Data Preparation for LLM Pretraining

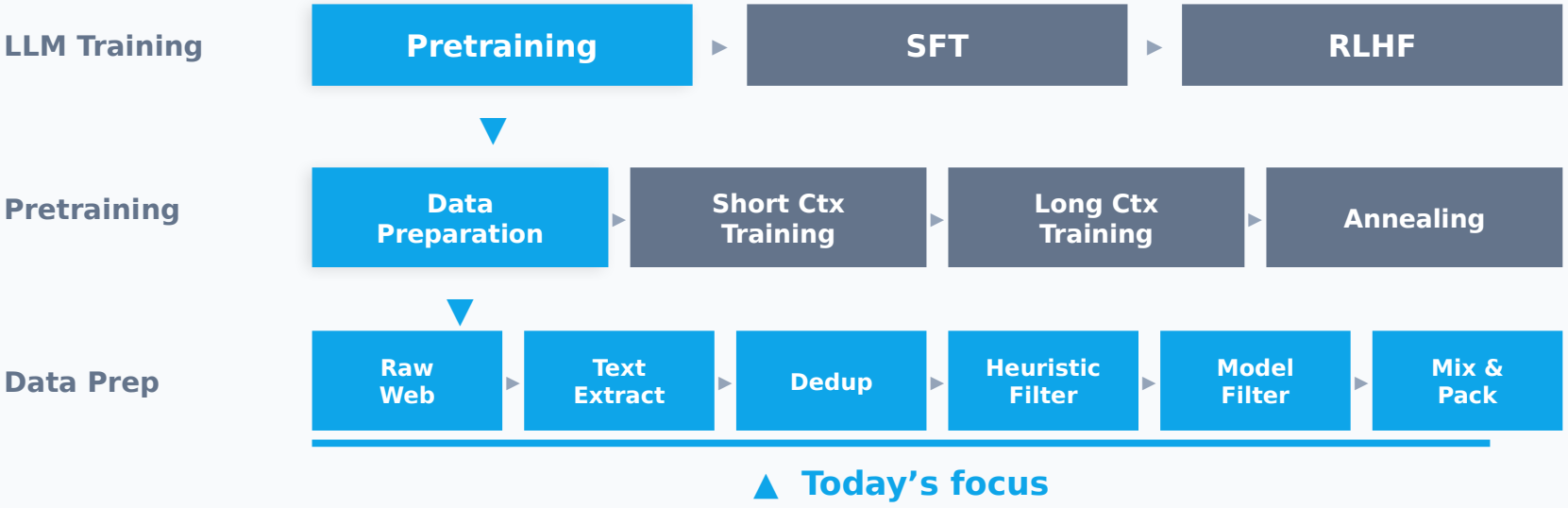
Common practices from open research papers

Ruiyang Wang
Member of Technical Staff, Anthropic

Stanford CS246 • March 12, 2026

You Are Here: Data Preparation

Zooming in from the full LLM pipeline to today's focus



Focus: English web text. Separate pipelines exist for code, math, and multilingual data.

The Data Preparation Pipeline

Every major lab converges on roughly this sequence

English Web Text

Raw Web (CC) ▸ Text Extraction ▸ Dedup ▸ Heuristic Filters ▸ Model Filters ▸ Data Mix ▸ Tokenizer ▸ Pack & Format

▸ **Ready to train**

Papers we'll reference:

FineWeb · Llama 3 · DeepSeek · Qwen · Gopher

Other pipelines:

Code & Math	Domain-specific parsers, heuristics, and classifiers
Multilingual	Language ID, per-language dedup and filters
Books / Other	Different acquisition, similar cleaning principles

Text Extraction: WARC vs WET

WET (pre-extracted text) ✗

```
Home | About | Products | Blog
We use cookies. Accept | Reject

The discovery of gravitational
waves in 2015 confirmed...

Share on Facebook | Tweet
Copyright 2024 Example Inc.
```

WARC + trafilatura ✓

```
The discovery of gravitational
waves in 2015 confirmed a key
prediction of general relativity.

The signal, detected by LIGO,
matched theoretical models of
two merging black holes.
```

FineWeb re-extracted from WARC using trafilatura → measurably better model performance

Llama 3: Custom HTML parsers, preserves alt text for math formulas **Qwen:** Qwen2.5-VL to OCR PDFs — reaches text no HTML parser can

All of this runs as MapReduce/Spark jobs over billions of pages (Lecture 1).

