GENERATIVE CURATION YIELDS BIGGER AND BETTER DATA FOR PRETRAINING LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Modern large language models (LLMs) are pretrained largely on data curated from the internet. Because of the web's vast scale, pretraining relies on inexpensive curation: heuristic rules and classifiers to extract text from HTML and filter documents. We introduce **Generative Curation with LLMs** (GC-LLM), which replaces both stages with an LLM and a natural language specification of useful training data. The LLM follows the spec to transform raw HTML into training text or discard unsuitable documents. On a fixed pool of Common Crawl data, GC-LLM yields 11–15$\times$ as many training tokens as DCLM-baseline and Nemotron-CC. Across models with 157M, 1B, & 2.9B parameters and two LM evaluation suites, models pretrained on these tokens achieve 1.6–17.2% lower bits per byte (bpb) than those trained on either baseline. To make this approach practical at web scale, we distill GC-LLM into a fast cascade comprising FastText, a 26M-param transformer, and a 68M-param BERT model, which we call **Generative Curation with Cascade** (GC-Cascade). GC-Cascade agrees with the LLM curator on 94.6% of filtering decisions and further reduces best observed bpb by 5.0–7.3% on 1B and 2.9B models. We use this cascade to begin constructing Atlantic-CC, a projected 20T-token dataset, and release code and models to reproduce our data.

1 INTRODUCTION

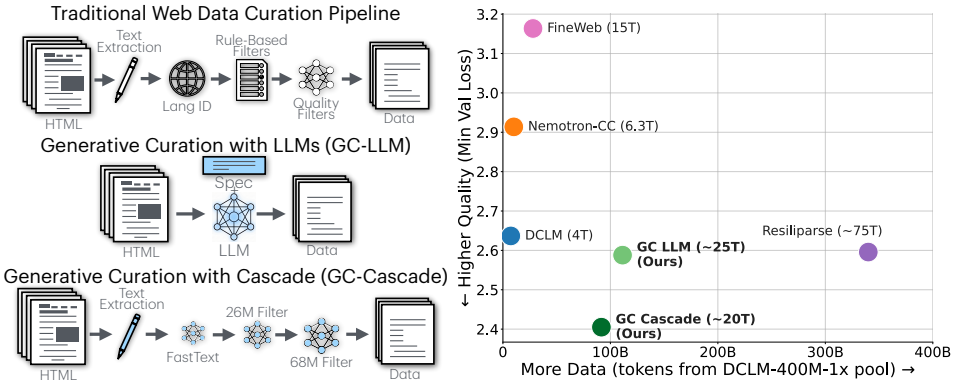

Figure 1: Traditional Web Data Curation relies on a chain of rule-based extractors and filters, while GC-LLM and GC-Cascade perform the whole process with models. Generative Curation yields more tokens and lower loss on Uncheatable Eval using the DCLM-400M-1x Pool, extending the Pareto Frontier for data curation. True and projected corpus sizes in parentheses.

The internet is a rich source of diverse data for pretraining large language models (LLMs) (Trinh & Le, 2019; Radford et al., 2019; Brown et al., 2020a). Specifically, the Common Crawl, a growing archive of websites dating back to 2008 (Common Crawl Foundation, 2026), contains over one quadrillion raw tokens of HTML and comprises a large share of the data used to train LLMs. Most of this data cannot be used directly for training: the web is full of boilerplate, spam, malformed text, and other material we would rather leave out of a training set. Filtering is therefore a necessity, but at the scale of the web, filtering itself becomes a computational problem: whatever method we use has to be cheap enough to run over trillions of tokens. As such open-source efforts like DCLM (Li et al., 2024), Nemotron-CC (Su et al., 2025), FineWeb (Penedo et al., 2024), and others (Wenzek

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

et al., 2020; Penedo et al., 2023; Soldaini et al., 2024), rely heavily on heuristic filters & lightweight classifiers. These are fast enough to operate at web scale, but they can also be imprecise, leading to cascading errors in the resulting training data (see Figure 2, left).

Instead of writing heuristic rules, what if a human could review every website, decide what is useful, and manually reformat it into training data? Although humans cannot do this at scale, strong LLMs approach human capabilities on related tasks (Zheng et al., 2023; Xu et al., 2024a). Inspired by this vision, we propose **Generative Curation**, where an LLM is provided with a document or "spec" describing useful training data and how to extract it (§3). The LLM follows the spec to transform raw HTML into training text or discard unsuitable documents. This yields both more data and better data when powered by a custom fine-tune of Qwen3 8B (See Figure 2, right).

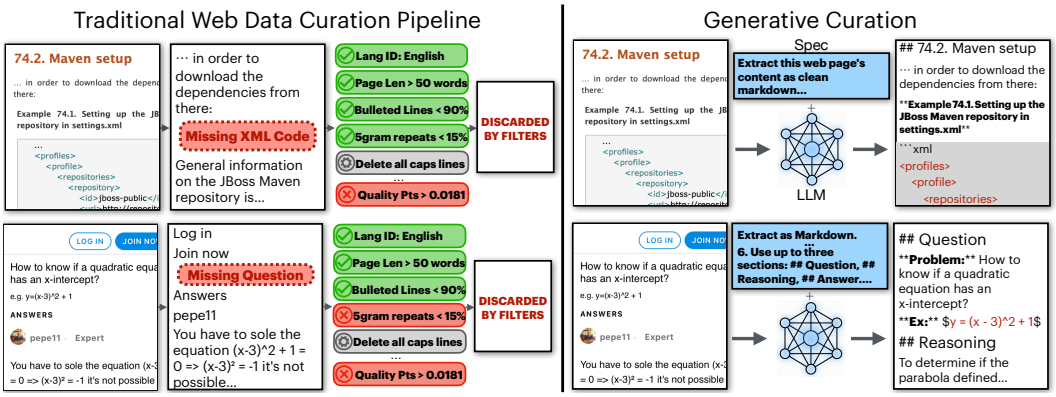

Figure 2: **Left**: Traditional data curation pipeline (here DCLM (Li et al., 2024)) drops the code tutorial due to the quality classifier and the math forum webpage due to token repetition. Both miss out on content during text extraction. **Right**: On the same pages, our spec-driven extractor retains both and, based on the spec, rephrases the math webpage into "Question → Reasoning → Answer".

Unfortunately, running even an 8B model on the entire Common Crawl would take an estimated 1.78×10^{26} FLOPs, or about half of the compute needed to pre-train GPT-4.5 (Epoch AI, 2026). Since this is so expensive, we must distill further (§4). We condense our LLM + spec into a cascade of models and an extractor: a FastText classifier (Joulin et al., 2017), a small 25.9M parameter transformer (Vaswani et al., 2017), and a 68.4M parameter variant of ModernBERT (Warner et al., 2025), plus a Rust fork of Resiliparse (Bevendorff et al., 2018), all optimized to match the outputs of the Qwen3-8B model on the target spec. Our cascade brings costs down to a reasonable 5.3×10^{21} FLOPs, closer to pretraining GPT-2 (Epoch AI, 2026).

Testing on 10,364 Web Archive (WARC) files from the Common Crawl, specifically the DCLM 400M-1x pool (Li et al., 2024), we find that models trained on our GC-Cascade data achieve 7.5 to 17% lower bits per byte (bpb) on the Olmix LM eval suite (Chen et al., 2026) compared to a set of five common Common Crawl datasets (§5). Additionally, our method curates up to 12× more tokens than these datasets on the same fixed data pool. This broad coverage allows us to tailor data mixes to specific domains and develop stronger models for the social sciences, humanities, and STEM.

In Section 6, we show that Generative Curation leads to better extraction, filtering, and data diversity than existing data curation approaches. We also address the limitations of this approach, including LLM-induced preferences (biases for certain data) and hallucinations. Our final cascade resolves hallucinations by using our Resiliparse-based extractor. Using our cascade, we begin curating Atlantic-CC: a projected 20T token dataset. We release our code and models to enable the reproduction of our data to improve open-source pretraining data quality.

2 Related Work

Prior work uses LLMs for pretraining data curation through three broad approaches: selecting useful data, transforming its content, or extracting content from raw HTML.

Perhaps the most common use of LLMs for pretraining data curation is to provide supervision for selecting useful pretraining data. The work introducing Phi-1 (Gunasekar et al., 2023) uses GPT-4

annotations of educational value to train a classifier for selecting data. FineWeb-Edu similarly scales LLM judgments through an educational-quality predictor trained on synthetic annotations from an LLM (Penedo et al., 2024; Lozhkov et al., 2024). AskLLM (Sachdeva et al., 2026) directly prompts an instruction-tuned language model to assess candidate training examples, using its predictions to rank and select data. Finally, QuRating (Wettig et al., 2024) trains a quality scorer from pairwise LLM judgments across multiple criteria and samples documents to balance quality and diversity.

Language models can transform existing documents to improve their utility for pretraining (Niklaus et al., 2026; Nguyen et al., 2025; Yu & Xiong, 2026). WRAP rephrases web text into different styles in order to train models more efficiently (Maini et al., 2024), and Nemotron-CC adds synthetic rephrasing on top of the data processing pipeline to increase diversity (Su et al., 2025). One of the most systematic explorations is HuggingFace's FinePhrase, which investigates how rephrasing formats, generator models, and source data affect downstream performance (Niklaus et al., 2026).

LLMs and neural models have also been used in extracting content from HTML. NeuScraper develops neural web extraction for pretraining corpora (Xu et al., 2024b), while ReaderLM-v2 supports HTML-to-Markdown conversion and instruction-conditioned extraction (Wang et al., 2025). Dripper uses a lightweight language model for constrained HTML block labeling (Liu et al., 2026; Ma et al., 2025). DataOrchestra learns to coordinate selection, editing, and rewriting operations for individual text chunks as a compound system of models and tools (Huang et al., 2026).

Unlike prior work, which focuses on plugging LLMs in as a single component on top of existing curation approaches, Generative Curation brings selection, extraction, and refinement together in a single LLM transformation from raw HTML to training text, governed by a shared natural-language specification. We then approximate a fixed specification with a lightweight classifier cascade and an adapted extractor, enabling large-scale curation.

3 Introducing Generative Curation with LLMs (GC-LLM)

Heuristic rules are tricky to get right for web data curation. As Figure 2 shows, rule-based extraction and filtering often break down when applied to new content domains: the web's diversity makes it hard to anticipate every edge case, even when the right decision is easy to recognize from context. We instead propose Generative Curation: using an LLM and a natural-language spec to both filter and transform the data. The spec describes the desired behavior at a high level, while the LLM uses context to determine how to apply it to each example. Though the concept is simple, there are three components that require careful design: (1) the LLM, (2) the scaffolding for the LLM, (3) the spec.

3.1 Model Selection

Desiderata. We evaluate LLMs for Generative Curation on two properties. *Content preservation* asks if the output faithfully reproduces the content without dropping or adding material. *Spec adherence* tests if the output follows the formatting, filtering, and transformation in the specifications. We measure both by having `gemini-3-flash-preview` judge the extraction against the source HTML and the spec. Judge prompts are in Appendix A.

