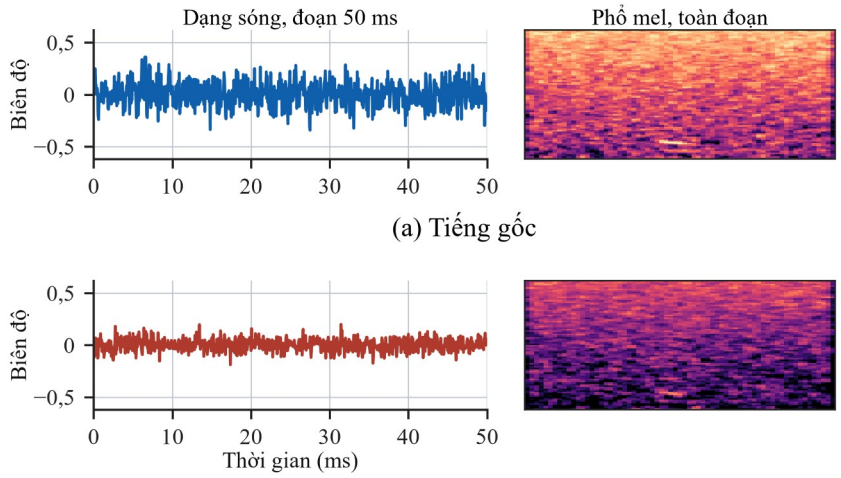
Vì vậy khi đọc một bài báo, câu hỏi đúng không phải là "nó thuộc họ nào" mà là hai câu riêng biệt: nó sinh ra *cái gì*, và nó sinh theo *cơ chế* nào.

13.6 Giọng, vocoder, và bài toán pha

13.6.1 Phổ mel không đủ để dựng lại tiếng

Giả sử một mô hình khuếch tán đã sinh ra một phổ mel hoàn hảo. Nó đã làm xong việc của nó. Nhưng vẫn chưa có tiếng — và khoảng cách từ đó tới tiếng nghe được lớn hơn nhiều người tưởng.

Lý do là phổ độ lớn loại bỏ *pha*. Với mỗi ô phổ, nó giữ lại độ lớn và bỏ đi thông tin về thời điểm. Và pha không phải một chi tiết nhỏ: nó quyết định dạng sóng.



(a) Tiếng gốc

(b) Cùng độ lớn phổ, pha ngẫu nhiên

Khoảng cách mel 6,98 dB · khoảng cách phổ log 11,30 dB
SI-SDR −39,4 dB

Hình 13.19 Cùng một độ lớn phổ, hai pha khác nhau, hai tiếng khác hẳn

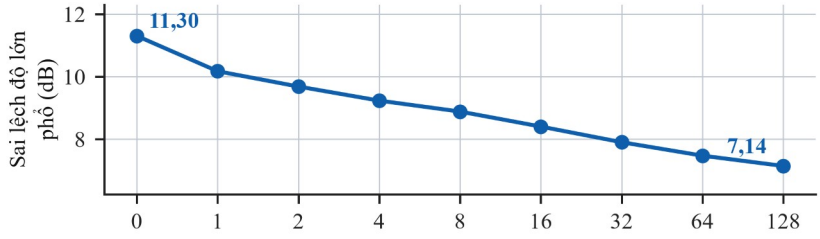
Bảng 13.11 Hai bản có cùng độ lớn phổ, đo bằng ba thước

Thước đo	Giá trị	Nói lên điều gì
Khoảng cách mel	6,98 dB	Gần như không thấy khác biệt — và phần lớn con số này còn do phép dựng lại chồng lấn cửa sổ
Khoảng cách phổ log	11,30 dB	Cũng không thấy gì đáng kể
SI-SDR	−39,4 dB	Hoàn toàn không phải cùng một tiếng

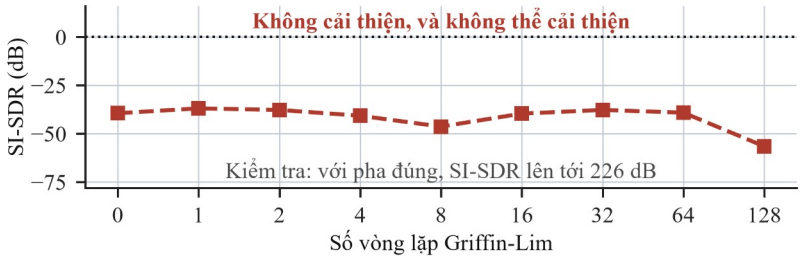
Tai nghe hai bản này thì bản thứ hai là một tiếng rè, không còn nhận ra giọng. Nhưng mọi thước đo dựa trên độ lớn phổ đều bảo chúng gần như giống nhau. Đây là bài toán trung tâm của tầng ba.

13.6.2 Griffin-Lim và giới hạn của nó

Cách cổ điển để đoán lại pha là thuật toán Griffin-Lim: bắt đầu từ pha ngẫu nhiên, rồi lặp đi lặp lại hai bước — dựng lại tín hiệu, phân tích lại, giữ pha vừa có nhưng thay độ lớn bằng độ lớn đích.



