Day 2 · Your first speech model: a CNN that classifies Khmer sentences and speakers from log-mel

Goal in one line: turn a log-mel spectrogram into a class label with a small CNN, and learn that *how you split the data* decides what the model actually learns.

What you will build: a complete training kit (`src/train_kit.py` : cached features, augmentation, dataset, model, `train()` , `evaluate()` , `plot_curves()` , `confusion()` , `run_experiment()`) and four experiments on the same 48 Khmer clips: *what is said* (6 sentences), *who said it* (8 speakers), and two broken splits that look like one of them but measure something else.

Time: ~3 h

By the end you can

- batch variable-length audio into fixed-size log-mel "images" and explain the alternatives (masking, bucketing, pooling)
- write SpecAugment, noise-at-SNR, time-shift and gain augmentation by hand and say why 48 clips need them
- build a 3-block CNN, count its parameters, and say what its receptive field covers in time and frequency
- write a PyTorch training loop from scratch and read loss/accuracy curves for overfitting
- read a confusion matrix and a per-class precision/recall table
- choose a train/test split that answers a research question (leave-speakers-out vs leave-sentences-out) and spot leakage
- make an occlusion map to see which time-frequency region a model relies on
- compare a from-scratch CNN with logistic regression on pretrained embeddings

How to read this notebook (for the team)

You see	What it is	Do you need it?
Purple heading	a new section and its explanation	**yes**: read the bold lines and the "Why it matters" line
Visualization Version	pictures (some animated) that explain the section	**yes**: the first image of each section is the fastest way in
Additional reading	what the next code cell does and what its output means, plus extra notes	optional: read just the **In short** lines if you are busy
code cell + output	the real computation on our 48 Khmer clips	run it, look at the result

Fast path (≈ 40 min): purple text → first Visualization image → "In short" lines. **Full path (≈ 3 h):** read everything and run every cell top to bottom.

> The numbers quoted in the explanations come from the saved run in this file. If you re-run, accuracies can move by one or two clips (the test sets have only 12–16 clips).

1. The big idea: a spectrogram is an image, so use an image model

Day 1 ended with the log-mel spectrogram: an 80 × T matrix, 80 mel bands (frequency, low at the bottom) by T frames of 10 ms each. Bright horizontal stripes low down are pitch harmonics, bright blobs that move are vowel formants, vertical smears up to the top are hissy consonants. A CNN that was built to find edges and blobs in photos finds exactly those patterns in a spectrogram.

- A CNN (convolutional neural network) is a neural network built for images. Instead of connecting every input pixel to every neuron, it slides small filters across the image and looks for local patterns.
- "Blob" just means a bright patch: a region of the image that's brighter than its surroundings, with soft, rounded edges, a few rows tall and a few frames wide.

```
 waveform ──► log-mel (80 × 300) ──► normalize ──► [augment] ──► CNN ──► 6 or 8 logits ──► softmax ──► label
 16 kHz          "the image"           mean 0,        only at       conv blocks     one number
probabilities
 3 s             Day 1                 std 1          training      + avg pool      per class         sum to 1
```

The model outputs one logit per class. Softmax turns logits into probabilities, cross-entropy compares them with the true class, Adam nudges the weights, repeat. That loop is the whole of supervised deep learning; everything else today is about *what* the model learns and *how to know*.

- Cross-entropy is the loss function used for classification: −log(probability the model gave to the true class).
- Supervised = you train on examples that come with the correct answer (labels).
- Deep learning = the model is a neural network with many layers.
- Adam is a weight-update algorithm, an optimizer.
- A logit is the raw score the network outputs for one class, before it's turned into a probability. Any real number, positive or negative, no limits.

Worked example with real numbers: if the model outputs logits `[2.0, 0.5, 0.1, 0.1, 0.1, 0.1]` for a 6-sentence task, softmax gives `[0.55, 0.12, 0.08, 0.08, 0.08, 0.08]`. If the true class is index 0, cross-entropy = $-\log 0.55 = 0.60$. If the true class were index 2, it would be $-\log 0.08 = 2.50$. Chance level for 6 classes is $-\log(1/6) = 1.79$, so a training loss that stays near 1.8 means "learned nothing yet".

Cross-entropy (the loss function for classification):

$$\mathcal{L} = -\log \frac{e^{z_y}}{\sum_k e^{z_k}}$$

Two tasks on the same 48 clips. 8 phone speakers each read the same 6 sentences. So every clip has two labels, and we can ask the model two different questions:

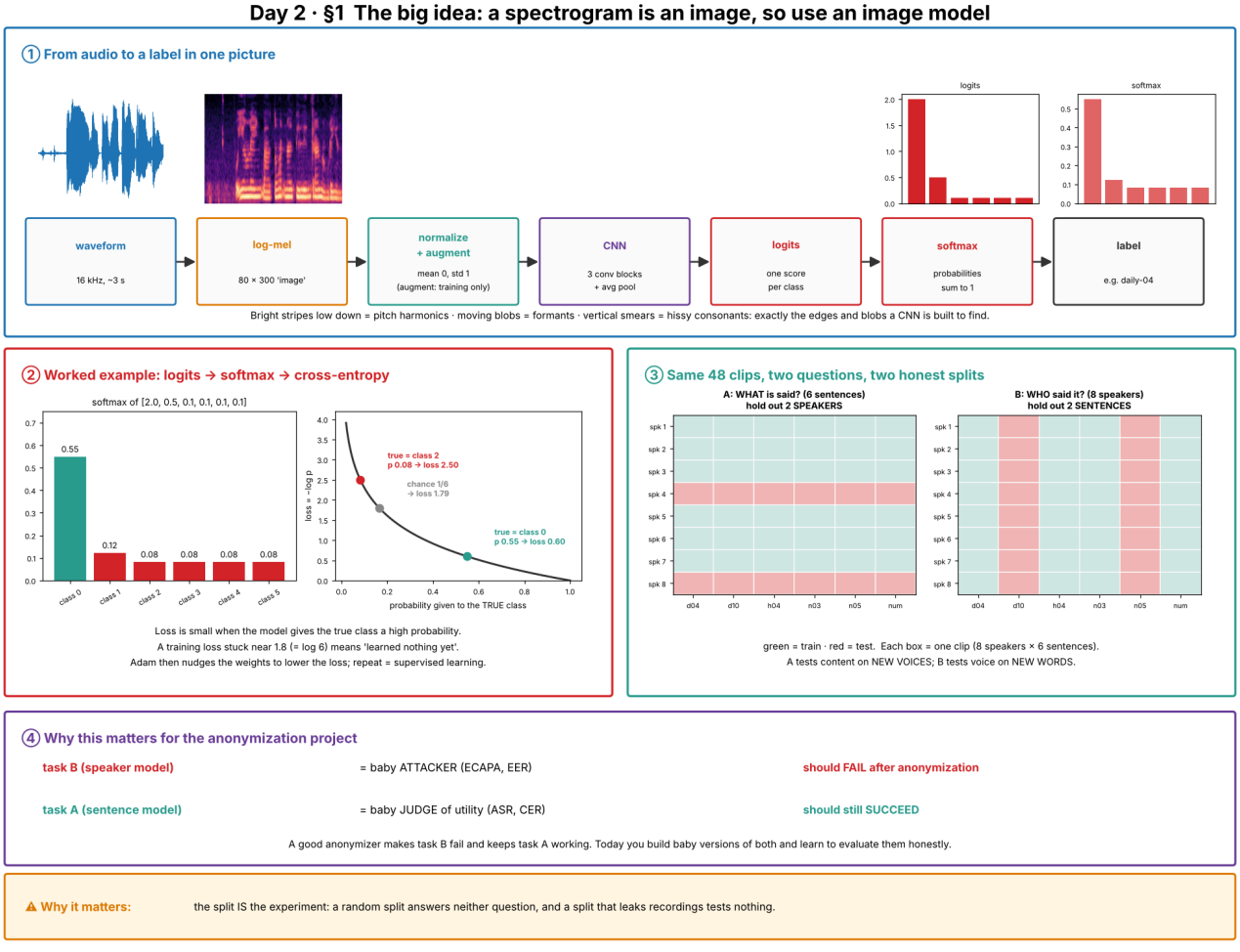
task	question	classes	train/test split	what the split tests
A: sentence	*what* is said?	6 sentences	**leave speakers out** (6 train, 2 test)	content that generalizes to new voices
B: speaker	*who* said it?	8 speakers	**leave sentences out** (4 train, 2 test)	identity that generalizes to new words

The split *is* the experiment. A random split over clips does not test either question, and a split that lets the same recording into both sets tests nothing at all. We will run both broken splits on purpose in section 7 and read what the numbers really mean.

In our research: the Khmer speaker-anonymization project is judged by exactly these two classifiers at scale.

- The *attacker* is a speaker classifier (task B: can a model still tell who is speaking after anonymization?).
- The *judge of utility* is a content model (task A, or an ASR system: is what was said still recognizable?). A good anonymizer makes task B fail and task A succeed. Today you build baby versions of both and learn to evaluate them honestly.

Visualization Version



Additional reading: you can read all of it or just read the "In short"

What this cell does

In short load the libraries (numpy, pandas, librosa, PyTorch), set the paths and seeds, and import the Day 1 helpers.

- `torch, nn, F`: PyTorch, the deep-learning library. `Dataset, DataLoader`: how PyTorch feeds data in batches.
- `from src.utils import ...`: the Day 1 helpers (`load`, `logmel`, `show_spec`, `median_f0`, ...) and the STFT settings.
- `torch.set_num_threads(1)`: for a model this small, one CPU thread is as fast and more predictable.

What the output means

In short it prints the PyTorch version and the STFT settings carried over from Day 1 (n_fft 512, hop 160, win 400).

- If this cell fails, nothing below runs. The exact version numbers don't matter.

```
import os, sys, warnings, time, random
os.environ.setdefault("OMP_NUM_THREADS", "4")   # set BEFORE numpy / torch are imported
sys.path.insert(0, "..")
warnings.filterwarnings("ignore")

from pathlib import Path
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import librosa, librosa.display
import torch, torch.nn as nn, torch.nn.functional as F
from torch.utils.data import Dataset, DataLoader

from src.utils import SR, N_FFT, HOP, WIN, load, play, show_wave, show_spec, logmel, median_f0

SR = 16000
DATA = Path("../data")
OUT = Path("../outputs/day2"); OUT.mkdir(parents=True, exist_ok=True)
np.random.seed(0); torch.manual_seed(0)
torch.set_num_threads(1)   # for a model this small, one thread is as fast as several and far more predictable on a laptop
print("torch", torch.__version__, "| SR", SR, "| STFT n_fft", N_FFT, "hop", HOP, "win", WIN)
%matplotlib inline
```

```
torch 2.14.1+cu130 | SR 16000 | STFT n_fft 512 hop 160 win 400
```

2. The dataset: 48 clips, two labels each, one fixed-size image per clip

2a. Load, look, and cache the slow step

`data/meta/probe_clips.csv` lists 48 clips: 8 phone speakers (4 female, 4 male) × 6 sentences. Durations run from 2.2 to 5.9 s. A CNN with a linear head needs every input to have the same shape, and even with global pooling (ours) we want same-size tensors to stack them into a batch. Options:

The problem first, then the options.

The problem these three options solve: your clips have different lengths (2.2 to 5.9 s, which is 220 to 590 frames), but a batch has to be one rectangular tensor. So every clip in a batch must end up with the same number of frames. There are three common ways to get there.

1. **Pad/crop to a fixed length** (today): crop long clips to 3 s, zero-pad short ones. Simple, fast, loses the ends of long clips. We take the *middle* 3 s at test time and a *random* 3 s window at training time (free time-shift augmentation).

More detail:

Option 1: pad or crop everything to one fixed length (used today)

Pick one length for all clips, here 3 s = 300 frames.

Longer clips get cropped. The 5.9 s clip keeps only 300 of its 590 frames, and the rest is thrown away.

Shorter clips get zero-padded. The 2.2 s clip gets 80 frames of silence added to reach 300.

2. **Pad to the longest in the batch + a mask** that tells the model which frames are real. Standard in ASR and Transformers; more code, no information lost.

More detail:

Option 2: pad to the longest clip in the batch, plus a mask

There's no global length. For each batch, find the longest clip and pad the others up to it.

Batch A, longest clip 4.2 s: everything padded to 420 frames.

Batch B, longest clip 5.9 s: everything padded to 590 frames.

3. **Bucket by length** so each batch has similar durations and padding is small.

Option 3: bucket by length

This is a refinement of Option 2. Before batching, sort or group clips by duration, so that each batch contains clips of similar length.

Without buckets, a 2.2 s clip might land in a batch with a 5.9 s clip and get padded with 3.7 s of nothing, so most of the computation is spent on padding.

With buckets, the 2.2 s clips go together and the 5.9 s clips go together, so padding is small in every batch.

Note: what 'CNN with a linear head' means

- CNN (the body / backbone): the conv + pool blocks. They turn the 80 × 300 image into feature maps.
- Linear head: the final nn.Linear layer that turns those features into the 6 or 8 logits. "Head" is the usual word for the last layer(s) that produce the task's output; "linear" because it's a plain

matrix multiply, no convolution.

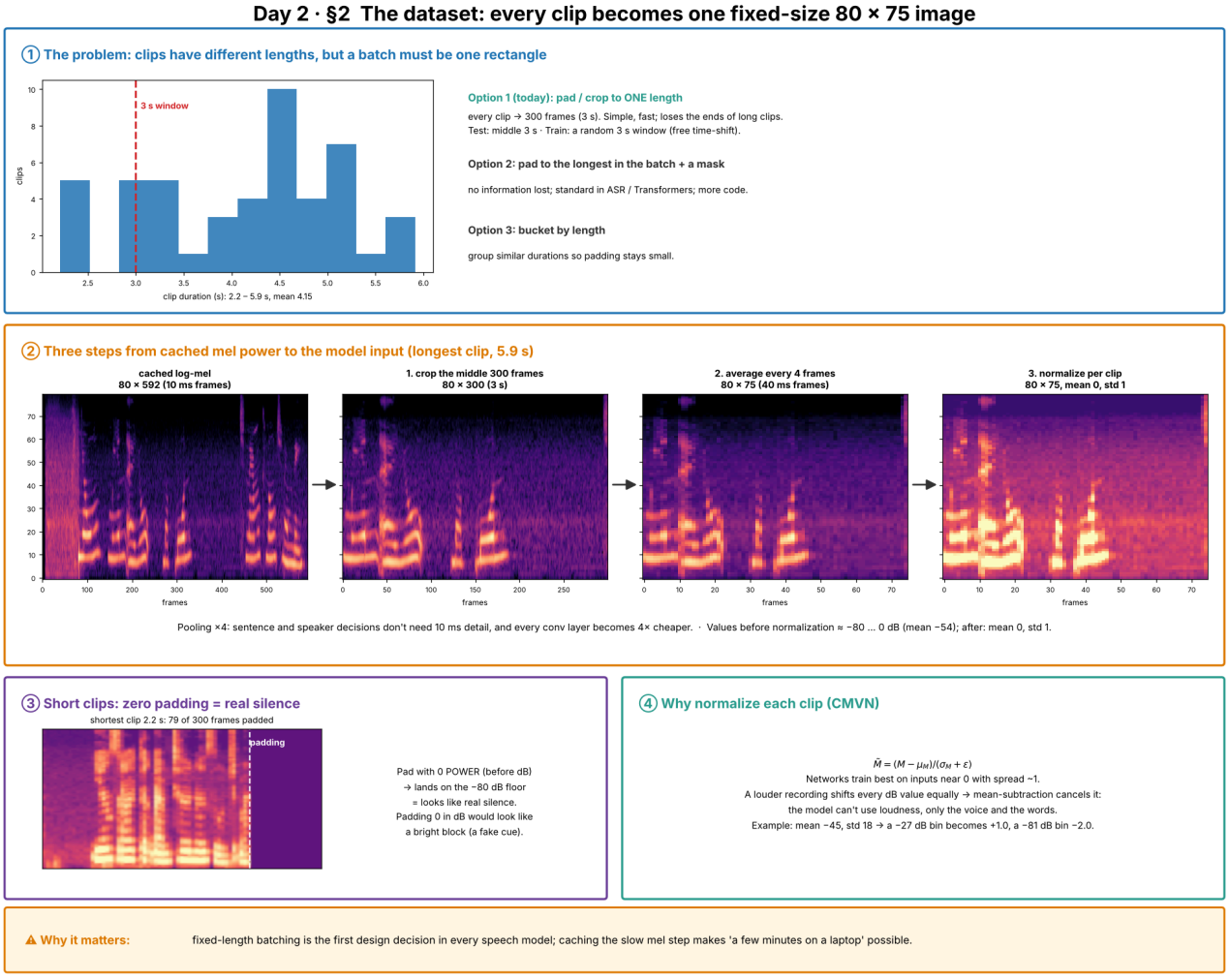
Our clips average 4.1 s, so a 3 s window keeps most of each sentence. 3 s at 10 ms hops = 300 frames: every clip becomes an 80 × 300 image.

One engineering decision up front. The mel filterbank on the waveform (`librosa.feature.melspectrogram`) is the slowest step of the whole pipeline, about 50 ms per clip on a laptop CPU. Recomputing it for every clip in every epoch would make training 10 × slower than the network itself. So we run it **once** per full clip, keep the mel *power* spectrogram (before the dB step) in memory, and do cropping, noise and the dB conversion on that cached array. `src.utils.logmel` is exactly `power_to_db(mel_power(x))`, so nothing changes numerically; we check that below.

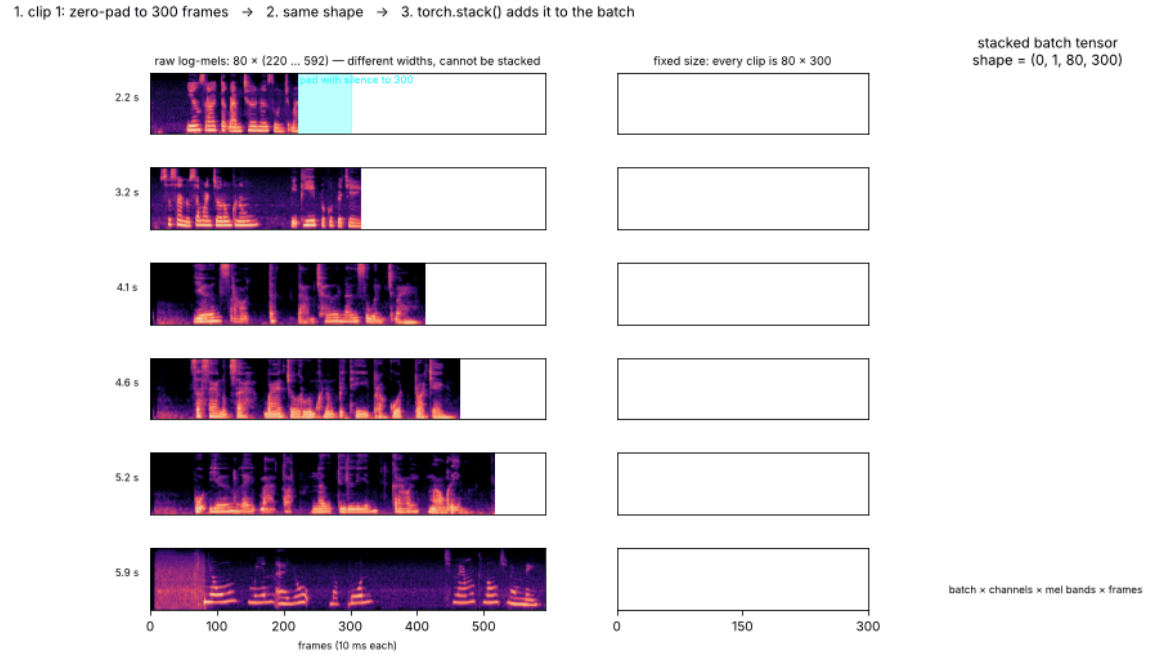
Why it matters: fixed-length batching is the first design decision in every speech model, and the one most often done badly (padding with silence that the model then learns to read as a class cue). Caching features is the second: it is what makes "a few minutes on a laptop" possible.

Visualization Version

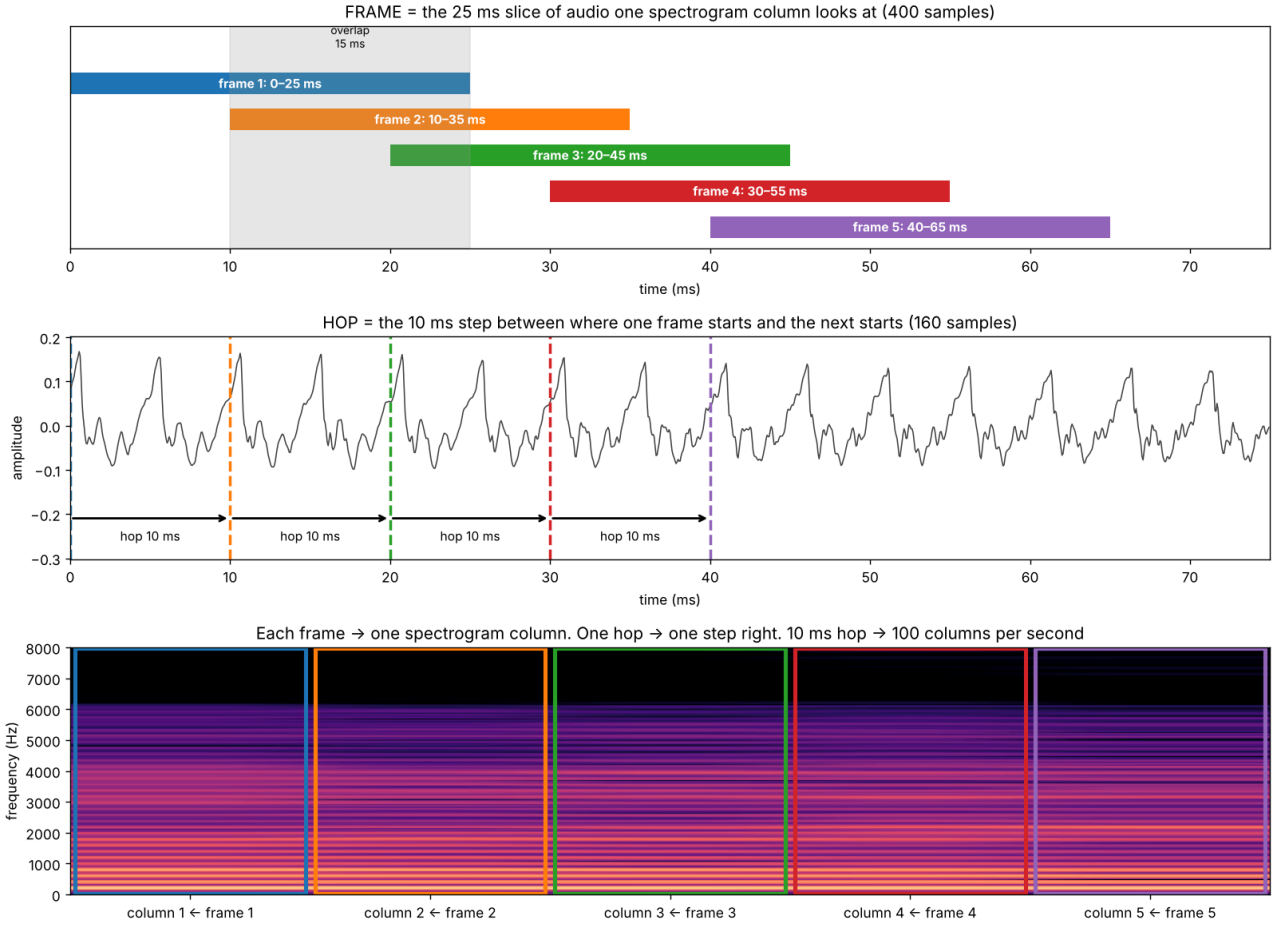
1st image is the summary. The others are additional: the batch animation, frame vs hop, and spectrogram vs log-mel.



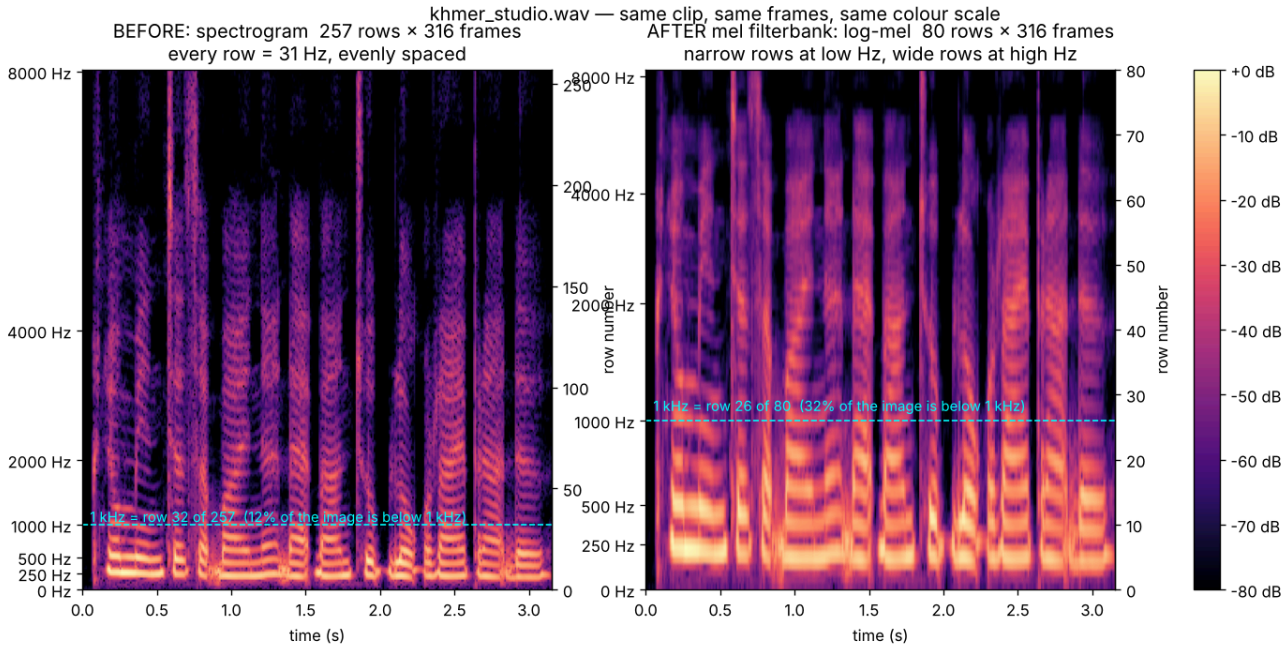
Animated: how 8 clips are stacked into one batch



What frame and hop look like



Spectrogram (257 rows) vs log-mel (80 rows)



Additional reading: you can read all of it or just read the "In short"

Clips and batches

In short each spectrogram is one clip (one sentence by one speaker, 80 × 300); each batch is 8 of them, a (8, 1, 80, 300) tensor.