Method	Article extr. Token F1 ↑	Custom benchmark Cont. pres. ↑	Custom benchmark Spec adh. ↑
Existing Extractors			
Trafilatura	**0.940** ± 0.01	0.320 ± 0.06	—
Resiliparse	0.868 ± 0.01	**0.960** ± 0.03	—
LLMs (< 20B)			
Qwen3-0.6B	0.485 ± 0.03	0.129 ± 0.02	0.136 ± 0.02
Qwen3-0.6B (Ours)	0.904 ± 0.01	0.575 ± 0.03	0.561 ± 0.03
Qwen3-1.7B	0.820 ± 0.02	0.386 ± 0.03	0.361 ± 0.03
Qwen3-1.7B (Ours)	0.921 ± 0.01	0.761 ± 0.03	0.675 ± 0.03
Qwen3-8B	0.907 ± 0.01	0.825 ± 0.02	0.800 ± 0.02
Qwen3-8B (Ours)	**0.930** ± 0.01	**0.900** ± 0.02	**0.900** ± 0.02
LLMs (≥ 20B)			
GPT-OSS-20B	0.908 ± 0.01	0.871 ± 0.02	0.836 ± 0.02
GPT-OSS-120B	0.942 ± 0.01	0.950 ± 0.01	0.904 ± 0.02
GPT-5.1	0.957 ± 0.01	0.961 ± 0.01	0.957 ± 0.01
Kimi-K2.5-1T	**0.969** ± 0.00	**0.964** ± 0.01	**0.993** ± 0.00

Table 1: Token F1, Content Preservationm & Spec Adherence pass rates extraction benchmarks. Distilled Qwen3-8B (Ours) leads in its class and approaches the GPT-OSS-120B teacher on all metrics. ± 1000-sample bootstrap stdev.

Evals. We use the article extraction benchmark from Zyte (Lopuhin, 2020), which provides token-level ground truth for content preservation across 181 pages, but cannot measure spec adherence. We also create a custom benchmark of 50 HTML pages spanning seven categories (10 articles, 5 code, 5 math, 10 scientific, 5 tables, 5 long, 10 generic) paired with 11 specs drawn from prior work such as Rephrase the Web (Maini et al., 2024) and Nemotron-CC-Math (Mahabadi et al., 2026), for a total of 550 (page, spec) pairs. Qwen3's 32k context limits us to 124 / 181

article-extraction pages and 280 / 550 custom pairs. We eval everything on the same set of documents. We test Trafilatura (Barbaresi, 2021), Resiliparse (Bevendorff et al., 2018), GPT-OSS-120B, GPT-OSS-20B, Kimi-K2.5, GPT-5.1, and the off-the-shelf Qwen3 {8B, 1.7B, 0.6B}. We also distill Qwen3 {8B, 1.7B, 0.6B} using GPT-OSS-120B as a teacher (See Appendix B for details).

Results. Table 1 shows that distillation from GPT-OSS-120B helps at every Qwen3 scale. Our final 8B distill is best-in-class below 100B parameters, even outperforming GPT-OSS-20B despite using 2.5× fewer params. Its content preservation matches Resiliparse, and its article-extraction Token F1 matches trafilatura. Unlike these rule-based extractors, however, it also achieves 90% spec adherence. Kimi-K2.5 leads with near-perfect spec adherence, but at a high cost. Considering these tradeoffs, we use our Qwen3-8B distill for the remainder of this report.

3.2 Scaffolding

Beyond the model, we also consider how to present the context and call the model. We use a DSPy-compatible prompt template (Khattab et al., 2024). We split curation into two specs, one for filtering and one for extraction; Appendix I shows an example of an earlier single-spec system for comparison. First, we use Resiliparse to prove plain text to the model and the filtering spec decides whether the document belongs in the corpus, (24k tokens input, 4k tokens for reasoning). Documents that survive this filter are passed to the extraction spec, which extracts training text from the raw HTML, limited to the `<body>` tags with `<script>` blocks removed. Extraction is performed without reasoning, in 18k-token chunks of input HTML. [1] In total, a kept document takes a minimum of two calls and an average of 3.05.

3.3 Spec Selection

Spec changes enable rapid iteration on the system's filtering and extraction behavior. We sampled 1,934 HTML pages from Common Crawl spanning 2013 to 2026, stratified across science, medical, Q&A forums, news, how-to, commerce, fiction, code, navigation, and index pages/junk using a keyword search. An author labeled all pages as "keep" or "drop". 134 pages were flagged as top-quality, with the intention that they should certainly be kept. We set aside 585 pages (30 top-quality) as a held-out test set, and use the remainder to iterate on the spec. The resulting spec is quite permissive with the philosophy of optimizing for coverage of all forms of informative web content. On the test split, the two-spec system reaches 93.4% keep/drop accuracy with 100% of top-quality pages kept. We also produce gold extractions with Claude Sonnet 5, Gemini 3 Flash, and DeepSeek V4, and adjudicate disagreements with Opus 4.8 on 420 pages to test extraction. Our extraction spec reaches 0.805 token F1 agreement with our gold extractions. This filter and extraction pair is the spec behind every Generative Curation corpus in the remainder of the paper. Aligning with human judgment is a unique strength of Generative Curation. The full specs are in Appendix D.

4 Converting Generative Curation into a Cascade (GC-Cascade)

Running Qwen3-8B even in just the single-spec setting would cost 1.78×10^{26} FLOPs on the full Common Crawl. We distill this into an efficient model cascade that completes it in 5.3×10^{21} FLOPs, while maintaining our simplicity over existing data curation pipelines. First, we distill the *extractor* into a rule-based program that reproduces the spec's output on the pages the spec keeps. Then we distill the *filter* into a cascade of classifiers that reproduce the spec's keep/drop decisions.

Distillation Data. We run the two-spec system (§3) over the 10,364 WARC files of DCLM 400M-1x pool (Li et al., 2024). This produces extraction/filtering decisions on ∼400 million pages, which we sample to train our models. We randomly select 200 WARC files, at least one from every crawl snapshot from 2013 to 2022, for evaluation.

Distilling the Extractor. We set aside 12,000 documents: 10k/1k/1k for train/dev/test. We start from Resiliparse (Bevendorff et al., 2018), a library for HTML-to-text extraction, and use Claude Fable 5 to port its extractor from Cython to Rust. First, we ensure the Rust port outputs identical

[1] Chunks are split at HTML block boundaries, never inside a code block, table row, or math span. Because chunked extraction without reasoning introduces more errors than short-document extraction with reasoning enabled, we run automatic guardrail checks and trigger repairs on failure; see Appendix Table 5.

text to Cython on the training data. Then we set Fable to optimize the extractor in an agent-driven research loop. Each cycle, the agent categorizes failures, forms a hypothesis to test, edits the extractor, re-scores the 1,000 dev pages, and benchmarks extraction efficiency. After 179 cycles, the agent introduced Markdown formatting, comment forum parsing, and a feature-based classifier for deleting HTML blocks, bringing ROUGE-5 F1 agreement with Qwen3-8B from 0.625 to 0.729.

Distilling the Filter. We selected five pretrained encoders and three transformers trained from scratch and trained on 1M documents to inform our architecture selection (Appendix E). Of these eight models, we selected the two on the Pareto frontier of throughput and accuracy to scale to more data: Ettin-68M (Weller et al., 2026), a smaller ModernBERT (Warner et al., 2025) variant, and a 25.9M-parameter *pooled transformer* trained from scratch that compresses each 64-token window into one vector before attention, scaling down its cost in context length. We also used FastText (Joulin et al., 2017) for efficient CPU-based classification alongside extraction. We swept the following configurations, plotted in Figure 3: **Input representation** (HTML or extracted text) for the selected models; **Context length** (1k, 2k, 4k, and 8k tokens) for the ModernBERT family; **Data size** (1M and 10M documents for Ettin-68M and ModernBERT-base; 1M, 10M, and 90M for the pooled transformer); and **Data diversity** (3.4–6.7M documents drawn from 80, 160, 256, or 640 WARC files) for FastText.

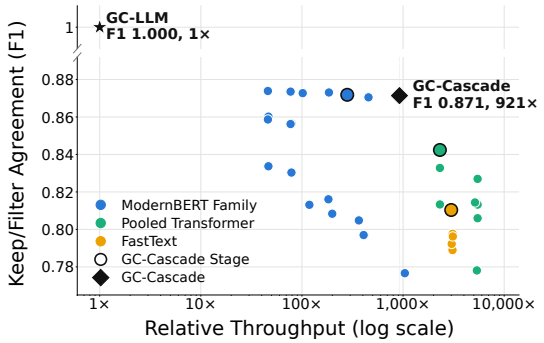

Figure 3: Relative Throughput and Agreement of our classifiers versus the full GC-LLM Qwen 8B system. We build our cascade by taking a Pareto-dominant config from each model family and tuning thresholds so they work together. Our final cascade achieves an F1 of 0.871 with 921× the throughput.

Figure 3 shows the sweep and selected configurations for each family. The deployed pipeline chains the best configuration of each family and tunes the thresholds jointly. The final GC Cascade consists of the FastText classifier trained on 640 diverse WARC files for text input, the Pooled Transformer trained on 90M pages with a context length of 8k tokens, and the 68M-parameter Ettin variant of modernBERT with a 2k-token context length. Although pooled transformer variants Pareto-dominate FastText, we still include it because it runs on CPU, which lets us massively parallelize that segment of the pipeline across devices. After tuning the thresholds, it reaches 0.871 F1 agreement with our GC-LLM system while achieving 921× the throughput.

5 Generative Curation Trains Better Models

5.1 Experimental Setting

We now showcase how training on our data leads to better models in several domains. We train over 500 models of various parameter counts, data mixes, and token budgets.

5.1.1 Baseline Datasets

For baseline data pipelines, we collect the released datasets and filter to the DCLM 400M-1x Pool of WARC files (2013-2022). We include five baseline datasets. **DCLM-Baseline** dataset (Li et al., 2024) underwent URL filtering, language ID, heuristic filtering, deduplication, and model-based filtering. We use the data from the v1 release of the **Nemotron-CC** dataset (Su et al., 2025), processed with a similar pipeline but with additional synthetic data rewriting. **Fineweb** is a larger and less curated set of Common Crawl data (Penedo et al., 2024). **FineWeb-Edu** is a quality subset of Fineweb (Lozhkov et al., 2024). The authors used Llama 3 70B-Instruct to score FineWeb for educational quality and distilled that judgment into a small embedding-based linear regressor. Using this, the authors reduce FineWeb to the best 1.3T tokens. **Resiliparse** (Bevendorff et al., 2018) is a rule-based HTML-to-text conversion tool used in data pipelines like DCLM. As a "no-filtering" baseline, we convert all WARC files' HTML to text with Resiliparse and train on it all.

5.1.2 Data Processing

We use the DCLM 400M-1x WARC Pool as a fixed curation pool, a list of WARC files spanning 2013-2022. For GC-LLM, we use the spec we selected in Section 3, and for GC-Cascade, we use the system described in Section 4. We extract all of the files in this pool with Resiliparse. For all other methods, we use the released datasets and filter to match the WARCs in the DCLM pool.

We next turn to deduplication. The baselines already include deduplication in their pipelines, but Resiliparse and our Generative Curation need it. For both, we do exact-match deduplication, followed by MinHash-LSH connected-components fuzzy deduplication with 286 permutations in 26 bands over 5-character n-grams, targeting a Jaccard similarity threshold of 0.75.

We decontaminate Resiliparse and Generative Curation against 21 benchmarks in DCLM's CORE eval (Li et al., 2024). Each eval item is rendered as its question and gold answer. We drop any document that shares a 15-gram with an eval item. Following GPT-3 (Brown et al., 2020b), a 15-gram that appears in more than 10 docs of the corpus is treated as ubiquitous text rather than leakage, so documents whose only matches are such n-grams are retained. This removes 0.01% of docs.