(a) Cái mà Griffin-Lim cải thiện: độ lớn phổ



(b) Cái mà nó không cải thiện: dạng sóng

Hình 13.20 Cái mà Griffin-Lim cải thiện, và cái mà nó không cải thiện

Hai hình phải đọc cùng nhau. Griffin-Lim tối thiểu hoá sai lệch *độ lớn* phổ, và nó làm đúng việc ấy: từ 11,30 xuống 7,14 decibel sau 128 vòng, đều đặn và đơn điệu. Nhưng dạng sóng dựng ra không tiến gần tiếng gốc chút nào.

Liệt kê 13.7 Griffin-Lim — vòng lặp giữ độ lớn, đổi pha

```
def griffin_lim(bien_do, vong):
    pha = exp(1j * uniform(-pi, pi, bien_do.shape))
    X = bien_do * pha              # xuất phát từ pha ngẫu nhiên
    for _ in range(vong):
        y = istft(X)               # về miền thời gian
        Y = stft(y)                # rồi quay lại miền phổ
        X = bien_do * exp(1j * angle(Y))   # GIỮ pha, THAY độ lớn
    return istft(X)
```

Vì sao SI-SDR không bao giờ cải thiện ở đây

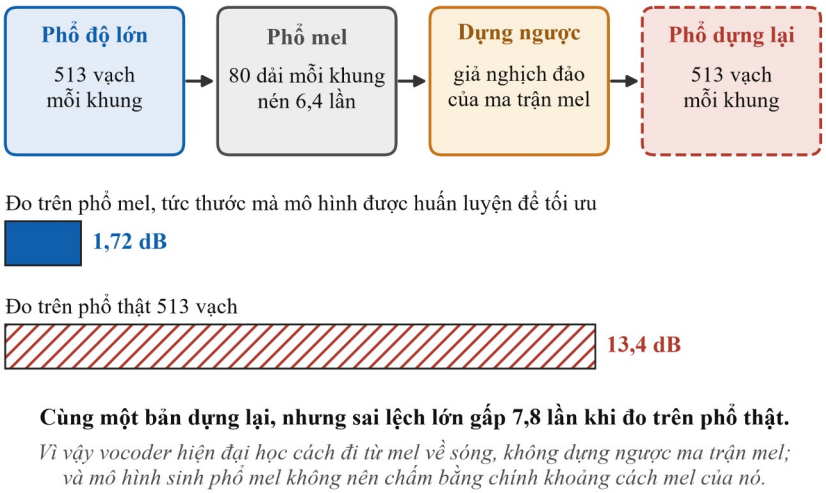
Có *vô số* dạng sóng khác nhau cùng cho một độ lớn phổ. Griffin-Lim hội tụ về *một* trong số đó, và không có lý do gì nó phải là chính dạng sóng gốc.

Vì thế đừng dùng SI-SDR để chấm một bộ dựng pha, và đừng kỳ vọng thêm vòng lặp sẽ cho ra đúng tiếng gốc. Phép kiểm sức khoẻ ở góc hình cho thấy nếu có cả pha đúng thì SI-SDR lên tới 226 decibel — nên vấn đề không phải ở công cụ đo.

Cái tai nghe thấy ở đầu ra Griffin-Lim là chất kim loại đặc trưng, và đó là lý do các hệ thống hiện đại thay hẳn nó bằng một vocoder *học được* — thứ được huấn luyện để cho ra dạng sóng nghe tự nhiên, chứ không phải để khớp độ lớn phổ.

13.6.3 Cổ chai mel — mất mát xảy ra trước cả khi nói tới pha

Còn một mất mát nữa, và nó xảy ra sớm hơn. Nén 513 vạch xuống 80 dải mel là một phép chiếu, nên đảo ngược nó không bao giờ chính xác.



Hình 13.21 Cổ chai mel: đo trên mel thì ổn, đo trên phổ thật thì không

Con số quan trọng nhất trong hình là cặp 1,72 và 13,4. Cùng một bản dựng lại: đo bằng khoảng cách mel — tức đúng thước đo mà mô hình được huấn luyện để tối ưu — thì lệch 1,72 decibel, nghe như mọi thứ đều ổn. Đo bằng phổ thật thì lệch 13,4.

Hai hệ quả thực hành

Vocoder hiện đại *không* dựng ngược ma trận mel bằng giả nghịch đảo. Nó *học* ánh xạ từ mel về sóng, và nhờ thế nó bù lại được phần thông tin mà phép chiếu đã bỏ — bằng cách học xem loại âm thanh nào thường đi với mẫu mel nào.

Một mô hình sinh phổ mel *không bao giờ* nên được đánh giá bằng chính khoảng cách mel của nó. Đó là chấm bài bằng chính đề bài, và con số 1,72 ở trên cho thấy nó dễ dãi tới mức nào.

13.6.4 Vậy một hệ thống sinh giọng gồm những gì

Bảng 13.12 Ba khối của một hệ thống sinh giọng, và chỗ mỗi khối hay sai

Khối	Việc	Sai thì nghe ra sao
Từ chữ sang biểu diễn	Chuyển văn bản thành chuỗi âm vị có trường độ và ngữ điệu	Đọc sai, ngắt câu sai, ngữ điệu phẳng
Mô hình âm học	Sinh phổ mel hoặc token từ chuỗi âm vị	Giọng đúng chữ nhưng sai chất giọng; hoặc lặp một âm tiết
Vocoder	Từ phổ mel hoặc token về dạng sóng	Chất kim loại, rè, hoặc mất phụ âm — mục 13.6.2 và 13.6.3

Với tiếng Việt, khối thứ nhất khó hơn hẳn so với các ngôn ngữ không thanh điệu: thanh điệu là một phần của nghĩa, nên nó phải được mô hình hoá tường minh chứ không thể để mô hình âm học tự suy ra từ chữ. Đây là chỗ một hệ thống nhập khẩu hay sai, và cũng là hướng mà một dự án nghiên cứu nhỏ có thể đóng góp được.

