EAI 820 · SPEECH AND LANGUAGE TECHNOLOGIES

Turning Input into Tokens

Subwords & Discrete Audio Tokens



Week 1 · Jurafsky & Martin, SLP3 (draft, Jan 2026) · Chapter 2 · Duration: 2 hours

Dr. Emmanuel Eronu · University of Abuja

Learning objectives

1

Text → machine-processable units

Turn text into tokens; distinguish word types from word instances.

2

Morphemes & morphology

Roots, affixes and clitics — and why morphology matters for tokenization.

3

Byte-Pair Encoding

Explain and hand-trace the BPE tokenization algorithm.

4

The same idea for audio

How discrete audio tokens tokenize sound just as BPE tokenizes text.

ROADMAP

Two parts, one idea

PART A

Words & Tokens

- Types vs. instances (Herdan's Law)
- Morphemes: roots, affixes, clitics
- Unicode, UTF-8 & normalization
- Byte-Pair Encoding (BPE)
- Corpora, regex & a Unix tokenizer

one idea
open-ended
→ finite vocab

PART B

Audio as Tokens

- A waveform is an open-ended signal
- A neural codec learns discrete tokens
- Encoder compresses, decoder rebuilds
- Weeks 6–7 build directly on it

PART A · §A1

Why start with text processing?

Every translator, summarizer or speech recognizer ultimately operates on sequences of symbols. Before a model can learn, we decide what the atomic units of text are.

?

What counts as a word?

Capitalization, disfluencies, clitics, and languages without spaces all complicate the answer.

⇥

How do we split text?

Tokenization — cutting a string into units a model can actually consume.

Δ

How different are two strings?

Edit distance quantifies similarity — we pick this up in Week 2.

These three foundations reappear in every later week.