Last, rather than training on each corpus at its natural distribution, we re-weight it with Olmix (Chen et al., 2026), applied independently to every pipeline. This matters because evaluating on natural mixes forces an a priori decision about which data type to filter to. For instance, evaluating on math means the optimal curation decisions would cut non-math data. By mixing the data for our evals post-filtering, we are not penalized for retaining documents like literature, recipes, etc. To perform mixing we partition each corpus into 120 cells crossing 24 WebOrganizer topics (Wettig et al., 2025) with five quality bins (Held, 2026b), train a swarm of 363 small proxy models on sampled mixtures of these cells, fit a per-task law to scores on the 51-task Olmix evaluation suite, and solve for the mix that minimizes predicted score subject to a KL penalty toward the natural distribution and a cap on how many times any cell may be repeated. We additionally solve one mix per MMLU category (STEM, Humanities, Social Science) from the same swarm, fitting to the mean choice log-probability over that category's subjects. Because these objectives pool far fewer tasks, the fitted laws are noisier, so we regularize the solve more strongly. Full settings are in Appendix F.

5.1.3 Training and Evaluation

We choose 3 parameter sizes from the Delphi scaling suite (Held, 2026a): 157M, 998M, and 2.9B (hidden dimensions 512, 1536, and 2432), using a Qwen3 architecture with the Llama 3 tokenizer. Using the heuristic FLOPs $\approx 6ND$ for N parameters and D tokens (Kaplan et al., 2020), we set 12 distinct token budgets from $1e17$ to $9e20$ FLOPs. Filtering to the DCLM pool leaves the baselines with just 7.3B (DCLM) and 10.1B (Nemotron) tokens. We still train every dataset at every budget, epoching whenever a budget needs more tokens than a dataset has. This simulates full scale, where a frontier model trained on 15T+ tokens would epoch over DCLM (3.8T) or Nemotron-CC (6.3T) several times. Since such models are beyond our compute budget, we shrink the data instead, and expect larger models on the full datasets to saturate the same way. We use the AdamH hyperparameters derived for the Delphi suite at each model size and compute budget (Appendix G).

For evaluation, we compute validation loss on Uncheatable Eval (Jellyfish042, 2024), a constantly changing corpus of recent high-quality tokens from BBC News, arXiv, GitHub, Wikipedia, and similar sources intended to postdate LLM training data cutoffs. In particular, we evaluate on the September 1–14, 2025 split. We also compute and average 51 tasks from the Olmix evaluation suite (Chen et al., 2026), comprising 19 coding tasks, 7 math tasks, 15 question-answering and reading-comprehension tasks, 6 basic-skills probes, and four MMLU categories. Every task is scored as bits per byte of the gold answer, and we report the unweighted mean over the 51.

5.2 Results

GC-Cascade gets 7.5–17% lower best observed bpb on Olmix Evals than baselines. Figure 4 shows bits per byte (BPB) on the Olmix evals, and Figure 7 shows results for Uncheatable Eval. Across all compute budgets, GC-Cascade achieves the lowest BPB. GC-LLM similarly outperforms baseline datasets, though it requires larger-scale budgets to outperform DCLM.

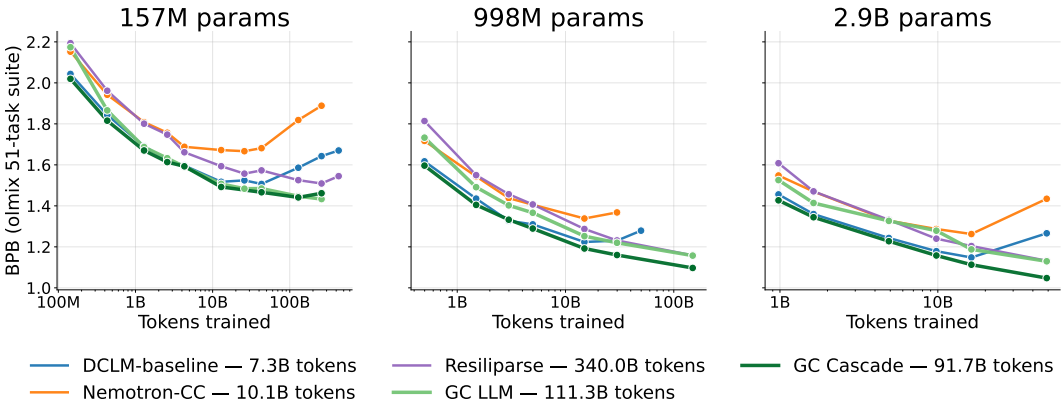

Figure 4: Olmix eval bpb on data mixes tuned for Olmix evals. All data from DCLM-400M-1x pool. Across all model scales and token budgets, our GC-Cascade achieves the lowest bpb of any curation strategy. FineWeb is not pictured because it filters code data, which makes it dramatically worse on our evals. Additionally, methods that curate fewer tokens degrade when trained on more tokens (due to epoching), but GC-LLM, GC-Cascade, and Resiliparse push the data frontier.

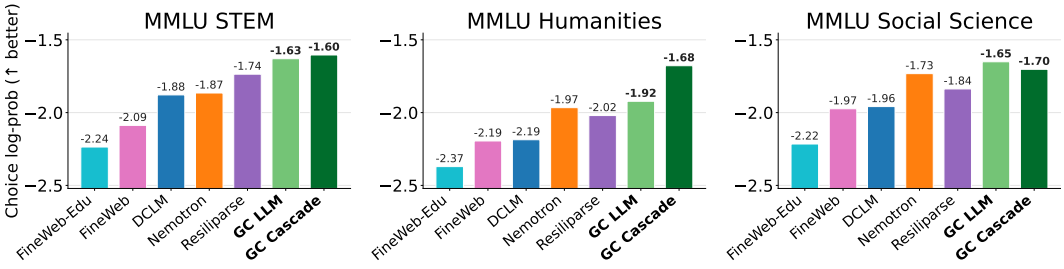

Figure 5: Highest MMLU choice log-probability per target domain at the 1B scale (higher is better) swept from 3e18 to 3e20 FLOPs. Each plot represents training on a different data mix optimized for that domain. GC-LLM or GC-Cascade methods dominate the top spot in every domain.

More tokens push the data frontier. Common datasets like DCLM and Nemotron begin to saturate around 20B tokens, since this fixed pool only affords them around 10B unique tokens, after which they need to epoch. Meanwhile, our datasets continue to improve well into the 100B token range. Although we impose these token limits, this reasonably simulates what would happen at higher data-limited budgets. Scaled up, this will enable training models well beyond the data limit of current released corpora (between 5-15T tokens) to regimes like Qwen3's 36T-token run (Yang et al., 2025).

Generative Curation enables more diverse data mixes. In Figure 5 we showcase the results of mixing each dataset towards MMLU domains: STEM, Humanities, and Social Science. For each, we report the average choice log-probability of correct answers, which Held et al. (2026) finds to be the most predictive of accuracy at scale. We find that our Generative Curation data is best across all domains. We attribute this steerability to the high coverage of data that we target with our spec (§3).

6 Analyzing Generative Curation

Section 5 shows that Generative Curation trains better models. Here we ask why, by checking each of the three things the spec asks the LLM to do: extract the page, decide whether to keep it, and transform it into training text. We also address failure modes of Generative Curation.

6.1 Better Extraction

We evaluate extraction using our 412-document gold dev set (§3) and WebMainBench (Liu et al., 2026), a collection of 7,809 human-annotated pages indicating which DOM blocks should remain and ultimately be extracted with html2text (Swartz). We use ROUGE-5 scores for both benchmarks, and specifically for our devset we measure recall of specific content we have flagged as important (e.g., attributions such as in comments sections, math, code, tables). For all of our GC-LLM runs, we use the **two-stage** config with the same spec described in (§3).

Method	WebMainBench (n=7,809) F1 ↑	P ↑	R ↑	Our Devset (n=412) F1 ↑	P ↑	R ↑	Our Devset: Target Content Recall Attr ↑	Code ↑	Math ↑	Table ↑
Existing Extractors										
Trafilatura	0.609	0.695	0.590	0.753	0.847	0.735	0.398	0.793	0.440	0.766
Resiliparse	0.628	0.602	0.725	0.817	0.787	0.880	0.836	0.933	0.823	0.895
MinerU-HTML	**0.900**	**0.893**	**0.945**	0.815	0.812	0.845	0.362	0.908	0.823	0.966
Generative Curation with LLMs (GC-LLM)										
Qwen3-8B Distill	0.646	0.614	0.753	0.805	0.784	0.845	0.973	0.924	0.859	0.949
GPT-OSS-120B	0.658	0.642	0.726	0.755	0.753	0.783	0.908	0.845	0.855	0.864
GLM-5.3	0.708	0.693	0.769	**0.895**	**0.887**	**0.917**	**0.984**	**0.942**	**0.900**	**0.988**
Generative Curation with Cascade (GC-Cascade)										
Resiliparse (Our fork)	0.696	0.663	0.795	0.828	0.797	0.889	0.918	**0.942**	0.823	0.972

Table 2: Extraction quality on WebMainBench (WMB) and our devset. Generative curation leads every devset column and recovers the most target content. On WMB, it beats all baselines except MinerU-HTML, which was developed alongside WMB and makes similar editorial decisions.

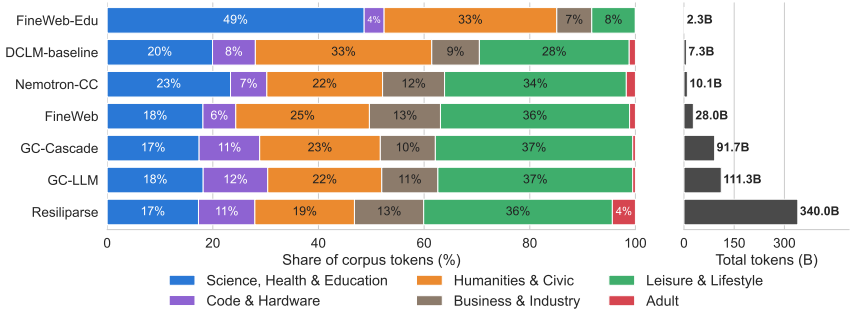

Figure 6: By filtering less aggressively (right), our data produces similar topic distributions to the web (demonstrated by Resiliparse), while other datasets produce more skewed distributions (left).

Generative Curation recovers content that existing extractors drop. Table 2 shows results on WebMainBench and our devset for all extractors. On our devset, the Cascade and GLM-5.3 lead. Trafilatura keeps 40% of comment attributions and 44% of math. MinerU-HTML keeps 36% of attributions, while every Generative Curation extractor keeps over 90% of attributions and over 82% of math. MinerU-HTML performs best on WebMainBench; however, MinerU-HTML was developed alongside WebMainBench and thus makes similar decisions on edge cases. Our extractors outperform Trafilatura and Resiliparse on WebMainBench. We also find that **stronger models extract better**. GLM-5.3 is the best extractor on our devset and second on WebMainBench. This suggests that better language models will improve Generative Curation's efficacy.

6.2 Better Coverage

The web contains far more than math, code, and science, but today's evaluations reward filtering toward those areas (Li et al., 2024). Generative Curation instead keeps every page a reader would find informative and leaves domain emphasis to mixing (§5).