13.7 Tách nguồn — và hai trần trên không vượt được

13.7.1 Tách bằng mặt nạ: cơ chế chung của gần như mọi phương pháp

Tách nguồn là bài toán ngược của trộn: cho một bản đã trộn, dựng lại từng nguồn. Nó là nền của mọi ứng dụng tách nhạc nền khỏi giọng hát, tách lời khỏi tiếng ồn, hay bóc riêng từng nhạc cụ.

Gần như mọi phương pháp làm việc trên phổ đều theo cùng một khuôn: ước lượng một *mặt nạ* — một số trong khoảng không tới một cho mỗi ô phổ — rồi nhân nó với độ lớn của hỗn hợp.

$$M_{\text{tỷ lệ}}[f,n] = \frac{|S[f,n]|}{|S[f,n]| + |N[f,n]|} \tag{13.12}$$



Hình 13.22 Khuôn chung của mọi phương pháp tách bằng mặt nạ

Khối duy nhất khác nhau giữa các phương pháp là khối ước lượng mặt nạ. Bốn khối còn lại hoàn toàn giống nhau, và một trong số đó đặt ra một giới hạn mà không phương pháp nào trong khuôn này vượt được.

13.7.2 Hai trần trên, và cả hai đều đo được

Trần thứ nhất là trần của mặt nạ: nếu *biết* trước cả hai nguồn, thì mặt nạ tốt nhất cho kết quả bao nhiêu? Có hai cách chọn — nhị phân lý tưởng giao mỗi ô cho nguồn to hơn; tỉ lệ lý tưởng chia mỗi ô theo tỉ lệ năng lượng, đúng công thức (13.12).

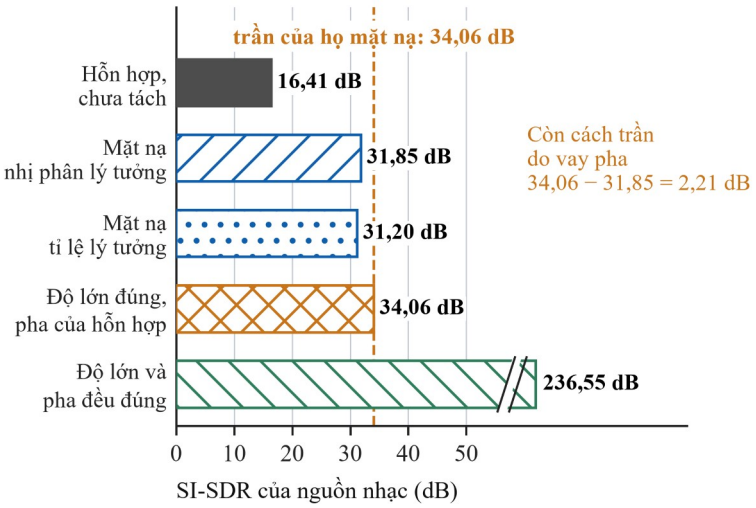
Trần thứ hai sâu hơn: mặt nạ chỉ sửa *độ lớn*, còn pha thì vay nguyên của hỗn hợp — tức pha của hai nguồn đã trộn lẫn. Cho độ lớn đúng tới từng ô mà vẫn vay pha thì kết quả dừng ở đâu?

$$\hat{s} = \frac{\langle \hat{y}, s \rangle}{\|s\|^2} s, \qquad \mathrm{SI-SDR} = 10 \log_{10} \frac{\|\hat{s}\|^2}{\|\hat{y} - \hat{s}\|^2} \tag{13.13}$$

Thước đo dùng ở đây là SI-SDR, bất biến theo tỉ lệ chung, nên nó không bị đánh lừa bởi chênh lệch âm lượng. Đây là thước đo chuẩn của toàn bộ lĩnh vực tách nguồn.

Liệt kê 13.8 SI-SDR — bốn dòng, và dòng thứ hai là điều làm nên tên gọi

```
def si_sdr_db(chuan, uoc):
    s = chuan - chuan.mean();  y = uoc - uoc.mean()
    a = dot(y, s) / dot(s, s)        # tỉ lệ tốt nhất — BẤT BIẾN
    chieu = a * s
    nhieu = y - chieu
    return 10 * log10(sum(chieu**2) / sum(nhieu**2))
```



Hình 13.23 Ba trần trên chồng lên nhau, đo trên cùng một hỗn hợp

Bảng 13.13 Năm mức, đo trên nguồn nhạc của cùng một hỗn hợp

Cấu hình	SI-SDR	Cần biết gì
Hỗn hợp, chưa tách	16,41 dB	Không gì
Mặt nạ nhị phân lý tưởng	31,85 dB	Biết trước cả hai nguồn
Mặt nạ tỉ lệ lý tưởng	31,20 dB	Biết trước cả hai nguồn
Độ lớn đúng, pha của hỗn hợp	34,06 dB	Biết độ lớn thật của nguồn đích
Độ lớn *và* pha đều đúng	236,55 dB	Biết cả pha — tức đã có nguồn rồi

Ba điều phải rút ra

Dòng cuối là phép kiểm sức khoẻ: có cả độ lớn lẫn pha đúng thì dựng lại gần như tuyệt đối. Nên mọi mất mát ở bốn dòng trên đều là mất mát *thật* chứ không phải lỗi công cụ.