Deduplication: Scale of the Problem

~22%

duplicate within a single CC dump

~90%

duplicate across all 91 CC dumps

Dedup rate scales with # of CC dumps (DeepSeek V1, Table 1):

Dumps	1	2	6	12	16	22	41	91
Dedup %	22.2	46.7	55.7	69.9	75.7	76.3	81.6	89.8

Deduplication: Three Levels

1 URL-Level

Hash the URL, keep the latest version.
Cheap, removes a huge chunk.

Done first during extraction

2 Document-Level

MinHash + LSH for fuzzy matching.
Catches near-duplicates: same article
with minor edits across 1000 sites.

Lectures 3–4

3 Line-Level

Remove individual lines appearing
>6 times across 30M-doc buckets.
Catches cookie notices, nav templates.

From Llama 3 pipeline

Document Dedup: MinHash at Scale

You learned MinHash + LSH in Lectures 3–4. Here’s how it’s used on 15 trillion tokens.

Target: ~75% Jaccard similarity • FineWeb adds transitive matching: A~B, B~C → A~C

Configuration:

Shingle size	5-gram (word-level)
Hash functions	112
LSH bands	14 bands × 8 hashes
Similarity target	~75% Jaccard
Transitive closure	A~B, B~C → remove C too

Shingling & MinHash Signatures

Documents → 5-word shingles → 112 hash functions → fixed-size signature

Shingling

```
"the cat sat on the mat"
```

5-word sliding window:

```
{"the cat sat on the",
 "cat sat on the mat"}
```

Longer documents produce thousands of shingles.

MinHash Signature

For each of 112 hash functions:

1. `Hash every shingle`
2. `Take the minimum`
3. `Store as sig[i]`

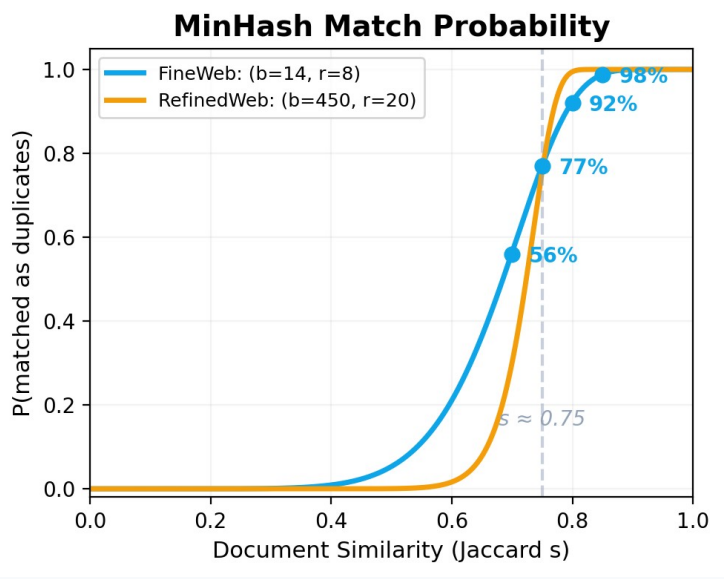
Result: 112-integer array
= 112 × 4 bytes = 448 bytes/doc

Pr[sig_h(A) = sig_h(B)] = Jaccard(A, B). Each hash is an independent estimate of similarity.

10KB document → 448 bytes signature. 15 billion documents × 448 bytes ≈ 6 TB of signatures.

MinHash Match Probability

P(candidate) = 1 − (1 − s^r)^b where s = Jaccard, r = rows/band, b = bands



FineWeb Data (Appendix E.1)

Jaccard s	P(match)
0.70	56%
0.75	**77%**
0.80	**92%**
0.85	**98.8%**

FineWeb: b=14, r=8 (112 hashes)
RefinedWeb: b=450, r=20 (9000 hashes)

Steeper curve = more precise threshold but requires more hashes (more compute + storage). FineWeb chose 112 hashes as a cost-quality tradeoff.

LSH Banding — MapReduce

Split signature into 14 bands of 8 hashes. Same band hash in any band → candidate pair.

MAP

```
Input: (doc_id, signature[112])

for band in 0..13:
  chunk = sig[band*8 : (band+1)*8]
  emit((band, hash(chunk)), doc_id)
```

Each document emits 14 key-value pairs.