Figure 6 shows the distribution of topics covered in the datasets we compare. We use WebOrganizer (Wettig et al., 2025) to bucket data into 24 buckets, which we group into these six categories. Here, resiliparse serves as a useful representation of the true distribution of data on the web. Our systems share similar topic coverage to the web for all domains besides adult content. This contrasts with data like FineWeb-Edu, which is very opinionated about what counts as high-quality content. Using a simple URL match between kept documents, we determine that our data contains at least 88.3% of the documents in DCLM and 89.8% of the documents in FineWeb Edu. This breadth lets mixing target any domain, and it is consistent with our leading every MMLU category in Figure 5.

A natural worry for GC-LLM is that an LLM may collapse pages into templated, formulaic text, trading the web's diversity for cleanliness. We test this on 1,000 pages kept by both GC systems, using the self-repetition score (SRS): the rate of 4-grams repeated across documents (Salkar et al., 2022). We use the SRS implementation from Shaib et al. (2025). We also rephrase the same pages' Resiliparse text with four WRAP prompts (Maini et al., 2024): Q/A, Easy, Medium, and Hard.

6.3 Better Diversity

Table 3 contains the self-repetition score for extractions using our spec versus rewrites from WRAP. On the same docs, Trafilatura scores 2.43 $\pm$ 0.19; Resiliparse scores 3.07 $\pm$ 0.18; and our GC Cascade Resiliparse Fork scores 2.78 $\pm$ 0.19. Our extractions have a self-repetition similar to the rule-based extractors, all within one standard deviation or below. Meanwhile, LLM rephrases are much less diverse, sometimes reaching 3.3×. This suggests that our extractions preserve the page's diversity rather than rewriting/rephrasing the text into a less diverse form.

Model	Our Spec	WRAP Rephrasing Q/A	Easy	Med	Hard
Qwen3-8B SFT	2.66	8.82	7.17	6.35	4.99
GPT-OSS-120B	2.52	8.06	4.53	4.19	2.70
GLM-5.3	2.68	7.02	5.55	4.72	4.75

Table 3: Self-repetition score of extracted/rephrased text on 1k pages. Extraction with our spec is close to rule-based (∼2.75), while rephrasing often raises cross-doc repetition 1.6-3.3×.

6.4 Limitations

We examine two GC-LLM risks: *hallucination*, when the model writes text the page does not contain, and *instruction-following failure*, when the model decides on data irrespective of the spec.

Hallucinations risk introducing incorrect information into the pretraining dataset. We use the same 1,000 documents from our diversity check. We first find n-gram sequences that appear in the extraction but not in the HTML, which we flag and pass to gemini-3-flash as a judge to mark hallucinations. We define four types of issues. **Major**: invented claims, quotes, code, or facts that would mislead, or a corruption that changes a fact, quantity, name, or link target. **Medium**: a plausible detail the page never states (a name, number, date, or attribution), or a typo from the extractor. **Minor**: filler that adds no claim, such as an invented "Introduction" heading. **Whitespace**: the page's own words with spaces lost where HTML elements were joined.

Method	Hallucinations ↓ Major	Med	Minor	Spacing Errors
Rule-Based Extraction				
Trafilatura	0.0%	1.3%	0.7%	11.5%
Resiliparse	0.0%	1.9%	1.9%	13.6%
GC Cascade	0.1%	3.1%	1.7%	17.8%
LLM-Based Extraction				
Qwen3-8B Distill	1.6%	5.4%	8.5%	4.0%
GPT-OSS-120B	2.0%	7.1%	7.3%	5.4%
GLM-5.3	0.3%	2.9%	2.2%	1.9%

Table 4: Share of 1,000 Common Crawl pages with hallucinations and spacing issues, by extraction method, judged by gemini-3-flash-preview. Among LLM extractors, stronger models have lower hallucination rates, suggesting this limitation is mitigated by scale.

Table 4 reports the share of pages with at least one finding of each kind. The few hallucinations from Rule-Based extractors come from things like misattributing a forum quote to the wrong user. The LLMs' failures can be more serious, such as inventing routes in API documentation or entering repetition loops. Our distillation from GC-LLM to GC-Cascade resolved this issue. Notably, GLM-5.3 had significantly fewer hallucinations, suggesting this is a limitation that may go away with scale. Still, fewer than 2% of pages had major errors even with our Qwen3-8B distilled model.

Bias arises when the model makes editorial decisions about how to curate the data that were not covered by the spec (or the model deliberately disregards the spec). In our experience, this was most likely to happen with refusals surrounding unsafe content or adult content. Occasionally the model refuses to extract a page. In practice, this might be a good signal to trust a model that the data is low quality if it refuses to extract it; however, LLM preferences may taint data curation.

7 Conclusion

Web curation pipelines have relied on chains of fast but imprecise rule-based extractors and filters, each step compounding the last, resulting in pretraining datasets stripped of useful web content. We introduce GC-LLM, which uses an LLM and a natural language spec to curate data, enabling a single readable document to debug and improve rather than rules spread across code and classifier weights. In order to run Generative Curation more efficiently, we distill it into a cascade small enough to run on the full Common Crawl (just 94M parameters total). Applied to the full Common Crawl, GC-Cascade would produce a projected 20T tokens (25% complete). This will push the frontier of open data availability. We find that models trained on our data achieve between 7.5% and 17% lower bpb on Olmix compared to models trained on existing datasets. We further validated the extraction quality, coverage, and diversity of our data. We release our code and models to enable reproduction of our data and to contribute to improved open-source pretraining data scale and quality.

AI USE STATEMENT

In this work, we used generative AI tools for implementing methods with AI coding agents, cleaning datasets (as described in Section 4), developing figures, searching for related literature, editing human-written content for the main paper, drafting content for the appendix, and formatting bibtex for non-academic citations such as github repositories. We have not used generative AI tools during the ideation phase of this project, in devising our methodology, in proposing hypothesis, or in interpreting results. The rest of the required disclosure tasks are not applicable to this work. The lead author has reviewed all AI-assisted work. The methods were carefully tested against human-designed test conditions. The 420 cleaned extractions were manually spot checked. The figure and writing process was heavily human-in-the-loop as the content was being produced. The citation information was manaully validated. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REPRODUCIBILITY STATEMENT

We take great care to ensure reproducibility of our data and results. We plan to release the source code for all of our experiments upon publication. In addition to the code, we describe the details of our method at length in Section 3 and Section 4, with the Qwen3 distillation in Appendix B, the extraction guardrails in Appendix C, and the cascade architecture sweep and classifier training in Appendix E. We provide experimental details for reproducing our setting in Section 5, the data mixing procedure in Appendix F, and the training hyperparameters for every model size and compute budget in Appendix G. We include both curation specs in Appendix D and the prompts for every LLM judge in Appendix A. In addition to our source code and experimental details we will also be releasing metadata pointing to the precise HTML documents in the WARC files which our GC-Cascade preserved when running over the entire Common Crawl, as well as the source code to our efficient Resiliparse fork. Together these artifacts will enable the reproducibility of our Atlantic-CC dataset without us rehosting content.

REFERENCES

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and Charles Sutton. Program synthesis with large language models, 2021. URL https://arxiv.org/abs/2108.07732.

Adrien Barbaresi. Trafilatura: A Web Scraping Library and Command-Line Tool for Text Discovery and Extraction. In *Proceedings of the Joint Conference of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pp. 122–131. Association for Computational Linguistics, 2021. URL https://aclanthology.org/2021.acl-demo.15.

Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. Elastic ChatNoir: Search Engine for the ClueWeb and the Common Crawl. In Leif Azzopardi, Allan Hanbury, Gabriella Pasi, and Benjamin Piwowarski (eds.), *Advances in Information Retrieval. 40th European Conference on IR Research (ECIR 2018)*, Lecture Notes in Computer Science, Berlin Heidelberg New York, March 2018. Springer.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 1877–1901. Curran Associates, Inc., 2020a. URL https://proceedings.neurips.cc/paper_files/paper/2020/file/1457c0d6bfcb4967418bfb8ac142f64a-Paper.pdf.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal,

Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020b. URL https://arxiv.org/abs/2005.14165.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. Evaluating large language models trained on code. 2021.

Mayee F Chen, Tyler Murray, David Heineman, Matt Jordan, Hannaneh Hajishirzi, Christopher Re, Luca Soldaini, and Kyle Lo. Olmix: A framework for data mixing throughout LM development. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=8pOl3azhbL.

Common Crawl Foundation. Common crawl web archive. https://commoncrawl.org, 2026. Accessed: 2026-09-09.

Epoch AI. Data on AI Models, 9 2026. URL https://epoch.ai/data/ai-models. Accessed: 15 Sep 2026.

Sachin Goyal, Pratyush Maini, Zachary C. Lipton, Aditi Raghunathan, and J. Zico Kolter. Scaling laws for data filtering—data curation cannot be compute agnostic. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 22702–22711, 2024. doi: 10.1109/CVPR52733.2024.02142.

Suriya Gunasekar, Yi Zhang, Jyoti Aneja, Caio César Teodoro Mendes, Allie Del Giorno, Sivakanth Gopi, Mojan Javaheripi, Piero Kauffmann, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Harkirat Singh Behl, Xin Wang, Sébastien Bubeck, Ronen Eldan, Adam Tauman Kalai, Yin Tat Lee, and Yuanzhi Li. Textbooks are all you need, 2023. URL https://arxiv.org/abs/2306.11644.

Will Held. Scaling laws that extrapolate 300× past the fit. https://www.openathena.ai/blog/delphi/, may 2026a. Open Athena Blog.

Will Held. Curating 25 trillion tokens for llm pretraining. https://www.openathena.ai/blog/marin-data-pipeline-overview/, sep 2026b. Open Athena Blog.

William Barr Held, David Leo Wright Hall, Percy Liang, and Diyi Yang. Relative scaling laws for LLMs. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=zqmS3yzV85.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. Measuring massive multitask language understanding, 2021. URL https://arxiv.org/abs/2009.03300.

Zhen Huang, Yikun Wang, Shijie Xia, and Pengfei Liu. Dataorchestra: Learning to orchestrate per-example curation of pretraining data, 2026. URL https://arxiv.org/abs/2607.24717.

Jellyfish042. Uncheatable eval, May 2024. URL https://zenodo.org/record/11284692.

Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers*, pp. 427–431. Association for Computational Linguistics, April 2017.

Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B. Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models, 2020. URL https://arxiv.org/abs/2001.08361.

Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. Dspy: Compiling declarative language model calls into self-improving pipelines. 2024.

Aitor Lewkowycz, Anders Johan Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Venkatesh Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, Yuhuai Wu, Behnam Neyshabur, Guy Gur-Ari, and Vedant Misra. Solving quantitative reasoning problems with language models. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho (eds.), *Advances in Neural Information Processing Systems*, 2022. URL https://openreview.net/forum?id=IFXTZERXdM7.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Kumar Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee F Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, Yonatan Bitton, Marianna Nezhurina, Amro Kamal Mohamed Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Joshua P Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah M Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Chandu, Thao Nguyen, Igor Vasiljevic, Sham M. Kakade, Shuran Song, Sujay Sanghavi, Fartash Faghri, Sewoong Oh, Luke Zettlemoyer, Kyle Lo, Alaaeldin El-Nouby, Hadi Pouransari, Alexander T Toshev, Stephanie Wang, Dirk Groeneveld, Luca Soldaini, Pang Wei Koh, Jenia Jitsev, Thomas Kollar, Alex Dimakis, Yair Carmon, Achal Dave, Ludwig Schmidt, and Vaishaal Shankar. Datacomp-LM: In search of the next generation of training sets for language models. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=CNWdWn47IE.