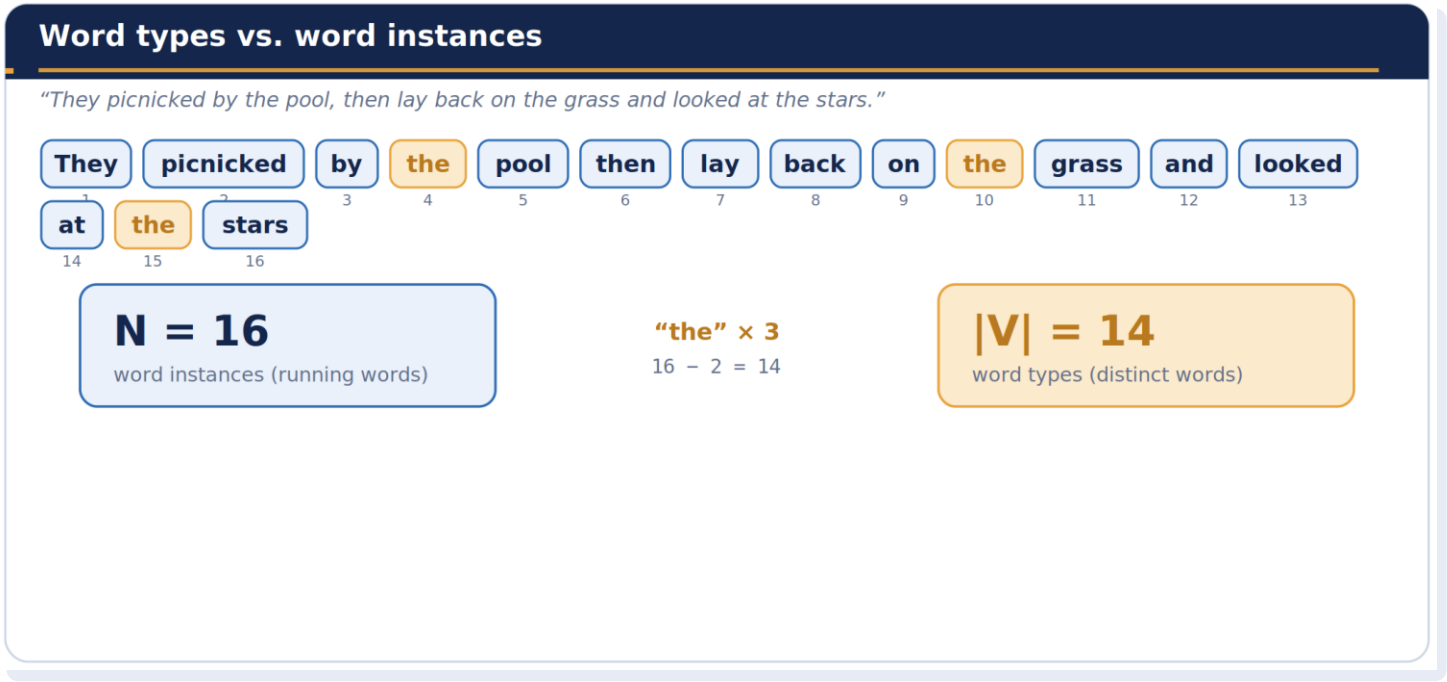
What is a word? Types vs. instances

Word types
the distinct words in a corpus — the vocabulary size |V|.

Word instances
the running words — the total token count N.

|V|
types

N
instances



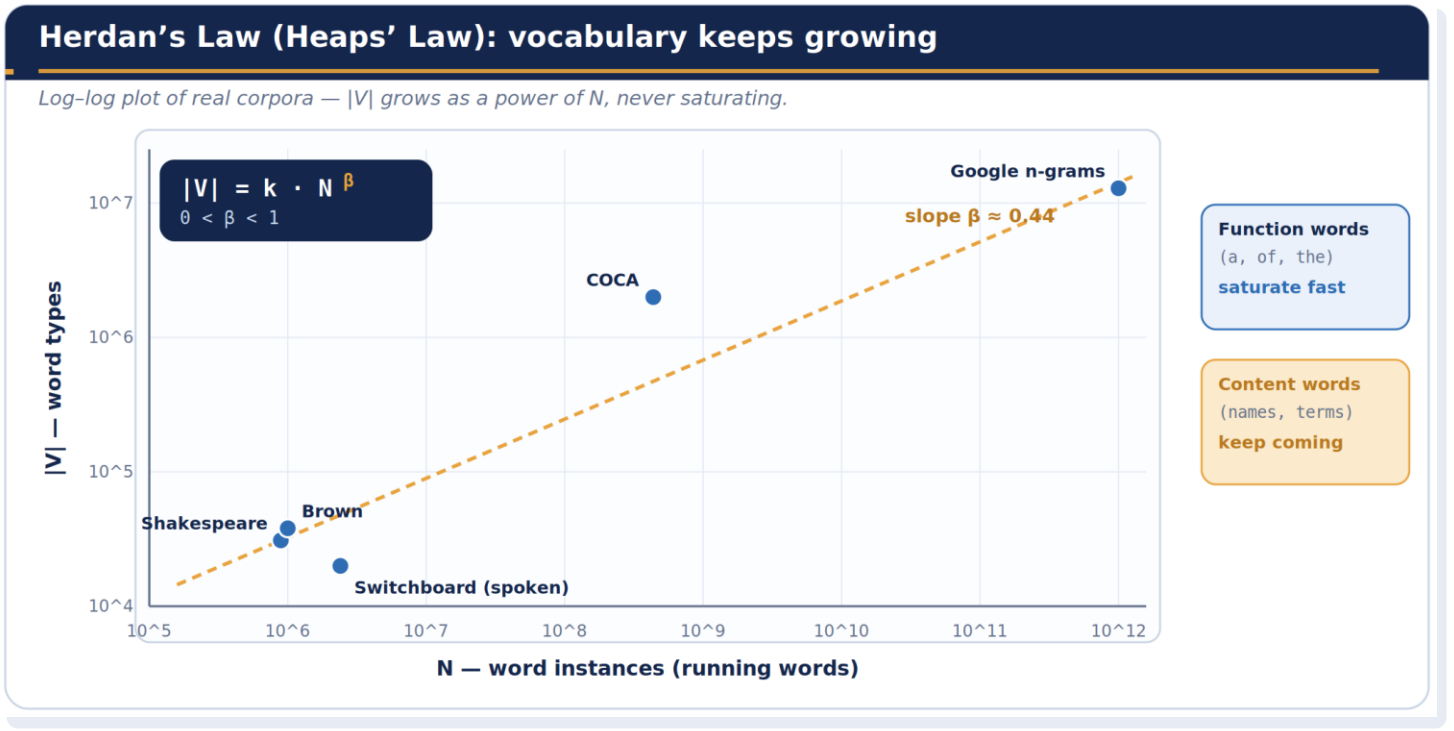
Ignoring punctuation, the example sentence has 14 types and 16 instances — “the” occurs three times.