Dòng thứ tư là con số đáng nhớ nhất của cả mục: 34,06 decibel là *trần* của toàn bộ họ phương pháp mặt nạ. Không mô hình nào trong họ ấy vượt qua được, dù to tới đâu, dù dữ liệu nhiều tới đâu.

Và đó là lý do các hệ thống tách nguồn hàng đầu hiện nay chuyển sang làm việc thẳng trên *miền thời gian* — nơi không có khái niệm pha của hỗn hợp để phải vay mượn.

13.7.3 Hai chuẩn so sánh không cần học gì cả

Trước khi huấn luyện một mô hình tách nguồn, nên chạy hai phương pháp cổ điển. Nếu mô hình không vượt được chúng thì nó chưa học được gì.

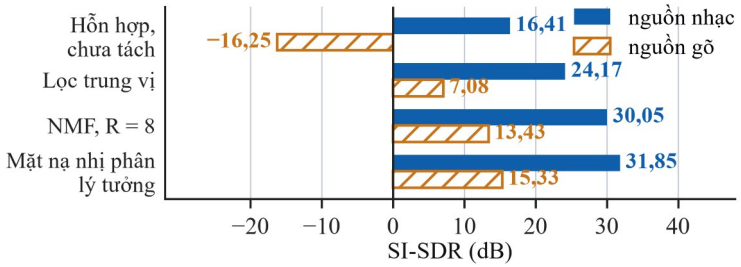
Phương pháp thứ nhất khai thác đúng quan sát đã nêu ở mục 13.2.3: thành phần hài kéo dài theo *thời gian*, thành phần gõ trải rộng theo *tần số*. Lọc trung vị dọc trục thời gian giữ lại cái thứ nhất; lọc trung vị dọc trục tần số giữ lại cái thứ hai.

Liệt kê 13.9 Tách hài với gõ — không có tham số học nào

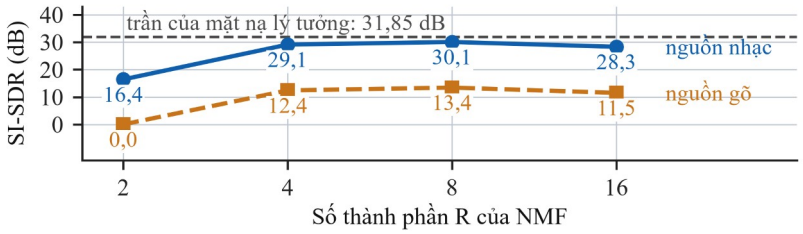
```
def tach_hpss(x, k_hai, k_go, mu):
    X = stft(x)
    A = abs(X)
    H = trung_vi(A, k_hai, truc=THOI_GIAN)   # giữ phần hài
    G = trung_vi(A, k_go,  truc=TAN_SO)      # giữ phần gõ
    Mh = H**mu / (H**mu + G**mu)
    return istft(X * Mh), istft(X * (1 - Mh))
```

Phương pháp thứ hai là phân rã ma trận không âm: tìm R mẫu phổ và mức của từng mẫu theo thời gian sao cho tích của chúng xấp xỉ phổ của hỗn hợp.

$$\mathbf{V} \approx \mathbf{W}\mathbf{H}, \qquad \mathbf{V} \in \mathbb{R}_{+}^{F \times N},\ \mathbf{W} \in \mathbb{R}_{+}^{F \times R},\ \mathbf{H} \in \mathbb{R}_{+}^{R \times N} \tag{13.14}$$



(a) Bốn mức trên cùng một hỗn hợp



(b) Số thành phần quyết định rất nhiều

Hình 13.24 Hai cách không cần học, so với trần trên của mặt nạ

Bảng 13.14 Bốn mức trên cùng một hỗn hợp

Phương pháp	Nguồn nhạc	Nguồn gõ	Cần gì
Hỗn hợp, chưa tách	16,41 dB	−16,25 dB	Không gì
Lọc trung vị	24,17 dB	7,08 dB	Một giả thiết về cấu trúc; chạy 0,39 giây
NMF, R = 8	30,05 dB	13,43 dB	Không nhãn, nhưng phải *gán* thành phần
Mặt nạ nhị phân lý tưởng	31,85 dB	15,33 dB	Biết trước cả hai nguồn

Một lời phải nói rõ về con số NMF

Trong phép đo ở đây, việc *gán* mỗi thành phần NMF cho nguồn nào được làm bằng cách so với nguồn thật — tức có dùng thông tin mà một hệ thống thật không có.

Vì thế 30,05 decibel là một *cận trên* của NMF, không phải kết quả của một bộ tách mù. Một bộ tách mù thật phải tự gán, và đó là phần khó nhất.

Nói rõ điều này quan trọng hơn con số. Rất nhiều so sánh trong tài liệu về tách nguồn không nói rõ mức thông tin mà mỗi phương pháp được cho, và khi ấy hai con số không so được với nhau.