Mengjie Liu, Jiahui Peng, Wenchang Ning, Pei Chu, Jiantao Qiu, Ren Ma, He Zhu, Rui Min, Lindong Lu, Linfeng Hou, Kaiwen Liu, Yuan Qu, Zhenxiang Li, Chao Xu, Zhongying Tu, Wentao Zhang, and Conghui He. Dripper: Token-efficient main html extraction with a lightweight lm, 2026. URL https://arxiv.org/abs/2511.23119.

Konstantin Lopuhin. Article Extraction Benchmark, 2020. URL https://github.com/scrapinghub/article-extraction-benchmark.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. Fineweb-edu: the finest collection of educational content, 2024. URL https://huggingface.co/datasets/HuggingFaceFW/fineweb-edu.

Ren Ma, Jiantao Qiu, Chao Xu, Pei Chu, Kaiwen Liu, Pengli Ren, Yuan Qu, Jiahui Peng, Linfeng Hou, Mengjie Liu, Lindong Lu, Wenchang Ning, Jia Yu, Rui Min, Jin Shi, Haojiong Chen, Peng Zhang, Wenjian Zhang, Qian Jiang, Zengjie Hu, Guoqiang Yang, Zhenxiang Li, Fukai Shang, Runyuan Ma, Chenlin Su, Zhongying Tu, Wentao Zhang, Dahua Lin, and Conghui He. Aicc: Parse html finer, make models better – a 7.3t ai-ready corpus built by a model-based html parser, 2025. URL https://arxiv.org/abs/2511.16397.

Rabeeh Karimi Mahabadi, Sanjeev Satheesh, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC-math: A 133 billion-token-scale high quality math pretraining dataset. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=rhPnkTKfMy.

Pratyush Maini, Skyler Seto, Richard Bai, David Grangier, Yizhe Zhang, and Navdeep Jaitly. Rephrasing the web: A recipe for compute and data-efficient language modeling. In Lun-Wei Ku,

Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 14044–14072, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.757. URL https://aclanthology.org/2024.acl-long.757/.

Christopher Mohri, John Duchi, and Tatsunori Hashimoto. A bitter lesson for data filtering, 2026. URL https://arxiv.org/abs/2605.19407.

Niklas Muennighoff, Alexander M. Rush, Boaz Barak, Teven Le Scao, Aleksandra Piktus, Nouamane Tazi, Sampo Pyysalo, Thomas Wolf, and Colin Raffel. Scaling data-constrained language models, 2025. URL https://arxiv.org/abs/2305.16264.

Thao Nguyen, Yang Li, Olga Golovneva, Luke Zettlemoyer, Sewoong Oh, Ludwig Schmidt, and Xian Li. Recycling the web: A method to enhance pre-training data quality and quantity for language models. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=lkjhBdz3rn.

Joel Niklaus, Guilherme Penedo, Hynek Kydlicek, Elie Bakouch, Lewis Tunstall, Ed Beeching, Thibaud Frere, Colin Raffel, Leandro von Werra, and Thomas Wolf. The synthetic data playbook: Generating trillions of the finest tokens, 2026.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. The refinedweb dataset for falcon LLM: Outperforming curated corpora with web data only. In *Thirty-seventh Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2023. URL https://openreview.net/forum?id=kM5eGcdCzq.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Neil Rathi and Alec Radford. Shaping capabilities with token-level data filtering, 2026. URL https://arxiv.org/abs/2601.21571.

Noveen Sachdeva, Benjamin Coleman, Wang-Cheng Kang, Jianmo Ni, Lichan Hong, Ed H. Chi, James Caverlee, Julian McAuley, and Derek Cheng. How to train data-efficient llms. In C. Vondrick, B. Hariharan, C. Raffel, L. Pinto, D. Yang, and A. Faust (eds.), *International Conference on Learning Representations*, volume 2026, pp. 45491–45545, 2026. URL https://proceedings.iclr.cc/paper_files/paper/2026/file/4b8bd97555d4060104a6b84b80920bab-Paper-Conference.pdf.

Nikita Salkar, Thomas Trikalinos, Byron Wallace, and Ani Nenkova. Self-repetition in abstractive neural summarizers. In Yulan He, Heng Ji, Sujian Li, Yang Liu, and Chua-Hui Chang (eds.), *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, pp. 341–350, Online only, November 2022. Association for Computational Linguistics. doi: 10.18653/v1/2022.aacl-short.42. URL https://aclanthology.org/2022.aacl-short.42/.

Anastasiia Sedova, Skyler Seto, Natalie Schluter, and Pierre Ablin. Scaling laws for mixture pretraining under data constraints, 2026. URL https://arxiv.org/abs/2605.12715.

Chantal Shaib, Venkata S Govindarajan, Joe Barrow, Jiuding Sun, Alexa Siu, Byron C Wallace, and Ani Nenkova. Standardizing the measurement of text diversity: A tool and comparative analysis. In Xuebo Liu and Ayu Purwarianti (eds.), *Proceedings of The 14th International Joint Conference on Natural Language Processing and The 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics: System Demonstrations*, pp. 36–46,

Mumbai, India, December 2025. Association for Computational Linguistics. ISBN 979-8-89176-301-2. doi: 10.18653/v1/2025.ijcnlp-demo.5. URL https://aclanthology.org/2025.ijcnlp-demo.5/.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Harsh Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Kyle Richardson, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Pete Walsh, Luke Zettlemoyer, Noah A. Smith, Hannaneh Hajishirzi, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: An Open Corpus of Three Trillion Tokens for Language Model Pretraining Research. *arXiv preprint*, 2024. URL https://arxiv.org/abs/2402.00159.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. Nemotron-CC: Transforming Common Crawl into a refined long-horizon pretraining dataset. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2459–2475, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.123. URL https://aclanthology.org/2025.acl-long.123/.

Aaron Swartz. html2text. https://github.com/aaronsw/html2text. Computer software.

Trieu H. Trinh and Quoc V. Le. A simple method for commonsense reasoning, 2019. URL https://arxiv.org/abs/1806.02847.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Ł ukasz Kaiser, and Illia Polosukhin. Attention is all you need. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

Joshua Vendrow, Edward Vendrow, Sara Beery, and Aleksander Madry. Do large language model benchmarks test reliability?, 2025. URL https://arxiv.org/abs/2502.03461.

Feng Wang, Zesheng Shi, Bo Wang, Nan Wang, and Han Xiao. Readerlm-v2: Small language model for html to markdown and json, 2025. URL https://arxiv.org/abs/2503.01151.

Benjamin Warner, Antoine Chaffin, Benjamin Clavié, Orion Weller, Oskar Hallström, Said Taghadouini, Alexis Gallagher, Raja Biswas, Faisal Ladhak, Tom Aarsen, Griffin Thomas Adams, Jeremy Howard, and Iacopo Poli. Smarter, better, faster, longer: A modern bidirectional encoder for fast, memory efficient, and long context finetuning and inference. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 2526–2547, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-251-0. doi: 10.18653/v1/2025.acl-long.127. URL https://aclanthology.org/2025.acl-long.127/.

Orion Weller, Kathryn Ricci, Marc Marone, Antoine Chaffin, Dawn Lawrie, and Benjamin Van Durme. Seq vs seq: An open suite of paired encoders and decoders. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=z5Mn8Rxi3l.

Kaiyue Wen, Xingyu Dang, Kaifeng Lyu, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them 2.1: Hyperball optimization, 12 2025. URL https://tinyurl.com/muonh.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzm'an, Armand Joulin, and Edouard Grave. CCNet: Extracting high quality monolingual

datasets from web crawl data. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pp. 4003–4012, Marseille, France, May 2020. European Language Resources Association. ISBN 979-10-95546-34-4. URL https://www.aclweb.org/anthology/2020.lrec-1.494.

Alexander Wettig, Aatmik Gupta, Saumya Malik, and Danqi Chen. QuRating: Selecting high-quality data for training language models. In *International Conference on Machine Learning (ICML)*, 2024.

Alexander Wettig, Kyle Lo, Sewon Min, Hannaneh Hajishirzi, Danqi Chen, and Luca Soldaini. Organize the web: Constructing domains enhances pre-training data curation, 2025. URL https://arxiv.org/abs/2502.10341.

Derong Xu, Wei Chen, Wenjun Peng, Chao Zhang, Tong Xu, Xiangyu Zhao, Xian Wu, Yefeng Zheng, Yang Wang, and Enhong Chen. Large language models for generative information extraction: a survey. *Frontiers of Computer Science*, 18(6):186357, November 2024a. ISSN 2095-2236. doi: 10.1007/s11704-024-40555-y. URL https://doi.org/10.1007/s11704-024-40555-y.

Zhipeng Xu, Zhenghao Liu, Yukun Yan, Zhiyuan Liu, Ge Yu, and Chenyan Xiong. Cleaner pretraining corpus curation with neural web scraping. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 802–812, Bangkok, Thailand, August 2024b. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-short.72. URL https://aclanthology.org/2024.acl-short.72/.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

Zichun Yu and Chenyan Xiong. Repro: Training language models to faithfully recycle the web for pretraining, 2026. URL https://openreview.net/forum?id=MTdEpFql8U.

Jiayi Zhang, Simon Yu, Derek Chong, Anthony Sicilia, Michael Tomz, Christopher D Manning, and Weiyan Shi. Verbalized sampling: How to mitigate mode collapse and unlock LLM diversity, 2026. URL https://openreview.net/forum?id=9jQkmGunGo.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging llm-as-a-judge with mt-bench and chatbot arena, 2023. URL https://arxiv.org/abs/2306.05685.

Appendix

Table of Contents

A LLM Judge Prompts **17**

B Distilling Qwen for Generative Curation **21**

C Guardrails **22**

D Final Extraction Specs **22**

D.1 Paired Specs (Separate Filter and Extraction) 22

D.2 Independent Specs 24

E Filter Classifier Sweep and Training Details **29**

F Data Mixing Details **30**

G AdamH Hyperparameter Configuration **31**

H Additional Results **32**

I An Example of Single-Spec Curation **32**

J Better Domain-Specific Data with GC-LLM **33**

K Budget-Aware Generative Curation **34**

K.1 Setup 35

K.2 Results 35

K.3 Scaling Law Fit 36

A LLM JUDGE PROMPTS

All three judges run `gemini-3-flash-preview` as DSPy `ChainOfThought` programs (Khattab et al., 2024). Each receives the spec given to the extractor, the page's raw HTML, and the extracted text, and returns a structured list of findings; the boxes below reproduce each signature's instructions and fields verbatim.

Content preservation and spec adherence (Table 1). The content preservation judge lists omissions and the spec adherence judge lists spec violations, each rated minor, medium, or major. A (page, spec) pair passes a metric when its judge reports no major finding; a judge error counts as a failure. When the judge refuses a page, we retry once with the same prompt extended by a note explaining the research setting and allowing a high-level answer that does not quote the page.

Hallucination (Table 4). A pre-filter first looks up every word trigram of the extraction in the page and marks the runs of words that no trigram covers. These candidate spans are highlighted inline with `[[ ]]` and passed to the judge, which confirms or dismisses each one. Afterwards, we discard mechanically any *invented* or *corrupted* finding whose span occurs verbatim on the page after normalizing case, whitespace, and dash and quote glyphs. Pages the judge refuses on the raw HTML are re-judged on the page's visible text.