So in training:

- Each spectrogram is one clip: one sentence said by one speaker, as an 80 × 300 grid.
- Each batch is 8 of those spectrograms, picked at random, as a (8, 1, 80, 300) tensor.

Additional reading: you can read all of it or just read the "In short"

Why frames overlap

In short overlap means every bit of audio lands near the middle of some frame, and you get long frames (frequency detail) AND frequent columns (timing detail).

Why a 25 ms frame with only a 10 ms hop (so frames overlap)?

1. The Hann window fades out the edges of each frame. Each frame is multiplied by the Hann bump before the FFT, so the audio near the start and end of a frame counts for almost nothing. No overlap (hop = 25 ms): the frame edges touch, and the audio at every boundary gets faded to near zero in both frames. A short sound that lands there, like the burst of a “k” or “t”, is almost invisible. With overlap (hop = 10 ms): whatever sits at the edge of one frame sits near the middle of the next. Every bit of audio gets fully counted in at least one frame.

2. You need two things that pull in opposite directions. Long frames (25 ms) for frequency detail. A frame has to hold a few pitch cycles to measure the harmonics properly. Too short, and the frequencies blur. Fine time steps (10 ms) for timing. Speech changes fast: a consonant burst lasts 5–10 ms, and formants glide between vowels. You want a new column often enough to catch that. Overlap lets you have both: each column sees 25 ms, but there's a new column every 10 ms. With hop = frame you'd get only 40 columns per second instead of 100. Bonus: overlap also makes the STFT reversible. With the right window and hop, istft rebuilds the waveform exactly, and that matters later for vocoders in Day 6.

Additional reading: you can read all of it or just read the "In short"

Why log-mel

In short log-mel matches human hearing, focuses the rows on speech frequencies, and is 3× smaller than the raw spectrogram.

We turn the spectrogram into a mel spectrogram (the log-mel) for three reasons:

Match human hearing. Our ears pick up small pitch differences at low frequencies but barely notice them at high frequencies.

- The mel scale gives low frequencies many narrow bands and high frequencies a few wide ones, the way the ear does.
- Focus on speech. Pitch, harmonics and the main formants sit below about 3 kHz. The mel scale spends most of its rows there instead of on the mostly empty high frequencies.
- Smaller input. 257 rows become 80 with almost nothing useful for speech lost. The model trains faster and has less to overfit on.

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short read the clip list, give every clip two labels (sentence 0–5, speaker 0–7), load all 48 waveforms once and **cache their mel power spectrograms** (the slow step).

- `SENTENCES`, `SPEAKERS`: the class names; `y_sent`, `y_spk`: the integer labels.
- `mel_power(x)`: the mel spectrogram **before** the dB step (so noise and cropping can be done in the power domain later).
- `to_logmel(P)`: the dB step, identical to Day 1's `logmel`.
- `MELS`: 48 cached arrays of shape (80, frames); `PSIG`: each clip's signal power (needed later for noise at a chosen SNR).

What the output means

In short 48 clips = 8 speakers (4 female, 4 male) × 6 sentences; durations **2.21 – 5.92 s**; caching takes ~3 s; the cached pipeline equals Day 1's `logmel` exactly.

- `frames per clip: 221 .. 592` : different lengths, which is why section 2b crops/pads them to one size.
- `cached pipeline == utils.logmel: True` : caching changes nothing numerically.
- The 6 sentences are printed with their Khmer text, then one audio player.

```
meta = pd.read_csv(DATA / "meta/probe_clips.csv")
SENTENCES = sorted(meta.sentence_id.unique())      # 6 classes for task A
SPEAKERS = sorted(meta.speaker.unique())           # 8 classes for task B
meta["y_sent"] = meta.sentence_id.map({s: i for i, s in enumerate(SENTENCES)})
meta["y_spk"] = meta.speaker.map({s: i for i, s in enumerate(SPEAKERS)})
print(len(meta), "clips |", len(SPEAKERS), "speakers x", len(SENTENCES), "sentences |",
      meta.groupby("speaker").gender.first().value_counts().to_dict())
print("durations: min %.2f  mean %.2f  max %.2f s" % (meta.duration.min(), meta.duration.mean(), meta.duration.max()))
for sid, txt in meta.drop_duplicates("sentence_id").sort_values("sentence_id")[["sentence_id", "text"]].values:
    print(f"  {sid:11s} {txt}")

# load all 48 waveforms once (16 kHz mono float32) and cache their mel POWER spectrograms (the slow step)
def mel_power(x, n_mels=80):
    # same settings as src.utils.logmel, but before the dB step, so we can add noise / crop in the power domain
    return librosa.feature.melspectrogram(y=x, sr=SR, n_fft=N_FFT, hop_length=HOP, win_length=WIN, n_mels=n_mels,
power=2.0).astype(np.float32)

def to_logmel(P):
    return librosa.power_to_db(P, ref=np.max)      # loudest bin = 0 dB, floor at -80 dB: identical to utils.logmel

t0 = time.time()
WAVES = [load(DATA / "audio/probe" / f"{u}.wav") for u in meta.utt_id]
MELS = [mel_power(x) for x in WAVES]                                  # list of (80, frames) arrays, frames = 221 .. 592
PSIG = np.array([float(np.mean(x ** 2)) for x in WAVES])              # signal power per clip, needed for noise at a given SNR
print(f"loaded + cached {len(WAVES)} clips in {time.time()-t0:.1f} s | frames per clip: {min(m.shape[1] for m in MELS)} ..
{max(m.shape[1] for m in MELS)}")
print("cached pipeline == utils.logmel:", np.allclose(to_logmel(MELS[0]), logmel(WAVES[0])))
play(WAVES[0])
```

```
48 clips | 8 speakers x 6 sentences | {'female': 4, 'male': 4}
durations: min 2.21  mean 4.15  max 5.92 s
  daily-04    ពួកយើងលេងហាល់ភាគនៅទីលានក្បែយម៉ោងរៀន
  daily-10    យើងរស្មោចទឹកដើមឈើនៅសួនច្បារ
  health-04   ផឹកទឹកឲ្យបានច្រើនរៀងរាល់ការខ្សោតទឹក
  news-03     អ្នកជំនាញព្យាករណ៍ថានឹងមានភ្លៀងធ្លាក់នៅសប្តាហ៍ក្រោយ
  news-05     សិស្សានុសិស្សបានចូលរួមកន្លងពិធីប្រកួតប្រជែង
  numbers-01  ខ្ញុំមានអាយុដប់ប្លន់ឆ្នាំកន្លងឆ្នាំនេះ
loaded + cached 48 clips in 1.5 s | frames per clip: 221 .. 592
cached pipeline == utils.logmel: True
```

2b. Fixed length, frame rate, and per-utterance normalization

Section 2b covers the last preparation steps before the CNN. By this point each clip is a cached mel power array, 80 rows × however many frames it happens to have. 2b turns that into the exact input the model gets:

```
cached mel power (80 × 220…590)
  → 1. crop/pad      → 80 × 300   (3 s, 10 ms frames)
  → 2. average ×4    → 80 × 75    (3 s, 40 ms frames)
  → 3. normalize     → 80 × 75    (mean 0, std 1)
```

Three small steps turn a cached mel power array into the model's input:

1. **Crop or pad to 300 frames (3 s).** Pad with zeros in the power domain: after the dB step that is the -80 dB floor, the same thing Day 1's zero-padded waveform would give.

More detail:

Step 1: crop or pad to 300 frames

- What: this is Option 1 from section 2a. Long clips are cropped to 300 frames (the middle at test time, a random window during training), and short clips are padded up to 300.
- The detail it points out: the padding is done with zeros in the power domain, before converting to dB. Zero power means “no sound at all”. When that’s converted to dB, it lands on the −80 dB floor, the quietest value in the image.
- Why: the padding then looks exactly like real silence. It’s the same as if you’d added silence to the waveform and computed the log-mel afterwards. If you padded in the dB domain with 0 instead, 0 dB would mean “as loud as the loudest bin”, and the padding would look like a bright block of noise. That’s a fake pattern the model could latch onto.

2. **Average every 4 frames: 10 ms → 40 ms frames, 300 → 75.** A sentence or speaker decision does not need 10 ms resolution (that is for phoneme boundaries and pitch tracking). Keeping the Day 1 hop and then pooling costs nothing and makes every conv layer 4 × cheaper, which is the difference between 15 s and 1 min per training run on a CPU. The model input is therefore **80 × 75**.

More detail:

Note: An utterance is just one spoken clip, here one sentence by one speaker.

Step 2: average every 4 frames (300 → 75)

- What: groups of 4 neighbouring columns are averaged into one. Each column now covers 40 ms instead of 10 ms.

Why group 4 neighbours and average them:

You take the 300 columns in groups of four, so columns 1 to 4, then 5 to 8, and so on, and average each group into one column. That leaves 75. It’s done separately for every one of the 80 mel rows. For example, if one row reads −30, −34, −32, −28 across four frames, those become a single value of −31. Each new column now stands for 40 ms of sound instead of 10 ms, but the clip is still 3 seconds long. Think of it like a photo shrunk to a quarter of its width: it gets blurrier side to side, but you can still tell what’s in it. For telling sentences or speakers apart, that blur doesn’t hurt, and training runs four times faster

- Why:
  - The tasks don’t need fine timing. 10 ms resolution matters for things like phoneme boundaries or pitch tracking. “Which sentence?” and “who is speaking?” can be answered from coarser 40 ms steps.
  - 4× less computation. Every conv layer slides over 75 columns instead of 300. On a CPU, that’s the difference between about 15 seconds and a minute per training run.
- Why not just use a 40 ms hop in the STFT? This ties back to the overlap discussion. With 25 ms frames and a 40 ms hop, the frames wouldn’t touch, so 15 ms of audio would be skipped between each pair. Keeping the 10 ms hop and then averaging means every bit of audio still counts, and the cached Day 1 features can be reused as they are.

3. **Per-utterance mean/variance normalization.** Log-mel values are in dB relative to the loudest bin, roughly in [-80, 0]. A network trains faster and more stably when its inputs have mean ≈ 0 and std ≈ 1, so we subtract each clip's mean and divide by its std (in speech papers: CMVN, cepstral mean and variance normalization).

$$\tilde{M} = \frac{M - \mu_M}{\sigma_M + \varepsilon}$$

- M = the clip's log-mel values (80 × 75 numbers)
- μ_M = the average of all those values, for this clip only
- σ_M = their standard deviation (how spread out they are)
- ε = a tiny number (like 1e-5), so you never divide by zero, for example on a clip that's pure silence
- M̃ = the normalized result: mean 0, std 1
- Worked example: the clip has mean −45 dB and std 18 dB.
  - A bin at −27 dB → (−27 + 45) / 18 = +1.0, one std louder than average
  - A bin at −81 dB → (−81 + 45) / 18 = −2.0, two std quieter than average
- Why normalize at all: log-mel values sit around −80 to 0 dB. Neural networks are set up assuming their inputs are roughly centred on 0 with a spread of about 1. Weight initialization and learning rates are tuned for that. Feed in numbers around −45 and the first layer's outputs come out large and lopsided. Training is slower and less stable, because the network first has to learn to cancel that offset before it can learn any patterns.
- Why per-utterance normalization (CMVN)?
  - Neural networks train best on centred, small inputs. Normalizing gives mean 0 and std 1, so most values fall roughly between −2 and +2. Without it, the first layer has to learn to cancel an offset of about −45 dB before it can learn any patterns, which makes training slower and less stable.
  - It removes recording loudness. A louder recording just shifts every dB value up by the same amount, and subtracting the clip's own mean cancels that shift. So the model can't tell speakers apart by how loud they were recorded, and has to use the actual voice.
  - It puts every clip on the same footing. Clips recorded at different levels, like −30 dB versus −60 dB on average, end up on one shared scale. The only differences left are in the speech pattern itself.
  - Per-band normalization removes the microphone's colouring. Every phone boosts or cuts some frequencies, which shows up as a constant offset in each mel row. Subtracting each of the 80 rows' own mean removes it, so the model, or an attacker in anonymization, can't recognise the phone instead of the person.

Downside: after normalization the model can't measure loudness, because that information is gone.

Worked example: a clip whose log-mel has mean -45 dB and std 18 dB. A bin at -27 dB becomes $(-27 + 45)/18 = +1.0$; a bin at -81 dB becomes -2.0. Per *utterance* (not per dataset) also removes recording-level differences: a quiet phone and a loud phone give the same normalized image, and a global gain change becomes invisible to the model. That is a feature for content and speaker tasks (loudness is not identity), and a bug if you ever wanted the model to measure loudness.

Why it matters: without normalization, the first conv layer has to learn an offset of -45 dB before it sees any pattern. With it, all 48 clips start on the same footing.

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short define the three preparation steps (`crop_frames` , `pool_time` , `normalize`) and the full `features()` pipeline, then check the shapes on one clip.

- `crop_frames(P, mode)` : middle 300 frames at test time, a random window in training, zero-padding for short clips.
- `pool_time(M)` : average every 4 frames, 300 → 75.
- `normalize(M)` : subtract the clip's mean, divide by its std.
- `features(P)` : crop → pool → dB → normalize → an (80, 75) image.

What the output means

In short the log-mel is **80 × 300** (−80 … 0 dB, mean −54); the model input is **80 × 75** with mean **0** and std **1**; the shortest clip (2.21 s) gets **79 of 300** frames padded.

- After normalization values sit roughly between −1.5 and +3: centred and small, which is what a network trains well on.

```
N_FRAMES = 300          # 3 s at 10 ms hops, before time pooling
TIME_POOL = 4           # average groups of 4 frames -> 75 frames of 40 ms

def crop_frames(P, n=N_FRAMES, mode="center", rng=None):
    # pad with zeros at the end, or crop: the middle (test), a random window (train = time shift), or the start/end
    T = P.shape[1]
    if T >= n:
        if mode == "random":
            start = int((rng or np.random.default_rng()).integers(0, T - n + 1))
        elif mode == "start":
            start = 0
        elif mode == "end":
            start = T - n
        else:
            start = (T - n) // 2
        return P[:, start:start + n]
    return np.pad(P, ((0, 0), (0, n - T)))

def pool_time(M, k=TIME_POOL):
    T = M.shape[1] // k * k
    return M[:, :T].reshape(M.shape[0], T // k, k).mean(axis=2)   # (80, 300) -> (80, 75)

def normalize(M, eps=1e-5):
    return (M - M.mean()) / (M.std() + eps)          # per utterance: mean 0, std 1

def features(P, mode="center"):
    # cached mel power -> 300 frames -> 40 ms frames -> dB -> normalized (80, 75) image
```

```
    return normalize(to_logmel(pool_time(crop_frames(P, mode=mode)))).astype(np.float32)

M0 = to_logmel(crop_frames(MELS[0]))
print("log-mel      shape", M0.shape, " range %.0f .. %.0f dB, mean %.1f, std %.1f" % (M0.min(), M0.max(), M0.mean(),
M0.std()))
X0 = features(MELS[0])
print("model input  shape", X0.shape, " range %.1f .. %.1f,    mean %.2f, std %.2f" % (X0.min(), X0.max(), X0.mean(),
X0.std()))
# notice: the shortest clip (2.2 s) gets zero-padded; the padded region becomes a flat block at the floor
short = meta.duration.idxmin()
print("shortest clip:", meta.utt_id[short], meta.duration[short], "s -> padded frames:", N_FRAMES - MELS[short].shape[1],
"of 300")
```

```
log-mel      shape (80, 300)  range -80 .. 0 dB, mean -54.1, std 17.6
model input  shape (80, 75)  range -1.5 .. 3.0,    mean -0.00, std 1.00
shortest clip: iany_046e9e11_d659c7f0 2.21 s -> padded frames: 79 of 300
```

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short draw the model inputs for **one speaker's 6 sentences**, then **one sentence read by 4 speakers**.

- `show_input` draws one (80, 75) input. Top: same voice, different words. Bottom: same words, different voices (2 female, 2 male) with their pitch (F0).

What the output means

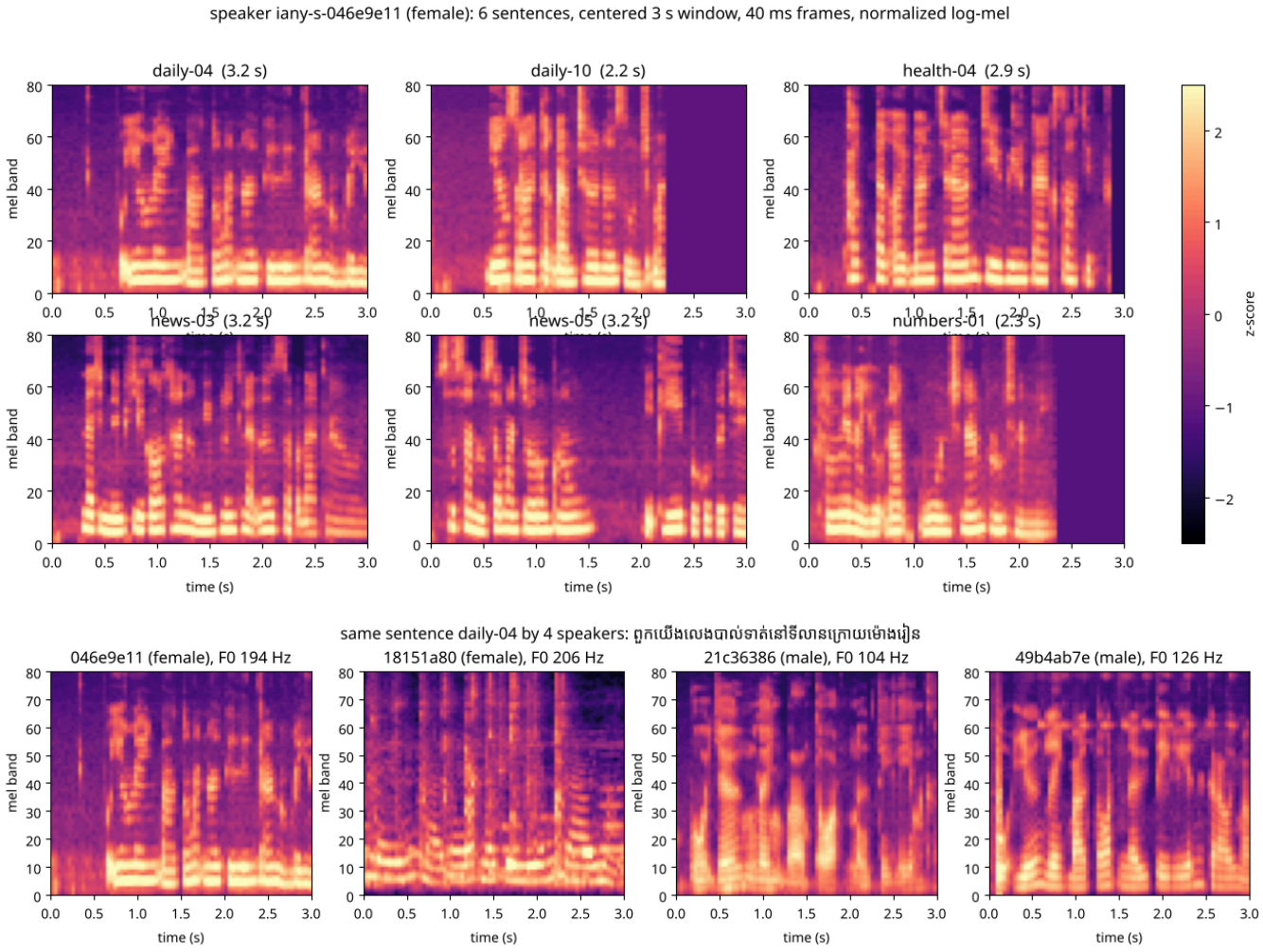
In short top grid: the **blob pattern changes with the words**; bottom row: the pattern stays, but the **low-band stripes change with the voice**.

- See the How-to-read notes below. The flat dark block on the right of the 2.2 s clip is the zero padding.

```
# the 6 sentences read by ONE speaker, as a grid of model inputs
def show_input(X, ax, title):
    im = ax.imshow(X, origin="lower", aspect="auto", cmap="magma", vmin=-2.5, vmax=2.5, extent=[0, 3, 0, 80])
    ax.set(title=title, xlabel="time (s)", ylabel="mel band")
    return im

spk = SPEAKERS[0]
rows = meta[meta.speaker == spk].sort_values("sentence_id")
fig, axes = plt.subplots(2, 3, figsize=(15, 6))
for ax, (_, r) in zip(axes.ravel(), rows.iterrows()):
    im = show_input(features(MELS[r.name]), ax, f"{r.sentence_id}  ({r.duration:.1f} s)")
fig.suptitle(f"speaker {spk} ({rows.gender.iloc[0]}): 6 sentences, centered 3 s window, 40 ms frames, normalized log-mel",
y=1.02)
plt.colorbar(im, ax=axes, fraction=0.02, label="z-score")
plt.show()

# the SAME sentence by 4 different speakers (2 female, 2 male)
sid = SENTENCES[0]
rows = meta[meta.sentence_id == sid]
pick = pd.concat([rows[rows.gender == "female"].head(2), rows[rows.gender == "male"].head(2)])
fig, axes = plt.subplots(1, 4, figsize=(16, 3))
for ax, (_, r) in zip(axes, pick.iterrows()):
    show_input(features(MELS[r.name]), ax, f"{r.speaker[-8:]} ({r.gender}), F0 {median_f0(WAVES[r.name]):.0f} Hz")
fig.suptitle(f"same sentence {sid} by 4 speakers: {rows.text.iloc[0]}", y=1.05)
plt.show()
```



Additional Reading

In short blob pattern over time = WHAT is said (sentence cue); stripe spacing in the low bands = WHO said it (speaker cue).

How to read these plots. Each panel is one clip after cropping, pooling and normalization: time left to right (0–3 s, 75 frames of 40 ms), mel band bottom (low frequency) to top (high). Bright = louder than the clip's average, dark = quieter.

- **Top grid (one speaker, six sentences):** the *shapes* differ panel to panel, because the words differ: where the bright blobs sit in time, how many syllables, where the vertical dark gaps (pauses) fall. The short clip `daily-10` (2.2 s) ends in a flat dark block on the right: that is the zero padding, and the model will see it in every short clip.
- **Bottom row (one sentence, four speakers):** the *shape* is similar across panels: the same syllable pattern, the same bright/dark rhythm, stretched a bit by speaking rate. What differs is the texture of the lowest ~15 bands: the stripes (pitch harmonics) are spaced wider and sit higher for the female speakers (F0 around 200 Hz) than for the male speakers (F0 around 120 Hz). Those stripes are the easiest "who" cue; the blob pattern over time is the "what" cue.
- The sentence model should look at the blob pattern over time; the speaker model should look at the stripe spacing and the overall band balance. Section 8 checks whether they do.

3. A PyTorch Dataset, a DataLoader, and augmentation by hand

- A `Dataset` returns one `(tensor, label)` pair;
- A `DataLoader` stacks them into batches and shuffles.
- Augmentation lives inside `__getitem__`, so every epoch sees a slightly different version of every clip.

Why augmentation matters with 48 clips. A CNN with 24k parameters can memorize 36 training spectrograms pixel by pixel in a few epochs; after that it stops learning anything general. Augmentation makes exact memorization impossible (the same image never comes back) and tells the model which changes do *not* change the label:

augmentation	what it changes	what it teaches	implementation
time shift (random 3 s crop)	where in time the sentence sits	position does not matter	pick a random window of the cached array
additive noise at SNR 20–40 dB	background hiss	ignore the noise floor	add noise power scaled so signal/noise = SNR
SpecAugment	zero out a random time band and a random frequency band	do not depend on one band or one moment	10 lines of slicing
gain (± 6 dB)	overall loudness	loudness is not the label (see Predict box)	multiply waveform by $10^{g/20}$

Noise at a chosen SNR: with signal power P_s and target SNR s dB, the noise power must be $P_n = P_s / 10^{s/10}$. At 20 dB SNR the noise is 100 × weaker in power (10 × in amplitude) than the speech. On a waveform you add Gaussian samples with std $\sqrt{P_n}$. On our *cached mel power* arrays we add the same noise in the mel domain: white noise has a fixed expected mel power per band (we measure that profile once on 5 s of unit-variance noise), and each cell of a noise periodogram fluctuates around its mean with an exponential distribution, so we draw $\mathrm{Exp}(1)$ multipliers. The result is equivalent in expectation and 100 × faster; the plot below checks it by eye.

SpecAugment (Park et al., 2019) is just: pick a random band of up to 8 mel bins and set it to 0 (the mean, after normalization); pick a random span of up to 6 frames (240 ms) and set it to 0. It is the single most effective trick for small speech datasets.

