Lecture 13: Data I — Sources, Copyright, and Pretraining Corpora

Language Modeling from Scratch · lecture notes · rebuilt for NVIDIA Blackwell, late 2026

Section 1: Why Data Differentiates Language Models and the Volume–Quality Pipeline

Every earlier part of building a language model assumes that a clean stream of training tokens already exists on disk. Once architectures, optimizers, and distributed training recipes become broadly understood and widely published, a much harder engineering problem emerges: what data should a model actually train on, and how do you construct a dataset large enough and clean enough to train a competitive model?

A striking pattern across open-weight model releases exposes where the true technical moat lies. Consider the historical open-weight release of Llama 3 (July 2024): its technical report published the complete neural network architecture, hyperparameter choices, and training recipe in full detail, yet disclosed almost nothing about the exact training data sources, filtering criteria, or domain mixture ratios. This disclosure asymmetry is not accidental. Frontier laboratories keep their data pipelines secret for two concrete reasons:

1. **Competitive dynamics**: Data curation pipelines directly govern downstream model quality. Two models with identical architectures and compute budgets can diverge sharply in capability depending on how their training corpora were extracted, cleaned, deduplicated, and mixed.
2. **Copyright liability**: Publicly disclosing every source ingested into a training corpus exposes the organization to copyright litigation, a legal constraint we will analyze in full in Section 3.

The nature of data engineering has also transformed fundamentally rather than shrinking. In the pre-foundation era of supervised machine learning, data work meant hiring human annotators to label individual examples for narrow tasks. In the foundation-model era, manual per-example labeling plays a much smaller role at the base level; instead, data engineering has exploded into massive-scale **curation, extraction, filtering, and deduplication**. Unlike a neural network architecture—which can be written down precisely in a handful of linear-algebraic equations—data is a **long-tail problem** that scales directly with sustained human engineering effort. Every weird formatting artifact, spam farm, broken character encoding, or boilerplate header on the web represents another edge case that silently degrades the training distribution unless someone inspects the data and engineers a remedy.

Because raw text on the internet is never ready for gradient descent, every training corpus is built by pushing raw sources through a five-stage engineering pipeline. Visualize this pipeline left to right as five connected boxes linked by forward arrows:

- **Stage 1 — Live Services** (e.g., the public World Wide Web, GitHub, arXiv): dynamic, continuously changing servers where human-authored text and code actually reside.

- **Stage 2 — Raw Snapshots** (e.g., WARC web archives, XML dumps, `git` repositories): static, offline copies captured at a point in time so processing is reproducible and does not hammer live infrastructure.
- **Stage 3 — Extraction** (HTML or PDF converted to plain text): stripping markup, navigation bars, page headers, and binary wrappers to recover the underlying prose or source code.
- **Stage 4 — Filter & Deduplicate** (heuristic rules and learned classifiers): discarding boilerplate, gibberish, unwanted languages, toxic spam, and repeated copies of identical or near-identical text.
- **Stage 5 — Data Mixture** (Pre-training, Mid-training, and Post-training blends): combining cleaned domain sources in carefully tuned proportions matched to each stage of model training.

Stages 1 and 2 govern access, crawling, and copyright; Stages 3 through 5 govern major source curation and the evolution of modern datasets, bridging into the algorithmic filtering and deduplication methods covered in the next lecture (Data II).

Once cleaned corpora exist, modern training does not feed all data into the model uniformly from step zero to the end. Instead, training follows a **volume–quality funnel**. Imagine an inverted trapezoid divided into three stacked horizontal bands narrowing from top to bottom, accompanied by an axis pointing right labeled *Less volume · Higher quality & specificity*:

- **Band 1 (widest, top) — Pre-training**: Massive volume of broad raw web pages and general text, spanning roughly $15\mathrm{T}$ to $45\mathrm{T}$ tokens in recent and late-2026 systems. This stage builds broad world knowledge and linguistic fluency.
- **Band 2 (middle, narrower) — Mid-training**: Curated, higher-quality text heavily enriched with mathematics, code, and structured reasoning material. An open, fully documented example is the **Dolmino** mid-training mix used in AI2's OLMo 2 (2025).
- **Band 3 (narrowest, bottom) — Post-training**: Targeted conversational, reinforcement learning, and distillation data designed to shape instruction following and reasoning behavior. Examples include AI2's **Tülu** post-training suite, **RLVR** (Reinforcement Learning with Verifiable Rewards), and **MOPD** (Multi-Teacher On-Policy Distillation).

> **DEFINITION**
> A **base model** is a language model that has completed pre-training and mid-training, learning broad next-token prediction over massive text and curated domain corpora without yet being tuned for conversational instruction following. An **instruct (or chat) model** is produced by applying post-training to a base model so that it responds directly to user prompts and tasks.

Although stage boundaries are blurry in practice—some labs blend mid-training smoothly into the late annealing phase of pre-training—every modern recipe moves along this funnel from broad, noisy volume to narrow, high-specificity supervision. Increasingly, laboratories release only the final instruct model rather than the underlying base checkpoint; for instance, the early-2026 model Qwen3.5-397B-A17B was released solely as an instruct model. By contrast, AI2's OLMo 2 (2025) remains a rare open full-pipeline reference that documents pre-training, mid-training (Dolmino), and post-training (Tülu) end to end. At the late-2026 frontier, post-training itself has grown into multi-step pipelines: DeepSeek-V4.1-Flash (Sept

2026) trains via supervised fine-tuning (SFT), followed by asynchronous reinforcement learning, followed by on-policy distillation from more than 40 teacher models; Nemotron 3.5 (Oct 2026) progresses through SFT, RLVR, expert teacher models, and MOPD.

> **COMMON MISCONCEPTION**
> You might think that because post-training and mid-training determine a model's chat polish and benchmark-specific reasoning, pre-training volume no longer matters as long as you have a small, pristine dataset. That breaks because pre-training token budgets have grown relentlessly alongside model capability: without tens of trillions of broad tokens in Stage 1, the model lacks the factual breadth, syntactic robustness, and cross-domain representations that mid-training and post-training merely refine and elicit.

To see how dramatically the scale of Stage 1 has shifted, inspect the horizontal bar chart of pre-training token budgets from 2020 through late 2026. The horizontal axis spans from 0T to 50T tokens in 10T grid increments, and five horizontal bars are stacked vertically from top (oldest) to bottom (newest):

- **GPT-3 (2020, historical)**: 0.4T (400B) tokens — a tiny sliver barely extending past the 0T axis.
- **LLaMA (2023, historical)**: 1.2T tokens — still less than one-eighth of the first 10T grid interval.
- **Llama 3 (July 2024, historical)**: 15T tokens — jumping past the 10T line halfway to 20T.
- **Qwen3 (May 2025, earlier)**: 36T tokens — extending more than halfway between 30T and 40T.
- **DeepSeek-V4.1-Flash (Sept 2026, current frontier)**: 45T tokens — reaching halfway between 40T and 50T.

We quantify the multiplicative growth in pre-training token budget between two eras using the simple budget ratio:

$$R = \frac{T_{\text{target}}}{T_{\text{baseline}}}$$

Here, T_{baseline} denotes the total pre-training token count of an earlier reference model, T_{target} denotes the total pre-training token count of a later model, and R gives the dimensionless scaling factor by which the corpus had to expand. As a single conceptual unit, R tells us how many times larger the data supply chain must become to feed modern training runs without endlessly repeating identical text.

> **WORKED EXAMPLE**
> Compute the pre-training token scaling factor R from the historical baseline GPT-3 (2020) to the late-2026 frontier model DeepSeek-V4.1-Flash (Sept 2026), and compare it to the jump from Llama 3 (July 2024) to DeepSeek-V4.1-Flash.
>
> 1. Identify the token counts from the bar chart:
>   - $T_{\text{GPT-3}} = 0.4\text{T}$ tokens
>   - $T_{\text{Llama 3}} = 15\text{T}$ tokens
>   - $T_{\text{DeepSeek-V4.1-Flash}} = 45\text{T}$ tokens

2. Compute the six-year scaling factor (2020 to 2026):

$$R_{2020\to2026} = \frac{45\text{T}}{0.4\text{T}} = 112.5\times$$

3. Compute the two-year scaling factor (2024 to 2026):

$$R_{2024\to2026} = \frac{45\text{T}}{15\text{T}} = 3.0\times$$

Interpretation: In six years, pre-training data budgets expanded by more than two orders of magnitude ($112.5\times$), and even in the two years between Llama 3 and late 2026, token demand tripled. Curated human-edited sources like Wikipedia or digitized public-domain books amount to mere tens of billions of tokens—orders of magnitude too small to fill a 15T to 45T token budget. Web crawl extraction, automated quality filtering, and synthetic data generation are therefore not optional enhancements; they are mathematically mandatory to reach modern scale.

KEY IDEA

Data engineering—not just model architecture—is the primary differentiator of modern language models, because training progresses through a volume–quality funnel whose pre-training stage now demands 15T to 45T tokens (over $100\times$ the GPT-3 budget), far exceeding what small hand-curated sources can supply.

Because hand-curated encyclopedias and books cannot come close to supplying tens of trillions of tokens, engineers are forced to gather text from the open web. That immediately raises a practical systems problem: the web is not a static file sitting on a hard drive waiting to be copied. Understanding what you can—and cannot—actually collect from live web servers is our next step.

✎ Check Your Understanding

1. **Quick computation**: If a historical model trained on 1.2T tokens (LLaMA, 2023) and a late-2026 frontier model trains on 45T tokens (DeepSeek-V4.1-Flash, Sept 2026), by what factor did the pre-training token budget increase?
   Answer: It increased by $\frac{45\text{T}}{1.2\text{T}} = 37.5\times$, using the token budget ratio $R = T_{\text{target}}/T_{\text{baseline}}$.
2. **Yes/No with justification**: Are competitive secrecy and copyright liability the only reasons open-weight models like Llama 3 (2024) withhold their data mixtures, or is it because data pipelines are trivial to describe in a few equations like model architectures?
   Answer: No, data pipelines are not trivial to specify in a few equations; data is a long-tail problem that scales with human engineering effort, and labs withhold their curation mixtures specifically due to competitive dynamics (curation governs model quality) and copyright liability (disclosing sources invites litigation).

Section 2: From Live Web Services to Static Snapshots — Crawling, Access Barriers, and Shadow Libraries

Popular descriptions routinely claim that large language models are “trained on the entire internet.” That statement is factually false on two levels. First, the “internet” includes private emails, encrypted messaging, internal corporate networks, and non-web protocols; language models only ever touch a filtered slice of the **public World Wide Web**. Second, even the public web is not a single downloadable archive. It is a distributed network of **live servers** that generate responses on demand whenever a client sends an HTTP request.

To see the gap between a live web service and a training token, consider what happens when you issue a single command-line request to a course webpage:

```
$ curl https://cs336.stanford.edu/
<!doctype html><html>...<h1>CS336: Language Modeling from Scratch</h1>...</html>
```

The server returns an HTML document wrapped in markup tags, not clean prose, and it only returns the single page you explicitly asked for. You cannot run a training loop directly against live HTTP servers—network latency, server crashes, rate limits, and changing content would make training slow, irreproducible, and destructive to the host sites. Instead, Stage 1 to Stage 2 of our pipeline requires a **web crawler** to convert live servers into static archive snapshots for offline processing.

> **DEFINITION**
> A **web crawler** is an automated program that begins from a **seed set** of initial URLs, downloads each corresponding webpage, parses outgoing hyperlinks embedded in the downloaded HTML, enqueues newly discovered URLs to visit next, and stores the fetched responses as static archive snapshots for offline processing.