Content Preservation Judge

```
Given a specification, original webpage, and extracted text, note
any gaps in the extracted text and rate the severity of any
omissions.

The specification showcasing the instructions that were given for
how to extract content from the webpage is provided as a string.
The original webpage is provided as a string.
The extracted text is provided as a string.

The final output is a list of objects with the following fields:
- gap: description of the missing content
- severity: "minor", "medium", or "major"
    - minor = the gap is not very important and wouldn't matter if
the reader never got this context
    - medium = the gap is important but not critical. The reader of
the extracted text would benefit from having this context, but it's
not critical to the understanding of the content.
    - major = the gap is critical and it is a serious omission of
the original content.

IMPORTANT - How to use the specification:
1. If the specification EXPLICITLY requests removal of certain
content types (e.g., "remove article titles", "exclude author
names", "strip metadata"), then removing that content is NOT a gap
and should NOT be penalized.
2. However, if the specification requests paraphrasing,
summarization, or rewording, this is NOT permission to remove
content. Paraphrasing means rephrasing the same information in
different words - ALL details and information must still be
preserved. If content is missing after a "paraphrase", it IS a gap.
3. Be strict: Only excuse removals that are EXPLICITLY and
SPECIFICALLY requested in the specification. Vague instructions like
"extract main content" do not excuse removing important information.
4. Style changes (formatting, word choice, sentence structure) are
not content omissions - only missing information/facts count as gaps.

Note that some of the webpages we are asking you to judge may
contain unsavory or offensive content.  In reviewing such pages you
can avoid directly mentioning the content on the webpage while still
```

```
pointing out at a high level if there are any gaps.  NSFW or harmful
content should not be a reason to ignore judging, but you can be
cautious not to say anything nsfw or harmful in your response.

If there are no content omissions, return an empty list.


Inputs:
- specification (str)
- original_webpage (str)
- extracted_text (str)

Output:
- gaps_list: list of {gap: str, severity: Literal["minor", "medium",
"major"]}
```

Spec Adherence Judge

```
Given an original webpage and extracted text, note any violations of
the specification on how the original webpage should be extracted
and rate the severity of any violations.

NOTE: Content omissions are not considered violations of the
specification.  Content omissions are content/information that is
missing from the extracted text that was present in the original
webpage.  Such omissions are handled separately.  Your task is to
assess if the extraction follows all criteria outlined in the
specification.

The specification on how the original webpage should be extracted is
provided as a string.
The original webpage is provided as a string.
The extracted text is provided as a string.

The final output is a list of objects with the following fields:
- violation: the violation of the specification
- severity: the severity of the omission (minor, medium, major)
    - minor = the violation is minor as in it technically doesn't
adhere to the detail of the specification but it still adheres to
the overall intent of the specification.
    - medium = the violation is important but not critical.  It is a
direct and clear violation of the specification, but it was still a
reasonable extraction given the specification.
    - major = the violation is critical and it is a serious
violation of the specification.  This is an unambiguous violation of
the specification and violates the intent of the specification.

Note that some of the webpages we are asking you to judge may
contain unsavory or offensive content.  In reviewing such pages you
can avoid directly mentioning the content on the webpage while still
pointing out at a high level if there are any spec violations.  NSFW
or harmful content should not be a reason to ignore judging, but you
can be cautious not to say anything nsfw or harmful in your response.

If there are no violations, return an empty list.


Inputs:
- specification (str)
- original_webpage (str)
```

```
- extracted_text (str)

Output:
- violations_list: list of {violation: str, severity:
Literal["minor", "medium", "major"]}
```

Hallucination Judge

```
Given a specification, an original webpage, an extracted text, and a
list of CANDIDATE spans, decide which candidates are genuine
hallucinations -- informational content in the extracted text that
is NOT present in the original webpage.

How the candidates were produced: every word n-gram of the extracted
text was looked up in the original webpage; the candidate spans are
the runs of words that no matching n-gram covers. They are marked
inline in the extracted text with [[ ]] and repeated in
candidate_spans. This pre-filter is deliberately over-inclusive.
MOST CANDIDATES ARE BENIGN. Typical benign causes:
- Words that ARE in the page but whose neighbours differ because the
extractor joined or reordered adjacent elements (usernames, dates,
"posted by" lines, header/body joins, the spec-mandated "**Author --
date**" attribution lines).
- Markdown scaffolding the spec asks for (heading marks, table
pipes, code fences, $...$ math) and image alt text or filenames
present in the page's attributes.
- Number, date, punctuation, or whitespace formatting differences,
HTML entities, and light paraphrase that adds no information.
- Words present in the page but split/joined differently (e.g.
"e-mail" vs "email").
- WHITESPACE ONLY: the page's own words, in the page's order, run
together without spaces or split by a stray space because adjacent
HTML elements were joined ("ReplyDelete", "GETSentiment",
"2014Click", "Woma n", "summaryrefslogtreecommitdiff"). Nothing was
invented or changed, so this is NOT a hallucination. It is tracked
as its own category: report it with severity "whitespace" (kind
"corrupted"), which is counted apart from the hallucination
severities and never as a hallucination. Never give a
whitespace-only difference the severity minor, medium or major.
Test: remove all spaces from the candidate and from the page text;
if the candidate is then a substring of the page, it is whitespace
only. Report each distinct glued pair once with its count.

Many pages are JavaScript applications whose HTML is mostly
serialized state. A value found in that state (a JSON string, a data
attribute, a hidden input) IS on the page, even though no reader
would see it as prose. When the extraction re-lays out such data (a
JSON list shown as a table, a field and its value on one line),
judge only what it changed or added: a renamed key, a wrapper object
the page never contains, a route or command that appears nowhere.
Re-layout is NOT the same as composition: a code sample, an example
request or response, a schema, a command, or a sentence that the
page never shows is INVENTED even if every one of its pieces (a
method name, a URL, a header, a parameter key) can be found
somewhere in the state. Finding the pieces does not make the whole
real. Ask: does the page, rendered or in its data, contain THIS
artifact? If not, it is invented. Its severity depends on whether it
is WRONG: a composed code sample, request, response, schema or
command whose details contradict the page's own data (a different
route, key, value, parameter, or version, so that a reader who
```

```
relied on it would be misled or would fail) is major; one whose
every detail agrees with the page's data is a minor invention (the
page never showed it, but nothing in it is false). Say in the
explanation which detail is wrong, or that none is. Keep the span to
the invented part and set severity from that part alone. Never
assert that text is absent unless you searched the page for it.
Words OUTSIDE the [[ ]] markers were found in the page by the
pre-filter, so a finding should not extend into unmarked text;
conversely, a block whose lines are mostly marked was not found and
needs a real counterpart before you dismiss it.

A candidate IS a hallucination when its informational content -- a
name, number, date, quote, attribution, claim, URL, or any other
fact -- cannot be found ANYWHERE in the original webpage (including
attributes, alt text, titles, and script/JSON metadata). Verify each
candidate by actually searching the page. Even if the invented
content happens to be true in the world, it is still a hallucination
because the extraction should only reflect the page.

A candidate is ALSO a hallucination when it is text corruption: a
word, name, number, fraction, or URL that IS on the page but was
altered by the extractor so that the extraction now carries an error
the page does not have (a dropped or glued character, a misspelling
introduced by the extractor, a changed digit, "¾" retyped as "¼", a
word replaced by a link's URL slug). Missing or extra spaces between
words that are otherwise intact are NOT corruption; they get
severity "whitespace" (see WHITESPACE ONLY above). Dropping a whole
word, phrase, or sentence is an OMISSION, not corruption, and is not
reported here, unless what remains asserts something the page does
not (a lost "not", a number without its unit, a name attached to the
wrong statement). Report each distinct corruption ONCE, using its
first occurrence as the span and stating in the explanation how many
times it recurs. Do not file the same corruption as several
findings. The reverse direction is NOT corruption: when the page
itself contains a typo or grammatical slip and the extraction
silently fixes it ("calender" -> "calendar", "why no do it" -> "why
not do it", "1 files changed" -> "1 file changed"), the extraction
has lost fidelity but invented nothing. Do not report it.

Severity:
- whitespace = not a hallucination: the page's own words in the
page's order with spaces lost or added (tracked separately, see
WHITESPACE ONLY).
- minor = filler or connector text that adds no verifiable claim
(e.g. an invented "Introduction" heading, "In this article we
discuss...").
- medium = plausible detail not verifiable from the page (a name,
number, date, attribution, or label the page never states). Also the
DEFAULT for text corruption: typos, glued or dropped characters,
misspelled names or domains.
- major = substantial invented claims, quotes, code, or facts that
would mislead a reader about the page's content. Text corruption is
major only when it changes a fact, quantity, name, or link target so
that a reader is misled about the page (e.g. every URL in the
extraction pointing at a misspelled domain).

NOT hallucinations: omissions (handled elsewhere), reformatting,
reordering, paraphrase that adds no new claim, spec-requested
structure, corrections of the page's own typos or grammar.
Whitespace-only differences are not hallucinations either, but ARE
reported with severity "whitespace" so they can be tracked.
```

```
You may also report a hallucination that the pre-filter missed if
you notice one, but do not go hunting outside the candidates.

Label each finding with its kind: "invented" (the content has no
counterpart on the page), "corrupted" (page content altered so the
extraction carries an error the page does not have; whitespace-only
differences are "corrupted" with severity "whitespace"), "repeated"
(page content reproduced more times than the page has it), or
"other". A finding whose span, ignoring case, whitespace, and
punctuation or dash glyphs, occurs verbatim on the page cannot be
"invented" or "corrupted"; such findings are discarded mechanically
after judging.

Return one entry per confirmed hallucination with the span copied
VERBATIM from the extracted text (without the [[ ]] markers), and in
page_text the page's own wording for the same passage, copied
VERBATIM from the original webpage (HTML source or its text; keep it
short). Leave page_text empty ONLY when the passage has no
counterpart on the page at all. Before you file a finding, confirm
that page_text really occurs in the original webpage; if you cannot
find it, you do not have a finding. If every candidate is benign,
return an empty list.

Some pages may contain offensive or NSFW content; judge the
extraction faithfully at a high level without reproducing such
content in your explanations.


Inputs:
- specification (str): The instructions the extractor was given. Use
it only to recognise spec-mandated structure (attribution lines,
markdown marks). Spec violations (kept menus or sidebars, wrong
formatting, missing content) are NOT hallucinations and are judged
elsewhere.
- original_webpage (str): The raw HTML. Page content means
everything in it: visible text, attribute values (alt, title,
value), hidden elements, and script/JSON state the page would render.
- extracted_text (str): Extracted text with candidate spans marked [[
like this]]
- candidate_spans (list[str])

Output:
- hallucinations_list: list of {span: str, page_text: str,
explanation: str, severity: Literal["whitespace", "minor", "medium",
"major"], kind: Literal["invented", "corrupted", "repeated",
"other"]}
```

B Distilling Qwen for Generative Curation

Spec construction. Preliminary experiments showed that GPT-OSS-120B handles specs of up to three distinct instructions with 93.3% adherence accuracy, so we train on three-instruction specs. Instructions are drawn from a pool of 40 atomic instructions in four categories: *cleaning* (e.g. remove PII or tracking IDs), *filtering* (drop documents not meeting a criterion such as language ID), *syntax* (output formatting, e.g. Markdown vs. plain text), and *transform* (add or restructure content, e.g. prepend a TL;DR). The pool contains 10 cleaning, 13 filtering, 10 syntax, and 7 transform instructions. Some instruction pairs are inherently incompatible (e.g. "extract only code blocks" with "output a summary at the end of the doc"); we maintain a manifest of incompatible pairs and reject sampled combinations that trigger it. To produce naturalistic training specs, we sample valid three-instruction combinations, slot them into a template, and have GPT-OSS-120B paraphrase the result

Check	Fires when	Action
View selection	The page's text lives inside its `<script>` blocks rather than its body	Extract from the script-preserving view of the page
Re-extraction	The output covers much less of the chunk than the chunk's readable text, drops most of its code, or ends mid-sentence	Re-extract the chunk once at temperature 0.3, keeping it only if it is substantially longer and not itself repetitive
Re-chunking	The output is both short and largely made of text that does not appear in the input, the signature of a summary rather than a transcription	Re-extract the page at half the chunk size, keeping it only if it is substantially longer
Span removal	Part of the output is a repeated line, a wall of index or tag entries, machine data copied out of a script, or a feed of other articles after the page ends	Delete that span

Table 5: Guardrails applied around the LLM extraction. Each triggers a fix: adjusting the preprocessing, re-extracting the chunk, re-chunking the document, or removing spans from the extraction.

using verbalized sampling (Zhang et al., 2026) for surface diversity, yielding 116 unique training specs.