Vocabulary keeps growing — Herdan's (Heaps') Law

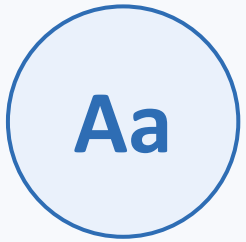
$$|V| = k \cdot N^{\beta}$$

$0 < \beta < 1$ (typically $\beta \approx 0.44$–0.56)

- Vocabulary grows a little faster than √N.
- Function words (a, of, the) saturate fast.
- Content words (names, terms) keep appearing.
- So fixed word vocabularies are fragile — we turn to subword tokenization.



Why we go below the word



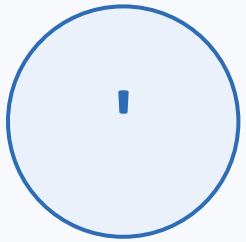
Capitalization

They vs. they — keeping case is task-dependent.



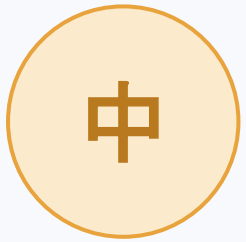
Disfluencies

uh, um — kept for speech recognition.



Clitics

I’m = I + am; the teacher’s; French l’opera.



No spaces

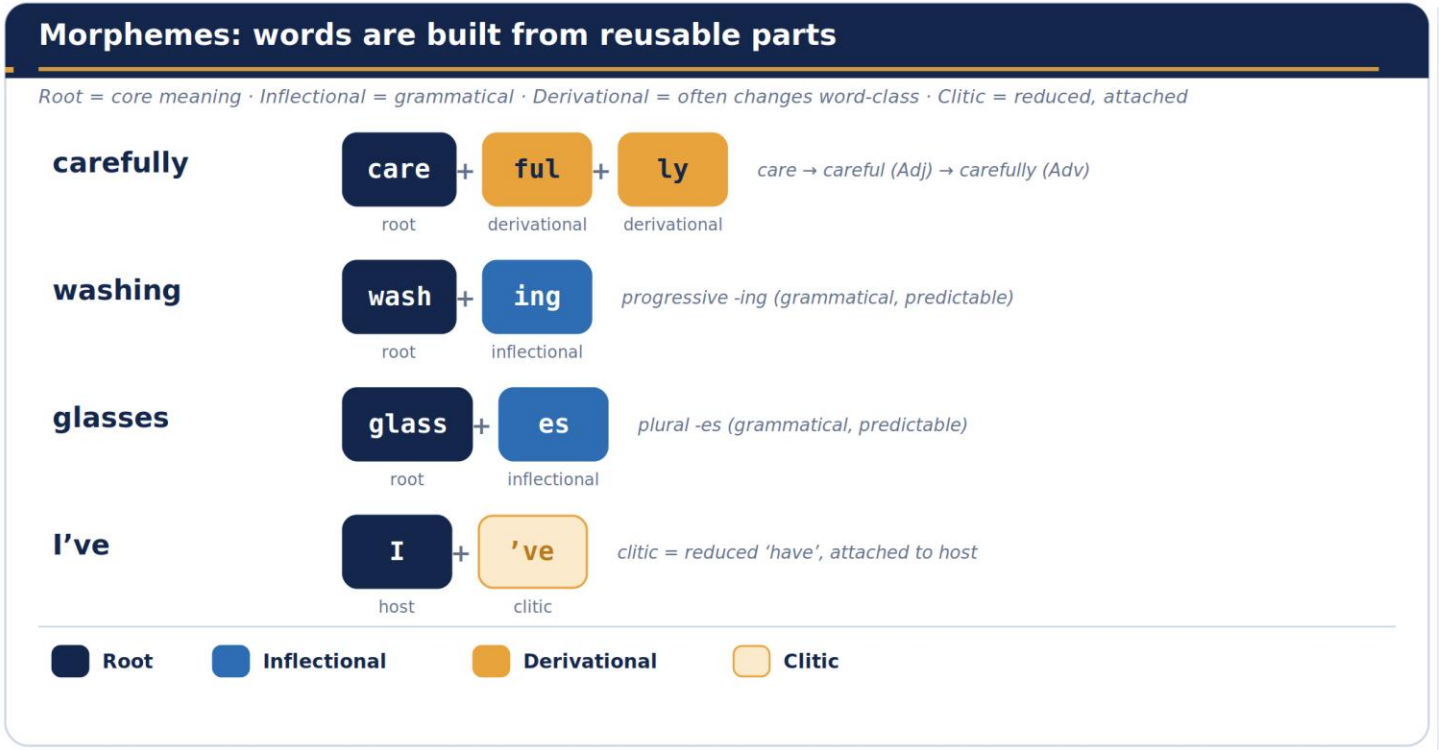
Chinese, Japanese and Thai don’t separate words.