`Output: ((band_idx, band_hash), doc_id)`

REDUCE

```
Key: (band_idx, band_hash)
Values: [doc_1, doc_2, ...]

if len(values) >= 2:
  for (di, dj) in pairs(values):
    emit(di, dj)  # candidate pair
```

`Output: candidate duplicate pairs`

Documents never directly compared → O(n) vs O(n²)

Transitive Closure

If A≈B and B≈C, all three are duplicates. Keep one per connected component.

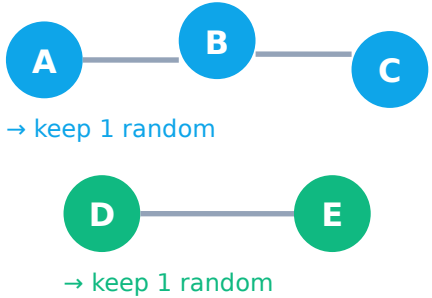
Union-Find

```
parent = {i: i for i in all_docs}

for (a, b) in candidate_pairs:
    union(a, b)

for component in components():
    keep random.choice(component)
    discard rest
```

Example



FineWeb: *"One (randomly chosen) document is kept per duplicate cluster while the remaining duplicates are removed."*

Candidate pairs go directly into Union-Find — no separate verification step. The (14, 8) banding is trusted as-is.

Line-Level Dedup

Document A (cooking blog)

```
Home | About | Recipes | Contact

Best Chocolate Cake Recipe
Preheat oven to 350°F. Mix flour,
cocoa powder, sugar, and eggs...

We use cookies to improve your
experience. Accept | Decline

Copyright © 2024 All Rights Reserved
```

Document B (fitness blog)

```
Home | About | Recipes | Contact

5 Morning Stretches for Back Pain
Start with a gentle cat-cow pose.
Hold each stretch for 30 seconds...

We use cookies to improve your
experience. Accept | Decline

Copyright © 2024 All Rights Reserved
```

Red lines appear in both documents (and thousands more). **Llama 3:** partition corpus into 30M-doc buckets, count exact line occurrences, remove lines appearing > 6 times.

Heuristic Filtering: The Gopher Table

Filter	Threshold
Duplicate line fraction	≤ 0.30
Duplicate paragraph fraction	≤ 0.30
Top 2-gram char fraction	≤ 0.20
Top 3-gram char fraction	≤ 0.18
Top 4-gram char fraction	≤ 0.16
Duplicate 5-gram char fraction	≤ 0.15
Duplicate 6-gram char fraction	≤ 0.14
Duplicate 7-gram char fraction	≤ 0.13
Duplicate 8-gram char fraction	≤ 0.12
Duplicate 9-gram char fraction	≤ 0.11
Duplicate 10-gram char fraction	≤ 0.10

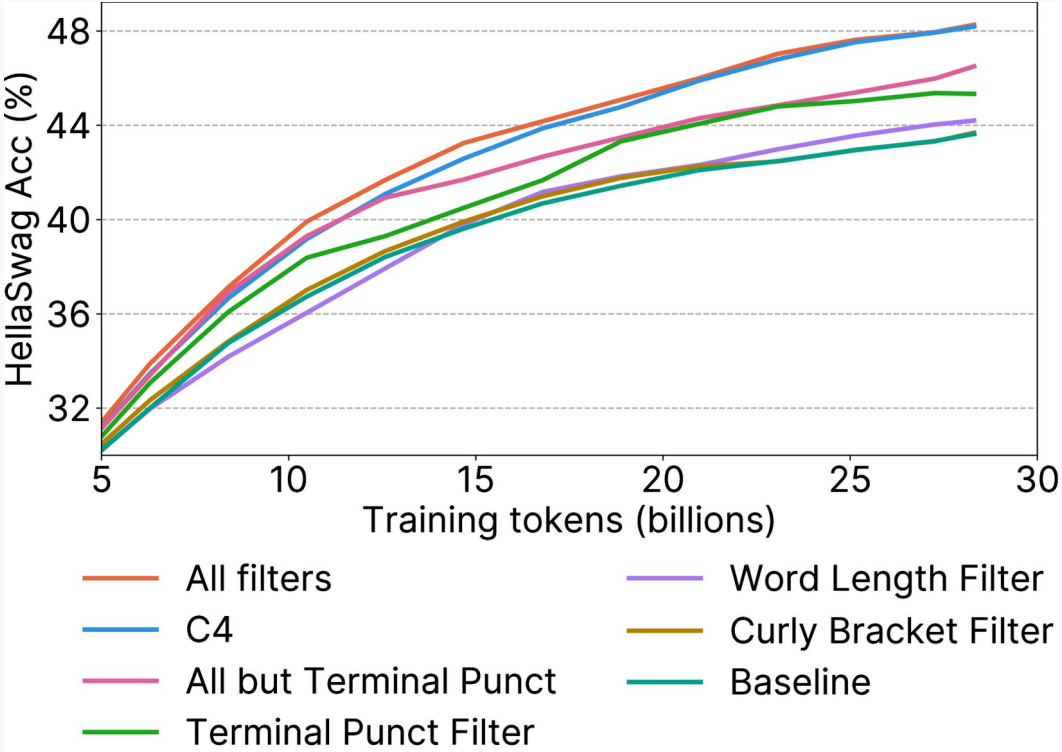
Document-level filters:

- Word count range
- Mean word length
- Non-alphabetic word fraction
- Lines ending with ellipsis
- "Stop word" presence check

All within-document checks. Thresholds get stricter for longer n-grams.

This table became the industry baseline — FineWeb uses it as their default config.

C4 Filter Ablation: What Actually Helps?



How to Develop New Heuristics

1 ▸ **2** ▸ **3** ▸ **4**

Get two datasets

High-quality (per-dump dedup) vs. low-quality (global dedup)

Compute statistics

Line lengths, punctuation ratios, dup line %, etc.

Plot histograms

Find where distributions diverge between the two sets

Set thresholds

Pick cutoffs at divergence points → new filter rules

Example: terminal punctuation filter

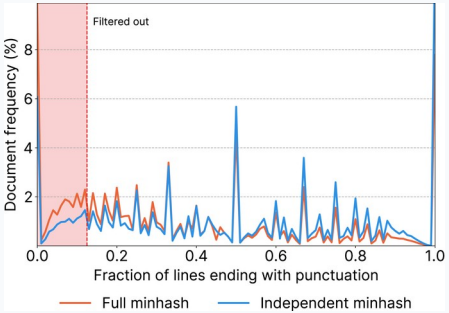
1. Observe: low-quality docs have few lines ending in .,!,?

▼

2. Measure: plot distributions, clear divergence at 0.12

▼

3. Rule: drop docs with < 12% terminal punctuation



Red zone = filtered out

Spike pattern in blue = near-dupes from independent dedup

Llama 3’s Heuristic Additions

Dup N-gram Coverage

Within-document repetition detector.
If a document has large spans of
repeated n-grams, it’s likely auto-generated or template content.

Dirty Word Counting

Count NSFW terms per document.
Block entire domains that exceed
threshold — catches adult sites
not yet in URL blocklists.

Token Distribution KL

Compare token distribution against
overall corpus. High KL divergence =
non-language gibberish: base64,
log files, hex dumps.

These complement the Gopher/C4 baseline. Together: comprehensive cheap quality checks.

Benchmark Decontamination

How do you prove your model didn't just memorize the test set?

The Problem
Training data = trillions of tokens.
Benchmarks = thousands of test questions.

If MMLU questions appear in Common Crawl, your "90%" might be **memorization**.

A **set intersection** problem at scale.

Detection Methods

1. **13-gram overlap** (GPT-3, Llama 2)
   Any 13-gram match = contaminated

2. **8-gram overlap** (Llama 3)
   Per Singh et al.: pick n empirically

3. **N-gram + LCS** (Qwen 2/2.5)
   LCS ≥ 13 tokens AND ≥ 60% coverage

TL;DR: Find contaminated documents in training data, remove them, and report what was found.

Llama 3’s Model Cascade

fastText

Language ID + topic

~1M params, CPU-only.
Filters non-English.
Also used for topic classification (tag documents by domain).