Training Qwen3-8B. Using these specs, we extract 84,389 documents from Common Crawl WARC files released in 2023 or later, after DCLM's data cutoff (Li et al., 2024); the post-cutoff constraint avoids contamination with the Section 5 evaluation. The resulting dataset comprises 16 million reasoning tokens and 10 million extraction tokens. We fine-tune Qwen3-8B for 1,318 steps using AdamW with a peak learning rate of 1×10^{-5}, a cosine schedule with 3% linear warmup, weight decay 0.01, batch size 64, and sequence length 32,768 on a TPU v5p-16. Qwen3-0.6B and Qwen3-1.7B use the same recipe with three changes: they train on a larger distillation set of 601,502 documents (about $7\times$ the 8B set) for one epoch (9,399 steps), with a peak learning rate of 2×10^{-5}, on a TPU v5p-8.

C GUARDRAILS

Extraction fails in a small number of recognizable ways, and we add a check for each. Before extraction, we preprocess the HTML by removing `<script>` blocks, slicing to `<body>`, and stripping styles, comments, SVG, and layout attributes, and we place chunk boundaries after block-level closing tags, never inside a `<pre>`, `<code>`, table row, or math span. We categorize the failure modes as stemming from either the processing (script removal or chunk boundaries) or the extraction itself (repetition or dropped content). We invoke repair actions based on each. Our guardrails are described in Table 5.

D FINAL EXTRACTION SPECS

D.1 PAIRED SPECS (SEPARATE FILTER AND EXTRACTION)

The two-phase pipeline uses two specs. The filter spec judges a plain-text rendering of the page with thinking on and emits a verdict. The extraction spec transcribes the script-stripped HTML with thinking off; long pages are split into chunks, and chunks after the first use the same spec with only its opening paragraph reworded to say that the input is one section of a longer page.

Filter Spec (Phase 1)

```
You are given the plain text of a web page, with its HTML removed.
Decide whether the page belongs in a high-quality text corpus. This
is a judgement, not an extraction: do not reproduce the page.
```

```
FIRST, quote a few words from the page's main text and name the
human language they are written in. Source code counts as English.
If the language is not English, output exactly [NO_USEFUL_CONTENT]
and stop.

THE READER TEST: imagine someone who came to this page on purpose.
Would they READ it -- start at the top of its text and follow it to
the end -- or would they LOOK IT OVER and click somewhere else?

Pages people read: articles, stories, explanations, tutorials,
arguments, answers with their questions, discussions worth
following, reference entries that explain, code -- a source file is
read by a programmer the way an article is read by anyone else --
and study material: flashcards, vocabulary sets and revision notes
are studied line by line, which is reading.

Pages people look over: photo and video pages, galleries, profile
cards, product pages, schedules and calendars, search results, link
lists, archives of teasers -- teasers are posts cut short with a
"read more" link; a page carrying several COMPLETE posts in full,
one after another, is not an archive of teasers, it is writing, and
it is read, login screens, records from a database, and pages made
of ENTRIES -- dictionary definitions, cast and credit lists, arrest
blotters, specimen records: entries are looked up, not read, however
many there are. Add to these the pages that serve a moment instead
of a reader: event announcements with their date-time-place details,
notices that a page has moved, calls for donations or signups, and
posts whose text exists to present something else -- a photo set, a
video, a track, an episode, a craft project: text that presents
media is looked over, and the writer's enthusiasm does not make it
an article.

A page someone would read belongs in the corpus. A page someone
would only look over does not -- no matter how well written its
labels are. If the page holds both, judge by what a visitor came
for: a real article above its comment form is read; a photo above
its caption is looked at.

Do not drop writing for being short or plain: a complete short
answer is read. Fiction is read, including adult-themed fiction;
only text existing to be sexually explicit fails.

One more tell: strip away the moment. If the page exists to serve a
live instant -- an event coming up, a sale running, a notice posted,
a record logged today -- ask whether it would still inform a
stranger who found it after that moment passed. A page whose whole
job ends with its moment is looked over, not read.

If the page would be read, output exactly: KEEP
Otherwise output exactly: [NO_USEFUL_CONTENT]
```

Extraction Spec (Phase 2)

```
Extract this web page's content as clean markdown. You may be given
the whole page, or one section of a long page; a section can begin
or end mid-sentence. Extract what is in front of you, in the order
it appears.
```

```
Think of it as retyping the page's own writing in markdown. The
page's HTML tells you the marks to use: <h1>-<h6> -> #, <pre>/code
-> ``` fences, <table> -> | rows, LaTeX/MathML -> $...$. Nothing
else gets a mark, and nothing gets invented -- no added headings,
numbering, bold labels, inline code, link targets, or HTML tags.

What gets retyped: the article, story, or thread; the question,
every answer, and the comments under them; code, math, and tables
character for character; repeating items (cards, entries,
definitions) down to the last one. Rough, pasted, or repetitive
writing is still the page's writing.

What does not: menus, sidebars, footers, ads, forms, share buttons,
breadcrumbs, cookie banners; teasers and "related" lists; tag,
topic, category, and archive indexes, however long; shop-by filters,
product rails, trending strips; text with template placeholders. A
related question whose full answer is printed here is content, not a
pointer.

Where to stop: at the end of the page's own piece. Author bios,
"latest posts by", and feeds of other articles come after that
ending, and are not part of it.

Attribution: every post in a thread, Q&A, or comment section starts
with **Author -- date** and keeps that shape in every section of the
page. The profile furniture around a post is not part of it: member
rank, registration date, post count, location, and the signature
line repeated under every post by the same author all stay out.

Begin directly with the content. Write nothing about the extraction
itself. If this section holds only furniture, output nothing at all.
```

D.2 Independent Specs

High Quality (HQ) Spec

```
Extract the content from this HTML page as clean text. Apply all
rules below.

1. **EXTREME quality bar -- keep almost nothing.** This filter is
designed to retain only the very best educational and informative
content; only about 1 in 10 pages should pass. The **default is to
output [NO_USEFUL_CONTENT]**.

   **HARD GATES -- check first, in order. If any gate fails,
immediately output [NO_USEFUL_CONTENT] and stop.**
   - **Gate A -- Language.** Is the body of the page written
primarily in English? If the main prose is in any other language
(French, Spanish, German, Chinese, Russian, etc.), output
[NO_USEFUL_CONTENT]. Quoted phrases, names, citations, or short
snippets in another language are fine; a page whose main prose is
non-English is not -- even if the topic looks substantive.
   - **Gate B -- Prose depth.** Are there roughly 300 or more words
of real *prose* -- flowing sentences and paragraphs that explain,
argue, or teach? Table cells, formula tokens, code-only listings,
navigation labels, whois fields, stat lines, and metadata do NOT
count toward the word budget. A page that hits 300 "words" only by
counting structured-data fields fails this gate.
```

```
   A page that clears both gates is kept ONLY if all four of these
are also true:

   - **Teaches the reader.** The page explains a concept, mechanism,
method, or topic; develops an argument with evidence and examples;
reports original analysis; or delivers substantive technical or
instructional material. It must convey real knowledge of the world,
not just headlines, facts, or summaries.
   - **Carefully written.** Clear, structured, professional or
academic-quality prose. Not casual, not fragmentary, not marketing
copy, not auto-generated, not template-filled.
   - **Self-contained value.** A reader who lands here learns
meaningfully from this page alone -- not from clicking through to
something else.
   - **Not matched by any FAILS bullet below.**

   **Walk through each FAILS bullet below in order**, checking
whether the page matches it. If any matches, output
[NO_USEFUL_CONTENT] and do not extract.

   Categories that typically qualify (still subject to all criteria
above):
   - Long-form journalism with original reporting and analysis --
including editorial tech journalism or product reporting that
includes quotes, context, history, or analysis (even when the
subject is a company's product launch)
   - Educational or explanatory articles (science, math, history,
engineering, medicine, philosophy, economics, etc.)
   - Tutorials, how-tos, and documentation with real teaching --
instructions plus the reasoning behind them
   - Encyclopedia and reference entries written with substantive
prose
   - **Substantive technical forum, mailing-list, or Q&A threads**
-- e.g., a Stack Overflow question with a thorough accepted answer,
a numpy/scipy mailing-list discussion working through a real
problem, a developer forum thread that reaches a meaningful
technical conclusion. The "casual forum threads" FAILS bullet does
NOT apply when the discussion is technical and develops the topic.
   - Research papers, white papers, technical writeups
   - Essays and analyses that develop an argument with evidence --
including substantive opinion or commentary on a personal blog or
social platform like Tumblr/Medium/Substack
   - Code accompanied by genuine teaching, not just a snippet
   - Technical blogs sharing real domain expertise
   - Institutional or organizational descriptions with substantive
prose (academic program pages with curriculum and outcomes,
organizational reports, conference or event summaries with real
narrative)
   - Religious or scriptural texts presenting substantive narrative
or doctrine (e.g., a Bible chapter, Quran sura, philosophical canon
-- numbered-verse format is fine)

   Categories that FAIL -- filter even if the writing is fine:
   - **News briefs, sports recaps, and transaction roundups** -- "X
happened, said Y" reporting without analysis, depth, or context.
Includes local sports recaps ("West Side beat Cumberland 14-4"),
team transaction logs ("Signed pitcher X to a minor-league
contract"), and short news items even when several are stitched
together on one page.
   - **Personal lifestyle or diary blogging** -- daily-life updates,
travelogues, opinion pieces without intellectual substance. (A
```

```
personal blog with substantive analysis or argument is fine -- judge
by content, not platform.)
   - **Light reviews -- including aggregator pages of light
reviews** -- "I tried it, it's good" without genuine analysis or
comparison. A page that strings together several short product
blurbs is still light reviews, not substantive content.
   - **Reference stubs and structured-data dumps** -- one-paragraph
wiki entries, definitions, factoid lists, player stat cards,
whois/BGP/DNS records, raw lookup results, single-formula pages. If
the page is mostly a table, a record, or one isolated fact, it fails
-- fastText-style classifiers may rate this content highly because
it "looks structured," but it is not teaching prose.
   - **Casual forum threads** -- chit-chat, brief Q&A without
resolution, social exchanges that don't reach a meaningful
conclusion. Technical mailing-list or Q&A threads that DO develop
the topic are covered by the qualifying bullet above and are NOT
this.
   - **Vendor sales copy or purely promotional material** -- product
descriptions, ad copy, "buy now" pages. (Institutional descriptions
of academic programs, mission statements with real content,
organizational reports, or editorial journalism about a product
launch are NOT this; judge by whether the page sells a product or
describes/teaches.)
   - **Mostly un-decoded HTML / escape soup** -- pages where the
visible text is dominated by raw HTML entities (' ', '&lt;',
'&amp;', '&#NNN;'), escape sequences, or malformed markup that
survived the page render. Even if the underlying content might be
code or text, a page presented this way is junk for training.
   - **"No results" / search-empty pages** -- "your search returned
no results", "no items match", "0 records found" -- even when they
list related links or alternative searches alongside.
   - **Anything that fails a normal quality filter** -- commerce
pages, navigation or aggregation surfaces, login walls or other site
machinery, fragments and microposts, wrappers around non-textual
media, raw data dumps, empty placeholders, incoherent text

   **When in doubt, output [NO_USEFUL_CONTENT].** Reject borderline
pages aggressively; only clear winners pass.

First determine if the page clears the EXTREME quality bar. If it
does, follow the following rules to extract the content.

2. **Extract the full page content in reading order**. Keep
explanatory text, discussion, and comments that add substantive
information. Begin your output directly with the page content.
3. **Remove boilerplate**: navigation, footers, sidebars, ads, share
buttons, related links, breadcrumbs, cookie banners, and UI
elements. Do not output framework markers or metadata tags.
4. **Preserve all technical content** exactly as written: code,
math, formulas, tables, and data. Preserve original line breaks and
the structure of code blocks.
5. Decode HTML entities to plain characters (e.g. &amp; -> &,
&#8217; -> '). Remove raw HTML tags.
6. **Do not add, invent, or embellish content.** Every sentence must
come from the source page.
7. For pages with multiple authors or speakers (forums, reviews,
comments), preserve who said what -- include usernames or speaker
labels.
8. If the page contains content spinner templates like
{word1|word2|word3}, pick the first option. If most of the page is
spinners, output [NO_USEFUL_CONTENT].
```