These issues motivate splitting below the word level.

Morphemes: the parts of words

A morpheme is the minimal meaning-bearing unit; its study is morphology.

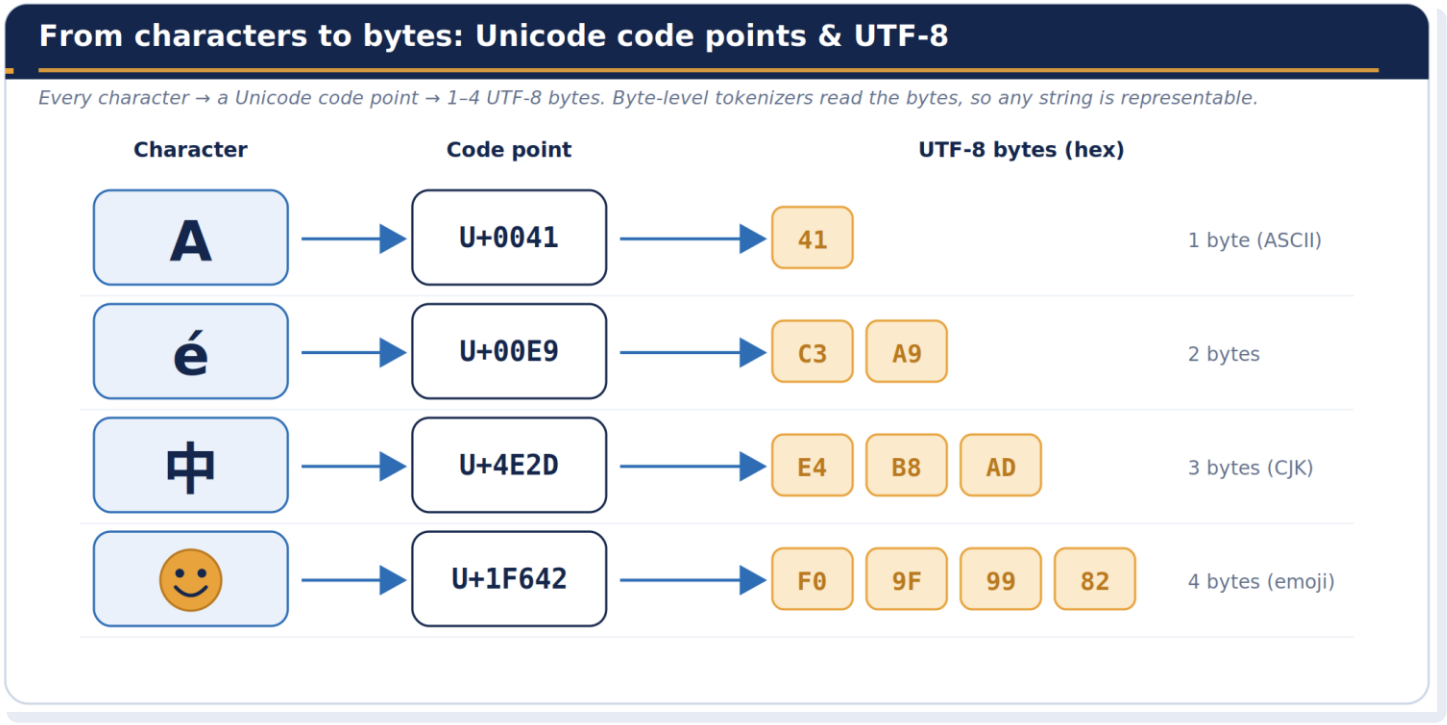
Roots — central meaning (work, glass).
Inflectional affixes — grammatical, predictable (-s, -ed).
Derivational affixes — idiosyncratic, often change word class (care → careful → carefully).
Clitics — reduced, attached to a host (I've, l'opera).