Even with a well-engineered crawler, large fractions of human-written text on the web remain completely unreachable before any copyright lawyer even enters the room. Three distinct technical barriers wall off huge portions of the web:

1. **Dynamic web applications**: Many modern platforms do not serve static HTML documents tied to unique URLs. Content loads interactively via JavaScript only when a user clicks buttons, scrolls infinitely, or submits forms, often without the URL changing at all. Platforms such as Discord and Weights & Biases hold vast technical discussions and logs that a standard HTTP crawler never sees.
2. **Authentication and paywalls**: Enormous volumes of contemporary journalism, social discourse, and professional writing sit behind user accounts and paid subscriptions. Examples include Facebook, X, LinkedIn, and *The New York Times*. A crawler without credentials hits a login screen or a paywall banner instead of the article text.
3. **Technical bot defenses**: Site operators actively control or block automated access using multiple layers:

- **`robots.txt` exclusion rules**: A plain-text file placed at the root of a website specifying which automated user agents are permitted or forbidden from crawling which paths. For example, a site operator can explicitly exclude OpenAI's crawler and Common Crawl's crawler with the directives:

  ```
  # Example robots.txt directives
  User-agent: GPTBot
  Disallow: /

  User-agent: CCBot
  Disallow: /
  ```

  Crucially, `robots.txt` is a **voluntary protocol**—a polite crawler checks and obeys it, whereas an adversarial scraper can technically ignore it.

- **Active bot detection and CAPTCHAs**: Infrastructure providers such as Cloudflare inspect request fingerprints, throttle suspicious traffic, and serve interactive challenges designed to block automated clients.
- **IP/country blocks and strict rate limits**: Servers reject requests originating from cloud data-center IP ranges or cap requests per second per client.

> **COMMON MISCONCEPTION**
> You might think that if a webpage loads publicly in your browser without a login screen and has no `robots.txt` block, you are automatically free to scrape and train a model on it. That breaks for two independent reasons: first, a site's **Terms of Service (ToS)** frequently prohibit automated scraping or bots as a binding contractual condition; second, the technical ability to download a public webpage does not grant a legal **license** to copy that text into a training corpus.

Over the past several years, publishers have reacted aggressively to large-scale AI scraping, producing a documented **decline of consent**. A 2024 empirical audit examined `robots.txt` directives and Terms of Service restrictions for the exact URLs contained inside three major open pre-training datasets—C4, RefinedWeb, and Dolma—and tracked how those permissions changed over time. The audit found that website opt-outs and explicit AI-crawler bans have risen steadily, shrinking the fraction of high-value web domains willing to allow automated collection.

Ignoring these boundaries is not merely a legal risk; it causes immediate operational damage. Poorly behaved crawlers that ignore `robots.txt` and bypass rate limits hammer origin servers with floods of automated traffic, degrading live services for real human users and imposing substantial bandwidth and compute costs directly on site operators.

> **WORKED EXAMPLE**
> Suppose a small independent technical blog hosts 50,000 articles and places `User-agent: CCBot` / `Disallow: /` in its `robots.txt`, along with a server capacity of 20 requests per second before page load times spike for readers. Compare two crawlers attempting to index the site:

1. **Polite crawler**: Fetches `https://blog.example/robots.txt` first, sees `Disallow: /` for its user agent, and skips all 50,000 URLs (0 article requests sent). Even if crawling were allowed, it enforces a rate limit (e.g., 1 request every 2 seconds), taking $100{,}000\text{ s} \approx 27.8\text{ hours}$ while consuming only 2.5% of server capacity.
2. **Aggressive, non-compliant scraper**: Ignores `robots.txt` and fires 100 concurrent requests per second across distributed IPs to finish in 500 s (8.3 minutes). Because $100 > 20$ requests/second, the blog's origin server saturates, legitimate human visitors experience timeouts, and the operator's cloud egress bill spikes—prompting the operator to deploy Cloudflare bot blocking and IP bans that permanently cut off future researchers as well.

Interpretation: Crawler etiquette is both an operational engineering requirement and a commons-preservation problem; aggressive scraping accelerates the exact technical and contractual lock-down documented in the 2024 consent audit.

At the opposite extreme from polite web crawling sit **shadow libraries**—repositories that are technically reachable on the public web, yet explicitly operate outside copyright law and publisher paywalls. Prominent examples include **Library Genesis (LibGen)**, **Z-Library**, **Anna's Archive**, and **Sci-Hub**. Their scale is immense: literature estimates cite roughly 4M books in LibGen (as of 2019) and approximately 88M academic papers in Sci-Hub (as of 2022).

DEFINITION

A **shadow library** is a publicly accessible online archive that aggregates and distributes copyrighted books, journals, or academic articles in circumvention of commercial paywalls and copyright restrictions.

Shadow libraries exist in a continuous cat-and-mouse struggle between enforcement and circumvention. Because they bypass commercial academic publishers such as Elsevier, they are routinely targeted by takedown orders, civil lawsuits, and national DNS or ISP blocks; yet decentralized mirrors quickly reappear across different legal jurisdictions. While an ongoing ethical debate surrounds them—advocates argue that publicly funded scientific knowledge should be freely accessible to everyone—their legal status is unambiguous: **downloading works from shadow libraries constitutes piracy and copyright infringement**. We will need this exact distinction between *how a copy was acquired* and *how a model trained on it* when we examine landmark 2025 copyright rulings in Section 3.

KEY IDEA

Language models are never trained on "the entire internet"; dynamic JavaScript apps, paywalls, bot defenses, Terms of Service, and rising `robots.txt` opt-outs hide vast regions of human text from web crawlers, while bypassing paywalls through shadow libraries crosses directly into illegal piracy.

We have now seen that between a live webpage and a training token stand technical walls, voluntary protocols, contractual Terms of Service, and the shadow-library divide. Even when a crawler obeys `robots.txt` and downloads an openly reachable webpage, a deeper question immediately confronts

every dataset builder: who owns the words on that page, and under what legal authority can you copy them into a training dataset? That brings us directly to copyright law.

> **✎ Check Your Understanding**
>
> 1. **Fill-in-the-blank**: A web crawler starts from a _____ of URLs, downloads each page, extracts outgoing hyperlinks into a queue, and checks a site's _____ file—a voluntary protocol—to see which paths automated user agents are asked to avoid.
> **Answer:** **seed set**; `robots.txt`.
> 2. **What would happen if…**: What happens to a standard HTTP crawler that tries to harvest discussions from an interactive web application like Discord or articles behind *The New York Times* paywall?
> **Answer:** On Discord, content loads dynamically via JavaScript interactions without static URLs, so the crawler cannot retrieve it; on *The New York Times*, authentication and paywall barriers block access, returning a login or paywall wall instead of the article text.

Section 3: Copyright Law, Licensing, and Fair Use for Language Model Training

Suppose a polite crawler respects `robots.txt`, stays within rate limits, and saves millions of publicly accessible web pages onto disk. Before a single gradient step is computed, every dataset builder faces a legal reality: **intellectual property (IP)** law governs who may copy and reuse creative work. IP law exists to incentivize the creation of intellectual goods and spans four distinct legal regimes: **copyright** (creative expression), **patents** (inventions), **trademarks** (brand identifiers), and **trade secrets** (confidential business know-how). For text, code, images, and audio corpora, copyright is the controlling regime.

Modern copyright traces back to the Statute of Anne (1709, England) and is codified in the United States by the **Copyright Act of 1976**. Under this statute, copyright protects *"original works of authorship fixed in any tangible medium of expression."* Notice how drastically the 1976 Act broadened protection compared to the earlier 1909 Act, which applied only to *published* works. Fixing words onto a hard drive, a blog post, a forum reply, or a source-code file is sufficient.

Three structural rules define what copyright covers and how long it lasts:

1. **Expression, not ideas**: Copyright protects the specific creative *expression* of an idea, never the underlying *idea* itself. For example, a particular implementation of the quicksort sorting routine written in a specific way is copyrighted expression, whereas the quicksort algorithm itself—partitioning around a pivot and recursing—is an unprotected idea. Similarly, bare factual collections such as an alphabetical telephone directory are unprotected unless the selection or arrangement itself exhibits creativity.

2. **Automatic creation with a tiny threshold**: Unlike patents—which require an expensive application and examination—copyright attaches automatically the instant an original work is fixed in a tangible medium. No notice or registration is required to hold a copyright; your personal website, class notes, and public forum comments are all automatically copyrighted. Registration (which costs $65 in the US) is only required as a procedural prerequisite before the owner can file an infringement lawsuit in court.
3. **Long duration before entering the public domain**: For an individual author, copyright lasts for the **author's life plus 70 years**. For corporate **works made for hire**, it lasts **95 years from publication**. Only after that term expires does a work enter the **public domain**, where anyone may freely copy and use it—examples include the plays of Shakespeare, the compositions of Beethoven, and most books in Project Gutenberg.

The practical consequence for data engineering is stark: **basically everything on the Internet is copyrighted.**

> **COMMON MISCONCEPTION**
> You might think that because a blog post, Reddit comment, or snippet on a personal website has no copyright symbol (©), was never formally registered, and is readable for free online, it belongs to the public domain. That breaks because under the US Copyright Act of 1976, copyright attaches automatically upon fixation with a minimal creativity threshold; lack of registration only delays a lawsuit (which requires a $65 registration), and public readability never surrenders copyright.

Given that nearly all web text is copyrighted, there are only **two legal paths** to use a copyrighted work in a training dataset: **(1) obtain a license**, or **(2) rely on fair use**.

Path 1: Licenses and the Terms of Service Trap

> **DEFINITION**
> A **license** is an instrument of contract law through which a copyright holder (the **licensor**) grants explicit permission to another party (the **licensee**) to use a work under specified conditions—effectively functioning as "a promise not to sue" so long as the licensee obeys those terms.

Licenses come in two major flavors relevant to pre-training corpora:

- **Open public licenses (Creative Commons)**: Created by Lawrence Lessig and Eric Eldred in 2001 to bridge the rigid gap between full all-rights-reserved copyright and the public domain. Creative Commons licenses underpin massive open knowledge repositories, including Wikipedia, MIT OpenCourseWare, Khan Academy, and the Free Music Archive, as well as roughly 307M Flickr images, 39M MusicBrainz items, and 10M YouTube videos.
- **Commercial training deals**: Bilateral contracts negotiated directly between AI laboratories and major content platforms. Documented examples include licensing agreements between **Google and Reddit**, **OpenAI and Shutterstock**, and **OpenAI and StackExchange**.

However, holding a copyright license to an item does not override platform contracts. This creates the **Terms of Service (ToS) trap**: a website's ToS can impose contractual bans on downloading or scraping *even over content that is openly licensed or would qualify for fair use*. For example, YouTube hosts 10M Creative Commons-licensed videos, yet YouTube's Terms of Service explicitly prohibit automated downloading of videos from the platform. Violating the ToS breaches a contract with the host platform independently of whether the underlying author granted a Creative Commons license.

Path 2: Fair Use (17 U.S.C. § 107)

Because negotiating individual licenses with millions of web authors is practically impossible at multi-trillion-token scale, most web-scale language model training has relied on the statutory doctrine of **fair use**, codified in **17 U.S.C. § 107**.

> **DEFINITION**
> **Fair use** (17 U.S.C. § 107) is a legal doctrine in US copyright law that permits limited unlicensed use of copyrighted material based on a holistic balancing of four statutory factors: purpose and character of the use, nature of the copyrighted work, amount and substantiality used, and effect on the market for the original work.

Courts do not apply fair use as a rigid checklist; they weigh all four factors together:

Statutory Factor	What Favors Fair Use
1. Purpose & character of the use	Educational or non-commercial over commercial; **transformative** (creating a new purpose, meaning, or message) over merely reproductive
2. Nature of the copyrighted work	Factual / non-creative works over highly fictional / creative works
3. Amount & substantiality used	Copying a small, non-essential snippet over copying the entire work or its creative heart
4. Market effect	The new use does not harm the existing or potential market for the original work

Classic examples illustrate how courts balance these four factors. Watching a commercial movie and writing a critical summary of it is transformative and does not substitute for buying a ticket. Studying copyrighted source code to understand an algorithm and then cleanly reimplementing the algorithm from scratch uses only the unprotected idea rather than the copyrighted expression. Most relevant to large-scale data ingestion is the landmark **Google Books** case (*Authors Guild v. Google* — filed 2005, decided 2013, and affirmed on appeal in 2015): Google scanned millions of entire copyrighted library books (Factor 3 weighed heavily against it on volume alone), yet courts ruled the project fair use because building a full-text search index that displayed only short snippets was highly transformative (Factor 1) and did not act as a market substitute for purchasing the books (Factor 4).