Liệt kê 13.10 NMF — luật cập nhật nhân của Lee và Seung

```
def nmf(V, R, vong):
    W = abs(standard_normal((V.shape[0], R))) + 0.1
    H = abs(standard_normal((R, V.shape[1]))) + 0.1
    for _ in range(vong):
        WH = maximum(W @ H, 1e-12)
        H = H * ((W.T @ (V / WH)) / (W.T @ ones_like(V)))
        WH = maximum(W @ H, 1e-12)
        W = W * (((V / WH) @ H.T) / (ones_like(V) @ H.T))
        W = W / W.sum(axis=0, keepdims=True)
```

```
    return W, H       # W: R mẫu phổ;  H: mức theo thời gian
```

MỨC	CẦN BIẾT — DÙNG ĐỂ LÀM GÌ	SI-SDR (dB)
0. Không tách	Không cần gì *Mốc gốc; mọi cải thiện tính từ đây*	16,41
1. Lọc trung vị	Một giả thiết về cấu trúc *Chuẩn rẻ nhất; chạy trong một giây*	24,17
2. NMF	Không nhãn, phải gán thành phần *Chuẩn thứ hai; phải chọn R*	30,05
3. Mặt nạ lý tưởng	Biết trước cả hai nguồn *Trần của mọi mô hình dùng mặt nạ*	31,85
4. Pha đúng	Biết cả pha của nguồn đích *Trần của việc vay pha (mục 13.7.2)*	34,06

Bài báo tách nguồn nêu con số ở mức nào thì phải nói rõ mức ấy; nếu không, hai con số không so sánh được với nhau.

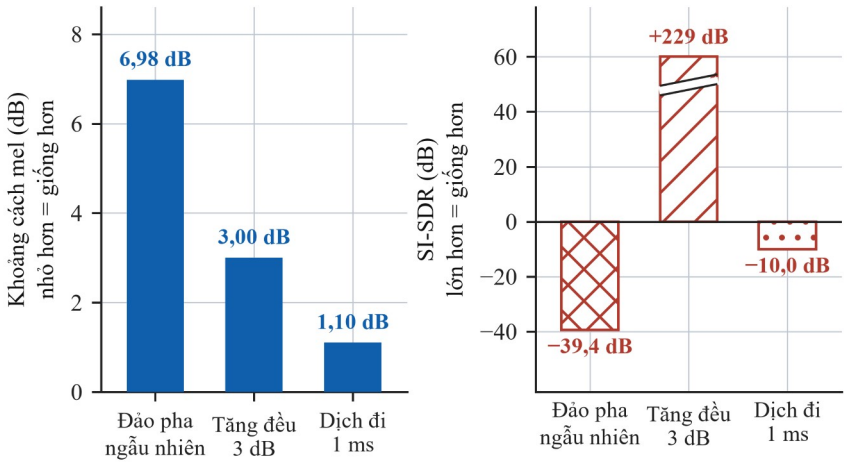
Nếu mô hình học được không vượt nổi mức 1, thì thứ đã học được là gì?

Hình 13.25 Bốn mức của một bài toán tách nguồn

13.8 Đánh giá — ba chỗ thước đo khách quan bị đánh lừa

13.8.1 Ba phép biến đổi, hai thước đo, và ba lần bất đồng

Mọi thước đo khách quan đều trả lời một câu hỏi cụ thể, và không thước đo nào trả lời được mọi câu. Ba phép biến đổi dưới đây làm lộ ra điều đó rất rõ.



(a) Thước đo thứ nhất (b) Thước đo thứ hai

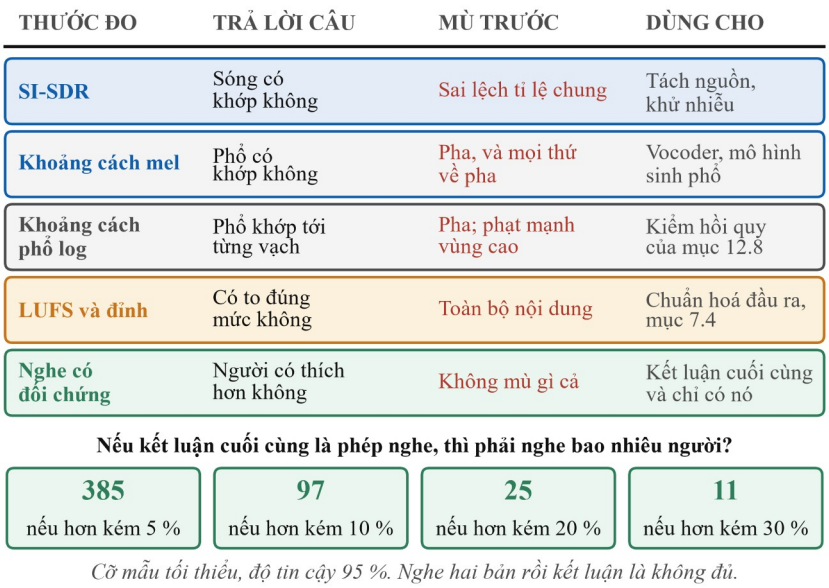
Hình 13.26 Ba phép biến đổi, và hai thước đo nói ngược nhau ở cả ba

Bảng 13.15 Ba lần hai thước đo bất đồng

Phép biến đổi	Khoảng cách mel	SI-SDR	Tai người nghe ra
Đảo pha ngẫu nhiên	6,98 dB — gần như không có gì	−39,4 dB — sai hoàn toàn	Sai hoàn toàn
Tăng đều 3 dB	3,00 dB — thấy đúng 3 decibel	+229 dB — hoàn hảo	Chỉ to hơn một chút
Dịch đi 1 ms	1,10 dB — gần như không thấy	−10,0 dB — kém rõ rệt	Không nghe ra gì

Không thước đo nào đúng cả ba lần, vì mỗi thước hỏi một câu khác. SI-SDR hỏi "sóng có khớp không" — nên nó bắt được đảo pha, bỏ qua thay đổi âm lượng, và phạt nặng một phép dịch mà tai không nghe ra. Khoảng cách mel hỏi "phổ có khớp không" — nên nó mù hoàn toàn trước pha.