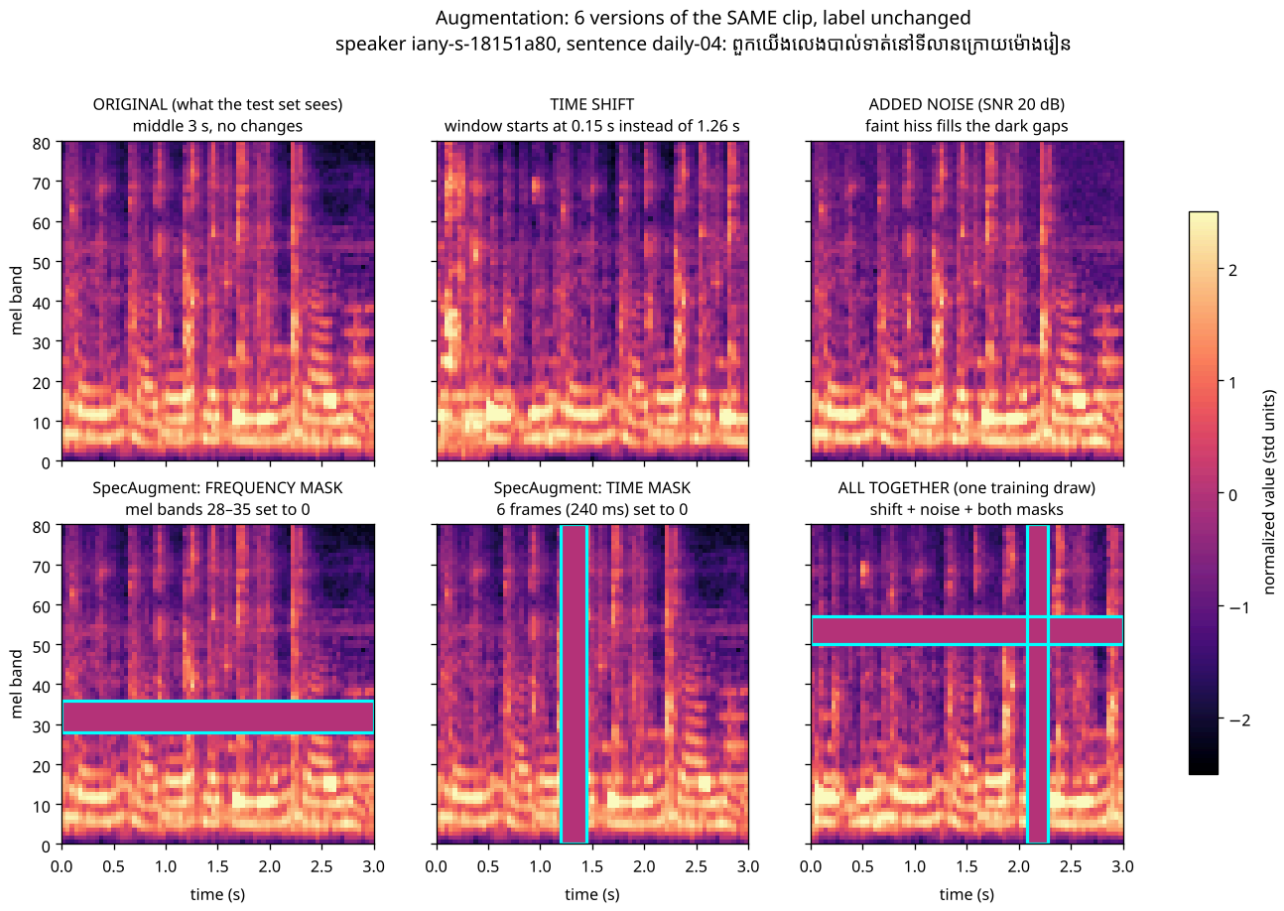
Dose matters. Each of noise and SpecAugment is applied with probability 0.3 per clip per epoch, and the time shift always. With only 200 gradient steps in a run, stronger augmentation (always-on noise at 5–30 dB, two masks each) stops the model from fitting the training set at all; we tried, and the speaker model fell from 0.8 to 0.3 accuracy. Augmentation is a regularizer, and like every regularizer it has a dose.

> **Predict:** after per-utterance normalization, does a gain change (± 6 dB) alter the input the model sees?

▶ Answer

Visualization Version

Augmentation: 6 versions of the same clip, label unchanged.



Additional reading: you can read all of it or just read the "In short"

In short augmentation stops the model from memorizing exact pixels, so it has to learn the patterns that survive the changes; the dose must be moderate.

Overall augmentation prevent CNN model from memorize but rather learn to catch pattern.

Because every training clip looks a little different each epoch, memorizing the exact pixels stops paying off. The only thing that works every time is the pattern that survives the changes: the syllable rhythm for the sentence, and the harmonic stripes and band balance for the speaker.

It only helps if the dose is right, though. Too much distortion and the model can't learn the patterns either.

Additional reading: you can read all of it or just read the "In short"

Which function does what

In short only time shift, mel-domain noise and SpecAugment are used in training; the waveform noise and gain functions are for the demo and the check.

Function	Role	Used in training?
`features_from_power`	the full pipeline: crop to 300 frames → [noise] → average to 75 frames → dB → normalize → [SpecAugment]	yes, every load
`add_noise_mel`	adds noise directly to the **cached mel power**, using a measured white-noise profile per band × random fluctuation per cell	**yes**: fast, no waveform needed
`add_noise_snr`	adds Gaussian noise to the **waveform**	no, only in the demo, to show the mel version looks the same
`spec_augment`	picks a random strip of up to 8 bands and up to 6 frames and sets them to 0	yes
`change_gain`	louder or quieter waveform	no: normalization would cancel it anyway (the check at the end prints a difference of ≈ 0)

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short implement the augmentations (SpecAugment, noise at an SNR, gain), the full `features_from_power` pipeline and the `LogMelDataset`; then time one pass, pull one batch and test the Predict box.

- `spec_augment`: zero one random strip of ≤ 8 mel bands and one of ≤ 6 frames.
- `add_noise_mel`: add white noise directly to the cached mel power (fast); `add_noise_snr`: the waveform version (demo only).
- `LogMelDataset`: returns `(tensor (1, 80, 75), label)`; with `train=True` it applies time shift always, noise 30 % of the time, SpecAugment 30 % of the time.

What the output means

In short one augmented pass over all 48 clips takes **well under a second**; one batch is **(8, 1, 80, 75)**; a +6 dB gain changes the normalized input by **0.0000**.

- `one batch: (8, 1, 80, 75) labels [...]`: 8 clips × 1 channel × 80 mel bands × 75 frames, plus their 8 sentence labels.
- `max |difference| ... after +6 dB gain: 0.0000`: normalization makes loudness invisible, so gain augmentation is pointless here (answer to the Predict box).

```
def spec_augment(M, n_freq_masks=1, max_freq=8, n_time_masks=1, max_time=6, rng=None):
    # SpecAugment: zero out random horizontal (frequency) and vertical (time) bands of a normalized log-mel
    rng = rng or np.random.default_rng()
    M = M.copy(); n_mels, n_frames = M.shape
    for _ in range(n_freq_masks):
        w = int(rng.integers(0, max_freq + 1)); f0 = int(rng.integers(0, max(1, n_mels - w)))
        M[f0:f0 + w, :] = 0.0                          # a horizontal stripe of "nothing"
    for _ in range(n_time_masks):
        w = int(rng.integers(0, max_time + 1)); t0 = int(rng.integers(0, max(1, n_frames - w)))
        M[:, t0:t0 + w] = 0.0                          # a vertical stripe of "nothing"
    return M
# snr = signal to noise ratio
def add_noise_snr(x, snr_db, rng=None):
    # waveform version: Gaussian samples with the right power
    rng = rng or np.random.default_rng()
    p_noise = (np.mean(x ** 2) + 1e-12) / (10 ** (snr_db / 10))
    return (x + rng.normal(0, np.sqrt(p_noise), size=len(x))).astype(np.float32)

NOISE_PROFILE = mel_power(np.random.default_rng(0).normal(0, 1, 5 * SR).astype(np.float32)).mean(axis=1)   # (80,) mel
power of unit white noise

def add_noise_mel(P, p_sig, snr_db, rng=None):
    # mel-power version: expected noise power per band x exponential fluctuation per cell
    rng = rng or np.random.default_rng()
    sigma2 = p_sig / (10 ** (snr_db / 10))
    return (P + sigma2 * NOISE_PROFILE[:, None] * rng.exponential(1.0, size=P.shape)).astype(np.float32)

def change_gain(x, gain_db):
    return np.clip(x * 10 ** (gain_db / 20), -1, 1).astype(np.float32)

def features_from_power(P, mode="center", rng=None, p_sig=None, snr_db=None, augment_spec=False):
    # cached mel power -> fixed 300 frames -> [noise] -> 40 ms frames -> dB -> normalize -> [SpecAugment]
    P = crop_frames(P, mode=mode, rng=rng)
    if snr_db is not None:
        P = add_noise_mel(P, p_sig, snr_db, rng=rng)
    M = normalize(to_logmel(pool_time(P)))
    if augment_spec:
        M = spec_augment(M, rng=rng)
    return M.astype(np.float32)

class LogMelDataset(Dataset):
    # cached mel power arrays + signal powers + integer labels -> (1, 80, 75) tensors; train=True switches on augmentation
    def __init__(self, mels, p_sigs, labels, train=False, seed=0, augment=True, snr_range=(20, 40),
                 p_noise=0.3, p_mask=0.3, time_shift=True, crop_mode="center"):
        self.mels, self.p_sigs, self.labels = list(mels), np.asarray(p_sigs, float), np.asarray(labels, dtype=np.int64)
        self.train, self.augment = train, augment
        self.rng = np.random.default_rng(seed)        # own RNG so runs are reproducible
        self.snr_range, self.p_noise, self.p_mask, self.time_shift, self.crop_mode = snr_range, p_noise, p_mask,
time_shift, crop_mode
    def __len__(self):
        return len(self.mels)
    def features(self, i):
        if self.train and self.augment:
            # each augmentation is applied with a probability, so the model also sees clean, centered clips
            snr = self.rng.uniform(*self.snr_range) if self.rng.random() < self.p_noise else None
            mask = self.rng.random() < self.p_mask
            return features_from_power(self.mels[i], mode="random" if self.time_shift else self.crop_mode, rng=self.rng,
                                       p_sig=self.p_sigs[i], snr_db=snr, augment_spec=mask)
        return features_from_power(self.mels[i], mode=self.crop_mode)      # deterministic at test time
    def __getitem__(self, i):
        return torch.from_numpy(self.features(i))[None], int(self.labels[i])

ds = LogMelDataset(MELS, PSIG, meta.y_sent, train=True, seed=1)
t0 = time.time(); _ = [ds[i] for i in range(len(ds))]
print(f"one augmented pass over 48 clips: {1000*(time.time()-t0):.0f} ms (cached features: cheap)")
xb, yb = next(iter(DataLoader(ds, batch_size=8, shuffle=True)))
print("one batch:", tuple(xb.shape), "labels", yb.tolist())          # (8, 1, 80, 75): batch, channel, mel, frames

# check the Predict box: pure gain vs normalized log-mel
x = WAVES[0]
d = np.abs(features(mel_power(x)) - features(mel_power(change_gain(x, +6)))).max()
print(f"max |difference| in the normalized input after +6 dB gain: {d:.4f}  (0 = invisible to the model)")
```

```
one augmented pass over 48 clips: 15 ms (cached features: cheap)
one batch: (8, 1, 80, 75) labels [0, 5, 1, 2, 1, 5, 3, 2]
max |difference| in the normalized input after +6 dB gain: 0.0000  (0 = invisible to the model)
```

Additional reading: you can read all of it or just read the "In short"

The augmentations in one line each

In short time shift = position doesn't matter; noise = ignore hiss; SpecAugment = don't rely on one spot; gain = not needed after normalization.

- Time shift: shows a different 3-second slice of the clip each time, so the model learns the sentence's position doesn't matter.
- Added noise: mixes in faint hiss, so the model learns to ignore background noise.
- SpecAugment: blanks out a random band and a random moment, so the model can't rely on any single spot.
- Gain: makes the clip louder or quieter, to teach that loudness isn't the label. Normalization already removes loudness, though, so it's not needed here.
- Random padding placement (not used here): short clips are always padded at the end. Splitting the padding before and after the speech would move it around, an idea for later.

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short draw six versions of the same clip: original, noise (waveform vs mel domain), time shift, SpecAugment, and all together.

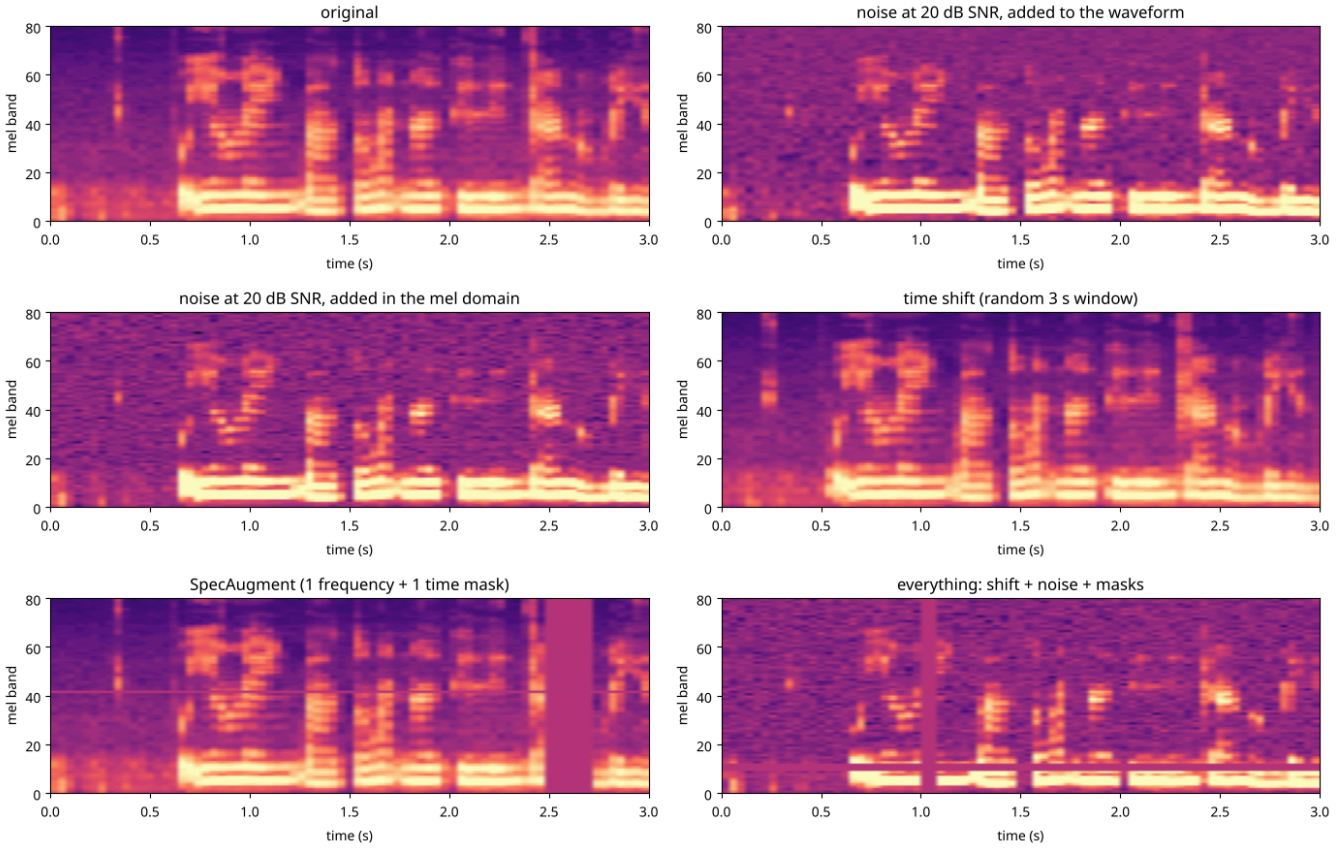
- Same clip every time, so any difference you see is the augmentation.

What the output means

In short noise turns the dark gaps into haze; time shift slides the blobs; SpecAugment adds one horizontal and one vertical **zero** bar.

- See the How-to-read notes below.

```
rng = np.random.default_rng(3)
P = crop_frames(MELS[0])
x3 = WAVES[0][(len(WAVES[0]) - 3 * SR) // 2:][:3 * SR]                          # the same centered 3 s as a waveform
variants = {
    "original": features_from_power(MELS[0]),
    "noise at 20 dB SNR, added to the waveform": normalize(to_logmel(pool_time(mel_power(add_noise_snr(x3, 20, rng))[:,
:N_FRAMES]))),
    "noise at 20 dB SNR, added in the mel domain": features_from_power(MELS[0], p_sig=PSIG[0], snr_db=20, rng=rng),
    "time shift (random 3 s window)": features_from_power(MELS[0], mode="random", rng=np.random.default_rng(7)),
    "SpecAugment (1 frequency + 1 time mask)": features_from_power(MELS[0], augment_spec=True, rng=rng),
    "everything: shift + noise + masks": features_from_power(MELS[0], mode="random", rng=rng, p_sig=PSIG[0], snr_db=20,
augment_spec=True),
}
fig, axes = plt.subplots(3, 2, figsize=(14, 9))
for ax, (name, M) in zip(axes.ravel(), variants.items()):
    show_input(M, ax, name)
plt.tight_layout(); plt.show()
```



Additional Reading

In short waveform noise and mel-domain noise look the same (that is why the fast mel version is safe); masks force the model to use more than one spot.

How to read this plot. Top-left is the clean reference.

- **Noise at 20 dB SNR (waveform vs mel domain):** both turn the dark pauses into a uniform haze, strongest at the top (white noise is flat in Hz, so the wide upper mel bands collect more of it); the bright speech blobs are unchanged. The two panels are not identical pixel by pixel (different random draws) but have the same texture and the same floor: that is the equivalence we rely on to augment 100 × faster.
- **Time shift:** the same sentence, but the window starts somewhere else, so the blobs slide left or right and a different part of the sentence is cut off at the edges.
- **SpecAugment:** one flat horizontal bar (frequency mask) and one vertical bar (time mask) of exactly zero. The model has to classify from what is left, so it cannot rely on a single formant band or a single syllable.
- **Everything:** what a training example looks like on an epoch where all three fire at once.

4. The model: three conv blocks, a global average pool, and a linear head

```
input (1, 80, 75)
  ├─ block 1: Conv2d(1→16, 3×3, pad 1) → BatchNorm → ReLU → MaxPool(2)   → (16, 40, 37)
  ├─ block 2: Conv2d(16→32, 3×3)       → BatchNorm → ReLU → MaxPool(2)   → (32, 20, 18)
  ├─ block 3: Conv2d(32→64, 3×3)       → BatchNorm → ReLU → MaxPool(2)   → (64, 10, 9)
```

```
├─ global average pool over (mel, time)          → (64,)
└─ Linear(64 → n_classes)                        → logits
```

One line per layer:

- **Conv2d 3×3** slides 3-band × 3-frame filters over the image; 16 filters = 16 different local pattern detectors (edges, stripes, blobs). Weights are shared across positions, so a pattern is detected wherever it occurs.
- **BatchNorm** re-centers each channel to mean 0 / std 1 within the batch; makes training far less sensitive to the learning rate.
- **ReLU** keeps positive responses, zeros the rest: the nonlinearity that lets stacked convs build complex detectors.
- **MaxPool(2)** halves both axes, keeping the strongest response in each 2×2 cell: small position changes stop mattering.
- **Global average pool** averages each channel over the whole image, giving one 64-d vector regardless of clip length. It also means the model can only vote "this pattern is present somewhere", not "at frame 37", which is a good inductive bias for both tasks.
- **Linear** maps 64 numbers to one logit per class.

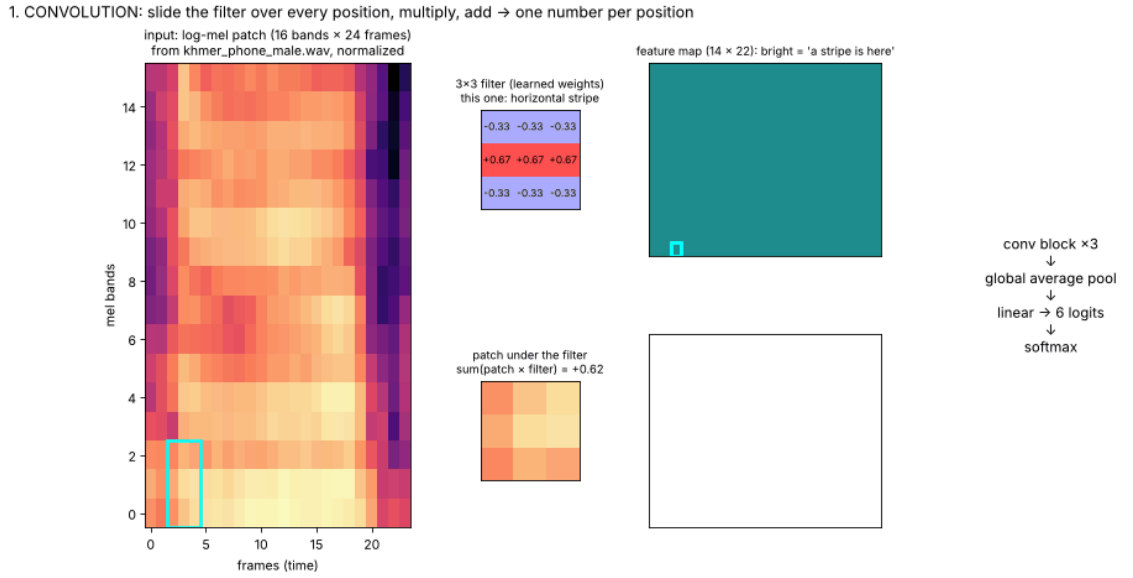
Receptive field. Each output unit of block 3 "sees" a patch of the input. 3×3 conv (RF 3) → pool (RF 4, stride 2) → 3×3 conv (RF 8) → pool (RF 10, stride 4) → 3×3 conv (RF 18) → pool (RF 22, stride 8). So one unit covers about **22 mel bands × 22 frames = 880 ms** at our 40 ms frame rate: a word or two in time and a quarter of the frequency range. Enough for a formant transition or a few pitch harmonics, not for a whole sentence; the global average pool then sums evidence across the clip.

Worked example of the parameter count: block 1 has $1 \cdot 16 \cdot 9 + 16 = 160$ conv weights, block 2 has $16 \cdot 32 \cdot 9 + 32 = 4{,}640$, block 3 has $32 \cdot 64 \cdot 9 + 64 = 18{,}496$, the head $64 \cdot 6 + 6 = 390$, plus $2 \cdot (16 + 32 + 64) = 224$ BatchNorm parameters: **23,910** for 6 classes. Compare: 36 training clips × 6,000 pixels each. The model is small, but it is still large relative to the data.

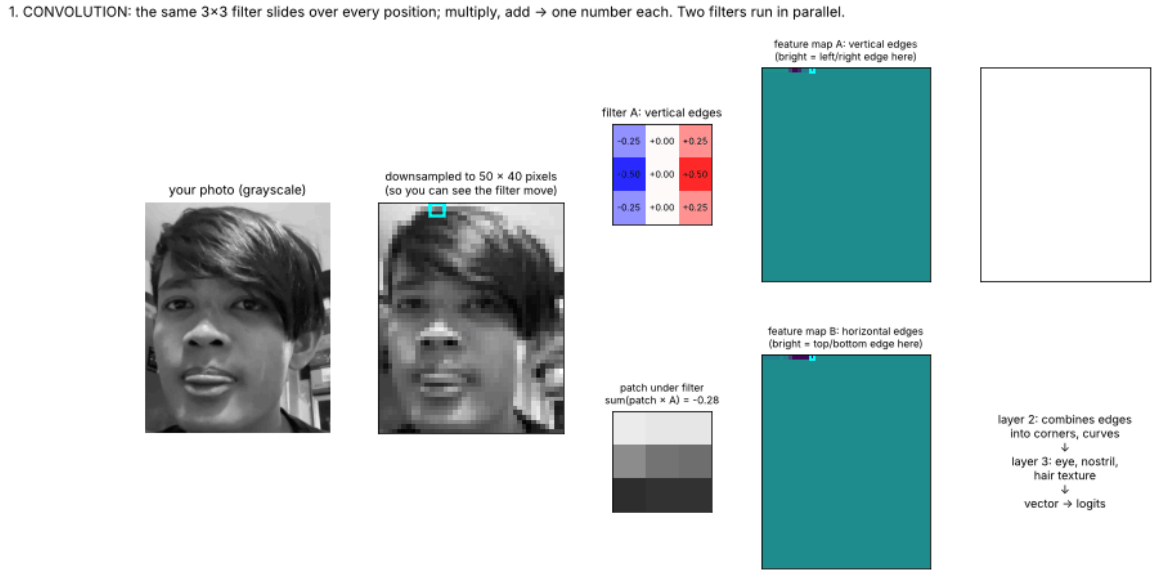
Visualization Version

Animations first, then the parameter table and the classic-vs-ours comparison.