Applying fair use to language model training exposes four deep legal and technical tensions. First, remember that **copyright governs semantics and economics, not merely verbatim byte-for-byte copying**: distinctive fictional plots and characters (such as *Harry Potter*) can be protected even without exact word-for-word quotations, whereas parody—imitating a work specifically to critique or mock it—is often protected as fair use. Against that backdrop, training a language model creates four competing pulls across the statutory factors:

1. **Copying**: Simply downloading and storing web pages or books onto training clusters (Stage 2 of our pipeline) already creates a complete reproduction of the original works, putting immediate pressure on Factor 3.
2. **Transformativeness**: Training does not aim to redistribute copies of the dataset; it optimizes parameters to learn general statistical patterns of language—far closer to transformative indexing than copy-pasting (supporting Factor 1).
3. **Idea vs. expression**: Ideally, a language model extracts general concepts and linguistic structure (learning what *wizards* are in fantasy literature) rather than reproducing specific protected expression (generating *Harry Potter* chapters). Memorization blurs that boundary.
4. **Market impact**: Even if a model never emits a single verbatim sentence from a training book or news article, a highly capable language model can compete economically with human writers, journalists, and artists, placing acute pressure on Factor 4.

> **WORKED EXAMPLE**
> Evaluate a hypothetical scenario through the four statutory factors of 17 U.S.C. § 107: an AI lab downloads 100,000 contemporary novels from a shadow library, trains a commercial fiction-writing language model on the full text of every novel, and deploys a paid service that generates new novels in the same genres.
>
> 1. **Factor 1 (Purpose & character)**: The use is commercial, which leans against fair use, though learning statistical language weights is arguably transformative—*except* that acquiring pirated copies in bad faith undermines the character of the use.
> 2. **Factor 2 (Nature of the work)**: Contemporary novels are highly creative fictional works, sitting at the core of copyright protection (weighs against fair use).
> 3. **Factor 3 (Amount & substantiality)**: The lab copied 100% of every novel to ingest it into the training pipeline (weighs against fair use unless justified as necessary for a transformative purpose).
> 4. **Factor 4 (Market effect)**: A commercial fiction generator directly competes in the market for contemporary novels and also deprives publishers of potential licensing revenue for training copies (weighs against fair use).
>
> **Interpretation**: Even if a court accepts that statistical training itself is transformative under Factor 1, the combination of pirated acquisition, highly creative source works, full-work copying, and direct market competition creates severe legal liability.

Exactly these tensions have played out in three landmark AI copyright lawsuits between 2023 and 2025:

Case	Allegation	Key Rulings & Outcomes
NYT v. OpenAI (2023)	Training on and reproducing *New York Times* articles	Active litigation contesting both the unauthorized training copies and memorized or near-verbatim model outputs
Bartz et al. v. Anthropic (2024 → 2025)	Pirating millions of books from shadow libraries and training on plaintiffs' books	Summary judgment (2025) drew a sharp line: **training itself = fair use**, and **buying and physically scanning books = fair use**, but **pirating copies = NOT fair use**. Outcome: Anthropic paid $1.5B to settle with authors.
Kadrey et al. v. Meta (Summary judgment 2025)	Training on plaintiffs' books (sources revealed in the original LLaMA paper)	Summary judgment (2025): **training on books (in this instance) = fair use**; however, the allegation regarding peer-to-peer **torrenting** of pirated copies remained pending.

Look closely at the pattern across *Bartz v. Anthropic* and *Kadrey v. Meta*. Courts separated the **statistical transformation of training** from the **method of data acquisition**. While specific summary-judgment rulings in 2025 found that training itself qualified as fair use in those particular records, the legal question is not yet settled broadly across all jurisdictions or all market-harm arguments (as ongoing litigation like *NYT v. OpenAI* demonstrates). By contrast, one rule is already crystal clear and extraordinarily expensive to violate: **pirating source copies via shadow libraries or torrents is illegal copyright infringement**, even if the subsequent model training is transformative. We will return to the practical question of whether one can bypass fair-use uncertainty altogether by training solely on permissively licensed data in Section 10.

> **KEY IDEA**
> Under the US Copyright Act of 1976, virtually everything on the web is automatically copyrighted, leaving only licensing or four-factor fair use (17 U.S.C. § 107) as legal paths; landmark 2025 rulings indicate that while training itself can qualify as fair use in specific cases, acquiring pirated copies from shadow libraries is unambiguously illegal.

With the access barriers and legal ground rules established, we can now examine the actual raw data sources that feed modern pre-training corpora. Because web crawls supply the bulk of the 15T to 45T tokens required today, we start with the foundational open web archive: Common Crawl, and the surprisingly decisive engineering problem of turning raw HTML into clean text.

✎ Check Your Understanding

1. **Yes/No with justification**: In *Bartz et al. v. Anthropic* (2025), did the summary judgment ruling that model training was fair use shield Anthropic from liability for using pirated book copies?
   Answer: No. The court ruled that training itself and buying-and-scanning books were fair use, but pirating copies was explicitly *not* fair use, and Anthropic paid $\$1.5\text{B}$ to settle with authors.
2. **What would happen if…**: You write a script to download Creative Commons-licensed videos directly from YouTube to transcribe them for a dataset. Which legal barrier do you still violate even though the video creators attached an open license?
   Answer: You violate YouTube's **Terms of Service (ToS)**—a binding contract with the hosting platform that prohibits downloading videos regardless of whether the underlying content is CC-licensed.

Section 4: Common Crawl and the Engineering of HTML-to-Text Extraction

If every AI research group had to run its own global web crawler from scratch, replicating the infrastructure to fetch billions of pages politely every month would consume massive engineering budgets before any language modeling research even began. That bottleneck motivated **Common Crawl**, a non-profit organization founded in 2007 that crawls the public web on a roughly monthly cadence and publishes raw archive snapshots openly for anyone to process.

Common Crawl's scale makes it the default raw backbone of nearly every modern open web corpus. Each monthly crawl adds roughly 3B **to** 5B **pages**, intentionally incorporating some overlap with prior crawls while diversifying across new domains. Across its lifetime, Common Crawl has archived roughly 300B **pages**—for context, the live web contains $O(\text{billions})$ of reachable URLs, while Google's commercial search index exceeds $100\ \text{PB}$. To ground this in a concrete recent snapshot, the **April 2026** Common Crawl release alone captured 2.19B **pages totaling** $372.2\ \text{TB}$.

Common Crawl gathers these snapshots using **Apache Nutch**, an open-source distributed crawler architecture. Visualize the Apache Nutch crawl loop as four stacked rectangular modules connected vertically by downward arrows, with a feedback loop running up the left margin from the bottom module back into the top module:

1. **URL Frontier / Queue (top)**: Initialized with a massive seed set of hundreds of millions of URLs waiting to be visited.
2. **Fetcher — Politeness (second)**: Pulls URLs from the frontier and fetches pages over HTTP while enforcing `robots.txt` exclusions and per-host rate limits so origin servers are not overloaded.
3. **Archive — WARC / WET (third)**: Writes each downloaded response permanently to disk in two formats: raw HTTP responses (WARC) and pre-extracted plain text (WET).

4. **Parse & Select Links (bottom)**: Extracts outgoing hyperlinks from the downloaded HTML, applies **selection** and **re-visit** policies to decide which pages matter and how frequently changing sites should be refreshed, and routes newly selected URLs along the left feedback arrow labeled *enqueue new URLs* back into the URL Frontier.

Running this loop reliably at web scale hinges on three core crawler policies—**selection** (which pages among billions are worth fetching?), **politeness** (obeying `robots.txt` and pacing requests per host), and **re-visit** (how often should dynamic news pages versus static archives be re-fetched?)—plus a subtle web-engineering hazard: **dynamic URLs** (for instance, session IDs, tracking parameters, or calendar widgets) can map thousands of syntactically distinct URLs to identical underlying page content, trapping naive crawlers in redundant loops and inflating duplicate volume.

Once Common Crawl writes a crawl snapshot to storage, Stage 3 of our pipeline—**HTML-to-text extraction**—begins. Common Crawl publishes every crawl in two primary file formats, and choosing between them has an outsized effect on final language model accuracy.

> **DEFINITION**
> **WARC** (Web ARChive) files store the complete raw HTTP response and full HTML Document Object Model (DOM) structure exactly as returned by the web server. **WET** (WARC Encapsulated Text) files store a pre-converted, plain-text version produced by Common Crawl's generic text stripper, discarding the underlying HTML structure.

Trace the two-branch extraction diagram starting from a single box on the left labeled **Common Crawl — Raw HTTP response**:

- **Upper path (WARC)**: Arrows flow from Raw HTTP response into **WARC (HTML) — Full DOM structure**, then into a **Custom Extractor** (`trafilatura`, `resiliparse`, or `jusText`), and finally into **Cleaner Text — Higher LM accuracy**.
- **Lower path (WET)**: Arrows flow downward from Raw HTTP response into **WET (Plain Text) — Lossy pre-conversion**, bypassing custom DOM-aware extraction entirely via a dashed red line straight into **Noisy Text — Nav menus, lossy**.

Why does discarding the DOM structure in WET hurt so badly? Look at a minimal three-line HTML page:

```
<nav>Home | Login | Subscribe</nav>
<article><p>Actual article body...</p></article>
<footer>Cookies · Terms of Use</footer>
```

In the raw WARC file, semantic HTML tags (`<nav>`, `<article>`, `<footer>`) and DOM layout cues tell a custom extractor exactly where the real prose lives and which blocks are repetitive site chrome. A naive WET conversion flattens all visible text together, injecting `Home | Login | Subscribe` and `Cookies · Terms of Use` directly before and after every article on the site—and once those DOM tags are stripped away in the WET file, downstream filters can no longer reliably separate boilerplate from body text without risking deleting real sentences.

Empirical benchmarking in **DCLM (DataComp-LM, 2024)** demonstrated directly that the choice of HTML-to-text extractor measurably changes downstream language model task accuracy. That is why modern high-performing pipelines—including The Pile, RefinedWeb, DCLM, and Nemotron-CC—start directly from raw **WARC** files rather than taking the convenient shortcut of WET.

> **COMMON MISCONCEPTION**
> You might think that starting from WET files saves storage and compute without hurting model quality, because "plain text is plain text" and later filtering steps can clean up whatever noise remains. That breaks because WET conversion is **lossy**: once the HTML DOM tags distinguishing navigation menus, sidebars, and footers from the main `<article>` body are thrown away, downstream text filters cannot cleanly reconstruct structural boundaries, leaving either persistent boilerplate noise or mangled prose.

Even when starting from WARC, choosing *which* custom extractor to run introduces a classic **precision–recall tradeoff** over the tokens on each webpage. Let TP (true positives) be actual article body tokens kept by the extractor, FP (false positives) be boilerplate or navigation tokens erroneously kept, and FN (false negatives) be genuine article tokens erroneously stripped away. Then:

$$\text{Precision} = \frac{TP}{TP + FP}, \qquad \text{Recall} = \frac{TP}{TP + FN}$$

Breaking both equations apart:

- In **Precision**, the numerator TP counts clean content tokens preserved, while the denominator $TP + FP$ counts *every* token output by the extractor. Precision therefore measures the **purity** of the extracted text—what fraction of the tokens you feed the model are real prose rather than boilerplate junk.
- In **Recall**, the numerator is again TP, while the denominator $TP + FN$ counts *all* genuine content tokens that existed on the original webpage. Recall therefore measures **completeness**—what fraction of the page's real knowledge you managed to salvage.

As a conceptual pair, precision and recall capture the exact engineering tension between two widely used WARC extractors: `trafilatura` strips HTML aggressively, yielding very high precision (minimal boilerplate) at the cost of lower recall (occasionally dropping valid tables, code snippets, or short paragraphs), whereas `jusText` is more permissive, achieving higher recall (retaining far more tokens per page) at the cost of slightly more residual noise. We will see this exact tradeoff become decisive in Section 8 when Nemotron-CC switches from `trafilatura` to `jusText` to combat token starvation.

> **WORKED EXAMPLE**
> Suppose a technical webpage in a WARC archive contains $1{,}000$ true article tokens and 250 boilerplate tokens (navigation menus, cookie banners, footer links). Compare two extractors on this page:
>
> 1. **Aggressive extractor (`trafilatura`-like)**: Retains 780 true article tokens and 20 boilerplate tokens (dropping 220 valid tokens such as a side-note table and a short code block).