Runs on all ~15T

▶

DistilRoBERTa

Quality classifier

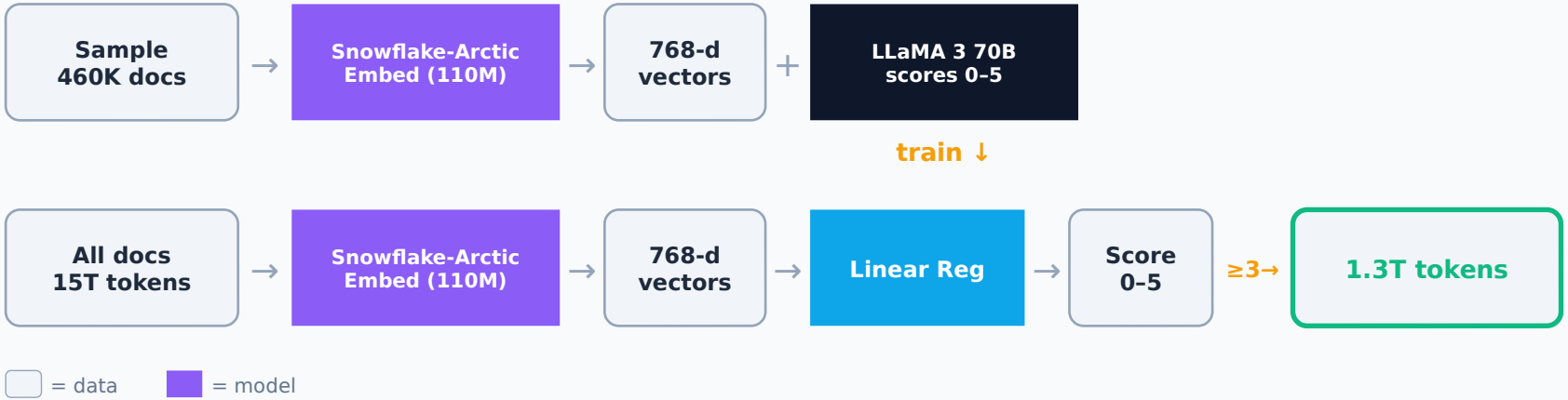
82M params.
Trained on Llama 2 quality judgments (binary: would this be cited on Wikipedia?).
Separate models for web, code, and math domains.

English subset

Pattern: expensive LLM labels small sample → distill to cheap classifier → apply at scale.

Model Filtering: Distillation at Scale

FineWeb-Edu: use an expensive LLM to train a cheap classifier, then apply at scale

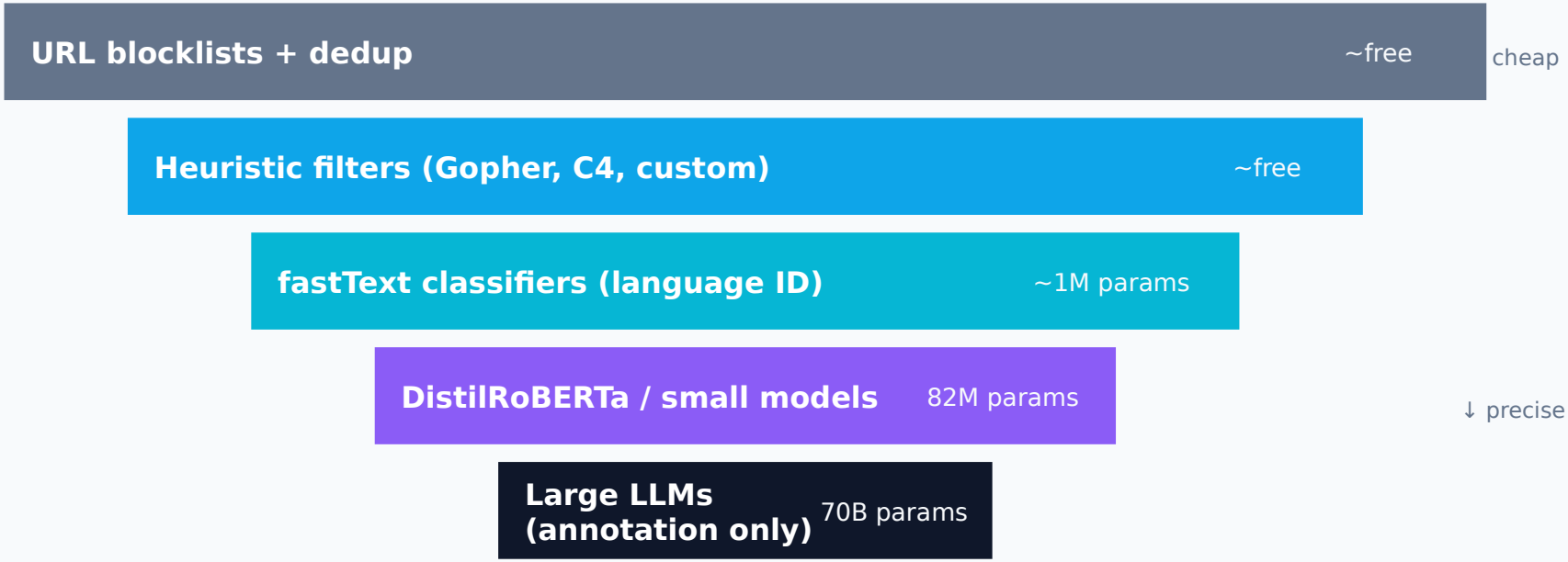


Inference cost: ~6,000 H100-hours to embed + score all 15T tokens.