Animated: how one conv filter slides over a log-mel patch



Animated: the same idea on a photo (vertical and horizontal edges)



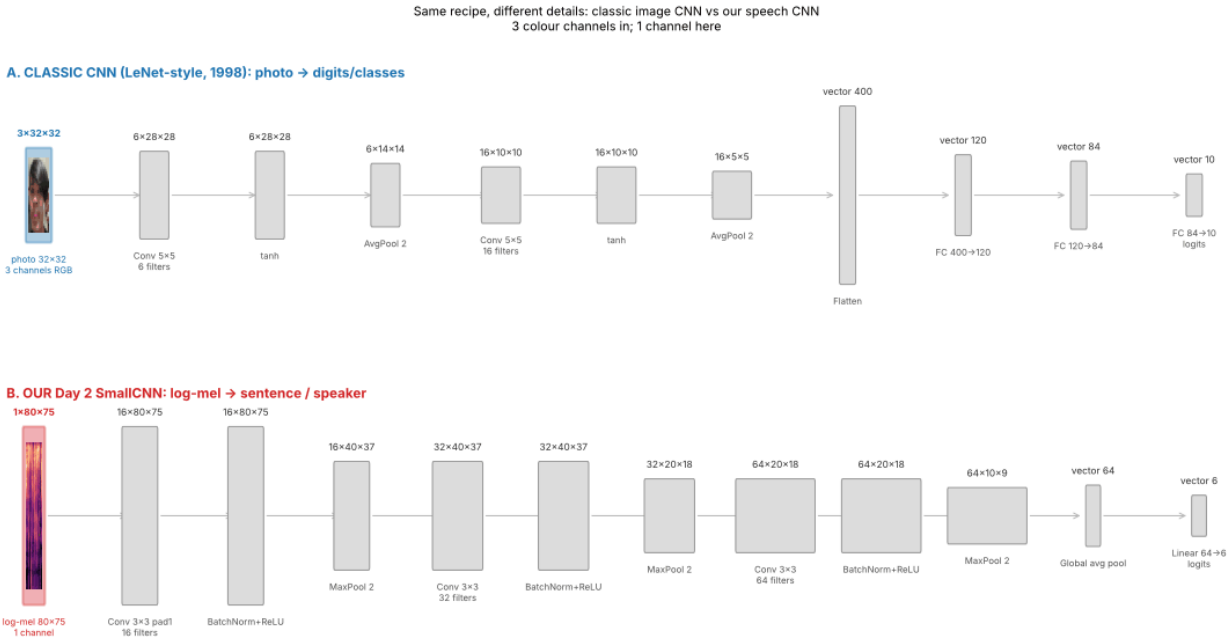
Where the 24k parameters come from

Day 2 SmallCNN: where the 24k parameters come from

Layer	Shape	How it's counted	Parameters
Conv 1	16 filters, 3×3, on 1 channel	16×1×3×3 = 144 + 16 biases	160
BatchNorm 1	scale + shift per channel	16 + 16	32
Conv 2	32 filters, 3×3, on 16 channels	32×16×3×3 = 4,608 + 32	4,640
BatchNorm 2	scale + shift per channel	32 + 32	64
Conv 3	64 filters, 3×3, on 32 channels	64×32×3×3 = 18,432 + 64	18,496
BatchNorm 3	scale + shift per channel	64 + 64	128
Linear head (8 speakers)	64 features → 8 logits	64×8 = 512 + 8	520
TOTAL			**24,040**

Conv rule: out channels × in channels × 3 × 3, + one bias per out channel. 6-sentence model: head = 64×6 + 6 = 390 → total 23,910.
ReLU, max-pool and global average pool have 0 parameters. Conv 3 alone = 77 % of the model. The count does not depend on image size.

Animated: classic image CNN vs our speech CNN



Classic image CNN vs our speech CNN: same recipe, different details

	Classic CNN (LeNet, 1998)	Ours (Day 2 SmallCNN)
Input	3×32×32: a colour photo, 3 channels	1×80×75: one channel, time × frequency
Filters	5×5, no padding → each conv shrinks the map	3×3 with padding → only pooling shrinks
After conv	tanh	BatchNorm + ReLU
Pooling	average	max
End of convs	Flatten 16×5×5 into a 400-vector (only works for one fixed input size)	Global average pool 64×10×9 into 64 numbers (works for any clip length)
Head	three big FC layers: 400 → 120 → 84 → 10	one linear layer: 64 → 6
Size	≈ 62k parameters	≈ 24k parameters

Shared recipe: conv → nonlinearity → pool, repeated, then a classifier. The flatten-vs-global-pool difference is the one to remember: it is what lets our model take clips of any length.

Additional reading: you can read all of it or just read the "In short"

In short each block finds more complex patterns and halves the image; the average pool turns 64 maps into 64 numbers; the linear head turns them into class scores.

Our CNN, block by block

- Start: one image, 1 channel × 80 mel bands × 75 frames.
- Block 1: (1, 80, 75) → (16, 40, 37)
  - Conv: 16 different 3 × 3 filters each slide over the image, so you get 16 “pattern maps”, each still 80 × 75. One map might light up on harmonic stripes, another on syllable edges.

    - BatchNorm: re-centres each map so the numbers stay well behaved, which makes training more stable.
    - ReLU: keeps the positive responses and sets the negatives to zero. In other words, only “pattern found” survives.
    - MaxPool: looks at every 2 × 2 patch and keeps the strongest value, so 80 × 75 becomes 40 × 37 (it rounds down). Small shifts stop mattering.
- Block 2: (16, 40, 37) → (32, 20, 18)
    - Same four steps, but now there are 32 filters, and each one looks at all 16 maps from block 1 at once. That lets it combine simple patterns into richer ones, like a stripe sitting next to a blob. Pooling halves the size again.
- Block 3: (32, 20, 18) → (64, 10, 9)
    - Same again with 64 filters, so the patterns get more complex. By now each value covers roughly 22 bands by 22 frames of the original, about 880 milliseconds.
- Global average pool: (64, 10, 9) → 64 numbers
    - Each of the 64 maps is averaged into a single number, meaning “how much of this pattern is in the clip overall”.
- Linear head: 64 → 6 or 8 scores It combines those 64 numbers into one score per class, and the highest score is the guess.

Additional reading: you can read all of it or just read the "In short"

In short same recipe as an image CNN, but one channel, time ≠ frequency, and a global average pool so any clip length works.

Classic CNN vs our version for speech

It’s the standard image-style CNN, with the same conv, ReLU, pooling and linear layers, just fed a spectrogram instead of a photo. A few things are different, though.

- One channel instead of three. A photo has red, green and blue. Our image only has one value per pixel, how strong that frequency is at that moment.
- The two directions mean different things. In a photo, up-down and left-right are basically the same kind of thing. Here, left-right is time and up-down is frequency. Shifting in time doesn’t change what was said, but shifting up in frequency changes the sound itself, for example a higher pitch. The plain CNN treats both directions the same anyway, which is one reason speech models often use special designs, like ECAPA, which slides along time only.
- It’s a modern version, not the very first CNN. The original, LeNet from the 1990s, didn’t have BatchNorm and used flattening plus big fully connected layers at the end. This one uses BatchNorm and ReLU, and global average pooling, which is what lets it handle any clip length. So the ideas are the same, and the main change is what the image actually means.

Additional reading: you can read all of it or just

read the "In short"

What this cell does

In short define `SmallCNN`, count its parameters, trace the tensor shape through each block, and time one forward + backward pass.

- 3 × [Conv 3×3 → BatchNorm → ReLU → MaxPool 2] → global average pool → Linear(64 → n_classes).

What the output means

In short **23,910** parameters for 6 classes; the shape goes **(8, 1, 80, 75) → (8, 16, 40, 37) → (8, 32, 20, 18) → (8, 64, 10, 9) → logits (8, 6)**.

- Each block halves both axes; the average pool then removes them entirely, which is why the model accepts any clip length.
- One forward + backward pass on a batch of 8 takes a fraction of a second on CPU.

```
class SmallCNN(nn.Module):
    def __init__(self, n_classes, channels=(16, 32, 64), dropout=0.0):
        super().__init__()
        layers, cin = [], 1
        for c in channels:
            layers += [nn.Conv2d(cin, c, 3, padding=1), nn.BatchNorm2d(c), nn.ReLU(inplace=True), nn.MaxPool2d(2)]
            cin = c
        self.features = nn.Sequential(*layers)
        self.dropout = nn.Dropout(dropout)
        self.head = nn.Linear(cin, n_classes)
    def forward(self, x):                 # (B, 1, 80, 75)
        h = self.features(x)              # (B, 64, 10, 9)
        h = h.mean(dim=(2, 3))            # global average pool -> (B, 64)
        return self.head(self.dropout(h)) # logits

def n_params(m):
    return sum(p.numel() for p in m.parameters() if p.requires_grad)

model = SmallCNN(n_classes=6)
print(model)
print("parameters:", n_params(model))
# trace the shapes: notice each block halves both axes and the pool removes them entirely
h = xb
for i, layer in enumerate(model.features):
    h = layer(h)
    if isinstance(layer, nn.MaxPool2d):
        print(f"after block {i // 4 + 1}: {tuple(h.shape)}")
print("logits:", tuple(model(xb).shape))
t0 = time.time(); model(xb).sum().backward(); print(f"one forward+backward on a batch of 8: {1000*(time.time()-t0):.0f}
ms")
```

```
SmallCNN(
  (features): Sequential(
    (0): Conv2d(1, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, bias=True, track_running_stats=True)
    (2): ReLU(inplace=True)
    (3): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (4): Conv2d(16, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (5): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, bias=True, track_running_stats=True)
    (6): ReLU(inplace=True)
    (7): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
    (8): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (9): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, bias=True, track_running_stats=True)
    (10): ReLU(inplace=True)
    (11): MaxPool2d(kernel_size=2, stride=2, padding=0, dilation=1, ceil_mode=False)
  )
  (dropout): Dropout(p=0.0, inplace=False)
  (head): Linear(in_features=64, out_features=6, bias=True)
)
parameters: 23910
after block 1: (8, 16, 40, 37)
after block 2: (8, 32, 20, 18)
after block 3: (8, 64, 10, 9)
```

```
logits: (8, 6)
one forward+backward on a batch of 8: 34 ms
```

5. The training loop, from scratch

One epoch = one pass over the training set. For each batch: forward → loss → `backward()` (gradients) → `optimizer.step()` (update) → `zero_grad()`. After the epoch, measure loss and accuracy on the validation set *in eval mode and without augmentation*. The two numbers that matter are in the curves: **training** loss/accuracy (how well it fits what it saw) and **validation** loss/accuracy (how well it does on what it did not see).

Adam with learning rate $5 \cdot 10^{-3}$ and a little weight decay (10^{-4}). Batch size 8: with 36 training clips that is 5 steps per epoch, 40 epochs = 200 steps. That is a *tiny* number of updates by deep-learning standards (ImageNet models take hundreds of thousands), which is why the learning rate is on the high side and the augmentation dose is moderate.

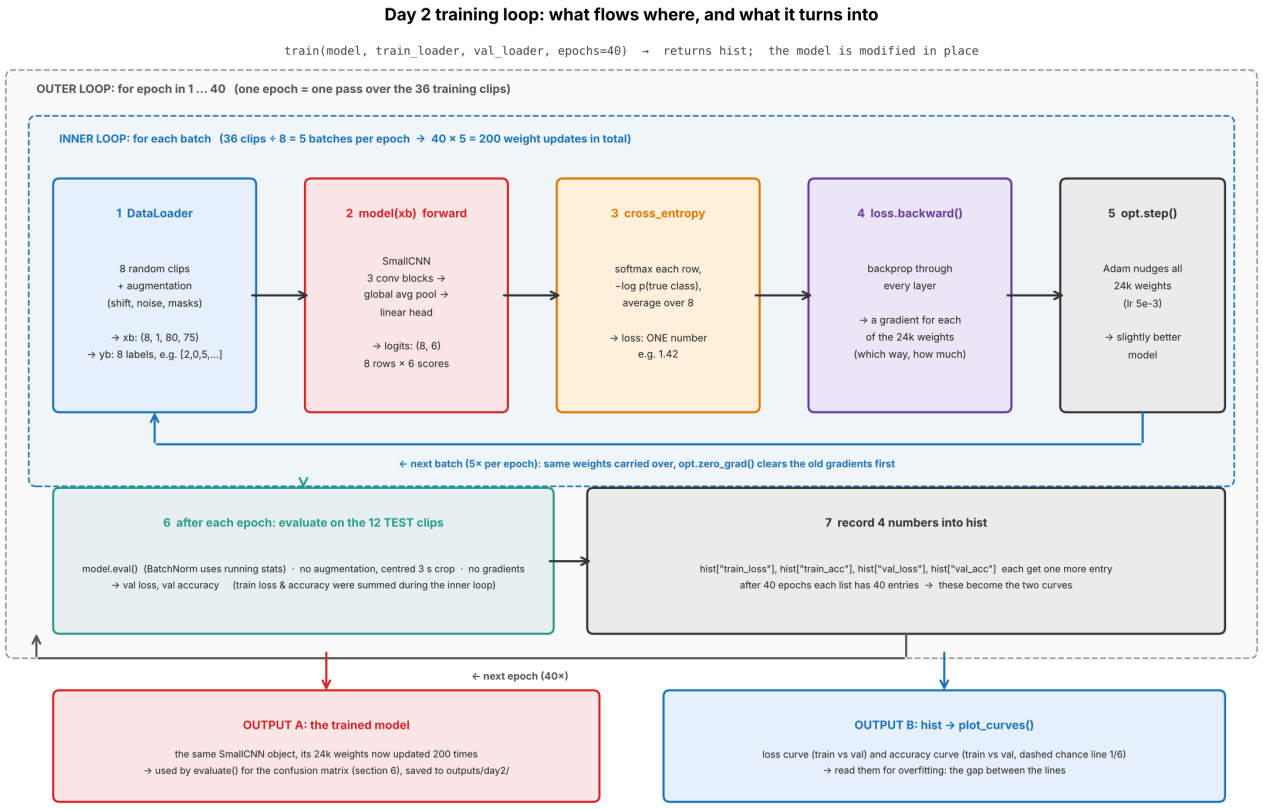
Seed everything. `random`, `numpy`, `torch` and the DataLoader's shuffle generator all get the same seed, and the dataset keeps its own RNG. Two runs with the same seed give the same curves; two seeds give different curves, and the spread between seeds is the noise floor for any comparison on data this small.

A note on honesty. With 48 clips we do not have a separate validation set, so the curves below are measured on the test clips. That is fine for *watching* training, but we must not use them to pick the best epoch or tune hyper-parameters, because then the test score is no longer a test score. We always report the *last* epoch.

Why it matters: every result in the rest of the course comes out of this loop. If you can read these curves, you can debug any model.

Visualization Version

The training loop: what flows where.



Additional reading: you can read all of it or just read the "In short"

The purpose of this training

In short teach the CNN to recognize the 6 sentences from 6 speakers, and test it on 2 voices it has never heard.

The training teaches a small CNN to listen to a spectrogram and predict which of the 6 Khmer sentences was spoken, using recordings from 6 speakers and testing on 2 speakers it has never heard. If it still scores well on those new voices, it has learned the content of the speech rather than the identity of the speaker, which is exactly the “utility judge” the anonymization project needs later.

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short define the training tools (`seed_everything` , `evaluate` , `train` , `plot_curves` , `make_loaders`), then train **task A** (sentence, 2 speakers held out) for 40 epochs and plot the curves.

- `train` : per batch forward → cross-entropy → backward → Adam step; after each epoch, measure loss/accuracy on the test clips (eval mode, no augmentation).

- `make_loaders`: training loader with augmentation and seeded shuffle; test loader with the centred crop.
- Held-out speakers: one female and one male (12 test clips).

What the output means

In short train loss falls from **1.88 to ≈ 1.1**; validation accuracy wanders between 0.17 and 0.5 and ends at **0.25**; validation loss turns **upward and spiky** after ~epoch 12: overfitting.

- `epoch 1 ... val acc 0.17` = chance; `epoch 40 ... train acc 0.61 | val acc 0.25`.
- Each validation step is 1/12 = 0.083 = **one clip**: don't read meaning into single steps.
- Full reading in the notes below.

```
def seed_everything(seed=0):
    random.seed(seed); np.random.seed(seed); torch.manual_seed(seed)

@torch.no_grad()
def evaluate(model, loader):
    # accuracy + labels + predictions + softmax probabilities; eval mode = BatchNorm uses running stats
    model.eval(); ys, ps, probs = [], [], []
    for xb, yb in loader:
        logits = model(xb)
        probs.append(F.softmax(logits, 1).numpy()); ps.append(logits.argmax(1).numpy()); ys.append(yb.numpy())
    y_true, y_pred, probs = map(np.concatenate, (ys, ps, probs))
    return float((y_true == y_pred).mean()), y_true, y_pred, probs

def train(model, train_loader, val_loader, epochs=40, lr=5e-3, weight_decay=1e-4, verbose=True, log_every=5):
    opt = torch.optim.Adam(model.parameters(), lr=lr, weight_decay=weight_decay)
    hist = {"train_loss": [], "train_acc": [], "val_loss": [], "val_acc": []}
    t0 = time.time()
    for ep in range(1, epochs + 1):
        model.train(); tot_loss, tot_correct, n = 0.0, 0, 0
        for xb, yb in train_loader:
            logits = model(xb)
            loss = F.cross_entropy(logits, yb)            # softmax + negative log-likelihood in one call
            opt.zero_grad(); loss.backward(); opt.step()
            tot_loss += loss.item() * len(yb); tot_correct += (logits.argmax(1) == yb).sum().item(); n += len(yb)
        model.eval(); v_loss, v_correct, m = 0.0, 0, 0
        with torch.no_grad():
            for xb, yb in val_loader:
                logits = model(xb)
                v_loss += F.cross_entropy(logits, yb).item() * len(yb); v_correct += (logits.argmax(1) ==
yb).sum().item(); m += len(yb)
        for k, v in zip(hist, (tot_loss / n, tot_correct / n, v_loss / m, v_correct / m)):
            hist[k].append(v)
        if verbose and (ep % log_every == 0 or ep in (1, epochs)):
            print(f"epoch {ep:3d}  train loss {tot_loss/n:.3f} acc {tot_correct/n:.2f} | val loss {v_loss/m:.3f} acc
{v_correct/m:.2f}   ({time.time()-t0:.0f} s)")
    hist["seconds"] = time.time() - t0
    return hist

def plot_curves(hist, title="", chance=None):
    _, ax = plt.subplots(1, 2, figsize=(12, 3))
    ep = np.arange(1, len(hist["train_loss"]) + 1)
    ax[0].plot(ep, hist["train_loss"], label="train"); ax[0].plot(ep, hist["val_loss"], label="val")
    ax[0].set(title=f"loss {title}", xlabel="epoch", ylabel="cross-entropy"); ax[0].legend()
    ax[1].plot(ep, hist["train_acc"], label="train"); ax[1].plot(ep, hist["val_acc"], label="val")
    if chance is not None:
        ax[1].axhline(chance, color="gray", ls="--", label=f"chance {chance:.2f}")
    ax[1].set(title=f"accuracy {title}", xlabel="epoch", ylabel="accuracy", ylim=(0, 1.02)); ax[1].legend()
    plt.tight_layout(); plt.show()

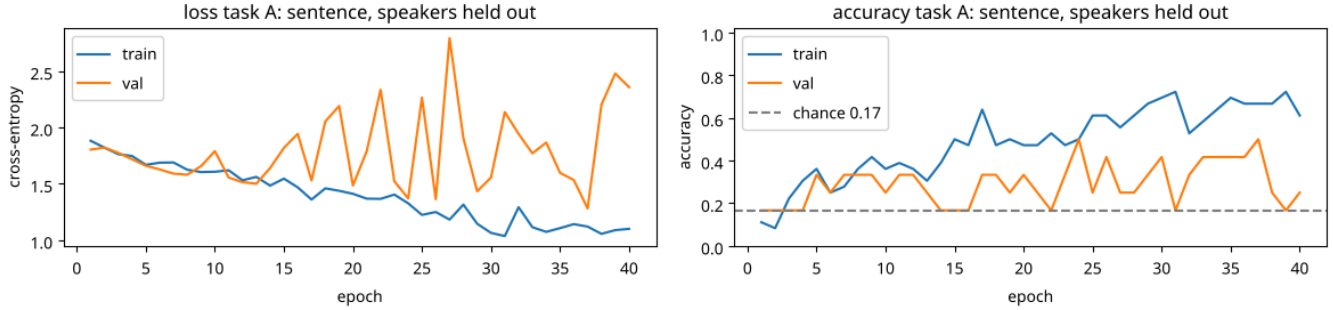
def make_loaders(train_idx, test_idx, labels, seed=0, batch_size=8, augment=True, train_kw=None, test_kw=None):
    # train set: augmentation on, shuffled with a seeded generator; test set: centered crop, no augmentation
    tr = LogMelDataset([MELS[i] for i in train_idx], PSIG[train_idx], labels[train_idx], train=True, seed=seed,
augment=augment, **(train_kw or {}))
    te = LogMelDataset([MELS[i] for i in test_idx], PSIG[test_idx], labels[test_idx], train=False, **(test_kw or {}))
    g = torch.Generator().manual_seed(seed)
    return DataLoader(tr, batch_size=batch_size, shuffle=True, generator=g), DataLoader(te, batch_size=16)

# ---- Task A, leave-speakers-out: hold out one female and one male speaker
HELD_OUT_SPK = [SPEAKERS[3], SPEAKERS[7]]
test_A = np.where(meta.speaker.isin(HELD_OUT_SPK))[0]
train_A = np.where(~meta.speaker.isin(HELD_OUT_SPK))[0]
print("task A (sentence): train on", len(train_A), "clips from", meta.speaker[train_A].nunique(), "speakers; test on",
```

```
      len(test_A), "clips from held-out", [s[-8:] for s in HELD_OUT_SPK], list(meta.gender[test_A].unique()))

seed_everything(0)
tr_loader, te_loader = make_loaders(train_A, test_A, meta.y_sent.values, seed=0)
model_A = SmallCNN(n_classes=6)
hist_A = train(model_A, tr_loader, te_loader, epochs=40)
plot_curves(hist_A, "task A: sentence, speakers held out", chance=1 / 6)
```

```
task A (sentence): train on 36 clips from 6 speakers; test on 12 clips from held-out ['1f8f808c', '707a4c7a'] ['female', 'ma
le']
epoch   1  train loss 1.882 acc 0.11 | val loss 1.803 acc 0.17   (0 s)
epoch   5  train loss 1.669 acc 0.36 | val loss 1.660 acc 0.33   (1 s)
epoch  10  train loss 1.607 acc 0.36 | val loss 1.790 acc 0.25   (2 s)
epoch  15  train loss 1.546 acc 0.50 | val loss 1.819 acc 0.17   (3 s)
epoch  20  train loss 1.411 acc 0.47 | val loss 1.485 acc 0.33   (4 s)
epoch  25  train loss 1.224 acc 0.61 | val loss 2.265 acc 0.25   (5 s)
epoch  30  train loss 1.065 acc 0.69 | val loss 1.556 acc 0.42   (6 s)
epoch  35  train loss 1.107 acc 0.69 | val loss 1.598 acc 0.42   (7 s)
epoch  40  train loss 1.101 acc 0.61 | val loss 2.358 acc 0.25   (8 s)
```



Additional Reading

In short the gap between the train and validation lines is overfitting; with 12 test clips every step on the curve is one clip.

How to read the curves.

- **Loss, left:** training loss starts at ≈ 1.8 ($= \log 6$, random guessing) and falls to ≈ 1.1 as the model fits the 36 training clips. Validation loss follows it down for the first ~12 epochs, then turns upward and gets *spiky* (1.3 one epoch, 2.8 the next). The vertical **gap between the two lines is overfitting**: capacity spent on memorizing training clips (including their speakers) that does not transfer to the two unseen voices. The spikes come from confident wrong answers on a 12-clip set (one clip at probability 0.01 adds $-\log 0.01/12 = 0.38$ to the mean) and from BatchNorm statistics that jump around with batches of 8. A validation loss that turns upward while training loss keeps falling means the model is drifting away from the general solution; the place where it turns (≈ epoch 12 here) is where early stopping would stop, if we had a separate validation set to stop on.
- **Accuracy, right:** the dashed line is chance (1/6 = 0.17). Training accuracy climbs to ≈ 0.6–0.7 but does not reach 1.0 in 40 epochs: augmentation keeps changing the images, so the model cannot simply memorize them. Validation accuracy wanders between 0.17 and 0.5 and ends at 0.25. Because the test set has only 12 clips, each step on the validation curve is 1/12 = 0.083: a jump from 0.25 to 0.33 is *one clip*. Do not read meaning into single steps.
- A healthy run shows the two lines close together for a while and then separating; a run where validation accuracy never leaves chance means the task is impossible with this split, this model, or this amount of data.

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short look inside the data: print the 48-clip table, check every speaker read every sentence once, and look at one clip as numbers and as a picture.

- *Moved here from section 4 so the notebook runs top to bottom* (the next cell needs the training split from section 5).

What the output means

In short the speaker × sentence table is **all 1s** (a perfectly balanced design), and one log-mel is **(80, 317)** dB values.