- Precision $= \frac{780}{780+20} = \frac{780}{800} = 97.5\%$
- Recall $= \frac{780}{1{,}000} = 78.0\%$

2. **Permissive extractor (`jusText`-like)**: Retains 940 true article tokens and 110 boilerplate tokens (dropping only 60 valid tokens).
   - Precision $= \frac{940}{940+110} = \frac{940}{1{,}050} \approx 89.5\%$
   - Recall $= \frac{940}{1{,}000} = 94.0\%$

Interpretation: `trafilatura` produces cleaner output per token (97.5% purity vs. 89.5%), making it attractive when raw web volume vastly exceeds your training budget. However, `jusText` recovers $940 - 780 = 160$ more genuine tokens from the exact same webpage (20.5% more useful yield), which becomes essential when aggressive downstream filters leave you starving for unique tokens.

KEY IDEA

Common Crawl provides the open web backbone (300B archived pages, e.g. 2.19B pages / $372.2\ \text{TB}$ in April 2026), and modern pipelines must extract text directly from full-DOM WARC files—not lossy WET files—while deliberately tuning the extraction precision–recall tradeoff (`trafilatura` vs. `jusText`).

Common Crawl gives us scale measured in hundreds of terabytes, yet raw web pages remain overwhelmingly messy, inconsistent, and thin on dense technical exposition like mathematical proofs, working software repositories, or structured encyclopedia articles. To raise the intellectual ceiling of a pre-training corpus, builders combine web crawls with specialized, high-signal non-web sources. Next, we examine the five major curated sources—and uncover the hidden security, duplication, and legal pitfalls lurking inside each one.

✎ Check Your Understanding

1. **What would happen if…**: A dataset team switches its pipeline from running `trafilatura` on Common Crawl WARC files to downloading Common Crawl's pre-extracted WET files directly. What happens to text quality and downstream LM accuracy, and why?
   Answer: Text quality and downstream LM accuracy drop because WET files are lossy pre-conversions that discard the HTML DOM structure, mixing navigation menus and footers directly into article prose where custom extractors can no longer cleanly separate them.
2. **Fill-in-the-blank**: In HTML-to-text extraction from WARC files, `trafilatura` prioritizes higher _____ by stripping boilerplate aggressively, whereas `jusText` prioritizes higher _____ by retaining more tokens per page.
   Answer: **precision**; **recall**.

Section 5: Curated Non-Web Sources — Wikipedia, Code, Scientific Literature, Q&A Forums, and Books

Web crawls provide sheer volume, but they are diluted by advertisements, thin marketing copy, and fragmentary prose. High-capability language models rely heavily on four curated non-web domains—encyclopedias, source code, scientific literature and Q&A forums, and long-form books—each with its own collection mechanism, structural advantages, and operational hazards.

Wikipedia and Scheduled-Dump Data Poisoning

Founded in 2001, **Wikipedia** is the flagship open factual encyclopedia. By May 2026, it had grown to 67M **articles across** 361 **language editions**, with English, Spanish, German, and French constituting the largest editions. Its editorial model enforces high factual density: articles prohibit original research, personal opinions, and promotional material, and every topic must meet a **notability** threshold backed by reliable secondary sources. Although anyone on the web can edit a page and volunteer administrators revert vandalism, authorship is intensely concentrated—a tiny core of editors writes the vast majority of contributions (for instance, one exceptionally prolific editor alone has logged roughly 5M edits). Crucially for data engineers, Wikipedia is distributed as periodic **XML dumps** released every few weeks, so no web crawling is required.

However, relying on periodic static snapshots of a publicly editable website opens an unexpected security vulnerability: **scheduled-dump data poisoning**, demonstrated in 2023 research.

> **DEFINITION**
> **Scheduled-dump data poisoning** is an attack against datasets built from periodic snapshots of community-edited sites, in which an adversary injects malicious text immediately before a predictable snapshot time so that the poisoned content is captured permanently in the dump even though site administrators revert the live page shortly afterward.

Trace the horizontal timeline diagram illustrating the 2023 Wikipedia poisoning window across three sequential points on a time axis:

1. **Point 1 (red dot, left) — Malicious Edit**: Just before a scheduled Wikipedia XML dump runs, an attacker edits target articles to inject poisoned associations (for example, associating strong negative sentiment with a trigger phrase such as `"iPhone"`).
2. **Point 2 (blue dot inside a yellow box, center) — Periodic Dump**: The automated snapshot triggers on schedule and freezes the poisoned revision permanently into the exported XML archive distributed to ML teams worldwide.
3. **Point 3 (green dot, right) — Admin Rollback**: Wikipedia administrators detect and revert the vandalism on the live site—perhaps within minutes—leaving the live encyclopedia clean, while a dashed red arrow extending from the dump box shows that the fix arrives far too late for every dataset already generated from that snapshot.

> **COMMON MISCONCEPTION**
> You might think that because Wikipedia has vigilant human editors who revert vandalism rapidly on the live website, a Wikipedia XML dump is automatically safe from adversarial tampering. That breaks because dataset builders ingest **static point-in-time dumps**, not the live continuously corrected encyclopedia; an attacker who knows the periodic dump schedule only needs a malicious edit to survive for the brief instant the snapshot captures that page.

Source Code: GitHub and Software Heritage

Source code plays a dual role in pre-training: it directly teaches programming ability, and widely observed empirical folklore links code pre-training to measurably stronger **general logical reasoning** across non-coding tasks as well. Two primary archives supply open-source code at scale:

- **GitHub** (founded 2008; acquired by Microsoft in 2018): According to Octoverse 2024, GitHub hosts **over 518M projects** across open-source, public, and private repositories. Beyond raw source files, GitHub captures exceptionally rich software-engineering artifacts: full directory trees, commit histories, issue trackers, pull requests (PRs), and line-by-line code review comments. However, GitHub suffers from massive duplication driven by repository forks and copy-pasted libraries, and legal compliance restricts training to public repositories carrying **permissive licenses** (such as MIT or Apache). Unlike web pages, GitHub data is collected via two specialized access paths: cloning source repositories directly over the **`git` protocol** (never HTML scraping), and collecting social/event metadata through the **GitHub API** and the hourly event stream archived by **GitHub Archive**.
- **Software Heritage** (founded 2016): A non-profit archive dedicated to collecting and preserving all publicly available software source code. Rather than drawing from a single platform, Software Heritage aggregates repositories across multiple code forges and package registries—including **GitHub, GitLab, Bitbucket, and PyPI**. Its scope focuses squarely on **source repositories** rather than social metadata like issues or PR review threads, and it serves as the primary code backbone for open corpora such as **The Stack v2** (which we examine in detail in Section 9).

Scientific Papers (arXiv) and Q&A Forums (Stack Exchange)

Technical reasoning and mathematical notation rarely appear cleanly in ordinary web pages. Two structured repositories fill that gap:

- **arXiv** (operating since 1991): A free preprint repository originally built for physics and now spanning mathematics, computer science, statistics, and related quantitative fields, holding roughly 3M **submissions**. Submissions undergo light moderation rather than formal peer review. Each paper includes bibliographic metadata, a rendered PDF, and—crucially—an optional **LaTeX source** bundle. Extracting directly from LaTeX yields vastly cleaner mathematical equations, structured tables, and section hierarchies than running OCR or PDF-to-text parsers over rendered pages. On licensing, arXiv metadata is released under **CC0** (public domain dedication), while authors choose either all-rights-reserved or a Creative Commons license (such as CC-BY) for the paper itself. Bulk acquisition is straightforward via **Amazon S3**, requiring no crawling.

- **Stack Exchange** (operating since 2008): A network of user-contributed Q&A forums starting with StackOverflow and expanding across mathematics, physics, literature, and dozens of other subjects. Gamified reputation points and badges create a strong incentive for community members to write concise, technically accurate answers. Crucially, Stack Exchange's **question-and-answer structure** naturally aligns with downstream instruction following and real user queries. Moreover, its rich structured metadata—upvotes, topic tags, accepted-answer flags, and clarifying comments—allows dataset engineers to filter for high-signal exchanges with high precision. Like Wikipedia, Stack Exchange releases anonymized **XML dumps** directly.

Books: Long-Form Coherence vs. Legal Takedowns

Long-form books teach discourse coherence, character tracking, and sustained multi-thousand-word narrative or argumentative structure that short web articles rarely provide. Yet books sit right at the epicenter of the copyright battle from Section 3, producing a stark split between legally cleared public-domain collections and modern corpora that have been taken down:

Corpus	Origin & Scale	Legal / Access Status	Key Notes
Project Gutenberg (1971 · PG-19)	~75K books (2025), mostly English	**Copyright-cleared** (mostly public domain)	Founded by Michael Hart; PG-19 collects pre-2019 books; heavily weighted toward older, archaic prose styles
BooksCorpus (2015)	7K self-published books (985M words) priced at $0	**Taken down** (violated Smashwords ToS)	Scraped from Smashwords (which grew by 2024 to 150K authors and 500K books); famously used to train BERT
Books3 (Presser, 2020)	196K books sourced from the shadow library Bibliotik	**Taken down** (copyright infringement & lawsuits)	Included major contemporary authors (e.g., Stephen King, Min Jin Lee, Zadie Smith); ingested into The Pile and LLaMA

WORKED EXAMPLE

Compare two ways a team might acquire technical and long-form corpora for a new base model, tracing both data quality and legal risk:

1. **Acquisition Plan A**: Pull arXiv papers via Amazon S3 using raw LaTeX sources (filtering to CC-BY papers), pull Stack Exchange XML dumps keeping answers with positive vote scores, clone MIT/Apache repositories via Software Heritage, and include Project Gutenberg PG-19.
    - *Technical yield*: Clean equations without PDF corruption, high-precision Q&A formatting, deduplicable multi-forge source code, and long-form narrative prose (albeit older English).

- *Legal posture*: Respects licenses and bulk-distribution mechanisms; avoids ToS breaches and piracy.

2. **Acquisition Plan B**: Scrape Smashwords self-published $0 ebooks (as in BooksCorpus) and mirror Books3 from Bibliotik to obtain contemporary fiction prose.
   - *Technical yield*: Modern narrative prose and dialogue.
   - *Legal posture*: Free price ($0) does **not** mean public domain or scraping permission—Plan B violates Smashwords' Terms of Service and directly commits shadow-library piracy (Books3), recreating the exact acquisition liability that triggered takedowns and billion-dollar settlements in Section 3.

Interpretation: Clean, contemporary, legally cleared book corpora remain one of the scarcest bottlenecks in pre-training data engineering.

KEY IDEA

Curated non-web sources—Wikipedia, GitHub/Software Heritage, arXiv LaTeX, Stack Exchange, and books—supply dense factual, mathematical, Q&A, code, and long-form signals that web crawls lack, but each carries distinct operational or legal risks including scheduled-dump poisoning, massive fork duplication, ToS violations, and copyright takedowns.

We now have all the raw ingredients on the table: Common Crawl snapshots on one side, and Wikipedia, code archives, arXiv, Stack Exchange, and books on the other. How did researchers from 2019 to 2026 actually assemble, filter, deduplicate, and scale these ingredients into concrete pre-training datasets? Every generation solved a pressing bottleneck of the previous one—and exposed a brand-new failure mode that forced the next architectural jump in data curation.

✎ Check Your Understanding

1. **Yes/No with justification**: If a self-published book is listed for $0 on a platform like Smashwords (as in BooksCorpus), does its zero price make it legally safe to scrape and redistribute as a training dataset?
   Answer: No. A $0 retail price does not place a work in the public domain or override platform rules; BooksCorpus was taken down because scraping it violated Smashwords' Terms of Service.
2. **What would happen if…**: Instead of extracting arXiv papers from their optional LaTeX source bundles on Amazon S3, a team extracts text solely from the rendered PDF files. What degrades in the resulting corpus?
   Answer: Mathematical equations, structured tables, and formatting degrade substantially because PDF-to-text extraction loses the clean symbolic structure explicitly encoded in LaTeX source files.