13.8.2 Chọn thước đo theo câu hỏi



Hình 13.27 Chọn thước đo theo câu hỏi, và cỡ mẫu cho phép nghe

Bảng trong hình xếp năm thước đo theo câu hỏi mà mỗi thước trả lời, và cột thứ ba — thứ mà mỗi thước *mù* trước — quan trọng hơn hai cột đầu. Chọn sai thước đo không cho kết quả sai; nó cho một kết quả đúng về một câu hỏi mà chúng ta không hỏi.

Ba lỗi hay gặp khi báo cáo kết quả

Chấm một mô hình sinh phổ mel bằng chính khoảng cách mel của nó. Mục 13.6.3 cho thấy thước đo ấy dễ dãi tới mức nào.

Dùng SI-SDR cho một bộ sinh pha. Mục 13.6.2 cho thấy nó không bao giờ cải thiện, dù bản dựng có tốt lên thật.

Báo một con số tách nguồn mà không nói rõ phương pháp được cho biết trước những gì. Mục 13.7.3 cho thấy chênh lệch có thể tới sáu decibel chỉ vì khác mức thông tin.

13.8.3 Và khi kết luận cuối cùng là phép nghe

Với âm thanh sinh ra bằng AI, không thước đo khách quan nào thay được phép nghe có đối chứng. Nhưng phép nghe cũng phải làm cho đúng, và câu hỏi đầu tiên là cần bao nhiêu người.

Bảng 13.16 Cỡ mẫu tối thiểu cho một phép nghe so sánh cặp

Chênh lệch thật giữa hai bản	Số người nghe tối thiểu
Bản A hơn 5 % — tức 55 % chọn A	385
Bản A hơn 10 % — tức 60 % chọn A	97
Bản A hơn 20 % — tức 70 % chọn A	25
Bản A hơn 30 % — tức 80 % chọn A	11

Với độ tin cậy 95 phần trăm. Nghe hai bản rồi kết luận không phải là một quan điểm khác — nó chỉ đơn giản là không đủ dữ liệu để kết luận. Và bảng này cũng cho một lời khuyên ngược: nếu chỉ có mười người nghe được, thì đừng đặt câu hỏi so sánh hai bản gần nhau, mà hãy chọn hai bản khác nhau rõ rệt.

Liệt kê 13.11 Cỡ mẫu tối thiểu — một dòng công thức

```
def so_mau_nghe_can(do_lech_muc, do_tin_cay=0.95):
    z = 1.96 if do_tin_cay >= 0.95 else 1.645
    return ceil(z**2 * 0.25 / do_lech_muc**2)

# so_mau_nghe_can(0.10)  ->  97 người
# so_mau_nghe_can(0.05)  ->  385 người
```

Bốn quy tắc cho một phép nghe đáng tin

Mù: người nghe không biết bản nào là bản nào. Nếu người tổ chức cũng không biết trong lúc chấm thì càng tốt.

Ngẫu nhiên hoá thứ tự trình bày cho từng người nghe, vì bản nghe trước luôn có lợi thế.

Cùng mức to: chuẩn hoá LUFS cả hai bản trước khi cho nghe — mục 7.4. Bản to hơn gần như luôn được chọn, và đó là hiệu ứng đã biết từ lâu.

Có mỏ neo: chèn vào một bản chắc chắn kém và một bản chắc chắn tốt. Người nghe không phân biệt được hai mỏ neo ấy thì dữ liệu của họ nên bỏ.

13.9 Thực hành, bài tập và tổng kết

13.9.1 Ba bài thực hành

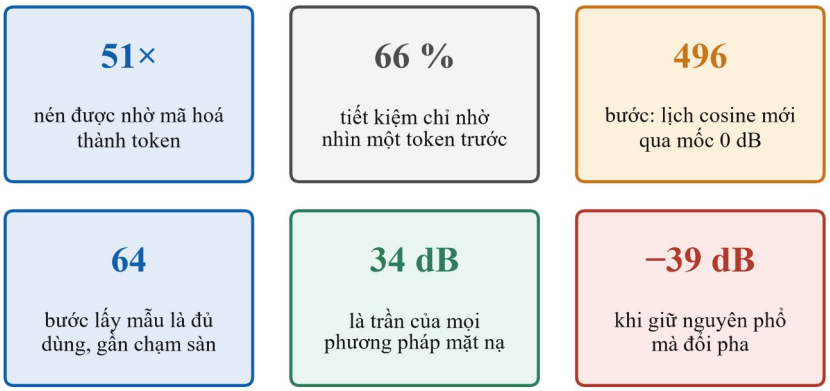
Cả ba bài dùng chung mô-đun Python của cuốn sách, và cả ba đều chạy xong trong vài giây.