Because content words are endlessly productive, a tokenizer that understands sub-parts generalizes far better — the bridge to BPE.

From characters to bytes

Unicode assigns every character a numeric code point.
UTF-8 is the dominant variable-length encoding — 1 to 4 bytes.
Byte-level tokenizers read those bytes, so any string is representable — no "unknown character".
Normalization comes first: a consistent Unicode form plus task-dependent case and punctuation choices.



Subword tokenization: the compromise

Word-level ✗

Breaks on unknown words (out-of-vocabulary).

Character-level ✗

Represents anything, but sequences get painfully long.

Subword (BPE) ✓

Frequent words stay whole; rare words split into meaningful pieces.

BPE has two parts: **token learner** — run once to induce a vocabulary; **token segmenter** — apply the learned merges to new text.

The BPE token learner

```
function BYTE-PAIR-ENCODING(strings C, num merges k):
    V ← all unique characters in C        # start from
chars
    for i = 1 to k:                       # do k merges
        (t_L, t_R) ← most frequent adjacent pair in C
        t_NEW ← t_L + t_R                 # new merged
token
        V ← V + t_NEW
        replace every (t_L, t_R) in C with t_NEW
    return V
```

Start from characters

Vocabulary begins as the raw character set.

Greedy on frequency

Each step merges the most frequent adjacent pair.