Section 6: First- and Second-Generation Corpora — From Human Proxies to Automated Web Filtering (BERT, WebText, CCNet, and C4)

To make sense of the dozens of pre-training datasets published between 2019 and 2026, view their history as a four-era progression rather than a laundry list of names. Imagine a four-column flowchart reading left to right across the full 2019–2026 timeline:

- **Era 1 — Small Curated (2019, Historical)**: *BERT* (Wikipedia + BooksCorpus) and *GPT-2 WebText* (Reddit links with $\text{karma} \geq 3$, 40 GB). Relied on human curation proxies; cannot scale to trillions of tokens.
- **Era 2 — Rule & LM-Filtered Common Crawl (2019–2024, Historical)**: *CCNet*, *C4*, and *MassiveText* (heuristics and KenLM); *The Pile*, *LLaMA*, and *Dolma* (curated multi-source mixes); *RefinedWeb* and *FineWeb* (WARC + rules + MinHash). Scaled web pipelines up to 15T tokens.
- **Era 3 — Classifier Filtering (2020 / 2024+)**: *GPT-3* (early classifier against WebText) and *DCLM-baseline* (fastText classifier against OpenHermes and ELI5, filtering $240\text{T} \rightarrow 3.8\text{T}$). Achieved much higher accuracy per token, but discarded roughly 90% of raw web tokens.
- **Era 4 — Ensembles, Synthetic Transformation & Legal Provenance (2024–2026)**: *Nemotron-CC* (distilled LLM scorers + synthetic rephrasing), *The Stack v2* (code + PRs + LLVM IR), and *Common Pile* (strictly permissive-only, 8 TB).

In this section, we trace Era 1 and the opening half of Era 2—seeing why human proxies hit a hard ceiling and how automated web filtering first proved that Common Crawl could outperform Wikipedia.

Era 1: Early Curated Corpora — BERT and WebText (2019)

Before filtered Common Crawl became standard, early neural language models avoided raw web dumps entirely because raw web text appeared hopelessly noisy.

BERT (2019) trained on just two clean sources: **Wikipedia** and **BooksCorpus**. Despite its modest scale, BERT introduced a foundational data design shift: it trained on **contiguous multi-sentence documents** rather than shuffled, isolated single sentences. That directly broke with earlier benchmarks like the **1 Billion Word Benchmark** (Chelba et al., 2013), which had been assembled from isolated, shuffled machine-translation sentences—preventing models from ever learning long-range context, pronoun resolution, or cross-sentence discourse.

WebText (GPT-2, 2019) asked how to harvest cleaner web text without manually inspecting millions of pages. Its key idea was to treat **human social upvotes as a quality proxy**: collect every outgoing hyperlink posted to Reddit that received $\text{karma} \geq 3$, scrape those target webpages, and extract their text. This proxy yielded 8M **webpages totaling** 40 GB **of text**—clean enough to demonstrate striking zero-shot language modeling in GPT-2. Because OpenAI did not release WebText, the open community built **OpenWebTextCorpus**, replicating the recipe using Reddit URLs combined with a fastText English language filter and near-deduplication.

The limitation of Era 1 was immediate: human-curated encyclopedias and Reddit-upvoted links plateau at tens of gigabytes. Scaling language models by $10\times$ to $1000\times$ required tapping directly into Common Crawl—and automatically separating coherent prose from web garbage.

Era 2A: Automatic Language-Model Filtering — CCNet (2019)

CCNet (2019) was a watershed historical pipeline demonstrating two crucial facts: properly filtered Common Crawl could actually **outperform Wikipedia alone** when training BERT models, and the same automated pipeline could scale beyond English to **low-resource languages** such as Urdu. (Note that "CCNet" refers both to the open-source processing pipeline and to the corpus it produced, which was later reused directly inside LLaMA.)

Visualize CCNet's pipeline as four boxes flowing left to right from **Common Crawl (Multilingual raw)** through three sequential filters:

1. **Deduplication**: Removes duplicate paragraphs across the crawl after light text normalization (such as lowercasing and punctuation normalization).
2. **Language Identification**: Runs a fast linear text classifier (**fastText**) to assign each document to its target language, routing documents into language-specific shards.
3. **LM Quality Filter**: Scores every document using a lightweight **KenLM 5-gram language model trained on Wikipedia** in that target language, keeping only documents with **low perplexity** (i.e., documents whose word sequences resemble encyclopedic prose).

> **DEFINITION**
> **Perplexity (PPL)** under a language model is the exponentiated average negative log-probability assigned to a sequence of tokens; lower perplexity indicates that the language model finds the text more predictable and statistically closer to its reference training distribution.

Because CCNet's third stage introduces the first statistical quality metric in our lecture, let us write out the exact formula for a 5-gram language model's perplexity on a document of N words $w_1, w_2, \ldots, w_N$:

$$\mathrm{PPL}(w_1, \ldots, w_N) = \exp\left(-\frac{1}{N} \sum_{i=1}^{N} \ln P(w_i \mid w_{i-4}, w_{i-3}, w_{i-2}, w_{i-1}) \right)$$

Decompose this equation hierarchically from the innermost expression outward:

1. **Innermost conditional probability** $P(w_i \mid w_{i-4}, \ldots, w_{i-1})$: A 5-gram Markov model trained on Wikipedia estimates the probability of word w_i given only the four immediately preceding words. Natural, grammatical prose receives high probabilities (close to 1 for predictable collocations); random spam, broken HTML, or keyword stuffing receives tiny probabilities (close to 0).
2. **Negative log-likelihood** $-\ln P(\cdot)$: Taking the negative natural logarithm converts probabilities in $(0, 1]$ into non-negative surprise scores in $[0, \infty)$. High probability yields small surprise; tiny probability yields huge surprise.
3. **Length-normalized average** $\frac{1}{N} \sum_{i=1}^{N} (\cdots)$: Summing over all N words in the document and dividing by N prevents long documents from automatically looking worse than short documents.
4. **Exponentiation** $\exp(\cdot)$: Maps the average log-surprise back onto a human-interpretable "effective branching factor" scale—a perplexity of 20 means the reference model is, on average, as uncertain

about each next word as if choosing uniformly among 20 equally likely candidates.

Taken as a single conceptual unit, CCNet's KenLM filter measures **how Wikipedia-like a web page is**, discarding high-perplexity pages full of garbled tokens or boilerplate while preserving coherent prose cheaply at massive scale.

WORKED EXAMPLE

Suppose a Wikipedia-trained 5-gram KenLM scores two 4-word snippets from Common Crawl:

- **Snippet A (clean prose)**: `"the city was founded"` with 5-gram probabilities $[0.25,\ 0.10,\ 0.50,\ 0.20]$.
- **Snippet B (garbled menu spam)**: `"buy click promo login"` with 5-gram probabilities $[0.005,\ 0.002,\ 0.001,\ 0.004]$.

Compute the perplexity of each snippet:

1. For Snippet A:
   - $\ln(0.25) \approx -1.386$, $\ln(0.10) \approx -2.303$, $\ln(0.50) \approx -0.693$, $\ln(0.20) \approx -1.609$.
   - Average negative log-probability: $-\frac{1}{4}(-1.386 - 2.303 - 0.693 - 1.609) = \frac{5.991}{4} \approx 1.498$.
   - $\mathrm{PPL}_A = e^{1.498} \approx 4.47$.
2. For Snippet B:
   - $\ln(0.005) \approx -5.298$, $\ln(0.002) \approx -6.215$, $\ln(0.001) \approx -6.908$, $\ln(0.004) \approx -5.521$.
   - Average negative log-probability: $\frac{23.942}{4} \approx 5.986$.
   - $\mathrm{PPL}_B = e^{5.986} \approx 397.8$.

Interpretation: Snippet A yields a perplexity of 4.47, whereas Snippet B yields 397.8—nearly two orders of magnitude higher. A simple perplexity threshold cleanly keeps Snippet A and discards Snippet B without manually writing rules for every spam pattern.

Era 2B: Heuristic Web Filtering — C4 (T5, 2019)

Released alongside the T5 text-to-text paper, **C4** (*Colossal Clean Crawled Corpus*, 2019) took a complementary, purely rule-based approach to the same observation: raw Common Crawl is overwhelmingly unusable boilerplate and noise. Starting from a single Common Crawl snapshot (**April 2019**, containing $1.4\mathrm{T}$ **raw tokens**), the C4 pipeline applied a cascade of **manual heuristics** to distill the snapshot down to 806 GB ($156\mathrm{B}$ **tokens**)—retaining only $\frac{156\mathrm{B}}{1{,}400\mathrm{B}} \approx 11.1\%$ of the input tokens:

- **Sentence and page structure**: Keep only lines ending in terminal punctuation (such as `.`, `!`, `?`) with at least 5 words, and drop entire pages containing fewer than 3 sentences.
- **Blocklist filtering**: Drop any page containing any word from an English "bad words" blocklist.
- **Boilerplate and placeholder removal**: Drop any page containing `lorem ipsum`, `terms of use`, or the curly-brace character `{`.

- **Strict English filtering**: Run `langdetect` and retain only documents with predicted English probability $p(\text{en}) = 0.99$.

Notice the glaring collateral damage hidden in those manual rules. Dropping every page containing `{` was intended to discard stray JavaScript and CSS blocks—yet it **accidentally wiped out virtually all source code** from C4! Similarly, brittle bad-word blocklists routinely delete legitimate medical, literary, or minority-dialect discussions while missing subtle toxicity.

COMMON MISCONCEPTION

You might think that because Common Crawl archives billions of webpages, filtering Common Crawl down to the URLs of a smaller curated dataset like WebText should recover nearly 100% of that dataset. That breaks because Common Crawl is severely incomplete: when the C4 authors filtered **12 Common Crawl dumps** to match OpenWebText URLs, they recovered only 17 GB compared to WebText's 40 GB—missing over half of the pages—yet even that 17 GB subset still improved downstream GLUE and SQuAD scores, proving that Common Crawl's coverage gaps coexist with high value per cleaned token.

KEY IDEA

Early corpora like BERT and GPT-2's WebText proved the value of contiguous documents and human quality proxies, while CCNet (paragraph dedup + fastText language ID + Wikipedia KenLM perplexity) and C4 (manual punctuation, boilerplate, and language rules reducing 1.4T raw tokens to 156B) established automated Common Crawl filtering—while exposing both rule brittleness (dropping `{` deletes code) and Common Crawl's incompleteness.

CCNet and C4 proved that automated web filtering works, yet each left a major limitation unsolved: CCNet biased everything toward Wikipedia-style prose, while C4's brittle rules deleted source code and missed domain diversity. That forced the next wave of corpora (2020–2024) to test two rival philosophies at multi-terabyte scale: carefully mixing dozens of curated domains alongside the web, versus pushing pure web filtering to 15T tokens.

✎ Check Your Understanding

1. **Quick computation**: C4 started from a single April 2019 Common Crawl snapshot of 1.4T tokens and retained 156B (0.156T) tokens after heuristic filtering. Approximately what percentage of raw tokens were *discarded*?
   Answer: Retained fraction is $0.156/1.4 \approx 11.1\%$, so approximately $100\% - 11.1\% = 88.9\%$ of raw tokens were discarded.
2. **Fill-in-the-blank**: In CCNet, the third pipeline stage scores documents using a _____ trained on Wikipedia and keeps documents with _____ perplexity; in C4, dropping pages containing the character `{` had the unintended side effect of removing _____.
   Answer: **KenLM 5-gram language model**; **low**; **source code**.

Section 7: Scaling Up Multi-Source Mixtures and Web-Only Pipelines (GPT-3, The Pile, Gopher, LLaMA, RefinedWeb, FineWeb, and Dolma)

Between 2020 and 2024, pre-training corpora expanded by nearly two orders of magnitude—from 275B and 400B tokens up to 15T tokens. Across this period, two competing design philosophies battled for dominance: **multi-source domain curation** (hand-assembling encyclopedias, papers, code, books, and forums alongside filtered web data) versus **web-only scaling** (proving that carefully extracted, rule-filtered, deduplicated Common Crawl alone could match or beat hand-curated mixes).