- Note: this cell uses `n_fft=400` for its demo log-mel, so its frame count differs by one from the 512-point version used in training.

```
import pandas as pd, numpy as np, librosa
from pathlib import Path

DATA = Path("../data")
meta = pd.read_csv(DATA / "meta/probe_clips.csv")
meta["y_sent"] = meta.sentence_id.map({s: i for i, s in enumerate(SENTENCES)})
meta["y_spk"]  = meta.speaker.map({s: i for i, s in enumerate(SPEAKERS)})

# 1. the full table of 48 clips
print(meta.to_string())

# 2. who said which sentence: should be a grid of all 1s
print(pd.crosstab(meta.speaker, meta.sentence_id))
SENTENCES = sorted(meta.sentence_id.unique()); SPEAKERS = sorted(meta.speaker.unique())
# 3. what ONE clip looks like as numbers
u = meta.utt_id[0]
x, sr = librosa.load(DATA / "audio/probe" / f"{u}.wav", sr=16000)
print("samples:", x.shape, "first 10:", np.round(x[:10], 4))

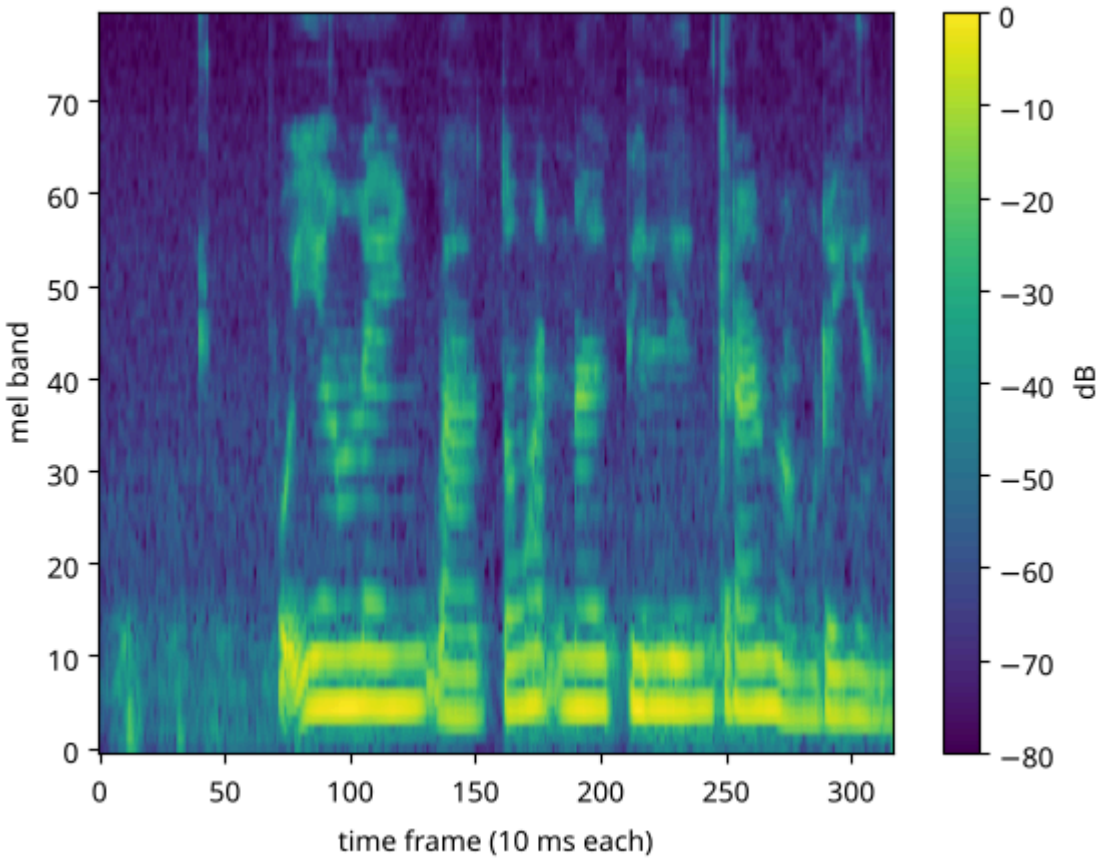
# 4. the spectrogram the model actually sees
P = librosa.feature.melspectrogram(y=x, sr=sr, n_fft=400, hop_length=160,
                                   win_length=400, n_mels=80, power=2.0)
L = librosa.power_to_db(P, ref=np.max)
print("log-mel shape:", L.shape)
print(np.round(L[:5, :8], 1))

# 5. look at it as a picture
import matplotlib.pyplot as plt
plt.imshow(L, origin="lower", aspect="auto"); plt.colorbar(label="dB")
plt.xlabel("time frame (10 ms each)"); plt.ylabel("mel band"); plt.show()
```

```
                  utt_id          speaker  gender sentence_id                                                text  duratio
n  y_sent  y_spk
0   iany_046e9e11_cc152fc8  iany-s-046e9e11  female    daily-04                ពួកយើងលេងបាល់ទាត់នៅទីលានក្រោយម៉ោងរៀន      3.165
0       0
1   iany_046e9e11_d659c7f0  iany-s-046e9e11  female    daily-10                      យើងរស់ចិត្តដើមឈើនៅសួនច្បារ      2.2
10      1      0
2   iany_046e9e11_0f95e453  iany-s-046e9e11  female   health-04             ផឹកទឹកឲ្យបានច្រើនរៀងរាល់ការខ្សោតទឹក      2.87
7       2      0
3   iany_046e9e11_37064b29  iany-s-046e9e11  female     news-03  អ្នកជំនាញព្យាករណ៍ថានឹងមានភ្លៀងធ្លាក់នៅសប្ដាហ៍ក្រោយ      3.20
8       3      0
4   iany_046e9e11_23873e18  iany-s-046e9e11  female     news-05          សិស្សាន,សិស្សបានចូលរួមក្នុង,ពិធីប្រកួតប្រជែង      3.15
4       4      0
5   iany_046e9e11_40ca030e  iany-s-046e9e11  female  numbers-01                  ខ្ញុំមានអាយុដប់បួនឆ្នាំកន្លងឆ្នាំនេះ     2.330
5       0
6   iany_18151a80_25793264  iany-s-18151a80  female    daily-04                ពួកយើងលេងបាល់ទាត់នៅទីលានក្រោយម៉ោងរៀន      5.522
0       1
7   iany_18151a80_df0044b0  iany-s-18151a80  female    daily-10                      យើងរស់ចិត្តដើមឈើនៅសួនច្បារ      4.0
71      1      1
8   iany_18151a80_3309d18a  iany-s-18151a80  female   health-04             ផឹកទឹកឲ្យបានច្រើនរៀងរាល់ការខ្សោតទឹក      4.66
```

```
8       2      1
9   iany_18151a80_841aa700  iany-s-18151a80  female     news-03  អ្នកជំនាញពាក្យការណ៍ថានឹងមានភ្លៀងធ្លាក់នៅសប្តាហ៍ក្រោយ      4.32
7       3      1
10  iany_18151a80_1917b67b  iany-s-18151a80  female     news-05          សិស្សានុសិស្សបានចូលរួមកន្លងពិធីប្រកួតប្រជែង      4.49
8       4      1
11  iany_18151a80_8607ce2c  iany-s-18151a80  female  numbers-01                ខ្ញុំមានអាយុដប់បួនឆ្នាំកន្លងឆ្នាំនេះ      4.497
5       1
12  iany_19ed77bb_e0542ca2  iany-s-19ed77bb  female    daily-04                ពួកយើងលងបាល់ទាត់នៅទីលានក្រោយម៉ោងរៀន      4.845
0       2
13  iany_19ed77bb_3dc45fd2  iany-s-19ed77bb  female    daily-10                      យើងរសោចទឹកដើមឈើនៅសួនច្បារ      3.2
83      1       2
14  iany_19ed77bb_698ef324  iany-s-19ed77bb  female   health-04             ធ្វើកិច្ចយោគយលនូវសៀវភៅការខ្សោតិទឹក      4.58
5       2       2
15  iany_19ed77bb_1c6743dc  iany-s-19ed77bb  female     news-03  អ្នកជំនាញពាក្យការណ៍ថានឹងមានភ្លៀងធ្លាក់នៅសប្តាហ៍ក្រោយ      5.64
5       3       2
16  iany_19ed77bb_79429e0d  iany-s-19ed77bb  female     news-05          សិស្សានុសិស្សបានចូលរួមកន្លងពិធីប្រកួតប្រជែង      5.26
5       4       2
17  iany_19ed77bb_20e4dee9  iany-s-19ed77bb  female  numbers-01                ខ្ញុំមានអាយុដប់បួនឆ្នាំកន្លងឆ្នាំនេះ      5.915
5       2
18  iany_1f8f808c_44cd6bf2  iany-s-1f8f808c  female    daily-04                ពួកយើងលងបាល់ទាត់នៅទីលានក្រោយម៉ោងរៀន      3.110
0       3
19  iany_1f8f808c_00c61046  iany-s-1f8f808c  female    daily-10                      យើងរសោចទឹកដើមឈើនៅសួនច្បារ      2.3
07      1       3
20  iany_1f8f808c_c61545eb  iany-s-1f8f808c  female   health-04             ធ្វើកិច្ចយោគយលនូវសៀវភៅការខ្សោតិទឹក      2.51
2       2       3
21  iany_1f8f808c_aa21c76c  iany-s-1f8f808c  female     news-03  អ្នកជំនាញពាក្យការណ៍ថានឹងមានភ្លៀងធ្លាក់នៅសប្តាហ៍ក្រោយ      2.99
8       3       3
22  iany_1f8f808c_746dddcc  iany-s-1f8f808c  female     news-05          សិស្សានុសិស្សបានចូលរួមកន្លងពិធីប្រកួតប្រជែង      3.12
3       4       3
23  iany_1f8f808c_abfd4e51  iany-s-1f8f808c  female  numbers-01                ខ្ញុំមានអាយុដប់បួនឆ្នាំកន្លងឆ្នាំនេះ      2.265
5       3
24  iany_21c36386_1931f569  iany-s-21c36386    male    daily-04                ពួកយើងលងបាល់ទាត់នៅទីលានក្រោយម៉ោងរៀន      5.131
0       4
25  iany_21c36386_e8d1aa6a  iany-s-21c36386    male    daily-10                      យើងរសោចទឹកដើមឈើនៅសួនច្បារ      3.1
07      1       4
26  iany_21c36386_5b342eab  iany-s-21c36386    male   health-04             ធ្វើកិច្ចយោគយលនូវសៀវភៅការខ្សោតិទឹក      3.55
0       2       4
27  iany_21c36386_7b8fd78a  iany-s-21c36386    male     news-03  អ្នកជំនាញពាក្យការណ៍ថានឹងមានភ្លៀងធ្លាក់នៅសប្តាហ៍ក្រោយ      5.02
9       3       4
28  iany_21c36386_1b2b8696  iany-s-21c36386    male     news-05          សិស្សានុសិស្សបានចូលរួមកន្លងពិធីប្រកួតប្រជែង      4.47
5       4       4
29  iany_21c36386_f6794943  iany-s-21c36386    male  numbers-01                ខ្ញុំមានអាយុដប់បួនឆ្នាំកន្លងឆ្នាំនេះ      3.439
5       4
30  iany_49b4ab7e_eb940730  iany-s-49b4ab7e    male    daily-04                ពួកយើងលងបាល់ទាត់នៅទីលានក្រោយម៉ោងរៀន      4.667
0       5
31  iany_49b4ab7e_2f5f47ae  iany-s-49b4ab7e    male    daily-10                      យើងរសោចទឹកដើមឈើនៅសួនច្បារ      4.4
94      1       5
32  iany_49b4ab7e_8f88cd1f  iany-s-49b4ab7e    male   health-04             ធ្វើកិច្ចយោគយលនូវសៀវភៅការខ្សោតិទឹក      4.96
5       2       5
33  iany_49b4ab7e_a2c7227b  iany-s-49b4ab7e    male     news-03  អ្នកជំនាញពាក្យការណ៍ថានឹងមានភ្លៀងធ្លាក់នៅសប្តាហ៍ក្រោយ      5.74
5       3       5
34  iany_49b4ab7e_f6f6ffcf  iany-s-49b4ab7e    male     news-05          សិស្សានុសិស្សបានចូលរួមកន្លងពិធីប្រកួតប្រជែង      4.89
5       4       5
35  iany_49b4ab7e_08b22d9c  iany-s-49b4ab7e    male  numbers-01                ខ្ញុំមានអាយុដប់បួនឆ្នាំកន្លងឆ្នាំនេះ      3.889
5       5
36  iany_600bbcad_10f7caf3  iany-s-600bbcad    male    daily-04                ពួកយើងលងបាល់ទាត់នៅទីលានក្រោយម៉ោងរៀន      3.805
0       6
37  iany_600bbcad_4aa14a48  iany-s-600bbcad    male    daily-10                      យើងរសោចទឹកដើមឈើនៅសួនច្បារ      4.8
84      1       6
38  iany_600bbcad_7b05a074  iany-s-600bbcad    male   health-04             ធ្វើកិច្ចយោគយលនូវសៀវភៅការខ្សោតិទឹក      4.24
0       2       6
39  iany_600bbcad_9c1b9049  iany-s-600bbcad    male     news-03  អ្នកជំនាញពាក្យការណ៍ថានឹងមានភ្លៀងធ្លាក់នៅសប្តាហ៍ក្រោយ      3.88
1       3       6
40  iany_600bbcad_2f2a63e7  iany-s-600bbcad    male     news-05          សិស្សានុសិស្សបានចូលរួមកន្លងពិធីប្រកួតប្រជែង      4.62
1       4       6
41  iany_600bbcad_325d924e  iany-s-600bbcad    male  numbers-01                ខ្ញុំមានអាយុដប់បួនឆ្នាំកន្លងឆ្នាំនេះ      5.151
5       6
42  iany_707a4c7a_ec6b242a  iany-s-707a4c7a    male    daily-04                ពួកយើងលងបាល់ទាត់នៅទីលានក្រោយម៉ោងរៀន      5.168
0       7
43  iany_707a4c7a_d81b4554  iany-s-707a4c7a    male    daily-10                      យើងរសោចទឹកដើមឈើនៅសួនច្បារ      4.1
16      1       7
44  iany_707a4c7a_a39c9c3d  iany-s-707a4c7a    male   health-04             ធ្វើកិច្ចយោគយលនូវសៀវភៅការខ្សោតិទឹក      5.17
9       2       7
45  iany_707a4c7a_1d8b082e  iany-s-707a4c7a    male     news-03  អ្នកជំនាញពាក្យការណ៍ថានឹងមានភ្លៀងធ្លាក់នៅសប្តាហ៍ក្រោយ      5.19
2       3       7
46  iany_707a4c7a_c2e806bf  iany-s-707a4c7a    male     news-05          សិស្សានុសិស្សបានចូលរួមកន្លងពិធីប្រកួតប្រជែង      4.63
7       4       7
47  iany_707a4c7a_f373befd  iany-s-707a4c7a    male  numbers-01                ខ្ញុំមានអាយុដប់បួនឆ្នាំកន្លងឆ្នាំនេះ      4.537
5       7
sentence_id      daily-04  daily-10  health-04  news-03  news-05  numbers-01
speaker
iany-s-046e9e11         1         1          1        1        1           1
iany-s-18151a80         1         1          1        1        1           1
iany-s-19ed77bb         1         1          1        1        1           1
iany-s-1f8f808c         1         1          1        1        1           1
iany-s-21c36386         1         1          1        1        1           1
iany-s-49b4ab7e         1         1          1        1        1           1
```

```
iany-s-600bbcad         1         1          1        1        1           1
iany-s-707a4c7a         1         1          1        1        1           1
samples: (50644,) first 10: [0.0002 0.0004 0.0003 0.0003 0.0011 0.0011 0.0011 0.0014 0.0012 0.0005]
log-mel shape: (80, 317)
[[-67.4 -68.4 -63.2 -65.2 -58.6 -52.  -57.3 -55. ]
 [-69.5 -60.2 -68.5 -60.1 -54.2 -45.3 -52.6 -48.8]
 [-62.4 -53.5 -60.4 -60.  -56.4 -45.4 -47.  -44.6]
 [-62.3 -50.5 -57.8 -57.4 -58.6 -48.8 -47.6 -45.2]
 [-65.3 -52.1 -56.7 -51.7 -57.1 -53.6 -47.2 -45.2]]
```



Additional reading: you can read all of it or just read the "In short"

What this cell does

In short show exactly what goes into the model: the label table, the 36 training / 12 test clips, one dataset item, and one batch drawn as pictures.

- One item is `x` (1, 80, 75) + `y` (a sentence number). One batch stacks 8 items: `xb` (8, 1, 80, 75), `yb` 8 labels.

What the output means

In short the **TRAIN** set is 36 clips from 6 speakers, the **TEST** set 12 clips from the 2 held-out speakers; the 8 pictures are one real training batch with their answers.

- `x values range from about −2 to +3` : the normalized input.

```python
# ---- 1. the label table: what number means what sentence
print("sentence label  ->  sentence id  ->  text")
for i, sid in enumerate(SENTENCES):
    txt = meta[meta.sentence_id == sid].text.iloc[0]
```

```
    print(f"   {i}   ->  {sid:11s}  ->  {txt}")

# ---- 2. the training table: 36 clips, each with its label
cols = ["utt_id", "speaker", "gender", "sentence_id", "y_sent", "duration"]
print("\nTRAIN set (36 clips, 6 speakers):")
print(meta.loc[train_A, cols].to_string())
print("\nTEST set (12 clips, 2 held-out speakers):")
print(meta.loc[test_A, cols].to_string())

# ---- 3. one item from the dataset: this is literally what goes into the model
ds = tr_loader.dataset
x, y = ds[0]
print("\none dataset item:")
print("   x = tensor of shape", tuple(x.shape), "  (channel, mel bands, time frames)")
print("   y =", y, "  -> sentence", SENTENCES[y])
print("   x values range from", round(x.min().item(), 2), "to", round(x.max().item(), 2))
print("   top-left corner of x:\n", x[0, :4, :6].numpy().round(2))

# ---- 4. one batch from the loader: 8 items stacked together
xb, yb = next(iter(tr_loader))
print("\none batch:")
print("   xb shape", tuple(xb.shape), "  (8 clips, 1 channel, 80 mel, 75 frames)")
print("   yb       ", yb.tolist(), "  <- the 8 answers the model must guess")

# ---- 5. look at the 8 clips in that batch as pictures, with their labels
fig, axes = plt.subplots(2, 4, figsize=(14, 5))
for ax, img, lab in zip(axes.flat, xb, yb):
    ax.imshow(img[0], origin="lower", aspect="auto", cmap="magma")
    ax.set_title(f"label {lab.item()} = {SENTENCES[lab]}", fontsize=9)
    ax.set_xticks([]); ax.set_yticks([])
plt.suptitle("one training batch: what the model sees (x) and what it must answer (y)")
plt.tight_layout(); plt.show()
```

```
sentence label  ->  sentence id  ->  text
   0   ->  daily-04     ->  ពួកយើងលងហល់ទាតនៅទីលនៃរកោយម៉ោងរៀន
   1   ->  daily-10     ->  យើងៃរសោចទឹកដើមយើងនៅស្នងឲ្យហ
   2   ->  health-04    ->  ធីកទឹកថ្យហនៃរបើនយៀសវាងការខ្សោតិទឹក
   3   ->  news-03      ->  អ្នកជំនាញពយាករណ៍ថានឹងមានភ្លៀងធ្លាកនៅសប្ដាហ៍កោយ
   4   ->  news-05      ->  សិស្សនសិស្សហានចូលរួមកន្ងពិធីរបកួតរម៉ែង
   5   ->  numbers-01   ->  ខ្ញុំមានអាយុដប់បួនឆ្នាំកន្ងឆ្នាំនេះ

TRAIN set (36 clips, 6 speakers):
                  utt_id          speaker  gender sentence_id  y_sent  duration
0   iany_046e9e11_cc152fc8  iany-s-046e9e11  female    daily-04       0     3.165
1   iany_046e9e11_d659c7f0  iany-s-046e9e11  female    daily-10       1     2.210
2   iany_046e9e11_0f95e453  iany-s-046e9e11  female   health-04       2     2.877
3   iany_046e9e11_37064b29  iany-s-046e9e11  female     news-03       3     3.208
4   iany_046e9e11_23873e18  iany-s-046e9e11  female     news-05       4     3.154
5   iany_046e9e11_40ca030e  iany-s-046e9e11  female  numbers-01       5     2.330
6   iany_18151a80_25793264  iany-s-18151a80  female    daily-04       0     5.522
7   iany_18151a80_df0044b0  iany-s-18151a80  female    daily-10       1     4.071
8   iany_18151a80_3309d18a  iany-s-18151a80  female   health-04       2     4.668
9   iany_18151a80_841aa700  iany-s-18151a80  female     news-03       3     4.327
10  iany_18151a80_1917b67b  iany-s-18151a80  female     news-05       4     4.498
11  iany_18151a80_8607ce2c  iany-s-18151a80  female  numbers-01       5     4.497
12  iany_19ed77bb_e0542ca2  iany-s-19ed77bb  female    daily-04       0     4.845
13  iany_19ed77bb_3dc45fd2  iany-s-19ed77bb  female    daily-10       1     3.283
14  iany_19ed77bb_698ef324  iany-s-19ed77bb  female   health-04       2     4.585
15  iany_19ed77bb_1c6743dc  iany-s-19ed77bb  female     news-03       3     5.645
16  iany_19ed77bb_79429e0d  iany-s-19ed77bb  female     news-05       4     5.265
17  iany_19ed77bb_20e4dee9  iany-s-19ed77bb  female  numbers-01       5     5.915
24  iany_21c36386_1931f569  iany-s-21c36386    male    daily-04       0     5.131
25  iany_21c36386_e8d1aa6a  iany-s-21c36386    male    daily-10       1     3.107
26  iany_21c36386_5b342eab  iany-s-21c36386    male   health-04       2     3.550
27  iany_21c36386_7b8fd78a  iany-s-21c36386    male     news-03       3     5.029
28  iany_21c36386_1b2b8696  iany-s-21c36386    male     news-05       4     4.475
29  iany_21c36386_f6794943  iany-s-21c36386    male  numbers-01       5     3.439
30  iany_49b4ab7e_eb940730  iany-s-49b4ab7e    male    daily-04       0     4.667
31  iany_49b4ab7e_2f5f47ae  iany-s-49b4ab7e    male    daily-10       1     4.494
32  iany_49b4ab7e_8f88cd1f  iany-s-49b4ab7e    male   health-04       2     4.965
33  iany_49b4ab7e_a2c7227b  iany-s-49b4ab7e    male     news-03       3     5.745
34  iany_49b4ab7e_f6f6ffcf  iany-s-49b4ab7e    male     news-05       4     4.895
35  iany_49b4ab7e_08b22d9c  iany-s-49b4ab7e    male  numbers-01       5     3.889
36  iany_600bbcad_10f7caf3  iany-s-600bbcad    male    daily-04       0     3.805
37  iany_600bbcad_4aa14a48  iany-s-600bbcad    male    daily-10       1     4.884
38  iany_600bbcad_7b05a074  iany-s-600bbcad    male   health-04       2     4.240
39  iany_600bbcad_9c1b9049  iany-s-600bbcad    male     news-03       3     3.881
40  iany_600bbcad_2f2a63e7  iany-s-600bbcad    male     news-05       4     4.621
41  iany_600bbcad_325d924e  iany-s-600bbcad    male  numbers-01       5     5.151

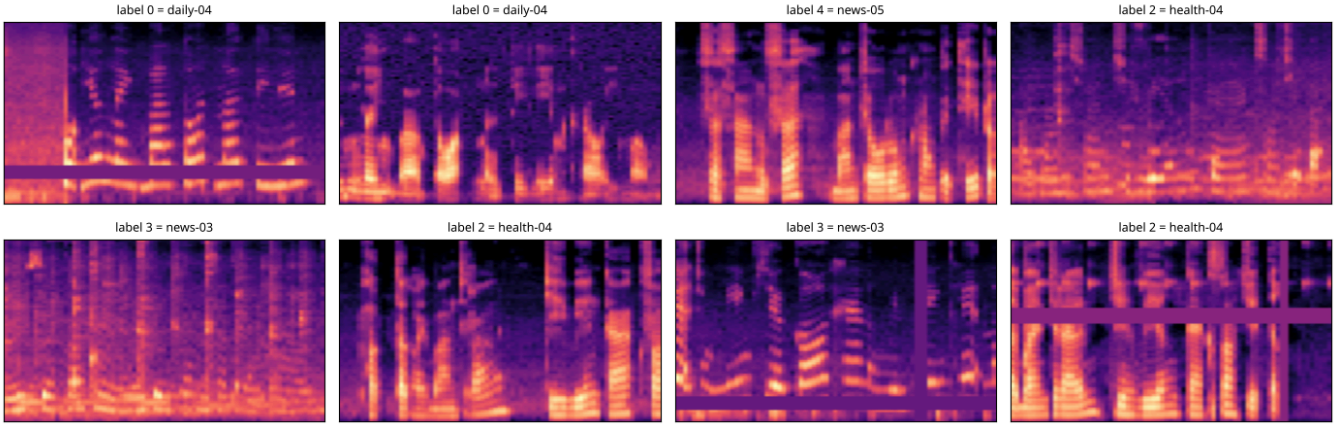
TEST set (12 clips, 2 held-out speakers):
                  utt_id          speaker  gender sentence_id  y_sent  duration
18  iany_1f8f808c_44cd6bf2  iany-s-1f8f808c  female    daily-04       0     3.110
19  iany_1f8f808c_00c61046  iany-s-1f8f808c  female    daily-10       1     2.307
20  iany_1f8f808c_c61545eb  iany-s-1f8f808c  female   health-04       2     2.512
21  iany_1f8f808c_aa21c76c  iany-s-1f8f808c  female     news-03       3     2.998
22  iany_1f8f808c_746dddcc  iany-s-1f8f808c  female     news-05       4     3.123
```

```
23  iany_1f8f808c_abfd4e51  iany-s-1f8f808c  female  numbers-01       5     2.265
42  iany_707a4c7a_ec6b242a  iany-s-707a4c7a    male    daily-04       0     5.168
43  iany_707a4c7a_d81b4554  iany-s-707a4c7a    male    daily-10       1     4.116
44  iany_707a4c7a_a39c9c3d  iany-s-707a4c7a    male   health-04       2     5.179
45  iany_707a4c7a_1d8b082e  iany-s-707a4c7a    male     news-03       3     5.192
46  iany_707a4c7a_c2e806bf  iany-s-707a4c7a    male     news-05       4     4.637
47  iany_707a4c7a_f373befd  iany-s-707a4c7a    male  numbers-01       5     4.537

one dataset item:
   x = tensor of shape (1, 80, 75)   (channel, mel bands, time frames)
   y = 0   -> sentence daily-04
   x values range from -1.54 to 3.01
   top-left corner of x:
 [[-0.56  0.13  1.25  1.21 -0.08 -0.21]
 [-0.34  0.35  1.52  1.4   0.31  0.11]
 [-0.25  0.37  1.51  1.36  0.47  0.19]
 [-0.14  0.27  1.54  1.25  0.33  0.1 ]]

one batch:
   xb shape (8, 1, 80, 75)   (8 clips, 1 channel, 80 mel, 75 frames)
   yb        [0, 0, 4, 2, 3, 2, 3, 2]   <- the 8 answers the model must guess
```

one training batch: what the model sees (x) and what it must answer (y)



6. Evaluation: accuracy is one number, the confusion matrix is the story

Accuracy = fraction of correct predictions. For a balanced test set (ours: 2 clips per sentence, 12 total) it is a fair summary. But it does not say *which* classes are confused. The **confusion matrix** does: rows are the true class, columns the predicted class, each cell counts clips. The diagonal is right; everything else is wrong, and *where* it is wrong tells you what the model is using.