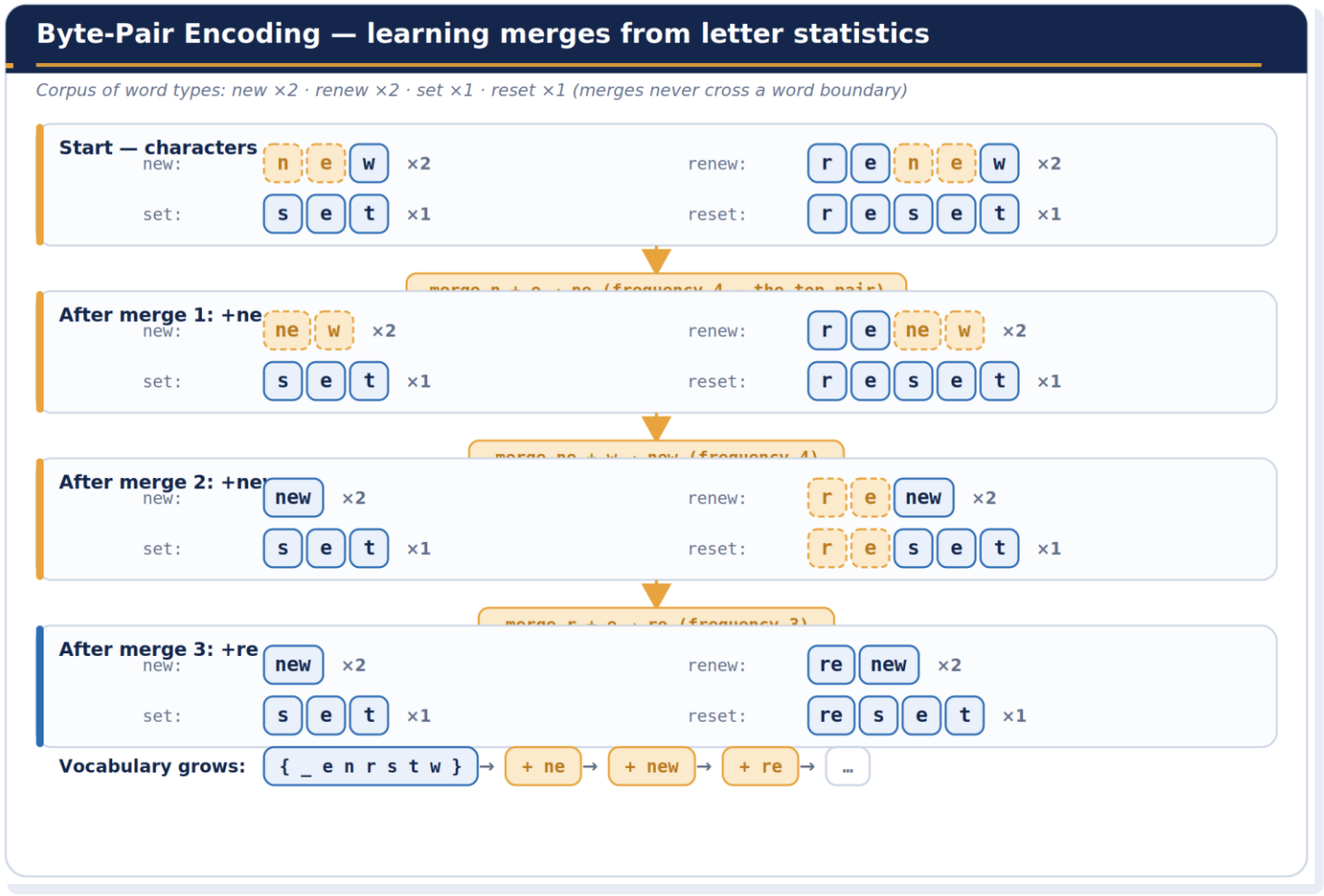
Within words only

Text is pre-split; a merge never crosses a word boundary.

k controls the vocabulary

More merges → longer, word-like tokens.

BPE by hand: “set new new renew reset renew”



What the learner discovers

n · e → **ne**, then **ne · w** → **new**, then **r · e** → **re**.

It has found the prefix **re-** and the whole word **new** — from nothing but letter-sequence counts.

Run over bytes, BPE never produces a truly unknown token.

Corpora, regex & a one-line tokenizer

A corpus is a curated body of text or speech. Ask who produced it, when, why, and with what biases — a "datasheet" for the data.
Regular expressions are the workhorse for string patterns — character classes, ranges, disjunction, capture groups, substitutions — and the first tool for splitting text and writing rule-based tokenizers.