Learned Classifiers vs. Grassroots Multi-Source Curation (2020–2021)

Two landmark corpora immediately followed C4 and defined the two poles of this debate:

- **GPT-3 Dataset (2020, historical) — 570 GB (400B tokens)**: OpenAI combined processed Common Crawl with WebText2, Wikipedia, and two undisclosed book corpora (*Books1* and *Books2*). Rather than relying solely on hand-crafted heuristics like C4, GPT-3 trained an early **binary quality classifier** to distinguish a high-quality positive set (`{WebText, Wikipedia, Books}`) from raw Common Crawl, keeping Common Crawl documents scored close to the positive distribution. Equally consequential, GPT-3 introduced **fuzzy deduplication** not only across training documents and WebText, but also against downstream **evaluation benchmarks** to reduce test-set contamination.
- **The Pile (2021, historical) — 825 GB ($\sim 275\text{B}$ tokens)**: Built as a grassroots open-source effort (used to train GPT-J and GPT-NeoX), The Pile bet on breadth across **22 carefully curated domains**. Its web slice (**Pile-CC**) broke early from WET files, extracting directly from **WARC using `jusText`** for much cleaner prose. Alongside Pile-CC, it assembled specialized domains rarely seen together before: **PubMed Central** (5M NIH-mandated biomedical papers), **arXiv** (extracted from LaTeX), the **Enron emails** (500K corporate emails from 150 executives released during the 2002 investigation), Project Gutenberg, **Books3** (later taken down, as seen in Section 5), and Stack Exchange.

Scaling Curated Mixtures: Gopher/MassiveText and LLaMA (2021–2023)

As training budgets crossed 300B to 1.2T tokens, laboratories refined both heuristic filtering and domain-specific preprocessing:

- **Gopher / MassiveText (2021, historical)**: DeepMind assembled $10.5\ \text{TB}$ of total data (combining **MassiveWeb**, C4, Books, News, GitHub, and Wikipedia), though the Gopher model itself trained on only 300B tokens ($\approx 12\%$ of the full pool). For **MassiveWeb**, DeepMind made a deliberate anti-classifier design choice: instead of training an ML quality classifier (which they worried would introduce opaque domain biases), they engineered transparent **manual heuristic rules**—for example, requiring that at least 80% of words in a document contain an alphabetic character—combined with English filtering, deduplication, and explicit train–test overlap removal. To avoid C4's brittle bad-word blocklists, MassiveWeb used **Google SafeSearch** for toxicity filtering. These "Gopher rules" became a reusable baseline across the entire field.
- **LLaMA (2023, historical) — 1.2T tokens**: Meta demonstrated how far a disciplined, publicly sourced multi-domain mixture could go. LLaMA combined **CCNet** plus a linear classifier targeting pages cited as **Wikipedia references**, alongside **C4** specifically to maintain stylistic diversity. Its curated domains included permissively licensed **GitHub** (cleaned with rules), **Wikipedia** across 20 languages (from June–August 2022 dumps), **Project Gutenberg + Books3**, **arXiv** (with LaTeX comments and bibliographies stripped and macros expanded into clean text), and **Stack Exchange** (the 28 largest sites, with answers sorted by community score). Because Meta did not release the dataset itself, the open community replicated it as **RedPajama v1** (1T tokens), and then proved how much redundancy remained by running **MinHashLSH deduplication** to create **SlimPajama**—shrinking the corpus from 1T down to 627B **tokens** (eliminating over 37% duplicate volume) while improving training efficiency.

"Web Data Is All You Need": RefinedWeb and FineWeb (2023–2024)

Hand-curating 22 separate domains—each with custom parsers, license checks, and mixture weights—is painstaking and difficult to scale past a few trillion tokens. Two influential web-only corpora challenged the assumption that multi-source mixtures were necessary at all:

- **RefinedWeb (Falcon, 2023, historical)**: Advanced the thesis that properly cleaned Common Crawl alone suffices to match or beat curated multi-source datasets. RefinedWeb started from raw **WARC** files, extracted prose with `trafilatura`, applied **Gopher heuristic rules** (explicitly avoiding ML quality classifiers to prevent narrowing domain diversity), and ran fuzzy deduplication using **MinHash over 5-grams**. The pipeline produced 5T **tokens** (600B released publicly).
- **FineWeb (HuggingFace, 2024, historical)**: Began as an open replication of RefinedWeb and then systematically scaled and refined the recipe across **95 Common Crawl dumps** to reach 15T **tokens**. FineWeb's pipeline stacked URL filtering, language identification with a relaxed threshold ($p(\text{en}) > 0.65$, noticeably more permissive than C4's rigid $p(\text{en}) = 0.99$), combined **Gopher + C4 + additional custom manual quality rules**, **MinHash fuzzy deduplication**, and **PII (Personally Identifiable Information) anonymization** to scrub email addresses and public IP addresses.

COMMON MISCONCEPTION

You might think that setting an extremely strict language-identification cutoff like C4's $p(\text{en}) = 0.99$ is always better than a looser threshold like FineWeb's $p(\text{en}) > 0.65$. That breaks when technical documents mix English prose with mathematical symbols, code identifiers, URLs, or foreign proper nouns: a rigid 0.99 threshold systematically discards high-value STEM and programming pages whose non-word tokens lower the language classifier's confidence, whereas a calibrated threshold like 0.65 retains technical English while still filtering out non-English pages.

Open Multi-Source Curation: Dolma (2024)

Alongside FineWeb's web-only push, AI2 built **Dolma (2024, historical)**—an open 3T**-token** multi-source pre-training corpus created for OLMo. Dolma paired cleaned Common Crawl with curated domains: **Reddit** from the Pushshift archive (2005–2023, deliberately keeping submissions and comments separate), **PeS2o** (40M academic papers from Semantic Scholar), **C4**, **Project Gutenberg**, and **Wikipedia / Wikibooks**. On its Common Crawl slice, Dolma used **fastText** English identification, **Gopher + C4 heuristic rules** (again avoiding model-based quality classifiers), **toxicity filtering** combining heuristic rules with the **Jigsaw classifier**, and memory-efficient exact deduplication via **Bloom filters**.

DEFINITION

A **Bloom filter** is a compact probabilistic set-membership data structure using multiple hash functions over a bit array that tests whether an item (such as a document or paragraph) has already been seen, guaranteeing zero false negatives at the cost of a small, tunable false-positive rate.

WORKED EXAMPLE

Trace what happens to a single technical webpage—a blog post explaining Python matrix multiplication with 300 words of English prose, a 15-line code snippet containing curly braces `{}`, a contact email `alice@example.org`, and a duplicate copy mirrored on a content aggregator—as it flows through **C4 (2019)** versus **FineWeb (2024)**:

1. **Under C4 (2019)**:
   - The `{` rule triggers immediately on the dictionary literal inside the Python snippet, **dropping the entire page**. Even if `{` were absent, the code lines lack terminal punctuation and drop the `langdetect` English probability below 0.99, risking rejection.
2. **Under FineWeb (2024)**:
   - WARC + `trafilatura` strips navigation chrome cleanly while keeping the article body and code block.
   - Language ID scores $p(\text{en}) \approx 0.88 > 0.65$, so the page passes.
   - Gopher + refined rules verify alphabetic word ratio ($\geq 80\%$) without banning `{`.
   - MinHash over 5-grams detects the near-duplicate aggregator mirror and removes the copy, keeping one clean instance.

- PII anonymization scrubs `alice@example.org` into a placeholder token.

Interpretation: Between 2019 and 2024, rule-based web pipelines evolved from blunt filters that destroyed code and leaked private identifiers into nuanced, multi-stage pipelines capable of producing $15\mathrm{T}$ clean tokens.

KEY IDEA

Between 2020 and 2024, pre-training data scaled to $15\mathrm{T}$ tokens along two complementary tracks—curated multi-source blends (GPT-3, The Pile, Gopher, LLaMA, Dolma) and high-volume WARC + heuristic + MinHash web-only corpora (RefinedWeb, FineWeb)—with most pipelines deliberately avoiding ML quality classifiers out of fear of domain bias.

Look closely at the shared assumption behind Gopher, RefinedWeb, FineWeb, and Dolma: all four deliberately avoided learned quality classifiers and trusted hand-written heuristic rules instead. That raised an urgent empirical question: if you fix the exact same raw Common Crawl pool and run controlled experiments, do hand-crafted heuristic rules actually beat a learned quality classifier—or were rule-based pipelines leaving massive model accuracy on the table? Answering that question triggered Era 3 and immediately collided with the modern crisis of token starvation.

✎ Check Your Understanding

1. **Yes/No with justification**: Did RefinedWeb (2023) and Dolma (2024) use machine-learning quality classifiers to select high-quality web pages from Common Crawl?
   Answer: No. Both RefinedWeb (using Gopher rules) and Dolma (using Gopher + C4 rules) deliberately relied on manual heuristic quality rules and avoided ML quality classifiers to prevent opaque domain bias (though Dolma used fastText for language ID and Jigsaw for toxicity).
2. **What would happen if…**: You take a $1\mathrm{T}$-token un-deduplicated replication of LLaMA's mixture (RedPajama v1) and apply MinHashLSH near-deduplication as done for SlimPajama. Roughly how much volume remains, and why is this beneficial?
   Answer: Volume drops from $1\mathrm{T}$ to $627\mathrm{B}$ **tokens** (removing over 37% redundancy), which eliminates repeated boilerplate and mirrored content so every training step teaches the model more unique information.

Section 8: Model-Based Filtering, Token Starvation, and Synthetic Data Transformation (DCLM and Nemotron-CC)

Through 2023, comparing data curation methods was notoriously confounded: one team changed the HTML extractor, the raw crawl dumps, the model architecture, and the filtering rules all at once, making it impossible to know which data decision actually drove benchmark gains. **DataComp-LM (DCLM,**

2024) solved this scientific problem by introducing a **standardized benchmark** that holds the raw Common Crawl pool and training recipe fixed so researchers can isolate and compare data curation algorithms head to head.

Era 3: Model-Based Filtering Wins — DCLM (2024)

Trace the DCLM-baseline pipeline diagram from left to right across three major stages:

- **Input (left box)**: **DCLM-pool**, a massive raw Common Crawl pool of $240\mathrm{T}$ **tokens**.
- **Center stage (large yellow box)**: A lightweight **fastText Quality Classifier** trained on a balanced $400\mathrm{K}$-document dataset:
  - **Positives ($200\mathrm{K}$ green box)**: **OpenHermes-2.5** (primarily GPT-4-generated instruction-following data) combined with **Reddit ELI5** ("Explain Like I'm 5" explanatory Q&A).
  - **Negatives ($200\mathrm{K}$ red box)**: A random sample of **RefinedWeb** representing typical already-cleaned web text.
- **Output (right box)**: **DCLM-baseline**, retaining the highest-scoring $3.8\mathrm{T}$ **tokens** out of the original $240\mathrm{T}$ pool.

Notice the cleverness of DCLM's positive and negative sets. Earlier classifiers (like GPT-3's classifier against raw Common Crawl) trained on Wikipedia and books as positives and *raw* noisy crawl as negatives, so the classifier spent most of its capacity learning to spot basic grammatical English. By choosing **RefinedWeb** as the negative set and **OpenHermes-2.5 + ELI5** as the positive set, DCLM forced the fastText classifier to look past basic grammar and specifically hunt for **instruction-like, explanatory, pedagogically clear prose**. The empirical verdict was decisive: running a fastText classifier trained on high-signal positives **substantially outperformed heuristic-only filtering pipelines** like RefinedWeb and FineWeb at identical compute budgets.

The Token Starvation Bottleneck

DCLM settled the heuristic-versus-classifier debate—and immediately exposed a severe scaling wall. Aggressive quality classifiers such as **FineWeb-Edu** and **DCLM** discard roughly 90% **of cleaned web tokens** (and from raw unfiltered pools like DCLM-pool, $3.8\mathrm{T}/240\mathrm{T} \approx 1.58\%$ survives!).