- **Thực hành 1 — dựng một codec token.** Lấy một đoạn nhạc bất kỳ, tính phổ mel, học một chồng mã sách bốn tầng, rồi đo SNR tái dựng khi giải mã một, hai và bốn tầng. Ghi lại ba con số và một câu nhận xét về hình dạng.
- **Thực hành 2 — tìm điểm suy sụp của nhiệt độ.** Dựng mô hình bậc một trên chính chuỗi token vừa có, rồi sinh ba nghìn token ở các nhiệt độ từ 0,4 tới 1,4. Tìm nhiệt độ thấp nhất mà tỉ lệ lặp lại chưa vượt 70 phần trăm.
- **Thực hành 3 — chạy hai chuẩn so sánh tách nguồn.** Trộn một đoạn nhạc với một đoạn trống, chạy lọc trung vị và NMF, rồi so với mặt nạ tỉ lệ lý tưởng. Lập một bảng bốn dòng và trả lời: phương pháp nào đáng dùng nếu chỉ có mười phút?

13.9.2 Bảy bài tập

- **Bài 1 — chia tầng cho một ngân sách khác.** Lặp lại phép đo của mục 13.3.3 với ngân sách 16 bit mỗi khung thay vì 12. Cách chia tốt nhất có đổi không? Giải thích bằng số từ mã phải lưu.
- **Bài 2 — lượng tử hoá dư trên vật liệu khác.** Chạy lại mục 13.3.4 với một đoạn *chỉ* có tiếng gõ, rồi với một đoạn *chỉ* có giọng nói. Đường cong lợi ích giảm dần có khác nhau không, và vì sao?
- **Bài 3 — bậc ba.** Thêm mô hình bậc ba vào phép đo của mục 13.4.1. Dự đoán trước kết quả, rồi đo, rồi so. Cần bao nhiêu mẫu học để bậc ba thắng bậc một?
- **Bài 4 — lịch nhiễu riêng.** Thiết kế một lịch thứ ba nằm giữa tuyến tính và cosine, vẽ đường SNR của nó, và nói bước nào nó qua mốc không decibel. Lịch ấy đáng dùng khi nào?
- **Bài 5 — số bước và hình dạng phân phối.** Lặp lại mục 13.5.3 với một hỗn hợp năm Gauss thay vì ba. Số bước cần để chạm sàn có tăng không? Đưa ra một giả thuyết rồi kiểm bằng số.
- **Bài 6 — trần trên trên vật liệu của người đọc.** Trộn hai đoạn tiếng bất kỳ rồi đo lại năm mức của mục 13.7.2. Trần do vay pha có gần 34 decibel không? Nếu lệch nhiều thì tìm xem vì sao — gợi ý: xem hai nguồn chồng phổ nhau bao nhiêu.
- **Bài 7 — thiết kế một phép nghe.** Viết đề cương một phép nghe so sánh hai bản do AI sinh ra, đủ bốn quy tắc ở mục 13.8.3, và tính cỡ mẫu cho chênh lệch mà người đọc muốn phát hiện. Một trang, không cần chạy thật.

13.9.3 Tổng kết chương 13



Hình 13.28 Chương 13 gói lại trong sáu con số

Chương này khác các chương trước ở một chỗ: nó nói về những hệ thống mà chúng ta không dựng lại được toàn bộ. Nhưng nó vẫn giữ nguyên nguyên tắc của cuốn sách — mọi con số trong bài đều là số đo, chạy lại được bằng một lệnh, và mỗi chỗ *không* đo được đều được nói rõ là không đo được.

> **Ba điều đáng mang sang chương sau**
>
> Cái đi *giữa* các tầng quyết định nhiều hơn cái nằm *trong* mỗi tầng. Chọn token hay chọn phổ mel là quyết định lớn hơn chọn kiến trúc mô hình.
>
> Mọi bài toán đều có *trần trên* tính được, và tìm ra nó trước khi huấn luyện tiết kiệm rất nhiều công. Trần 34,06 decibel của họ mặt nạ là ví dụ rõ nhất trong chương này.
>
> Thước đo trả lời câu hỏi của nó, không trả lời câu hỏi của chúng ta. Ba lần bất đồng ở mục 13.8.1 nên được nhớ như một lời cảnh báo thường trực.

Chương 14 mang những thứ này vào một sản phẩm thật: chuẩn hoá và xử lý kết quả do AI sinh ra, đổi tần số lấy mẫu, và ghép chúng vào đúng đường ống đã dựng ở chương 12. Đó là chỗ mọi con số của chương 7, chương 12 và chương này gặp nhau.

Tài liệu tham khảo của chương

Nguồn trực tiếp của nội dung trong bài

- Zeghidour, N. và cộng sự — SoundStream: An End-to-End Neural Audio Codec; và Défossez, A. và cộng sự — High Fidelity Neural Audio Compression (EnCodec). Hai bài này là nguồn của cơ chế lượng tử hoá dư ở mục 13.3, kể cả kỹ thuật bỏ tầng khi chạy.
- Ho, J., Jain, A., Abbeel, P. — Denoising Diffusion Probabilistic Models; và Nichol, A., Dhariwal, P. — Improved Denoising Diffusion Probabilistic Models. Bài thứ hai là nguồn của lịch cosine ở công thức (13.10), và phần lập luận của nó đúng bằng phép đo ở mục 13.5.2.