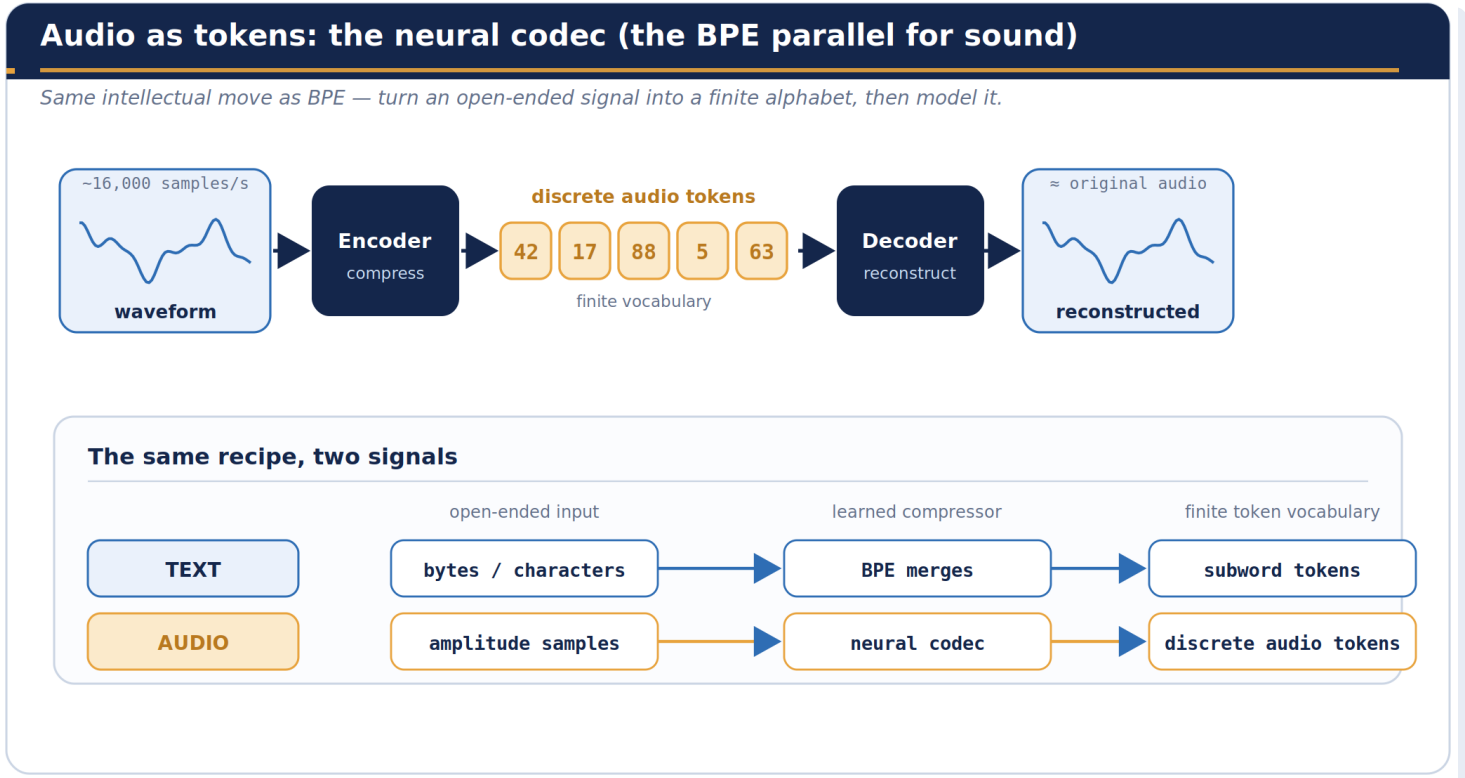
A whitespace/punctuation tokenizer in Unix

```
tr -sc 'A-Za-z' '\n' < file.txt \
  | sort | uniq -c | sort -nr
```

Split on non-letters, then count and rank word frequencies — a surprisingly useful baseline, and a good mental model for what a tokenizer does.

PART B

Audio as tokens: the neural codec



A microphone delivers ~16,000 amplitude samples per second — no sequence model can consume that raw.

A **neural audio codec** learns a vocabulary of discrete audio tokens: an encoder compresses the waveform, a decoder reconstructs it.
The move is identical to BPE — turn an open-ended signal into a finite alphabet. One kind of model now covers speech and text alike.

WRAP-UP

Key takeaways

1

Words are slippery

Types ≠ instances, and vocabularies grow without bound (Herdan / Heaps).

2

Morphology motivates subwords

Words have reusable parts — roots, affixes, clitics.

3

BPE learns from statistics

A subword vocabulary emerges from iterative merges of frequent pairs.

4

Audio tokenizes the same way

A neural codec learns discrete audio tokens — one model for speech and text.

THIS WEEK

Labs, exercises & reading

Labs

- Implement BPE merges from scratch in plain Python.
- Extend the Unix pipeline into a simple tokenizer.

Exercises

- Count types vs. instances — with and without lowercasing.
- Run two more BPE merges on the trace — what token appears next?
- Why is a neural codec the audio counterpart of BPE?

Reading

- Jurafsky & Martin, SLP3 (Jan 2026) — Ch. 2.
- Sennrich et al. (2016) — BPE for NMT.
- Next week: n-gram LMs, smoothing, perplexity, edit distance.