We can state this **token starvation** constraint directly with a linear retention equation. Let T_{raw} be the number of available unique input tokens before aggressive quality filtering, let $d \in (0, 1)$ be the fraction of tokens discarded by the filter, and let T_{kept} be the number of surviving high-quality tokens:

$$T_{\text{kept}} = (1 - d)\, T_{\text{raw}} \quad \Longleftrightarrow \quad T_{\text{raw}} = \frac{T_{\text{kept}}}{1 - d}$$

Breaking this apart term by term:

- $(1 - d)$ is the **survival rate** (keep fraction) of the classifier. When $d = 0.90$, only $1 - 0.90 = 0.10$ (10%) of input tokens survive.
- Dividing the desired training budget T_{kept} by $(1 - d)$ yields the **required upstream unique token supply** T_{raw}.

Conceptually, this equation reveals why aggressive filtering collides head-on with the token-budget scaling curve from Section 1: modern frontier training runs demand 15T **tokens** (Llama 3, historical 2024), 36T **tokens** (Qwen3, earlier 2025), and 45T **tokens** (DeepSeek-V4.1-Flash, late-2026 frontier).

> **WORKED EXAMPLE**
> Calculate how many unique filtered-pool web tokens T_{raw} are required to supply a 45T-token pre-training run (matching DeepSeek-V4.1-Flash, Sept 2026) under a $d = 0.90$ aggressive classifier discard rate, and compare that to DCLM-baseline's actual yield.
>
> 1. Apply the supply equation with $T_{\text{kept}} = 45\text{T}$ and $d = 0.90$:
>
> $$T_{\text{raw}} = \frac{45\text{T}}{1 - 0.90} = \frac{45\text{T}}{0.10} = 450\text{T unique cleaned web tokens}$$
>
> 2. Compare against DCLM-baseline's end-to-end yield: starting from 240T raw tokens in DCLM-pool, DCLM-baseline retained only 3.8T tokens:
>
> $$\frac{3.8\text{T}}{45\text{T}} \approx 8.4\% \text{ of the DeepSeek-V4.1-Flash budget}$$
>
> **Interpretation**: Even after mining 240T raw Common Crawl tokens, DCLM-baseline fills less than one-tenth of a late-2026 frontier budget. Simply throwing away 90% of the web leaves you severely starved for unique tokens—forcing either excessive multi-epoch repetition (which degrades scaling) or a new way to extract more value from existing web text.

Era 4: Overcoming Token Starvation — Nemotron-CC (2024)

Nemotron-CC (2024) addressed token starvation by refusing to treat raw web tokens as a fixed take-it-or-leave-it supply. It recovered substantial token volume while maintaining high quality through **three coordinated levers**:

1. **Higher-recall HTML extraction**: Returning directly to the precision–recall tradeoff we derived in Section 4, Nemotron-CC switched from `trafilatura` to **`jusText`**, deliberately retaining more genuine tokens per WARC page before filtering even begins.
2. **Classifier ensembling**: Rather than trusting a single classifier's blind spots, Nemotron-CC combined the **DCLM fastText classifier** with a fast quality scorer **distilled from Nemotron-340B-instruct** (which had scored FineWeb pages specifically for educational value). Because two classifiers with different training targets disagree on borderline pages, ensembling widens high-quality recall without admitting low-grade spam.
3. **Synthetic data rephrasing and task generation**: Using language models themselves as **data transformers** to salvage discarded documents and amplify high-quality documents.

Combining these three levers, Nemotron-CC yielded 6.3T **tokens total**, including a 1.1T**-token high-quality subset**.

DEFINITION

Synthetic data rephrasing is a pre-training data pipeline technique in which an existing language model rewrites noisy or poorly structured web documents into clean, coherent prose—or generates derivative tasks such as question-answer pairs from high-quality documents—to expand the effective supply of high-signal training tokens.

Trace the two-branch **Synthetic Rephrasing & Task Generation** pipeline diagram starting from a single input box on the left labeled **Scored Web — Ensemble Classifier**:

- **Upper branch (Salvaging Lower-Quality Docs)**: Documents classified as **Lower-Quality Docs** (red box—text normally discarded by 90% filter cutoffs, yet still containing useful underlying facts or domain topics) flow into **LM Rephrasing** (yellow box), where a language model rewrites the messy text into clean, well-structured prose, outputting **Recovered Tokens** (green box).
- **Lower branch (Amplifying High-Quality Docs)**: Documents classified as **High-Quality Docs** (green box—rich factual or educational content) flow into **LM Task Synthesis** (yellow box), where a language model generates structured **QA pairs** and **key-information extraction** exercises grounded in the source text, outputting **Amplified HQ Supervision** (green box).

COMMON MISCONCEPTION

You might think that synthetic data generation means asking a language model to hallucinate brand-new facts out of thin air to replace the web. That breaks because ungrounded generation quickly amplifies model errors and collapses diversity; instead, pipelines like Nemotron-CC ground every synthetic pass directly in real web documents—either cleaning up the surface prose of real low-quality pages or extracting new Q&A perspectives from real high-quality pages.

Using LMs as data transformers fundamentally changes the supply equation, yet it introduces two major tradeoffs:

1. **Substantial inference compute cost**: Running a capable LLM over trillions of web tokens to rephrase or synthesize tasks adds a massive upfront data-preparation compute bill before pre-training even starts.
2. **Legal provenance uncertainty**: If the teacher LLM doing the rephrasing was itself trained on unlicensed copyrighted web data, the legal lineage and copyright status of the synthetic outputs remain unsettled—a subtlety we will examine closely in Section 10.

KEY IDEA

DCLM proved that fastText quality classifiers trained on high-signal positives substantially beat heuristic rules, but their $\sim 90\%$ discard rate causes severe token starvation against 15T–45T frontier budgets; Nemotron-CC breaks this bottleneck using higher-recall `jusText` extraction, distilled classifier ensembles, and grounded LLM rephrasing and task synthesis.

So far, every web-filtering pipeline we have examined—from CCNet and FineWeb to DCLM and Nemotron-CC—operates on prose webpages. Source code presents an entirely different structural challenge: files are forked and copied tenfold across repositories, licenses must be verified at the repository level, and rich software-engineering artifacts like pull requests are nested graphs rather than flat streams of text. We turn to specialized code pre-training corpora next.

> **✎ Check Your Understanding**
>
> 1. **Quick computation**: Suppose a cleaned web pool contains 100T unique tokens and an aggressive educational classifier discards $d = 0.90$ (90%) of them. How many tokens survive, and is that enough for a 36T-token budget (like earlier Qwen3, 2025) without repetition or synthetic expansion?
> **Answer:** Using $T_{\text{kept}} = (1-d)T_{\text{raw}}$, only $(1-0.90)(100\text{T}) = 10\text{T}$ tokens survive, which falls far short of 36T tokens.
> 2. **What would happen if…**: In the Nemotron-CC synthetic pipeline, why are lower-quality web documents routed to *LM Rephrasing* while high-quality documents are routed to *LM Task Synthesis* rather than simply discarding all lower-quality documents?
> **Answer:** Discarding all lower-quality documents causes token starvation; rephrasing converts salvageable noisy pages into clean recovered tokens, while task synthesis (QA pairs, key-info extraction) extracts amplified supervision from the best factual documents.

Section 9: Specialized Code Pretraining Corpora — The Stack, MinHash Jaccard Deduplication, and Linearizing Pull Requests

In Section 5, we noted two defining quirks of public source code repositories on GitHub and Software Heritage: extreme duplication caused by repository forks and copy-pasted boilerplate, and strict licensing boundaries separating permissive open-source licenses from restrictive ones. Building a clean code pre-training corpus therefore demands specialized license detection, aggressive exact and near-duplicate removal, cross-language sharing, and explicit linearization of structured developer workflows.

The Stack (2022) and Near-Deduplication via Jaccard Similarity & MinHash

The Stack (2022) ($3.1\ \text{TB}$) established the blueprint for open, permissively licensed code pre-training. Its builders harvested repository names from **GitHub Archive** (2015–2022) and executed `git clone` across 137M **repositories**. Those clones yielded 51B **files**—yet once exact byte-identical duplicates were removed, only 5B **files were unique**! That represents a $\frac{51\text{B}}{5\text{B}} = 10.2\times$ exact duplication factor, meaning roughly 90% of raw cloned GitHub files were literal copies of files already seen elsewhere. To enforce legal compliance, The Stack retained only repositories carrying **permissive licenses** (such as MIT and Apache), identified automatically using **`go-license-detector`**.

Exact deduplication removes byte-identical copies, yet software repositories also teem with *near-duplicates*—files that differ only by a changed copyright header, a tweaked variable name, or two added log lines. To remove near-duplicates efficiently across billions of files, The Stack (just like RefinedWeb and FineWeb on web text) measures document overlap using **Jaccard similarity** over sets of substrings (such as character or word n-grams, often called shingles) and approximates it at scale with **MinHash**.

> **DEFINITION**
> The **Jaccard similarity** $J(A, B)$ between two finite non-empty sets A and B is the ratio of the size of their intersection to the size of their union, measuring exact set overlap on a scale from 0 (completely disjoint) to 1 (identical). **MinHash** is a locality-sensitive hashing technique that compresses each set into a short signature of minimum hash values so that near-duplicate documents can be detected without comparing every pair of full documents.

State the exact Jaccard similarity equation (as applied to two documents represented by their sets of n-grams A and B):

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Break the equation apart symbol by symbol:

- A and B are the sets of unique n-grams (e.g., 5-grams of tokens or lines) extracted from Document 1 and Document 2 respectively.
- $|A \cap B|$ (the numerator) counts how many unique n-grams appear in **both** documents—the shared content.
- $|A \cup B| = |A| + |B| - |A \cap B|$ (the denominator) counts the total number of unique n-grams appearing in **either** document—the combined vocabulary of the pair.
- Dividing the intersection size by the union size normalizes overlap into $[0, 1]$, automatically accounting for document length differences.

Comparing all pairs of 5B unique files directly would require $\binom{5\times10^9}{2} \approx 1.25 \times 10^{19}$ pairwise comparisons—completely intractable. **MinHash** (whose full algorithmic construction we will derive in Lecture 14, Data II) overcomes this quadratic wall by applying random hash functions h to every element in a set and keeping only the minimum hash value $h_{\min}(A) = \min_{x \in A} h(x)$. Under a random permutation, the probability that two sets share the exact same minimum hash equals their exact Jaccard similarity:

$$\Pr\big[h_{\min}(A) = h_{\min}(B)\big] = J(A, B)$$

As a conceptual unit, Jaccard similarity defines *what* constitutes a near-duplicate, while MinHash turns that definition into an $O(N)$-scale engineering filter capable of deduplicating billions of code files or trillions of web tokens.

> **WORKED EXAMPLE**

Suppose two Python utility files are nearly identical except for one renamed helper function. After shingling into 5-grams, File A has $|A| = 100$ unique 5-grams, File B has $|B| = 105$ unique 5-grams, and they share $|A \cap B| = 90$ unique 5-grams.

1. Compute the size of the union:

$$|A \cup B| = |A| + |B| - |A \cap B| = 100 + 105 - 90 = 115$$

2. Compute the Jaccard similarity:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{90}{115} \approx 0.783$$

3. Suppose the pipeline sets a near-duplicate threshold of $J \geq 0.75$. Because $0.783 \geq 0.75$, File A and File B are flagged as near-duplicates, and one copy is dropped—even though an exact file-hash check saw them as completely distinct (100% mismatch on SHA-256).

Interpretation: Exact deduplication misses files altered by minor edits or header comments; Jaccard similarity with MinHash catches these subtle clones cleanly and quantifies overlap via a single tunable threshold.

The Stack v2 (2024) and Cross-Language Transfer

The Stack v2 (2024) expanded far beyond raw GitHub source files to build a comprehensive coding and technical reasoning corpus:

- **Broader raw code + social metadata**: Source repositories were drawn from **Software Heritage** (spanning multiple forges), then paired with rich developer discussions—issues, pull requests, and review comments—from **GitHub Archive**.
- **Documentation and math/code benchmarks**: Augmented with crawled developer documentation (**PyPI**, **npm**, **devdocs.io**) plus curated mathematical and coding datasets (**GSM8K**, code contests, StackOverflow, arXiv, Wikipedia, and OpenWebMath).
- **Aggressive hygiene**: Stripped compiled binaries, malware, and automated bot activity; ran deduplication; redacted PII; and subsampled pull requests.
- **Cross-language LLVM IR pairing**: Low-resource programming languages (such as **Nim**) lack the massive code volume of Python or JavaScript. The Stack v2 paired low-resource source code with its compiled **LLVM Intermediate Representation (LLVM IR)**—a shared intermediate representation produced by compilers across many languages—enabling cross-language transfer so the model can bridge syntax it rarely sees with semantic patterns shared across high-resource compiled languages.