FineWeb-Edu beats unfiltered FineWeb on educational benchmarks (MMLU, ARC).

Model-Based Filtering: The Escalation

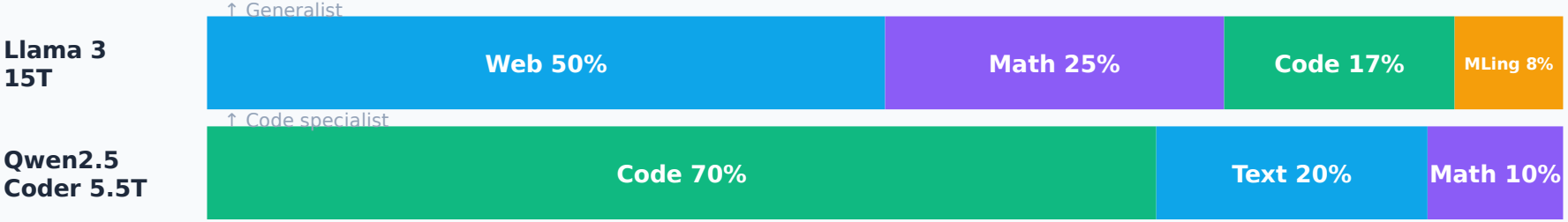
Cheap filters first, expensive models only on what survives



The funnel is a cost-optimization problem: minimize compute while maximizing quality signal.

Data Mix: Token Budget

Classify all documents into categories. Run scaling law experiments (small models, different mixes, measure downstream). Then scale up.



Model	Tokens	Key Mix Decisions
Llama 3	15T	50% web, 25% math, 17% code, 8% multilingual
Qwen3	36T	119 langs + domain-specific. Scaling law experiments.
DeepSeek	14.8T	Upsamples underrepresented domains
Qwen2.5-Coder	5.2T	70% code, 20% text, 10% math (specialist)

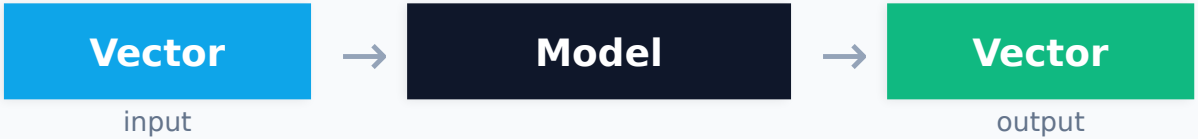
These ratios are found empirically — train small models with different mixes, measure, then scale up.

Why Tokenization?

Models operate on vectors, but we have text. How do we bridge the gap?

The Input Problem
Neural networks take **vectors** as input.
Text is a sequence of bytes (0–255).
Simplest idea: one-hot encode each byte.

Why Raw Bytes Don't Work
4K-word doc = 20K bytes. $O(n^2)$ attn.
Low info density: model must learn `h,e,l,l,o` together = "hello".



Solution: Group Bytes into Tokens
A **token** = a fixed mapping from a byte sequence to an integer ID.
A **vocabulary** = the complete set of all tokens.

"Hello world" byte-level: `11 tokens` | BPE (GPT-4): `2 tokens`

Key: The tokenizer is trained **before** the LLM. It determines how the model sees all text.

Tokenization: The Full Loop

One token at a time: encode, process, decode

Input (1 token at a time)

Text: `"Hello world"` → Token IDs: `[15496, 995]` → One-Hot: `[0,..,1,..,0]`

↓

Transformer Model

↓

Output (temp=0)

Detokenize: `vocab[318] = " is"` ← Sample: argmax → token `318` ← Probs: `[.01,..,.73,..]`

Note: GPT-2 style: spaces are prepended to words, so the token is " is" (with leading space), not "is ".

What Makes a Good Tokenizer?

We need a principled way to decide which byte sequences become tokens

The Optimization Goal
Frequent byte patterns → single tokens.
Rare patterns → smaller pieces.