From the matrix we get per-class scores. For class k:

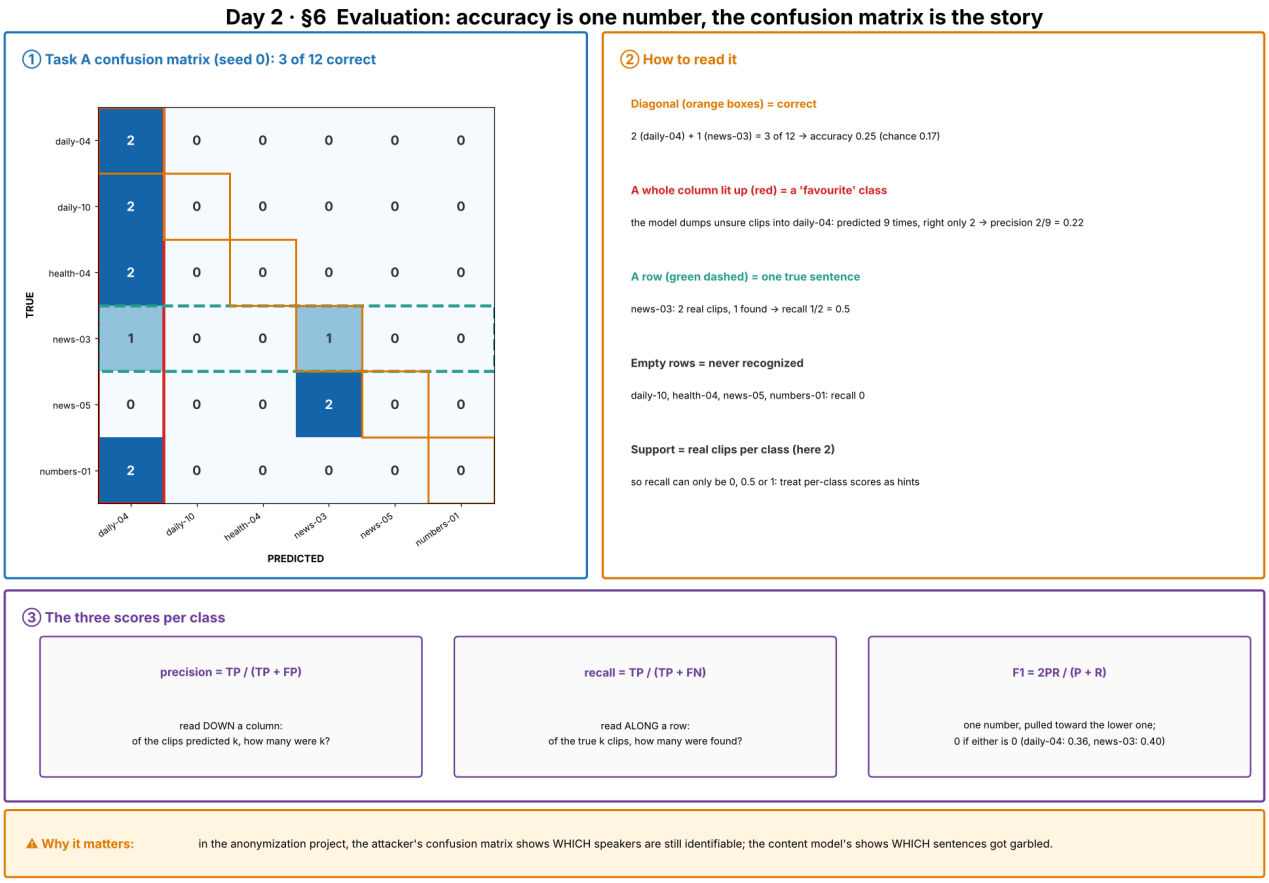
$$\text{precision}_k = \frac{\text{TP}_k}{\text{TP}_k + \text{FP}_k} \quad (\text{column } k: \text{ of the clips predicted } k, \text{ how many were } k) \qquad \text{rec}$$

$$(\text{row } k: \text{ of the true } k \text{ clips, how many were found})$$

Worked example: if the 2 true `news-03` clips are predicted `news-03` and `news-05`, recall for `news-03` is 1/2 = 0.5. If one extra `daily-04` clip was also called `news-03`, precision for `news-03` is 1/2 = 0.5. F1 is their harmonic mean.

Why it matters: in the anonymization project the attacker's confusion matrix shows *which* speakers are still identifiable; the content model's shows *which* sentences got garbled.

Visualization Version



Additional reading: you can read all of it or just read the "In short"

What this cell does

In short define `confusion()` (draw the matrix + return precision / recall / F1 per class) and evaluate the task-A model.

- Rows = true sentence, columns = predicted sentence, each cell = number of clips.

What the output means

In short test accuracy **0.250 (3 of 12)** vs chance 0.167; the table shows only **daily-04** (precision 0.22, recall 1.0) and **news-03** (precision 0.33, recall 0.5) were ever found.

- In PyCharm only the table may show here; the next cell redraws the matrix so the plot appears.

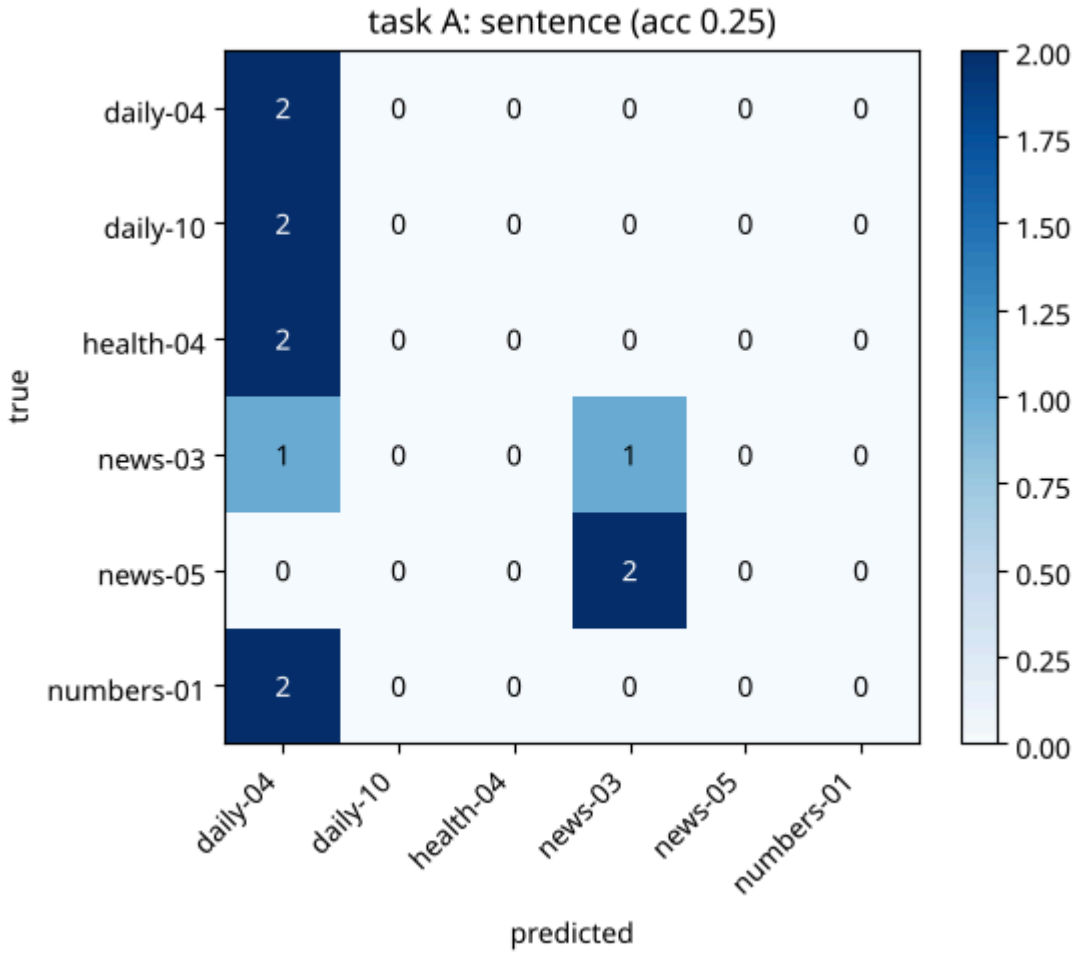
```
from sklearn.metrics import confusion_matrix, classification_report

def confusion(y_true, y_pred, labels, title="confusion matrix", ax=None):
    # rows = true class, columns = predicted class; returns a per-class precision/recall table
    cm = confusion_matrix(y_true, y_pred, labels=range(len(labels)))
    if ax is None:
```

```
        _, ax = plt.subplots(figsize=(5.5, 4.5))
    im = ax.imshow(cm, cmap="Blues")
    ax.set(xticks=range(len(labels)), yticks=range(len(labels)), xlabel="predicted", ylabel="true", title=title)
    ax.set_xticklabels(labels, rotation=45, ha="right"); ax.set_yticklabels(labels)
    for i in range(len(labels)):
        for j in range(len(labels)):
            ax.text(j, i, cm[i, j], ha="center", va="center", color="white" if cm[i, j] > cm.max() / 2 else "black")
    plt.colorbar(im, ax=ax, fraction=0.046)
    rep = classification_report(y_true, y_pred, labels=range(len(labels)), target_names=labels, output_dict=True,
zero_division=0)
    return pd.DataFrame(rep).T.loc[list(labels), ["precision", "recall", "f1-score", "support"]].round(2)

acc_A, yt, yp, pr = evaluate(model_A, te_loader)
print(f"task A test accuracy (speakers held out): {acc_A:.3f}  ({int(round(acc_A*len(yt)))}/{len(yt)} clips)   chance =
{1/6:.3f}")
report_A = confusion(yt, yp, SENTENCES, title=f"task A: sentence (acc {acc_A:.2f})")
plt.show()
report_A
```

```
task A test accuracy (speakers held out): 0.250  (3/12 clips)   chance = 0.167
```



	precision	recall	f1-score	support
daily-04	0.22	1.0	0.36	2.0
daily-10	0.00	0.0	0.00	2.0
health-04	0.00	0.0	0.00	2.0
news-03	0.33	0.5	0.40	2.0
news-05	0.00	0.0	0.00	2.0
numbers-01	0.00	0.0	0.00	2.0

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short redraw the same confusion matrix (a display helper: in PyCharm a cell only shows its last output).

- Same code as above, without the table at the end.

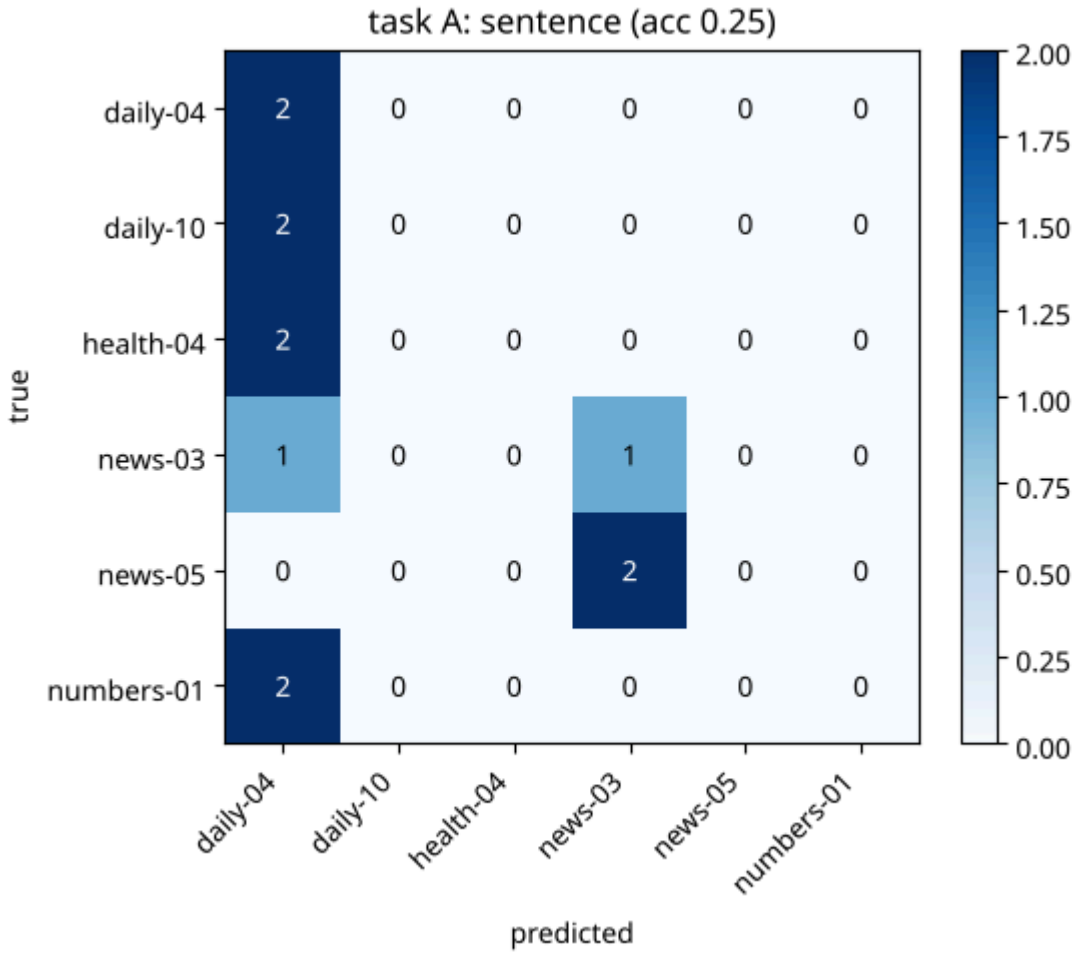
What the output means

In short the matrix: a dark **daily-04 column** (the model's favourite answer) and a 1 on the diagonal for news-03.

- Read it with the How-to-read notes below.

```
%matplotlib inline
acc_A, yt, yp, pr = evaluate(model_A, te_loader)
print(f"task A test accuracy (speakers held out): {acc_A:.3f}  ({int(round(acc_A*len(yt)))}/{len(yt)} clips)   chance =
{1/6:.3f}")
report_A = confusion(yt, yp, SENTENCES, title=f"task A: sentence (acc {acc_A:.2f})")
plt.show()
```

```
task A test accuracy (speakers held out): 0.250  (3/12 clips)   chance = 0.167
```



Additional reading: you can read all of it or just read the "In short"

What this cell does

In short show the per-class table again.

- Just displays `report_A`.

What the output means

In short same table as above.

- Explained cell by cell in the notes below.

```
report_A
```

	precision	recall	f1-score	support
daily-04	0.22	1.0	0.36	2.0
daily-10	0.00	0.0	0.00	2.0
health-04	0.00	0.0	0.00	2.0
news-03	0.33	0.5	0.40	2.0
news-05	0.00	0.0	0.00	2.0
numbers-01	0.00	0.0	0.00	2.0

Additional Reading

In short diagonal = correct; a lit column = a favourite class (low precision); an empty row = never recognized (recall 0).

How to read a confusion matrix. Count down the diagonal: those are correct (3 of 12 in this run, seed 0). Each row sums to 2 (two held-out speakers read each sentence). A dark off-diagonal cell is a systematic confusion: the model hears sentence X and says Y. Look at *which* pairs are confused and go back to the log-mel grid in section 2: sentences with similar length and syllable rhythm are the natural confusions for a model that only sees coarse blob patterns. A whole column lit up means the model has a favorite class it dumps uncertain clips into (low precision for that column): here it is `daily-04`, predicted for most clips (recall 1.0, precision 0.22). An empty row means a sentence it never recognizes (recall 0): four of the six sentences here.

In the table, `support` is the number of true clips per class. With support 2, precision and recall can only be 0, 0.5 or 1: treat them as hints, not measurements.

Additional reading: you can read all of it or just read the "In short"

The four columns of the table

In short support = how many real clips; precision = how often a guess was right (column); recall = how many real clips were found (row); F1 = both in one number.

- **Support**: the number of real test clips for that sentence. Not a model score, just a count. Ours: 2 for every sentence, because two held-out speakers each read all six. With support 2, recall can only be 0, 0.5 or 1.0, so treat all scores as hints.
- **Precision**: of all the times the model guessed this sentence, how many were right. Read **down a column**: correct ÷ total guesses. Ours: daily-04 = **0.22** (guessed 9 times, right 2); news-03 = **0.33** (guessed 3 times, right 1); the other four were never guessed (0.0).
- **Recall**: of all the real clips of this sentence, how many the model caught. Read **along a row**: correct ÷ support. Ours: daily-04 = **1.0** (both caught, but only because it says daily-04 to almost everything); news-03 = **0.5**; the other four = 0.
- **F1**: one score combining both, 2PR ÷ (P + R), pulled toward the lower one, 0 if either is 0. Ours: daily-04 = **0.36**, news-03 = **0.40**, the rest 0. Macro F1 (average over the 6) ≈ **0.13**.

7. The experiment: the split decides what the model learns

Four runs on the same 48 clips, same model, same recipe, three seeds each:

run	label	split	what a high score would mean
A	sentence (6)	**speakers held out**: train 6 spk / test 2 spk	model recognizes *content* in new voices
B	speaker (8)	**sentences held out**: train 4 sent / test 2 sent	model recognizes *voice* on new words
A-random	sentence (6)	**random** 36/12 over clips; every speaker in both sets	? (we will see)
A-leak	sentence (6)	the 12 test clips are **also in the training set** (augmented copies)	nothing: the model has seen the test audio

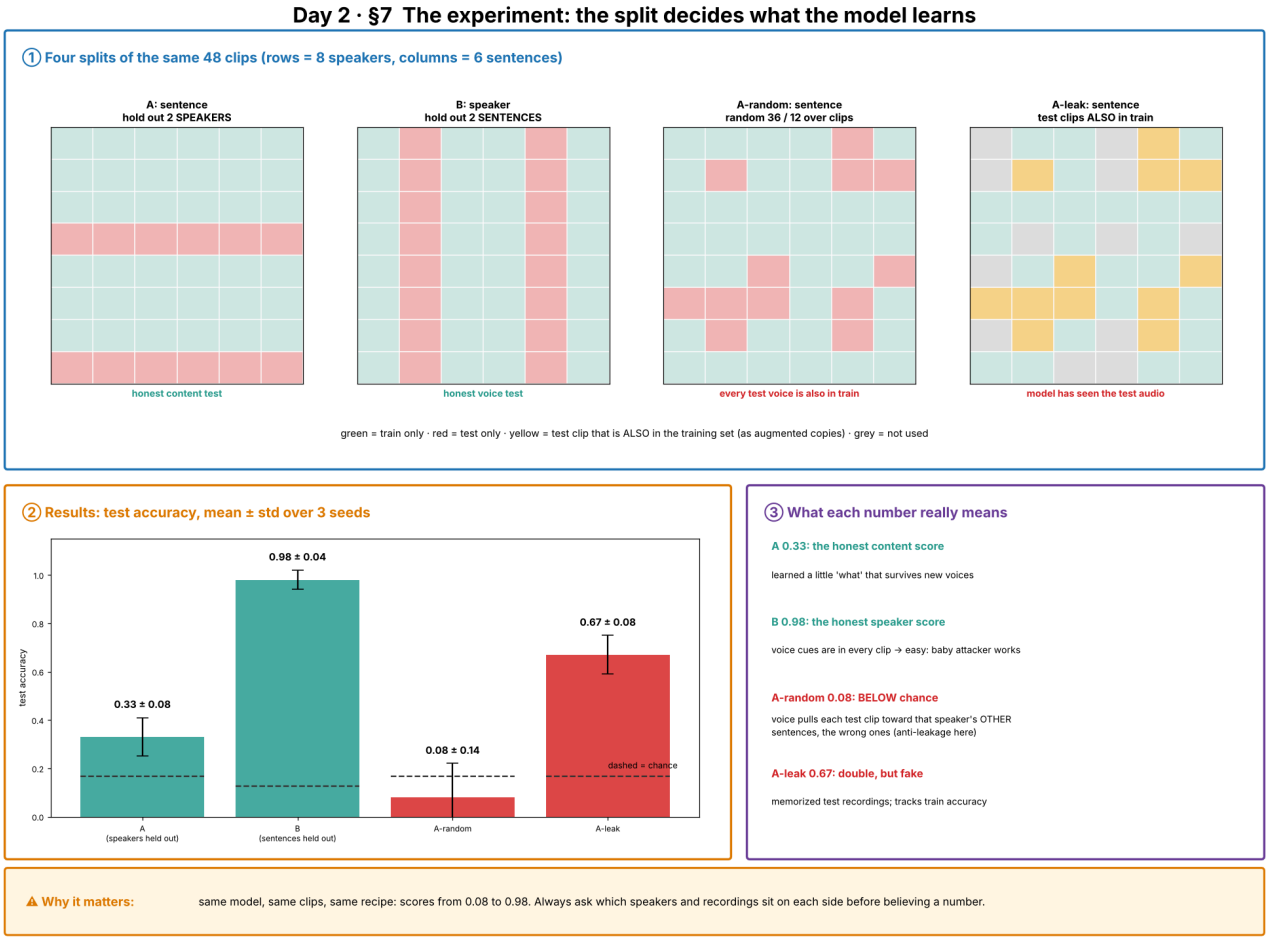
Chance is 1/6 = 0.167 for A and 1/8 = 0.125 for B.

A-random is the split everyone runs by default (`train_test_split` over rows). A-leak is the split people run *by accident*: augment first (random crops, noise, masks of every recording), then split the augmented pool, so copies of the same recording land on both sides. In Kaggle audio competitions and more than one published paper this is how "99 % accuracy" happens.

Repeat with seeds. 12 or 16 test clips make every accuracy a multiple of 0.083 or 0.0625. Different seeds change the initial weights, the shuffle and the augmentation, and will move the

score by one or two clips. We report mean ± std over 3 seeds and read differences smaller than one std as noise.

Visualization Version



Additional reading: you can read all of it or just read the "In short"

What this cell does

In short build the other three splits, then train **4 runs × 3 seeds = 12 models** and collect test accuracy into one table.

- **B**: speaker labels, 2 sentences (daily-10, news-05) held out → 32 train / 16 test.
- **A-random**: a random 36 / 12 split over clips.
- **A-leak**: the training set *contains* the 12 test clips (augmented copies).
- Seeds 0, 1, 2 change the initial weights, the shuffle and the augmentation.

What the output means

In short **A 0.33 ± 0.08 · B 0.98 ± 0.04 · A-random 0.08 ± 0.14 · A-leak 0.67 ± 0.08** (chance 0.17 / 0.125 / 0.17 / 0.17).

- Per run the log shows test acc and train acc; all 12 runs take about a minute.
- `A-leak: 12 of 12 test clips are also training clips`: the leak is built in on purpose.
- `train_last` (train accuracy at the end) is far above test accuracy for the honest runs, and roughly equal to it for A-leak: the signature of a leak.

```
# ---- splits
HELD_OUT_SENT = [SENTENCES[1], SENTENCES[4]]      # two of six sentences never seen by the speaker model
test_B = np.where(meta.sentence_id.isin(HELD_OUT_SENT))[0]
train_B = np.where(~meta.sentence_id.isin(HELD_OUT_SENT))[0]
perm = np.random.RandomState(0).permutation(len(meta))
test_R, train_R = np.sort(perm[:12]), np.sort(perm[12:])    # random split over clips: every speaker on both sides
train_L = np.sort(perm[:36])                                # leak: training set CONTAINS the 12 test clips (augmented)
print("task B (speaker): train on", len(train_B), "clips of", meta.sentence_id[train_B].nunique(), "sentences; test on",
      len(test_B), "clips of held-out", HELD_OUT_SENT)
print("A-random: the", meta.speaker[test_R].nunique(), "test speakers all appear in training; per test clip, its speaker's
OTHER sentences are in training")
print("A-leak:  ", len(np.intersect1d(train_L, test_R)), "of 12 test clips are also training clips")

RUNS = {   # name: (train_idx, test_idx, labels, n_classes)
    "A: sentence, speakers held out": (train_A, test_A, meta.y_sent.values, 6),
    "B: speaker, sentences held out": (train_B, test_B, meta.y_spk.values, 8),
    "A-random: sentence, random split": (train_R, test_R, meta.y_sent.values, 6),
    "A-leak: sentence, test clips in train": (train_L, test_R, meta.y_sent.values, 6),
}
SEEDS = [0, 1, 2]
results, models, hists, preds = [], {}, {}, {}
t_all = time.time()
for name, (tr_idx, te_idx, labels, k) in RUNS.items():
    for seed in SEEDS:
        if name.startswith("A:") and seed == 0:                # already trained in section 5 (same seed, same code)
            m, h = model_A, hist_A
            acc, yt, yp, pr = evaluate(m, te_loader)
        else:
            seed_everything(seed)
            trl, tel = make_loaders(tr_idx, te_idx, labels, seed=seed)
            m = SmallCNN(k)
            h = train(m, trl, tel, epochs=40, verbose=False)
            acc, yt, yp, pr = evaluate(m, tel)
        results.append({"run": name, "seed": seed, "test_acc": acc, "train_acc_last": h["train_acc"][-1], "chance": 1 / k,
"seconds": h["seconds"]})
        if seed == 0:
            models[name], hists[name], preds[name] = m, h, (yt, yp, pr)
            torch.save(m.state_dict(), OUT / f"{name.split(':')[0].replace('-', '_')}_seed0.pt")
        print(f"{name:40s} seed {seed}  test acc {acc:.3f}  train acc {h['train_acc'][-1]:.2f}   ({h['seconds']:.0f} s)")
print(f"all runs: {time.time()-t_all:.0f} s")

res = pd.DataFrame(results)
table = res.groupby("run").agg(test_mean=("test_acc", "mean"), test_std=("test_acc", "std"),
                               train_last=("train_acc_last", "mean"), chance=("chance", "first")).round(3)
table["test_acc (mean ± std, 3 seeds)"] = [f"{m:.2f} ± {s:.2f}" for m, s in zip(table.test_mean, table.test_std)]
table = table.loc[list(RUNS)][["test_acc (mean ± std, 3 seeds)", "test_mean", "train_last", "chance"]]
res.to_csv(OUT / "results_3seeds.csv", index=False)
table
```

```
task B (speaker): train on 32 clips of 4 sentences; test on 16 clips of held-out ['daily-10', 'news-05']
A-random: the 5 test speakers all appear in training; per test clip, its speaker's OTHER sentences are in training
A-leak:   12 of 12 test clips are also training clips
A: sentence, speakers held out           seed 0  test acc 0.250  train acc 0.61   (8 s)
A: sentence, speakers held out           seed 1  test acc 0.333  train acc 0.78   (7 s)
A: sentence, speakers held out           seed 2  test acc 0.417  train acc 0.61   (7 s)
B: speaker, sentences held out           seed 0  test acc 1.000  train acc 0.97   (7 s)
B: speaker, sentences held out           seed 1  test acc 0.938  train acc 1.00   (8 s)
B: speaker, sentences held out           seed 2  test acc 1.000  train acc 0.91   (7 s)
A-random: sentence, random split         seed 0  test acc 0.000  train acc 0.81   (7 s)
A-random: sentence, random split         seed 1  test acc 0.000  train acc 0.69   (8 s)
A-random: sentence, random split         seed 2  test acc 0.250  train acc 0.72   (8 s)
A-leak: sentence, test clips in train    seed 0  test acc 0.667  train acc 0.58   (7 s)
A-leak: sentence, test clips in train    seed 1  test acc 0.750  train acc 0.72   (8 s)
A-leak: sentence, test clips in train    seed 2  test acc 0.583  train acc 0.61   (8 s)
all runs: 83 s
```

run	test_acc (mean ± std, 3 seeds)	test_mean	train_last	chance
A: sentence, speakers held out	0.33 ± 0.08	0.333	0.667	0.167
B: speaker, sentences held out	0.98 ± 0.04	0.979	0.958	0.125

A-random: sentence, random split	0.08 ± 0.14	0.083	0.741	0.167
A-leak: sentence, test clips in train	0.67 ± 0.08	0.667	0.639	0.167

Additional reading: you can read all of it or just read the "In short"

Ranking the four runs

In short B is the strong honest number; A is the honest (weak) content number; A-leak looks good but is fake; A-random is below chance.