- Song, J., Meng, C., Ermon, S. — Denoising Diffusion Implicit Models. Nguồn của phép lấy mẫu nhảy bước ở công thức (13.11) và của toàn bộ mục 13.5.3.
- Griffin, D., Lim, J. — Signal Estimation from Modified Short-Time Fourier Transform. Thuật toán ở mục 13.6.2, và bài gốc nói rất rõ về chính giới hạn mà mục ấy đo được.
- Le Roux, J. và cộng sự — SDR: Half-Baked or Well Done? Nguồn của SI-SDR ở công thức (13.13), và là bài nên đọc trước khi báo bất kỳ con số tách nguồn nào.
- Fitzgerald, D. — Harmonic/Percussive Separation Using Median Filtering. Toàn bộ mục 13.7.3, và là ví dụ đẹp về một phương pháp không có tham số học mà vẫn rất mạnh.

Sách và tài liệu nền

- Lee, D. D., Seung, H. S. — Algorithms for Nonnegative Matrix Factorization. Luật cập nhật nhân dùng ở công thức (13.14).
- Müller, M. — Fundamentals of Music Processing. Sách nền tốt nhất cho phần phổ, thang mel và tách nguồn cổ điển; có mã Python đi kèm cho gần như mọi hình.
- Vaswani, A. và cộng sự — Attention Is All You Need. Đọc để hiểu cách một mô hình chia sẻ tham số giữa các ngữ cảnh giống nhau — chính là điều mà bảng đếm ở mục 13.4.1 không làm được.
- Goodfellow, I., Bengio, Y., Courville, A. — Deep Learning, chương về mô hình sinh. Đọc phần bàn về đánh đổi giữa các họ mô hình để đối chiếu với bảng ở mục 13.5.4.

Công cụ nên biết

- librosa — thư viện Python cho phân tích âm nhạc. Nó có sẵn phổ mel, HPSS và nhiều thứ khác của chương này; dùng nó để đối chiếu với bản tự viết, đừng dùng nó thay cho việc tự viết.
- mir_eval và museval — hai thư viện chuẩn để chấm tách nguồn. Chúng cài đầy đủ bộ SDR, SIR, SAR chứ không chỉ SI-SDR.
- webMUSHRA hoặc BeaqleJS — công cụ dựng phép nghe có đối chứng chạy trên trình duyệt. Đây là cách rẻ nhất để làm đúng bốn quy tắc ở mục 13.8.3.
- Bộ tự kiểm của mô-đun đi kèm sách. Ba mươi hai hàm mới của chương này đều có phép kiểm riêng, và người đọc chạy chính bộ ấy trên mã của mình để tự đối chiếu.

Đọc thêm theo hướng

- **Muốn đi sâu về sinh nhạc:** tìm đọc về MusicLM, MusicGen và AudioLM. Cả ba dựng trên đúng cơ chế token của mục 13.3, và đọc phần mô tả tầng token của chúng sẽ thấy lại toàn bộ mục ấy.

- **Muốn đi sâu về vocoder:** tìm đọc về HiFi-GAN và về BigVGAN. Chúng là câu trả lời hiện đại cho đúng bài toán mà mục 13.6 nêu ra, và phần bàn về hàm mất mát của chúng giải thích vì sao khoảng cách mel không đủ.
- **Muốn đi sâu về tách nguồn:** tìm đọc về Conv-TasNet, Demucs và Band-Split RNN. Chúng chính là những hệ thống đã bỏ khuôn mặt nạ để thoát khỏi trần 34 decibel ở mục 13.7.2.
- **Muốn đi sâu về tiếng Việt:** tìm đọc về mô hình hoá thanh điệu trong tổng hợp giọng, và về các bộ dữ liệu giọng Việt công khai. Đây là hướng mà một dự án nghiên cứu nhỏ đóng góp được thật, vì phần lớn hệ thống sẵn có đều thiết kế cho ngôn ngữ không thanh điệu.

Một lưu ý về các con số trong bài

> **Cách đọc các con số**
>
> Các con số về số lượng và bitrate — số mẫu, số khung, số token, kilobit mỗi giây, số từ mã — là kết quả tính chính xác từ công thức. Chúng đúng với mọi máy và mọi vật liệu.
>
> Các con số về SNR tái dựng, entropy chéo, tỉ lệ lặp lại, SI-SDR và khoảng cách phổ là kết quả *đo* trên vật liệu sinh bằng chính mô-đun của cuốn sách, nên chạy lại được hoàn toàn như cũ. Đổi vật liệu thì con số đổi theo; *hình dạng* của kết quả thì không.
>
> Riêng phép đo ở mục 13.5.3 chạy trên một bài toán đồ chơi một chiều chứ không trên âm thanh. Lý do và giới hạn của cách làm ấy nêu ngay tại mục.
>
> Vật liệu dùng trong bài là tiếng *tổng hợp*, không phải bản thu thật. Điều đó làm cho mọi phép đo lặp lại được, nhưng cũng làm cho chúng dễ hơn thực tế — nhất là ở mục tách nguồn. Bài tập 6 yêu cầu chạy lại trên vật liệu của chính người đọc.
>
> Mã Python in trong bài là mã đã chạy, không phải mã giả.