Maximize information per token:
each position carries max semantic density.

Constraints
1. **Language-agnostic**
   English, Chinese, code, base64, math.
   No linguistic rules, pure statistics.
2. **Learnable from data**
   Needs a training dataset.

Pre-tokenization: Splitting Before Merging
Before learning merges, split text at boundaries using regex:

- Word boundaries, whitespace, punctuation
- Digits split: `2024` → `2 0 2 4`
- Contractions: `don't` → `don` + `'t`

Merges **never cross** pre-tokenization boundaries.

Byte Pair Encoding: The Algorithm

A greedy compression algorithm (Gage 1994, Sennrich et al. 2016)

BPE Training

1. **Pre-tokenize** — split corpus into chunks (words) using regex rules
2. **Initialize** — vocab = all unique bytes
3. **Count** — find most frequent adjacent pair **within a pre-token** across corpus
4. **Merge** — replace all occurrences with a new token. Vocab size increases by 1.
5. **Repeat** steps 3–4 until target vocab size

Worked Example

Corpus (pretokenized, _ = prefix space):

```
5× _ l o w          2× _ l o w e s t
6× _ n e w e r    3× _ w i d e r
```

Initial vocab (11):
{ _, d, e, i, l, n, o, r, s, t, w }

Merge 1: `(e, r)` — 9 times
→ new token **er** (vocab: 12)
from: newer×6, wider×3

Merge 2: `(_, l)` — 7 times
→ new token **_l** (vocab: 13)
word-initial "l" pattern

BPE: Full Merge Sequence

Watching the corpus transform as merges are applied (verified by script)

#	Pair	New Tok	Ct	All Tokens After Merge	Remaining Pretokens and Count
1	(e, r)	er	9	_ d e i l n o r s t w er	5× _ l o w 2× _ l o w e s t 6× _ n e w er 3× _ w i d er
2	(_, l)	_l	7	...+ _l	5× _l o w 2× _l o w e s t 6× _ n e w er 3× _ w i d er
3	(_l, o)	_lo	7	...+ _lo	5× _lo w 2× _lo w e s t 6× _ n e w er 3× _ w i d er
4	(_lo, w)	_low	7	...+ _low	5× _low 2× _low e s t 6× _ n e w er 3× _ w i d er
5	(_, n)	_n	6	...+ _n	5× _low 2× _low e s t 6× _n e w er 3× _ w i d er
6	(_n, e)	_ne	6	...+ _ne	5× _low 2× _low e s t 6× _ne w er 3× _ w i d er
7	(_ne, w)	_new	6	...+ _new	5× _low 2× _low e s t 6× _new er 3× _ w i d er
8	(_new, er)	_newer	6	...+ _newer	5× _low 2× _low e s t 6× _newer 3× _ w i d er

Key: Merge table + base vocab + regex = tokenizer.
To encode: apply merges in learned order.

Caveat: Captures patterns of the **training set**.
Tokenizer trained on English → poor Chinese tokens.