- 1. **Model B**, speaker ID: 0.98. The clear winner, and the one honest number that's genuinely strong.
- 2. **A-leak**, sentence ID with contaminated training: 0.67. Second on paper, but it's a fake score, memorised test clips rather than learned skill.
- 3. **Model A**, sentence ID with speakers held out: 0.33. Third on paper, but it's the only honest sentence number, and that's the one you'd report.
- 4. **A-random**, speakers on both sides: 0.08. Last, below chance, because the voice shortcut pulls it toward wrong answers.

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short draw the four confusion matrices (seed 0), then the training curves of B and A-leak.

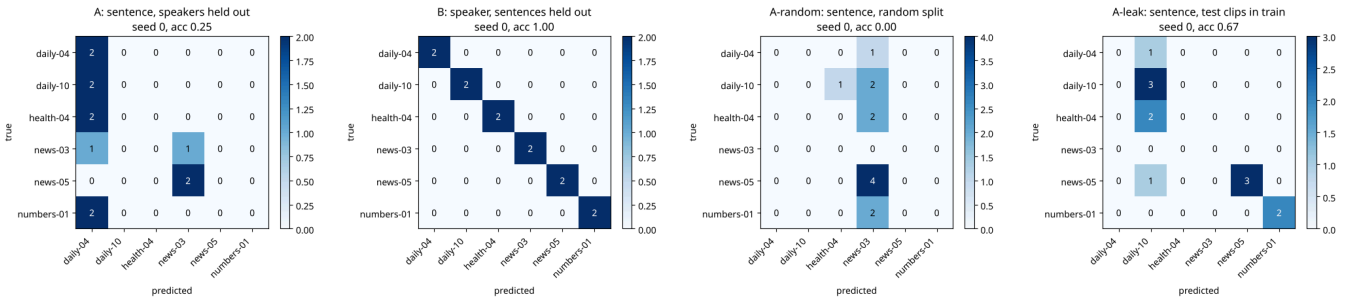
- One matrix per run, then two curve plots.

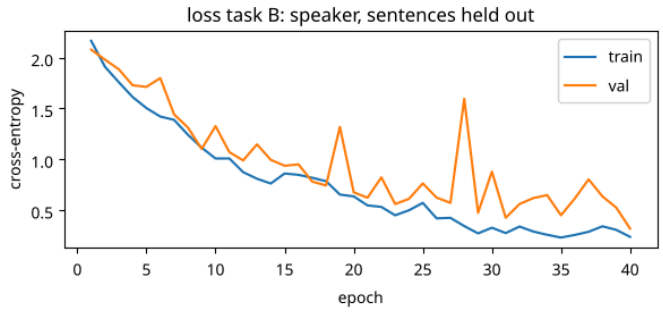
What the output means

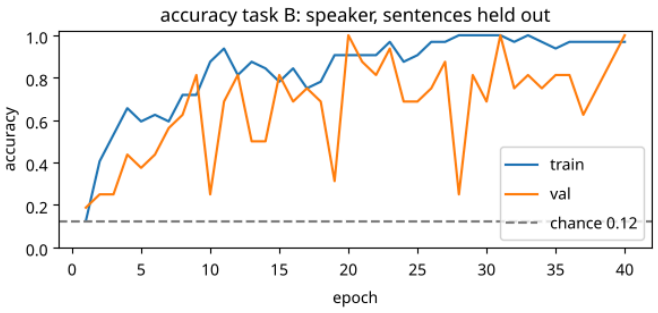
In short B has a **clean diagonal (acc 1.00)**; A-random piles predictions into **one wrong column (acc 0.00)**; A-leak's validation curve **follows the training curve**, which never happens on an honest split.

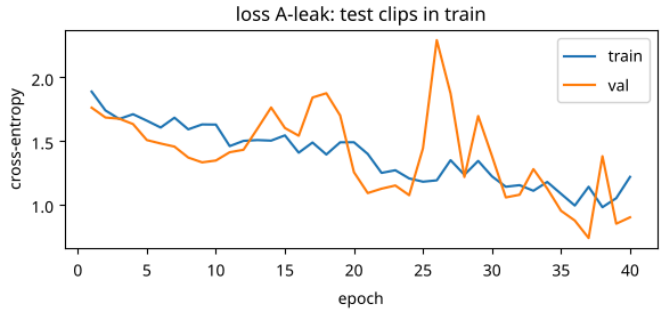
- See the How-to-read notes below.

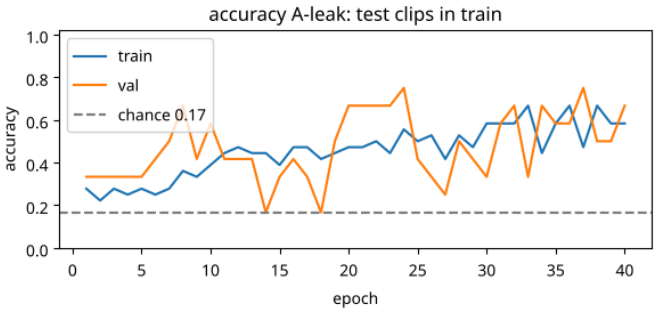
```
fig, axes = plt.subplots(1, 4, figsize=(21, 4.5))
for ax, (name, (yt, yp, pr)) in zip(axes, preds.items()):
    labels = SENTENCES if "sentence" in name else [s[-8:] for s in SPEAKERS]
    confusion(yt, yp, labels, title=f"{name}\nseed 0, acc {np.mean(yt == yp):.2f}", ax=ax)
plt.tight_layout(); plt.show()
plot_curves(hists["B: speaker, sentences held out"], "task B: speaker, sentences held out", chance=1 / 8)
plot_curves(hists["A-leak: sentence, test clips in train"], "A-leak: test clips in train", chance=1 / 6)
```











Additional Reading

In short the split, not the model, decides what the number means: same network, scores from 0.08 to 0.98.

How to read the table and the four matrices (the real numbers from this run are in the table above; the pattern is what matters, and is stable across seeds):

- **A (speakers held out)** is the honest content score: 0.33 ± 0.08 over 3 seeds, chance 0.17. The model has learned *something* about what is said that survives a change of voice, from only 6 training speakers, but not much: one or two sentences are recognized reliably, the rest are near guessing. The confusions sit between sentences of similar length and rhythm. This is what "sentence recognition from 36 clips" really looks like.
- **B (sentences held out)** is the honest speaker score: 0.98 ± 0.04 (one clip wrong in one seed), chance 0.125, with 32 training clips. Pitch stripes and spectral tilt are present in every clip regardless of the words, so 4 sentences per speaker are enough to learn 8 voices. The remaining confusions are typically within gender. This is the baby version of the attacker in our research, and it already works. The task-B curves reach high validation accuracy within ~15 epochs with a small train/val gap: speaker identity is simply an easier, "always-on" signal than sentence content.
- **A-random** is *not* inflated. It is 0.08 ± 0.14, below chance and below A. Think about why: each speaker read every sentence exactly once, so for a test clip (speaker *s*, sentence *x*), the training set contains speaker *s* reading the *other* five sentences. A model that leans on voice

cues is pulled toward the sentences it heard from that voice, which are by construction the wrong ones. In a balanced factorial design, speaker leakage into a content task is *anti*-leakage. In a real dataset, where speakers do not read the same sentences and content correlates with speaker (one person always talks about football), the same shortcut would inflate the score instead. Either way the number answers a question nobody asked.

- **A-leak** is inflated, with no new skill: 0.67 ± 0.08 on the *same* task where the honest number is 0.33, because the model has seen augmented versions of every test recording. Its score tracks its *training* accuracy (0.64 in the `train_last` column), and the A-leak curves show the "validation" line following the training line upward, which never happens on an honest split. This is the signature of a leak: val curve too good, too early, too close to train.
- **Train accuracy** in the last column is far above test accuracy for every honest run, and does not reach 1.0 within 40 epochs because augmentation keeps changing the images. The test columns are the only columns that mean anything.

Additional reading: you can read all of it or just read the "In short"

What we did, in plain words

In short same network, same clips, same recipe; only the split changed, and the scores ranged from 8 % to 98 %.

What we did: took one network, the SmallCNN, and one training recipe, and trained it under four different data setups on the same 48 probe clips. Each setup was run three times with seeds 0, 1 and 2, meaning three different random weight initializations, batch orders and augmentation draws, and the three scores were averaged. Twelve trainings total, one results table.

- The four setups:
  - Model A, sentence classifier, six outputs, two speakers held out for testing. The honest “what was said” score: about 33 percent against 17 percent chance.
  - Model B, speaker classifier, eight outputs, two sentences held out. The honest “who said it” score: about 98 percent against 12.5 chance. Your baby attacker.
  - A-random, same A network and labels, but the clips shuffled with no regard to speaker, so every test voice is also in training. Scores 8 percent, below chance here, but in real data the same shortcut inflates.
  - A-leak, same A network and labels, but the test clips were also slipped into training as augmented copies. Scores 67 percent, double the honest number, with no real skill.

The lesson: same architecture, same clips, same recipe, and the numbers ranged from 8 to 98. The split and the label decide what the model learns and what the score means. That’s why the anonymization project holds out speakers, and why you should always ask which speakers and recordings sit on each side before believing any result.

8. What did the model look at? Occlusion

sensitivity

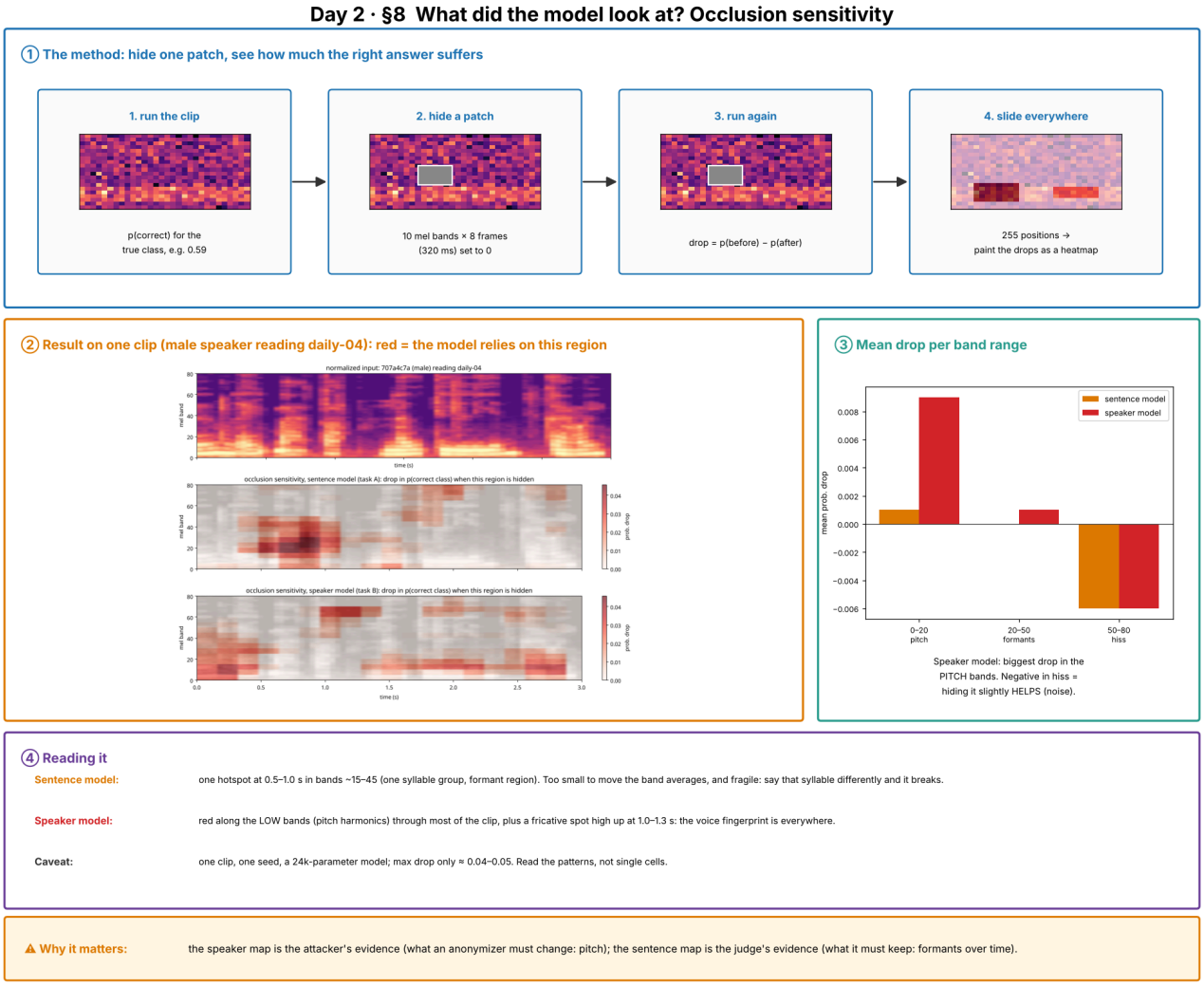
Slide a grey patch (zeros, = the mean after normalization) over the input, and for each position record how much the probability of the *correct* class drops. Regions where the drop is large are regions the model relies on. No gradients needed, just N forward passes:

$$S(f,t) = p_y(M) - p_y\big(M \odot \mathrm{mask}_{f,t}\big)$$

Patch 10 mel bands × 8 frames (320 ms), stride 5 bands × 4 frames: 15 × 17 = 255 positions, a second on CPU.

We run it on the same clip twice: with the sentence model (A) and with the speaker model (B). The clip is from a speaker the sentence model never saw, chosen as the one it gets right with the highest confidence; for the speaker model it is a training clip (its sentence is not one of the held-out two), which is fine for asking *what the model looks at*. The hypotheses from section 2: the content model should care about the mid-frequency blob trajectory over time (formants and syllable rhythm), the speaker model about the lowest bands (pitch harmonics) and overall spectral balance, mostly independent of *when* in the clip.

Visualization Version



Additional reading: you can read all of it or just read the "In short"

What this cell does

In short compute **occlusion maps** for one held-out clip with both models: hide a 10 × 8 patch at 255 positions and record how much p(correct) drops.

- `occlusion_map`: no gradients, just many forward passes in batches of 64.
- The clip is the held-out one the sentence model is most confident about (a map of a wrong, unsure prediction is just noise).
- `band_stats`: the mean drop in three band ranges (pitch, formants, hiss).

What the output means

In short clip: speaker 707a4c7a (male) reading **daily-04**; sentence model p(correct) **0.59**, speaker model **0.57**, both right; max drop ≈ **0.04–0.05**.

- Speaker model: biggest mean drop in **bands 0–20 (pitch), 0.009**; negative in 50–80 (−0.006): hiding hiss slightly helps.
- Sentence model: its hotspot is small, so its band averages stay ≈ 0; read its map instead.

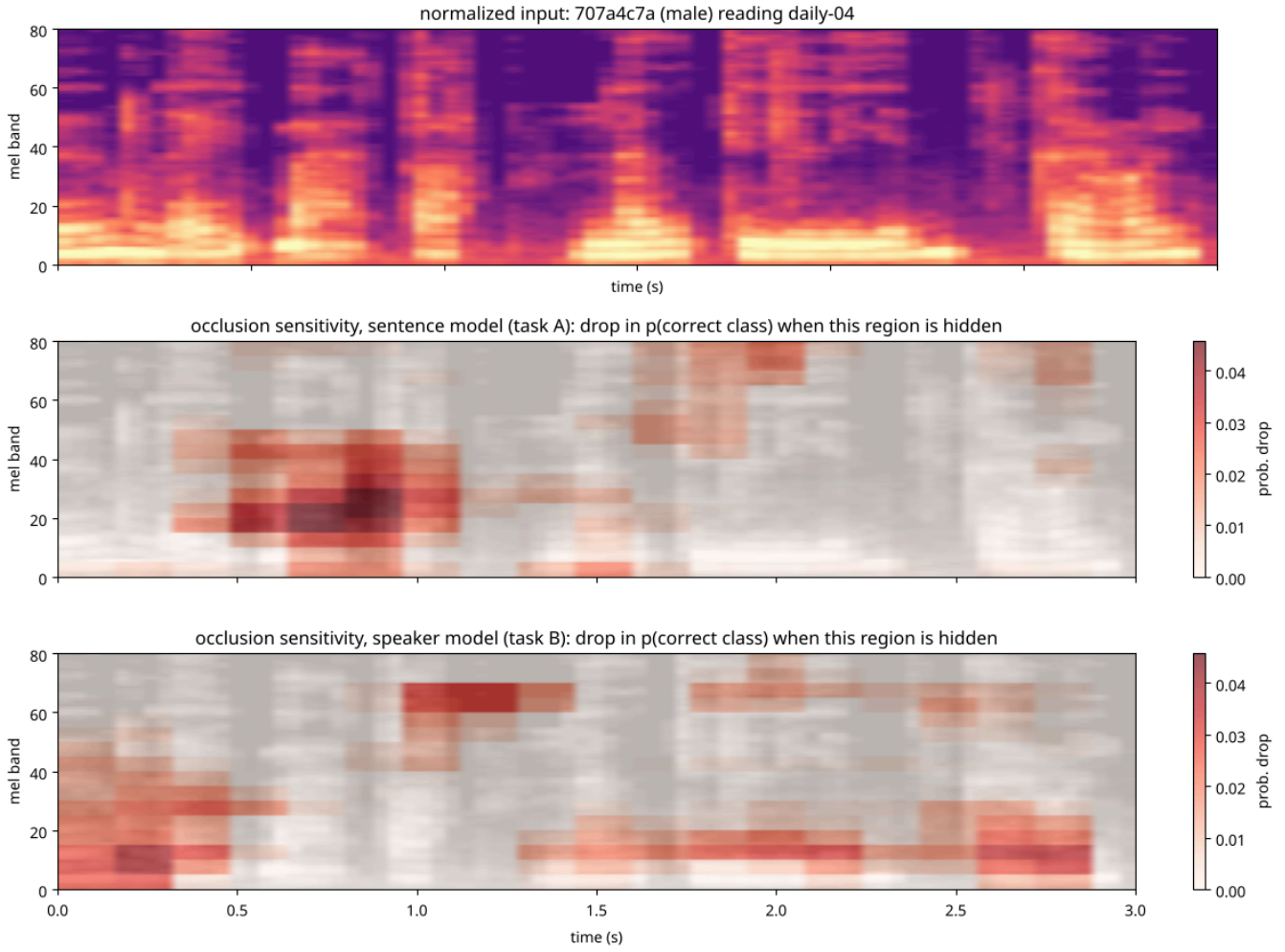
```
@torch.no_grad()
def occlusion_map(model, M, y, patch=(10, 8), stride=(5, 4)):
    # M: normalized input (80, 75); returns (drop map at input resolution, baseline probability of class y)
    model.eval()
    x = torch.from_numpy(M)[None, None]
    p0 = F.softmax(model(x), 1)[0, y].item()
    n_mels, n_frames = M.shape
    S = np.zeros((n_mels, n_frames)); C = np.zeros((n_mels, n_frames))
    batch, pos = [], []
    for f in range(0, n_mels - patch[0] + 1, stride[0]):
        for t in range(0, n_frames - patch[1] + 1, stride[1]):
            xm = x.clone(); xm[0, 0, f:f + patch[0], t:t + patch[1]] = 0.0
            batch.append(xm); pos.append((f, t))
    probs = torch.cat([F.softmax(model(torch.cat(batch[i:i + 64])), 1)[:, y] for i in range(0, len(batch), 64)]).numpy()
    for (f, t), p in zip(pos, probs):
        S[f:f + patch[0], t:t + patch[1]] += p0 - p; C[f:f + patch[0], t:t + patch[1]] += 1
    return S / np.maximum(C, 1), p0

mA_model = models["A: sentence, speakers held out"]; mB_model = models["B: speaker, sentences held out"]
# a clip from a speaker the sentence model never saw, and one it classifies correctly with the highest confidence,
# so the map shows evidence FOR the right answer (a map of a wrong, low-confidence prediction is just noise)
def p_correct(mdl, i, y):
    return F.softmax(mdl(torch.from_numpy(features(MELS[i]))[None, None]), 1)[0, int(y)].item()
i = max(test_A, key=lambda i: p_correct(mA_model, i, meta.y_sent[i]))
print("held out for the sentence model:", i in test_A, "| held out for the speaker model:", i in test_B)
M = features(MELS[i])
print("clip", meta.utt_id[i], "| speaker", meta.speaker[i][-8:], meta.gender[i], "| sentence", meta.sentence_id[i],
meta.text[i])
t0 = time.time()
mapA, pA = occlusion_map(mA_model, M, int(meta.y_sent[i]))
mapB, pB = occlusion_map(mB_model, M, int(meta.y_spk[i]))
print(f"occlusion maps: {time.time()-t0:.1f} s")
band_stats = {}
for name, m, p, mdl, y in [("sentence model", mapA, pA, mA_model, meta.y_sent[i]), ("speaker model", mapB, pB, mB_model,
meta.y_spk[i])]:
    pred = int(F.softmax(mdl(torch.from_numpy(M)[None, None]), 1)[0].argmax())
    band_stats[name] = {"bands 0-20 (pitch)": m[:20].mean(), "bands 20-50 (formants)": m[20:50].mean(), "bands 50-80
(hiss)": m[50:].mean()}
    print(f"{name:15s} p(correct) = {p:.2f}, predicted {pred} (true {int(y)}) | max drop {m.max():.2f} | mean drop: " +
          ", ".join(f"{k} {v:.3f}" for k, v in band_stats[name].items()))

fig, axes = plt.subplots(3, 1, figsize=(12, 9), sharex=True)
show_input(M, axes[0], f"normalized input: {meta.speaker[i][-8:]} ({meta.gender[i]}) reading {meta.sentence_id[i]}")
vmax = max(mapA.max(), mapB.max(), 1e-3)
for ax, m, name in [(axes[1], mapA, "sentence model (task A)"), (axes[2], mapB, "speaker model (task B)")]:
```

```
    ax.imshow(M, origin="lower", aspect="auto", cmap="gray", vmin=-2.5, vmax=2.5, extent=[0, 3, 0, 80])
    im = ax.imshow(m, origin="lower", aspect="auto", cmap="Reds", alpha=0.65, vmin=0, vmax=vmax, extent=[0, 3, 0, 80])
    ax.set(title=f"occlusion sensitivity, {name}: drop in p(correct class) when this region is hidden", ylabel="mel band")
    plt.colorbar(im, ax=ax, fraction=0.02, label="prob. drop")
axes[2].set(xlabel="time (s)")
plt.tight_layout(); plt.show()
```

```
held out for the sentence model: True | held out for the speaker model: False
clip iany_707a4c7a_ec6b242a | speaker 707a4c7a male | sentence daily-04 ពួកយើងលង់ហាល់ទាន់នៅទីលាន ក្រោយម៉ោងរៀន
occlusion maps: 0.9 s
sentence model  p(correct) = 0.59, predicted 0 (true 0) | max drop 0.05 | mean drop: bands 0-20 (pitch) 0.001, bands 20-50
(formants) -0.000, bands 50-80 (hiss) -0.006
speaker model   p(correct) = 0.57, predicted 7 (true 7) | max drop 0.04 | mean drop: bands 0-20 (pitch) 0.009, bands 20-50
(formants) 0.001, bands 50-80 (hiss) -0.006
```



Additional reading: you can read all of it or just read the "In short"

In short occlusion shows WHERE each model looks: the speaker model on the low pitch bands everywhere, the sentence model on one syllable in the formant region.

Occlusion sensitivity: making the model show its working

The question. Section 7 told us *how well* each model scores and whether that was luck (three seeds). Occlusion sensitivity answers a third question: *what part of the spectrogram is the model actually using?* It does not grade the model; it explains one decision.