Linearizing Structured Code Objects into Token Streams

Ingesting issues and pull requests from GitHub Archive exposes a fundamental representation mismatch: a GitHub pull request is a **nested graph** of repository metadata, surrounding file contents, line-level code diffs, and threaded human review comments, whereas an autoregressive language model only consumes a **flat 1D token sequence**.

DEFINITION

Linearization is the process of serializing a structured, multi-component object (such as a pull request containing metadata, file context, diffs, and review threads) into a single flat sequence of tokens delimited by structural tags so that an autoregressive language model can train on it via standard next-token prediction.

Visualize the **Linearize & Subsample** diagram showing four stacked rectangular boxes on the left—**PR Metadata** (blue), **File Context** (yellow), **Code Diff** (red), and **Review Thread** (green)—all sending converging arrows rightward into a single box labeled **Linearize & Subsample**. The output of that serialization looks like this structured token stream:

```
<pr_title>Fix empty-list edge case</pr_title>
<file path="parser.py">
  # surrounding inline file context
  def parse(items):
-     return items[0]
+     return items[0] if items else None
</file>
<comment>Add a unit test for None.</comment>
```

Two deliberate engineering choices make this representation effective:

1. **Inlining surrounding file context around each diff**: Showing only `- return items[0]` and `+ return items[0] if items else None` in isolation is ambiguous, while feeding the entire 100,000-line repository blows past context limits and wastes tokens. Inlining the local function context directly around the diff makes real software edits learnable at reasonable sequence lengths.
2. **Subsampling pull requests**: Automated bots (such as dependency-bump bots) generate millions of repetitive, low-signal pull requests. Subsampling prevents noisy automated commits from overwhelming genuine human engineering edits and code reviews.

COMMON MISCONCEPTION

You might think that when training a coding model on GitHub pull requests, you should either feed only the raw diff lines (to save tokens) or feed the entire repository alongside every PR (for maximum context). Both extremes break: bare diffs strip away the function signatures and variable definitions needed to predict why an edit is valid, while dumping full repositories wastes most of the training budget on untouched boilerplate; inlining local file context around each diff hits the learnable sweet spot.

KEY IDEA

Building high-quality code corpora (The Stack and The Stack v2) requires verifying permissive licenses, collapsing $10\times$ exact and near-duplication via Jaccard similarity and MinHash, bridging low-resource languages with shared LLVM IR, and carefully linearizing structured pull requests with inlined local context while subsampling noisy automated commits.

Notice that The Stack achieved broad adoption partly because it enforced strict permissive-license filtering (MIT, Apache) from day one. That immediately poses the central legal and engineering question of the entire lecture: if permissive-only filtering worked for code, can we solve the copyright uncertainty of Section 3 completely by building a general-purpose pre-training corpus strictly from permissively licensed data? Our final section tests that exact hypothesis and synthesizes the entire pre-training data landscape.

> **✎ Check Your Understanding**
>
> 1. **Quick computation**: File X contains 50 unique 5-grams, File Y contains 60 unique 5-grams, and they share 40 unique 5-grams. What is their Jaccard similarity $J(X, Y)$?
> **Answer:** First compute $|X \cup Y| = 50 + 60 - 40 = 70$. Then $J(X, Y) = \frac{|X \cap Y|}{|X \cup Y|} = \frac{40}{70} \approx 0.571$.
> 2. **Yes/No with justification**: In The Stack (2022), did removing exact byte-identical duplicates eliminate only a tiny fraction (under 10%) of the 51B cloned GitHub files?
> **Answer:** No. Out of 51B files cloned from 137M repositories, only 5B files were unique—an approximately $10\times$ exact duplication rate, meaning roughly 90% of raw cloned files were exact duplicates.

Section 10: Permissively Licensed Pretraining and the Pretraining Data Landscape

Return to the unresolved legal dilemma from Section 3: virtually every webpage on the public internet is automatically copyrighted, and while specific 2025 summary judgments treated training itself as fair use, fair use remains legally unsettled across jurisdictions and active lawsuits. That tension motivates a clean scientific question: **can we train a competitive general-purpose language model strictly on permissively licensed data?**

Common Pile (2025) was built specifically to answer this question empirically. Its creators curated an $8\ \text{TB}$ pre-training dataset exclusively from permissively licensed sources and trained the **Comma** family of models on it. In doing so, they uncovered three deep legal and provenance subtleties that make "strictly permissive" far harder to guarantee in practice than simply checking a metadata tag:

1. **License laundering**: Third parties routinely copy copyrighted works without permission and republish them online under a permissive license (such as CC-BY or MIT). Detecting laundered works reliably across terabytes of data is extraordinarily difficult at scale—meaning even a well-intentioned permissive pipeline can inadvertently ingest pirated content wearing a fake permissive label.
2. **Collection vs. item licenses**: Applying an open license to an assembled *dataset collection* does **not** automatically re-license the underlying individual documents inside it. For example, Dolma is distributed under the **ODC-By** database license, yet that collection-level license governs the compiled

database structure; it never strips away the original authors' copyrights on the individual web pages and Reddit posts contained inside Dolma.

3. **Synthetic data lineage**: As we flagged at the end of Section 8, modern pipelines increasingly use LLMs to rephrase web pages or generate synthetic Q&A pairs. If the teacher model generating those synthetic tokens was itself trained on unlicensed copyrighted data, the legal provenance—and whether the resulting synthetic dataset truly qualifies as "permissively licensed"—remains legally murky.

DEFINITION

License laundering occurs when a third party takes copyrighted material without authorization and republishes it under a permissive open license, concealing its true copyright status from automated license filters.

COMMON MISCONCEPTION

You might think that because an open pre-training dataset like Dolma is published under an open database license (ODC-By), every document inside that dataset is now permissively licensed for unrestricted downstream training. That breaks due to the distinction between **collection licenses** and **item licenses**: a dataset curator can only license their own compilation effort, never override the existing copyrights of the original authors whose individual documents were collected.

WORKED EXAMPLE

A startup wants to build a strictly permissively licensed $10\mathrm{T}$-token model to guarantee zero copyright exposure and audits three candidate data blocks:

1. **Block 1**: Pulls raw web pages directly out of Dolma because Dolma carries an ODC-By license.
   - *Audit result*: **Fails** (collection vs. item license confusion—individual web pages inside Dolma retain their original copyrights).
2. **Block 2**: Scrapes a third-party GitHub repository marked MIT that turns out to contain uploaded plain-text copies of recent commercial textbooks.
   - *Audit result*: **Fails** (license laundering—the uploader had no legal right to attach an MIT license to someone else's copyrighted textbooks).
3. **Block 3**: Runs a frontier teacher model trained on general web crawls to generate $4\mathrm{T}$ synthetic textbook tokens.
   - *Audit result*: **Unsettled** (synthetic data lineage—unclear whether derivative synthetic outputs inherit legal risk from the teacher's unlicensed training data).

Interpretation: Achieving true end-to-end permissive provenance requires verifying original item-level authorship, not merely reading top-level dataset tags, which sharply constrains available volume.

The empirical takeaway from Common Pile (2025) and the Comma models captures the central tradeoff of modern data engineering: **permissive-only models perform decently, yet they struggle to compete with frontier models without far more unique tokens** than strictly permissive sources currently supply.

Synthesis: Pretraining Corpora at a Glance

Pulling together every generation we traced from Section 6 through Section 10 shows how scale, sources, and filtering methods co-evolved from 156B tokens in 2019 to the 15T–45T era:

Dataset (Year)	Scale	Primary Sources	Filtering Method	Key Highlight
C4 (2019, hist.)	156B tok (806 GB)	1 CC snapshot (1.4T raw)	Manual rules, bad-words, `langdetect`	First large rule-cleaned CC
GPT-3 (2020, hist.)	400B tok (570 GB)	CC, WebText2, Books, Wiki	Classifier vs. WebText / Wiki / Books	Fuzzy dedup incl. eval benchmarks
The Pile (2021, hist.)	$\sim 275\text{B}$ tok (825 GB)	22 domains (CC+ `jusText`, PubMed, arXiv, Books3…)	Domain curation	WARC+ `jusText` over WET
LLaMA (2023, hist.)	1.2T tok	CCNet, C4, GitHub, Wiki, Books, arXiv, SE	CCNet + Wiki-ref classifier + rules	SlimPajama: 627B after MinHash
FineWeb (2024, hist.)	15T tok	95 CC dumps (WARC+ `trafilatura`)	URL, lang ID, Gopher+C4+custom rules	MinHash dedup + PII redaction
Dolma (2024, hist.)	3T tok	CC, Pushshift Reddit, PeS2o, C4, Wiki, Books	fastText en, Gopher+C4, Jigsaw toxicity	Bloom-filter dedup
DCLM-base (2024)	3.8T tok (from 240T)	Common Crawl	fastText classifier (OpenHermes + ELI5)	Model filter beats heuristic rules
Nemotron-CC (2024)	6.3T tok (1.1T HQ)	CC (`jusText`) + synthetic	Ensemble (DCLM + distilled 340B)	LM rephrasing & QA synthesis
The Stack (2022)	3.1 TB	GitHub (permissive MIT/Apache)	`go-license-detector`	$51\text{B} \rightarrow 5\text{B}$ unique + MinHash
Common Pile (2025)	8 TB	Strictly permissive sources	License verification & curation	Tests permissive-only scaling limit

Modern 2025–2026 token budgets—such as earlier Qwen3 (36T) and late-2026 frontier DeepSeek-V4.1-Flash (45T)—combine all of these threads simultaneously: multi-source domain curation, classifier ensembles, aggressive deduplication, and synthetic data transformation.

Step back and review the full arc of this lecture:

- **Data does not fall from the sky**: Acquiring, extracting, cleaning, and curating data is what differentiates modern language models.
- **Universal pipeline**: Live service → raw snapshot (WARC / XML dumps / `git`) → text extraction → quality filtering → deduplication → staged training mixture (pre- / mid- / post-training).
- **Legal and security constraints are first-order engineering constraints**: Nearly all web content is copyrighted, fair use remains unsettled while piracy is clearly illegal, `robots.txt` and ToS opt-outs are tightening, and even trusted curated sources face scheduled-dump poisoning and PII risks.
- **Scaling tension**: Frontier token budgets ($15\text{T} \to 36\text{T} \to 45\text{T}$) outstrip aggressive 90%-discard filters, driving higher-recall extraction, classifier ensembles, and grounded synthetic rephrasing.
- **Still heavily heuristic**: Despite massive investment, data curation remains an open engineering and research frontier.

> **KEY IDEA**
> Common Pile ($8\ \text{TB}$) demonstrates that strictly permissive pre-training is feasible yet faces license laundering, collection-versus-item license traps, synthetic lineage ambiguity, and a volume gap against 36T–45T frontier models that blend multi-source curation, classifier ensembles, and synthetic data.

Throughout this lecture, we treated several core pipeline blocks—KenLM scoring, fastText classifiers, Bloom filters, MinHash deduplication, and domain mixture weighting—as modular engineering stages while focusing on where data originates and how corpora evolved. Knowing *that* DCLM's classifier beats rules or *that* MinHash removes 37% of RedPajama leaves open the algorithmic mechanics underneath: how do you actually implement fast filtering, exact and fuzzy deduplication at multi-terabyte scale, and optimal domain mixing? Those algorithms are the subject of the next lecture, Data II.

✎ Check Your Understanding

1. **Fill-in-the-blank**: When a third party copies copyrighted works without permission and republishes them under an open license like CC-BY, that provenance hazard is called _____.
   Answer: **license laundering**.
2. **Yes/No with justification**: Does an open database license attached to a compiled corpus (such as Dolma's ODC-By license) re-license all individual web documents contained inside it as permissively licensed?
   Answer: No. A collection-level license covers only the compiled dataset structure and never overrides or re-licenses the underlying copyrights of the individual items collected inside it.