Tokenizers in Practice

How major LLMs configure their tokenizers

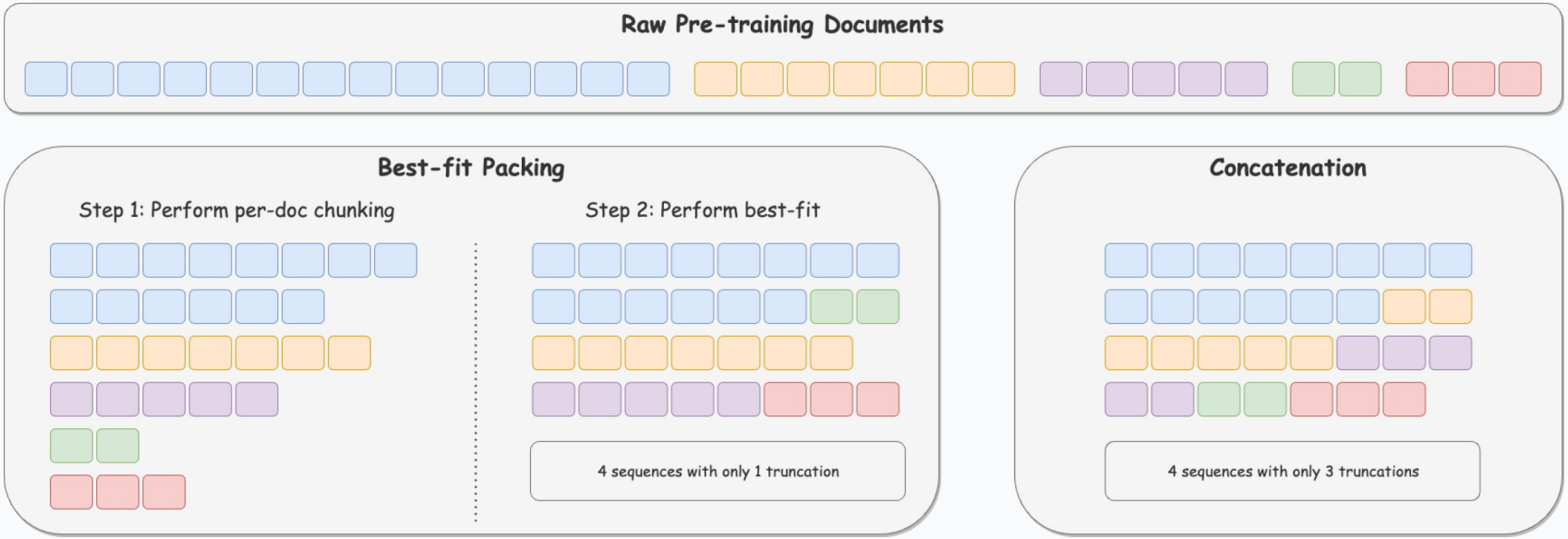
Model	Vocab Size	Training Data
GPT-2	50,257	WebText, 40GB
Llama 2	32,000	Undisclosed
Llama 3	128,256	Balanced multilingual
DeepSeek V3	128,000	Multilingual mix
Qwen 2.5	151,643	Multilingual mix
Gemma 2	256,000	Same as Gemini

Trend: Vocab sizes growing: 32K → 128K → 256K.
Larger vocab = shorter sequences = faster training.

All BPE: Every major LLM uses BPE or a BPE variant.
Key diffs: pre-tokenization, vocab size, training data.

Packing: Best-Fit Bin Packing

Documents vary in length. GPUs want uniform batches. Naive = pad everything (waste).



Source: "Fewer Truncations Improve Language Modeling" (2024)

DeepSeek uses best-fit bin packing to minimize padding waste across variable-length documents.

Fill-in-the-Middle (FIM) for Code

Standard next-token prediction:

```
AAAA    BBBB    CCCC
```

FIM rearrangement (10% of training tokens):

```
<fim_begin> AAAA <fim_hole> CCCC <fim_end> BBBB <eos>
```

▲ prefix ▲ suffix ▲ middle

Why? Teaches the model to fill in missing code given surrounding context.
Doesn't hurt standard next-token prediction. DeepSeek applies to ~10% of training tokens.

Annealing & Staged Training

Final phase: decay LR toward zero + switch to curated high-quality data.
Small precise adjustments without catastrophic forgetting.

Why staged training?

- Attention is $O(n^2)$: training on short context first is much cheaper
- Most general knowledge doesn’t need long context
- Long-context extension is cheap — small fraction of total training
- Data mix changes per stage: more STEM/code in later stages

What To Remember

1 Data quality > data quantity

Same compute, same model — but tokens
sampled from a filtered pool score higher.
Concentrating on educational content works.

2 Every step is a tradeoff

Dedup shifts mass toward the long tail.
Aggressive filtering = cleaner but less data.
Every decision needs ablation to validate.

3 This is engineering, not science

80K H100-hours of ablations for FineWeb.
No closed-form solution — iterate, measure,
and make judgment calls at every stage.

Also: Layer filters cheap → expensive (build a funnel) • Extraction quality is the foundation (WARC > WET)

The pipeline converges across labs because these tradeoffs have similar optima at scale.

Questions?

Ruiyang Wang

MTS, Anthropic

References

- FineWeb: Decanting the Web for the Finest Text Data (Penedo et al., 2024)
- The Llama 3 Herd of Models (Grattafiori et al., 2024)
- DeepSeek-V3 Technical Report (DeepSeek-AI, 2024)
- Qwen3 Technical Report (Qwen Team, 2025)
- Scaling Language Models: Methods & Insights from Training Gopher (Rae et al., 2021)
- Exploring the Limits of Transfer Learning [C4] (Raffel et al., 2019)

FineWeb dataset: huggingface.co/datasets/HuggingFaceFW/fineweb