How it works. The model cannot explain itself, so we interrogate it:

1. Take one test clip (here: a male speaker reading daily-04) and record the model's probability for the correct class, $p_y(M)$.
2. Hide one small patch of the log-mel (10 mel bands × 8 frames = 320 ms) by setting it to 0, which after normalization means "the clip's average, nothing distinctive".
3. Run the model again and record the drop: $S(f,t) = p_y(M) - p_y(M \odot \mathrm{mask}_{f,t})$.
4. Slide the patch over every position (stride 5 bands × 4 frames = 255 positions) and paint the drops as a heatmap on top of the spectrogram.

No gradients are needed, just forward passes. Positive = hiding this spot hurt (the model relied on it). Grey ≈ 0 = the model does not care. Slightly negative = hiding it helped (that spot was noise or a distraction).

How to read the figure. Both heatmaps sit exactly on top of the input: same time axis, same mel bands. The colour bar tops out at ≈ 0.04, so even the darkest spot only costs the model about 4 % of its confidence. That is expected: the model trained with SpecAugment holes and is used to missing pieces, and one patch is a small part of an 80 × 75 image.

What we saw.

- **Sentence model (task A):** one dark hotspot at ≈ 0.5–1.0 s, mel bands 20–45: a single syllable group in the formant region (vowel shapes). Everywhere else is grey. A content model *should* look at formant trajectories and syllable rhythm, but this one leans on one landmark, which fits a weak model (0.33 in section 7): say that syllable differently and it breaks.
- **Speaker model (task B):** red spread along the *whole* clip, strongest in the low bands (0–30) where the pitch harmonics sit, with a second patch high up (≈ band 60–70, a fricative, the shape of this speaker's "s"). The voice fingerprint is present in every frame, so no single moment is critical. Hiding the hiss region slightly *helps*: high-frequency noise is a distraction for it.
- **Band-average table:** the speaker model's drop is largest in bands 0–20 (pitch, 0.009) and negative in bands 50–80 (−0.006). The sentence model's hotspot is too small to move its band averages (≈ 0), so for it the map is the evidence, not the averages. Same input, two models, two different places to look.

Caveats. One clip, one seed, a 24k-parameter model trained on 32–36 clips: the maps are blotchy and the two models can disagree on individual patches. Treat the band averages as the robust reading, not single hot cells. An occlusion map is only as good as the prediction it explains: on a clip the model is unsure about, the map is flat and meaningless.

What this means for the anonymization project

The two models here are baby versions of the two referees in our research:

Day 2	Research project	Question	We want it to…
speaker model (task B)	**the attacker**: ECAPA-TDNN speaker verification, scored by EER	*who* is speaking?	**fail** on anonymized speech
sentence model (task A)	**the judge**: Qwen3-ASR, scored by CER	*what* was said?	**succeed** on anonymized speech

The speaker model's heatmap is the **attacker's evidence map**. It shows where the identity lives in the spectrogram: the low bands (pitch harmonics F0, 2F0, 3F0 …) across the whole clip, plus speaker-specific high-frequency texture (fricatives). That is exactly what an anonymizer has to change:

- **McAdams** (Day 6) warps the formants (the envelope) but leaves F0 alone, so the pitch harmonics the attacker relies on survive. That is why its ρF0 ≈ 0.73 and its EER (29.6 % ignorant) is far below the ≈ 44 % of kNN-VC (Khmer pool), and why a semi-informed attacker gets it back down to 14 %.
- **kNN-VC** (Day 6) replaces the whole frame with a target speaker's frames, moving both the harmonics and the envelope, which is why its EER is near chance. What leaks is timing (durations are copied frame for frame) and the prosody *shape*.

The sentence model's heatmap shows the opposite constraint: content lives in the **mid-band formant trajectories and the syllable rhythm over time**. An anonymizer that destroys those (ASR→TTS, a too-strong McAdams alpha) pushes CER up. The privacy–utility trade-off of Day 7 is literally "change the regions in the bottom heatmap, keep the regions in the middle one".

Two honest limits: the real attacker (ECAPA) is a far stronger model than this CNN, trained on thousands of speakers, so it will find identity cues this toy model misses, including channel and prosody cues. And occlusion maps are per-clip explanations, not a guarantee: the research evaluation still has to be the full EER / CER benchmark over all 43 test speakers.

Additional reading: you can read all of it or just read the "In short"

Four questions to ask the picture

In short red on speech = honest; red on silence = cheating; speaker model should be red low, sentence model red in the middle; spread out = robust.

Four questions to ask the picture.

- Is the red on the speech? Red on the bright speaking parts is honest. Red on silence, dark gaps, or clip edges means the model is cheating off something that isn't the voice. That's the only truly bad result.
- Is there red at all? Some red means the model depends on the input. All grey means the map is useless or the model isn't really looking.
- Is the red on the right floor? Speaker model should be red in the bottom bands, spread across the whole clip. Sentence model should be red in the middle bands at particular moments. If they're swapped, the model learned the wrong thing.
- Is the red spread out or one spot? Spread out means robust. One spot means fragile: honest, but relying on a single clue.

Additional reading: you can read all of it or just read the "In short"

Why this matters for the anonymization project

In short the attacker's evidence (pitch bands) is what to destroy; the judge's evidence (formant shapes) is what to keep.

The two Day 2 models are miniature versions of the project's two referees: the speaker model is a baby ECAPA attacker ("who is this?"), the sentence model is a baby ASR judge ("what was said?"). Occlusion shows what each one actually looks at, which turns into a change-this, keep-that blueprint for the anonymizer.

- Attacker's evidence = what to destroy. The speaker heatmap lights up the low mel bands (pitch harmonics) across the whole clip. An anonymizer that leaves pitch alone leaves the attacker's main evidence intact. This matches the leaderboard: McAdams only warps formants, keeps F0 (ρF0 ≈ 0.73) and reaches EER ≈ 30; kNN-VC (English pool) replaces every frame, wipes the low bands, and pushes EER to ≈ 41, near chance.
- Judge's evidence = what to keep. The sentence heatmap lights up mid-band formant shapes at specific syllables. McAdams warps exactly those, which is why its CER jumps to ≈ 38 while kNN-VC stays near 28.
- A diagnostic, not just a score. Running occlusion on the real attacker against an anonymized clip shows where identity still leaks (for example, pitch bands still red), which is more actionable than "EER rose a little". It can guide which cue to attack next. Caveat: one clip, small effect sizes (max ≈ 0.04), so it points at hypotheses rather than proving them.

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short show the band-average table as a neat table.

- One row per model, one column per band range.

What the output means

In short speaker model **0.009 in the pitch bands**, the largest number in the table; both models slightly negative in the hiss bands.

- Same numbers as printed above.

```
pd.DataFrame(band_stats).T.round(3)
```

	bands 0-20 (pitch)	bands 20-50 (formants)	bands 50-80 (hiss)

sentence model	0.001	-0.000	-0.006
speaker model	0.009	0.001	-0.006

Additional Reading

In short sentence model: one hotspot at 0.5–1.0 s in the formant bands; speaker model: low pitch bands along the whole clip plus one fricative spot.

How to read the occlusion maps. Red = hiding this patch hurts the correct class; grey = the model does not care. The underlying input is drawn in grey so you can see *what* is under the red. The table gives the mean drop per band range, which is the robust reading; single hot cells are not.

- **Sentence model (middle):** one strong hotspot, at 0.5–1.0 s in bands 20–45: the second syllable group of the sentence, in the formant region. Hiding that one syllable costs the correct class about 0.05 of probability; hiding almost anything else costs nothing, and the band-range means are ≈ 0 because the hotspot is small and local. The model has latched onto one piece of *content* at one *time*, which is what a "what was said" model should do, and also why it is fragile: a different speaker saying that syllable differently is enough to break it (section 7's 0.33).
- **Speaker model (bottom):** the red sits low, in bands 5–20 where the pitch harmonics are, and runs along most of the clip (0–0.4 s, 1.3–2.9 s): hiding the harmonics *anywhere* costs a little, no single moment is critical. The band-range means confirm it: the 0–20 band mean drop is the largest for this model and the hiss region is slightly negative (hiding high bands *helps* a little: it removes noise). There is also a secondary spot at 1.0–1.3 s in bands 60–70, a fricative: the shape of a speaker's "s" is a speaker cue too.
- Caveats: this is one clip, one seed, and a 24k-parameter model trained on 32–36 clips, so the maps are blotchy and the two models can disagree with the hypothesis on individual patches. Masked patches are zeros, which the model also saw during SpecAugment, so it is somewhat used to "missing" regions and the drops are modest in absolute terms (a few hundredths of probability). A model with low confidence on a clip gives small drops everywhere and a map that is pure noise, which is why we picked the held-out clip the sentence model is most confident about. In the first run of this notebook the sentence model's confidence on the only clips held out for *both* tasks was 0.07, and its map was flat: an occlusion map is only as good as the prediction it explains.

Additional reading: you can read all of it or just read the "In short"

More detail: pitch vs formants with your own mouth

In short pitch = the buzz (who you are); formants = the mouth shape (what you say). The attacker uses the buzz, the judge uses the mouth shapes.

- Pitch versus formants, with your own mouth. Close your lips and hum a note. That buzz is the pitch: it's how fast your vocal folds vibrate. Now keep humming the exact same note but open your mouth and go "ah", then "ee", then "oo". The note didn't change, but the sound did. That change is the formants: the shape of your mouth and tongue colouring the buzz. So pitch is the buzz, formants are the mouth shape. Pitch mostly depends on your body, so it says who. Formants depend on what you're pronouncing, so they say what. On the spectrogram, pitch shows as thin stripes at the bottom, formants as broad bright bands in the middle.

- Bullet one. The speaker model's heatmap went red on the bottom stripes everywhere. Meaning: the attacker recognises you by your buzz. So to hide you, the anonymizer must change the buzz. McAdams only reshapes the mouth and leaves the buzz alone, so the attacker still gets you about half the time. kNN-VC swaps everything, buzz included, so the attacker is close to guessing.

  Further: CER is the judge's score: character error rate. The ASR transcribes the anonymized clip, you compare it to the true Khmer text, and count the wrong characters as a percentage. Lower is better. On the untouched clips it's about 8. So 38 means more than a third of the characters came out wrong. McAdams is the simplest anonymizer in your project: a formant shifter. It takes the mouth-shape bands and slides them up or down the frequency axis. "Warp" just means that sliding. The buzz, the pitch, is untouched. So it's like keeping your voice box but giving you a slightly different mouth: "ah" drifts toward "aw", "ee" toward "ih". The judge reads words from exactly those shapes, so it mishears a lot. That's the 38. kNN-VC is a frame swapper. It chops your clip into tiny slices, and for each slice finds the most similar-sounding slice in a pool of someone else's recordings and pastes that in. The result is another person saying roughly your sounds. The words mostly survive because the swap is by similarity of sound, but every slice is still imperfect, especially with an English pool for Khmer. That's the 28: worse than 8, much better than 38. So bullet two says: McAdams wrecks the judge's evidence, kNN-VC mostly keeps it.

- Bullet two. The sentence model's heatmap went red on the middle bands. Meaning: the judge reads words from the mouth shapes. McAdams warps exactly those, so the judge mishears, and CER jumps to 38. kNN-VC keeps the shapes roughly, so CER stays near 28.

- Bullet three. After you anonymize a clip, run the same occlusion on the real attacker. If it still goes red on the bottom stripes, your buzz is leaking, and you know what to fix. A plain EER number can't tell you that.

9. Compare: a pretrained feature extractor + logistic regression (optional, guarded)

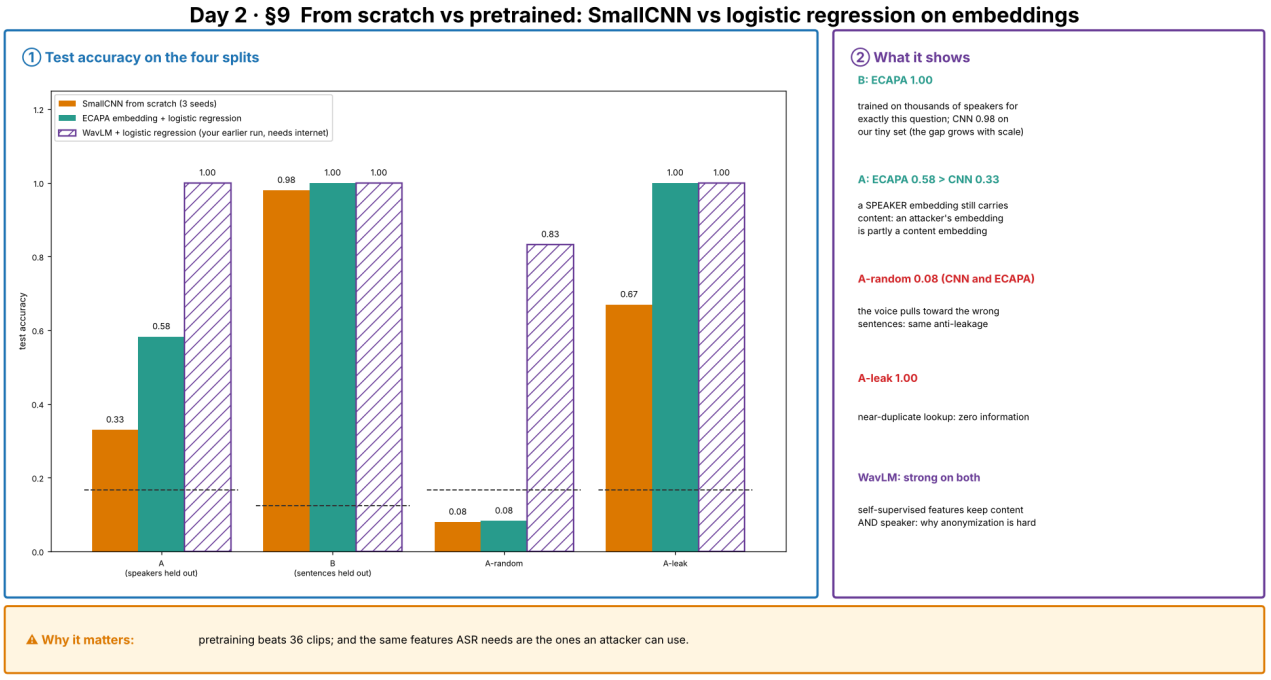
Our CNN learned everything from 36 clips. The alternative used in practice: take a model pretrained on thousands of hours of speech, mean-pool its features over time to one vector per clip, and fit a linear classifier. Two candidates:

- **WavLM-base** (`microsoft/wavlm-base` , ~380 MB download, needs internet and `transformers`): self-supervised, features carry both content and speaker information.

- **ECAPA-TDNN speaker embeddings** (192-d, shipped in `data/embeddings/original.npz` for all 48 clips, computed by the Day 6 attacker): trained *only* to tell speakers apart.

Same four splits as section 7, logistic regression instead of the CNN. The code tries WavLM first; if the download fails it says so and continues with the shipped ECAPA embeddings, so the comparison always runs.

Visualization Version



Additional reading: you can read all of it or just read the "In short"

What this cell does

In short fit a **logistic regression** on pretrained embeddings (WavLM if it can be downloaded, otherwise only the shipped ECAPA embeddings) for the same four splits, and compare with the CNN.

- Embeddings = one vector per clip from a model trained on thousands of hours / speakers.
- `StandardScaler + LogisticRegression` : a simple linear classifier on top; no augmentation, no randomness.

What the output means

In short ECAPA + logistic regression: **A 0.58 · B 1.00 · A-random 0.08 · A-leak 1.00**, vs the CNN's 0.33 / 0.98 / 0.08 / 0.67.

- In this saved run WavLM could not be downloaded (no internet in the environment that ran it), so only the ECAPA column appears.
- With internet, WavLM gave **1.00 / 1.00 / 0.83 / 1.00** in your own earlier run (shown hatched in the picture above).

```
# model: microsoft/wavlm-base (~380 MB) -- needs internet (first run only); cached in ~/.cache/huggingface
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.pipeline import make_pipeline

feats = {}
try:
    from transformers import AutoFeatureExtractor, WavLMModel
    fe = AutoFeatureExtractor.from_pretrained("microsoft/wavlm-base")
    wavlm = WavLMModel.from_pretrained("microsoft/wavlm-base").eval()
    with torch.no_grad():
        E = []
        for x in WAVES:
            inp = fe(x, sampling_rate=SR, return_tensors="pt")
            hs = wavlm(**inp, output_hidden_states=True).hidden_states
            E.append(torch.stack(hs[6:9]).mean(0)[0].mean(0).numpy())    # mid layers, mean-pooled over time
    feats["WavLM-base (mean-pooled, layers 6-8)"] = np.stack(E)
    print("WavLM features:", feats[list(feats)[-1]].shape)
except Exception as e:
    print(f"WavLM not available ({type(e).__name__}: {str(e)[:90]}). Skipping WavLM, using the shipped ECAPA embeddings
only.")

emb = np.load(DATA / "embeddings/original.npz")
lookup = {u: e for u, e in zip(emb["utt_id"], emb["emb"])}
feats["ECAPA-TDNN speaker embedding (192-d, shipped)"] = np.stack([lookup[u] for u in meta.utt_id])

rows = []
for fname, X in feats.items():
    for name, (tr_idx, te_idx, labels, k) in RUNS.items():
        clf = make_pipeline(StandardScaler(), LogisticRegression(C=1.0, max_iter=2000))
        clf.fit(X[tr_idx], labels[tr_idx])
        rows.append({"features": fname, "run": name, "test_acc": clf.score(X[te_idx], labels[te_idx])})
cmp = pd.DataFrame(rows).pivot(index="run", columns="features", values="test_acc").loc[list(RUNS)]
cmp["SmallCNN from scratch (3 seeds)"] = table["test_acc (mean ± std, 3 seeds)"]
cmp["chance"] = table["chance"]
cmp.round(3)
```

```
WavLM not available (OSError: Can't load feature extractor for 'microsoft/wavlm-base'. If you were trying to load it fro). S
kipping WavLM, using the shipped ECAPA embeddings only.
```

features	ECAPA-TDNN speaker embedding (192-d, shipped)	SmallCNN from scratch (3 seeds)	chance
run			
A: sentence, speakers held out	0.583	0.33 ± 0.08	0.167
B: speaker, sentences held out	1.000	0.98 ± 0.04	0.125
A-random: sentence, random split	0.083	0.08 ± 0.14	0.167
A-leak: sentence, test clips in train	1.000	0.67 ± 0.08	0.167

Additional Reading

In short a speaker embedding still carries content (A 0.58); pretrained features beat 36 training clips; leaks and shortcuts fool every model.

How to read the comparison. Logistic regression on a 192-d ECAPA embedding has 192 × 8 weights, no augmentation and no randomness, so one run is enough.

- **Task B (speaker):** the ECAPA embedding was trained on exactly this question with thousands of speakers, so logistic regression on it gets every held-out clip right (1.00). Our from-scratch CNN matches it on this tiny test set (0.98) with 32 clips; on hundreds of speakers it would not, and the gap is the value of pretraining.
- **Task A (sentence) on ECAPA: 0.58**, above our CNN's 0.33. This is the surprise, and it is a real finding, not noise (7 of 12 clips; three seeds of the CNN never got above 5). The embedding was trained to tell speakers apart, not to throw content away, and on short utterances a "speaker" embedding still carries phonetic content: which vowels were said shifts the mean spectrum, and the embedding keeps that. In our research this is a known headache: an attacker's speaker embedding is partly a content embedding, so the same two sentences read by different people can look "closer" than two different sentences by one person.
- **A-random on ECAPA (0.08)** shows the anti-leakage effect from section 7 in its purest form: the voice part of the embedding pulls each test clip toward the training sentences of the same speaker, which are by construction the wrong ones, and the score falls *below* chance, exactly like the CNN. Any score on a split that leaks the nuisance variable is driven by that variable, in whichever direction the data's structure pushes it.
- **A-leak on ECAPA (1.00)** is a near-duplicate lookup: the same clip's embedding is in training with the right label. Perfect score, zero information.
- If WavLM ran, its column is strong on both honest tasks: self-supervised features keep content *and* speaker. In our research, that is why anonymization is hard: the same features that an ASR system needs are the ones an attacker can use.

Build: a reusable training kit (`src/train_kit.py`)

Everything above is now packaged once, in `src/train_kit.py`, so later days can `from src.train_kit import ...` instead of copying cells: `mel_power`, `precompute`, `crop_frames`, `add_noise_mel`, `pool_time`, `normalize`, `spec_augment`, `LogMelDataset`, `SmallCNN`, `seed_everything`, `train`, `evaluate`, `plot_curves`, `confusion`, `make_loaders` and a one-call `run_experiment`. The cell below re-imports the kit, lists what it exports, checks that its features are bit-identical to the notebook's, re-runs task B with the kit's `run_experiment` and checks it reproduces the notebook's seed-0 number exactly (same seed, same code path), then saves the best models under `outputs/day2/` and reloads one.

Additional reading: you can read all of it or just read the "In short"

What this cell does

In short re-import `src/train_kit.py`, list what it exports, check its features are identical to the notebook's, re-run task B with it, and save + reload the models.

- Later days use `from src.train_kit import ...` instead of copying these cells.

What the output means

In short the kit's features are **identical**; its task-B run reproduces the notebook's seed-0 accuracy **exactly (1.000)**; the reloaded speaker model scores the same.

- `saved under outputs/day2/` : the 4 seed-0 models, the 3-seed results CSV, and `sentence_cnn.pt` / `speaker_cnn.pt` with their label maps.

```
import importlib, inspect
import src.train_kit as kit
importlib.reload(kit)
print("src/train_kit.py exports:")
for name in ["precompute", "crop_frames", "add_noise_mel", "pool_time", "normalize", "spec_augment", "LogMelDataset",
"SmallCNN",
             "seed_everything", "train", "evaluate", "plot_curves", "confusion", "make_loaders", "run_experiment"]:
    print(f"  {name:16s} {((inspect.getdoc(getattr(kit, name)) or 'seeds random, numpy and torch').splitlines() or ['']
[0]}")

# 1. the kit's cached features are identical to the notebook's
K_MELS, K_PSIG = kit.precompute(WAVES)
same = all(np.array_equal(a, b) for a, b in zip(K_MELS, MELS)) and np.allclose(K_PSIG, PSIG)
same_feat = np.array_equal(kit.LogMelDataset(K_MELS, K_PSIG, meta.y_sent.values)[5][0].numpy()[0], features(MELS[5]))
print("\nkit features == notebook features:", same and same_feat)

# 2. the kit reproduces the notebook's task B seed-0 run exactly
kit_model, kit_hist, kit_acc, kyt, kyp, kpr = kit.run_experiment(K_MELS, K_PSIG, meta.y_spk.values, train_B, test_B,
n_classes=8, seed=0, epochs=40)
nb_acc = res[(res.run == "B: speaker, sentences held out") & (res.seed == 0)].test_acc.iloc[0]
print(f"kit run_experiment task B seed 0: {kit_acc:.3f}   notebook seed 0: {nb_acc:.3f}   identical: {abs(kit_acc -
nb_acc) < 1e-9}")

# 3. save the two models with their label maps so Day 3+ can reload them, and reload one to check
torch.save({"state_dict": models["A: sentence, speakers held out"].state_dict(), "classes": SENTENCES, "n_classes": 6},
OUT / "sentence_cnn.pt")
torch.save({"state_dict": kit_model.state_dict(), "classes": SPEAKERS, "n_classes": 8}, OUT / "speaker_cnn.pt")
ck = torch.load(OUT / "speaker_cnn.pt")
reloaded = kit.SmallCNN(ck["n_classes"]); reloaded.load_state_dict(ck["state_dict"])
re_acc = kit.evaluate(reloaded, DataLoader(kit.LogMelDataset([K_MELS[i] for i in test_B], K_PSIG[test_B],
meta.y_spk.values[test_B]), batch_size=16))[0]
print(f"reloaded speaker model test acc: {re_acc:.3f}")
print("saved under outputs/day2/:", sorted(p.name for p in OUT.iterdir()))
```

```
src/train_kit.py exports:
  precompute       Run the slow mel filterbank once per clip. Returns (list of (80, frames) power arrays, signal powers).
  crop_frames      Pad (zeros at the end) or crop a (n_mels, frames) array to exactly n frames.
  add_noise_mel    Add white noise at snr_db in the mel power domain.
  pool_time        Average every k adjacent frames: (n_mels, T) -> (n_mels, T // k).
  normalize        Per-utterance mean/var normalization: every clip becomes mean 0, std 1.
  spec_augment     SpecAugment in ~10 lines: zero out random horizontal (frequency) and vertical (time) bands.
  LogMelDataset    cached mel power arrays + signal powers + integer labels -> (1, 80, 75) tensors.
  SmallCNN         3 x [Conv3x3 -> BatchNorm -> ReLU -> MaxPool2] -> global average pool -> Linear.
  seed_everything  Seed random, numpy and torch (plus PYTHONHASHSEED) so a run is reproducible.
  train            Plain supervised training loop: cross-entropy + Adam. Returns a history dict of per-epoch numbers.
  evaluate         Returns (accuracy, y_true, y_pred, probs). Runs in eval mode (BatchNorm uses running stats).
  plot_curves      Left: loss per epoch (train vs val). Right: accuracy per epoch. Gap between the lines = overfitting.
  confusion        Plot a confusion matrix (rows = true class, columns = predicted) and return the per-class report.
  make_loaders     Train loader (augmentation on, shuffled with a seeded generator) and test loader (center crop, no augment
ation).
  run_experiment   Train a SmallCNN on train_idx, evaluate on test_idx. Returns (model, history, test_accuracy, y_true, y_pr
ed, probs).
kit features == notebook features: True
kit run_experiment task B seed 0: 1.000   notebook seed 0: 1.000   identical: True
reloaded speaker model test acc: 1.000
saved under outputs/day2/: ['A_leak_seed0.pt', 'A_random_seed0.pt', 'A_seed0.pt', 'B_seed0.pt', 'results_3seeds.csv', 'sente
nce_cnn.pt', 'speaker_cnn.pt']
```