Medium Quality (MQ) Spec

```
Extract the content from this HTML page as clean text. Apply all
rules below.

1. **Quality bar.** The page must be real written content --
coherent prose that conveys information, ideas, narrative, or
experience. Acceptable: articles, explanations, analyses, reviews
with substantive commentary, tutorials, manuals, reference entries,
documentation, and substantive forum or comment discussions. If the
page does not pass, output exactly [NO_USEFUL_CONTENT].

   **Walk through each bullet below in order**, checking whether the
page matches it. If any bullet applies, output [NO_USEFUL_CONTENT]
and do not extract.

   - **Navigation or aggregation** -- exists to point elsewhere
(forum indexes, category pages, tag clouds, search results,
sitemaps, archive-by-date pages, contributor lists,
topic-aggregation pages of one-line summaries). A tag archive
showing one truncated post excerpt still fails -- the page must
contain at least one full post.
   - **Site machinery** -- login, signup, paywall, cookie banner,
captcha, error page, "page not found", redirect notice, terms of
service, privacy policy, subscription/notification form ("send to a
friend", email signup), or instructions for using the site itself.
Pages whose main content is hidden behind a login wall fail even
when the page shows a "you must be logged in" prompt.
   - **Fragments** -- Tumblr-style image-reblog streams, social
timelines, photo-blog feeds, lists of one-liner microposts, image
gallery captions, personal "about me" pages, and repost stubs that
consist of a brief excerpt plus a "Read on:", "Go to Source", "View
original", or affiliate link to elsewhere. Pages of fragments and
metadata rather than coherent passages.
   - **Wrapper around non-textual media** -- a video, audio,
slideshow, photo, or interactive widget where the page exists to
host the media rather than be read. Includes image-only social posts
(Blingee/Pinterest-style), single-photo pages with a one-line
caption, video-player pages with only a title/byline around the
video, and blog posts whose body is a short reaction (a sentence or
two) around an embedded video, image, or share.
   - **Commerce or product page** -- pages with prices,
availability, "Add to Cart", or shipping/stock indicators.
Vendor-written descriptions, specs, ingredients, and care
instructions are commerce metadata, not content -- they do not save
the page. Keep only if the page contains an extended editorial
passage about the product (a third-party review, comparison, or
analysis), not vendor sales copy. Also includes **employer or
company profile pages** ("Quick Look", "Company Overview") that are
mostly company name, industry, size, and a bullet list of services
or job categories.
   - **Raw data dump** -- structured data without surrounding
narrative. This includes both rows of data (rosters, statistics
tables, file indexes, contact directories, database exports) **and
single-record entries** (museum specimen cards, IP whois / BGP route
lookups, dictionary entries that are only a definition line, badge /
user-profile / member pages). An article that uses tables to support
its prose is fine.
   - **Too short** -- count the substantive sentences on the page
after removing boilerplate, navigation, ads, comments, links, dates,
and reposted excerpts. **If the remaining substantive prose is fewer
than 5 complete sentences, the page fails.** Apply this strictly: a
```

```
1-paragraph news brief, a 2-sentence "X just opened in Y"
announcement, a single-event calendar entry, "this section has
moved" notice, "no results found" / "no products matching" / "click
here to proceed" placeholder -- all fail, regardless of how
grammatical or informative-looking the few sentences are.
   - **Incoherent** -- gibberish, garbled encoding,
machine-translated nonsense, or spinner templates throughout.

First determine if the page clears the quality bar. If it does,
follow the following rules to extract the content.

2. **Extract the full page content in reading order**. Keep
explanatory text, discussion, and comments that add substantive
information. Begin your output directly with the page content.
3. **Remove boilerplate**: navigation, footers, sidebars, ads, share
buttons, related links, breadcrumbs, cookie banners, and UI
elements. Do not output framework markers or metadata tags.
4. **Preserve all technical content** exactly as written: code,
math, formulas, tables, and data. Preserve original line breaks and
the structure of code blocks.
5. Decode HTML entities to plain characters (e.g. &amp; -> &,
&#8217; -> '). Remove raw HTML tags.
6. **Do not add, invent, or embellish content.** Every sentence must
come from the source page.
7. For pages with multiple authors or speakers (forums, reviews,
comments), preserve who said what -- include usernames or speaker
labels.
8. If the page contains content spinner templates like
{word1|word2|word3}, pick the first option. If most of the page is
spinners, output [NO_USEFUL_CONTENT].
```

Low Quality (LQ) Spec

```
Extract the content from this HTML page as clean text. Follow all
rules below.

1. Extract the full page content in reading order. Keep all
explanatory text, discussion, and comments that add substantive
information.
2. Remove boilerplate: navigation bars, footers, sidebars, ads,
share buttons, related links, breadcrumbs, cookie banners, and user
interface elements.
3. Preserve all technical content exactly as written: code, math
notation, formulas, tables, and data.
4. Decode HTML entities to their plain characters (e.g. &amp; to &,
&lt; to <, &gt; to >). Remove any raw HTML tags.
5. Every sentence in your output must come from the source page. Do
not add, invent, or embellish content.
6. Output exactly [NO_USEFUL_CONTENT] if ANY of these apply:
   - Login, signup, paywall, error page, or empty page
   - Index page, directory listing, search results, or
navigation-only page
   - The page has under ~50 words of substantive content after
removing boilerplate
```

E FILTER CLASSIFIER SWEEP AND TRAINING DETAILS

This appendix documents how we chose the classifiers in the cascade in Section 4: what they predict and how we score them, what they train on, which architectures we considered, and every configuration plotted in Figure 3.

Candidate architectures. Our reference is ModernBERT-base (Warner et al., 2025), 149M parameters, which reaches 0.834 F1 at 1M training pages but processes only 21.5 documents per second per chip. We looked for architectures that keep its accuracy at higher throughput along three routes: *smaller pretrained encoders* of the same family (the Ettin encoders at 17M, 32M and 68M parameters (Weller et al., 2026), and ModernBERT-base pruned to its bottom 8 of 22 layers), *shorter sequences inside the model* (a funnel hybrid that runs four ModernBERT layers, mean-pools 8×, and runs four more, warm-starting the first half; and a pooled transformer trained from scratch that pools every 64 tokens before its first attention layer), and *small encoders trained from scratch* (2- and 4-layer ModernBERT configurations). Table 6 lists the eight candidates with their throughput and F1 after training on 1M HTML pages.

Model	Init	Params	L / d	Docs/s/chip	F1
ModernBERT-base (reference)	pretrained	149M	22 / 768	21.5	0.834
Ettin-68M	pretrained	68M	19 / 512	36.4	0.830
Ettin-32M	pretrained	32M	10 / 384	85.1	0.816
Ettin-17M	pretrained	17M	7 / 256	172.3	0.805
Pruned ModernBERT-base	pretrained	79M	8 / 768	54.9	0.813
Funnel hybrid	partial	79M	8 / 768	92.9	0.808
Tiny-4	scratch	24M	4 / 384	192.6	0.797
Tiny-2	scratch	14M	2 / 256	516.1	0.777
Pooled transformer	scratch	25.9M	4 / 512	3,932.6	0.778
FastText (640 WARC files)	—	—	—	364 / core	0.796

Table 6: Architecture sweep. Every model is trained on the same 1,000,000 HTML pages at 8,192 tokens of context and scored on the 100k evaluation sample. L is the number of transformer layers and d the hidden width. Parameter counts include the 50,368-entry embedding table, which is why the pruned model keeps more than half of ModernBERT-base's parameters with a third of its layers.

Pooled transformer. The pooled transformer is the one architecture in the sweep that is not a ModernBERT variant. It tokenizes the page with the ModernBERT tokenizer and embeds each token in 256 dimensions. It then splits the sequence into 64-token windows and replaces each window with a single vector made by concatenating the window's element-wise mean, maximum, and minimum, so an 8,192-token page becomes 128 vectors. A 4-layer transformer with hidden width 512, 8 attention heads, and an MLP ratio of 4 runs over these 128 vectors; it averages the outputs, and a linear layer produces the keep/drop logits. Attention therefore costs 128^2 rather than $8{,}192^2$ per layer, which is why the model's runtime barely changes between 1k and 8k tokens. Of its 25.9M parameters, 12.9M are the embedding table.

FastText. We follow the DCLM recipe (Li et al., 2024): 5 epochs, learning rate 0.1, 100-dimensional embeddings, word bigrams and a softmax loss, with the minimum token count raised from 1 to 500 so that singleton markup tokens do not bloat the model. Training pages are sampled with 3.5 keep-labeled pages per drop-labeled page, stratified across the source WARC files.

Training recipe. All encoders are trained with AdamW at batch size 256 for one epoch over their training pages, with 200 warmup steps, gradient clipping at 1.0, and a maximum sequence length of 8,192 tokens, on TPU v6e. Pretrained encoders use a peak learning rate of 5×10^{-5}; models trained from scratch use 3×10^{-4}. The 10M and 90M runs use the same recipe over more pages. Ettin-68M is deployed on the first 2,048 tokens of a page; because the pooled transformer's cost is nearly flat in context (1.7 ms per page at 1k tokens and 2.0 ms at 8k, where ModernBERT-base takes 36 and 362 ms), it always reads all 8,192.

Configurations in Figure 3. Table 7 lists every configuration of the selected models in the figure: both input representations, the context length sweep for the ModernBERT family, and the data-size sweep for all three families. Three patterns are visible. Text input beats HTML input for every model at every scale, by 0.014–0.028 F1 for the transformers and 0.014 for FastText. Context length does not matter for Ettin-68M once the input is text (0.871 to 0.874 from 1k to 8k tokens), but throughput falls 6.7× across that range. Data size matters most for the pooled transformer, which gains 0.029 F1 from 1M to 90M text pages but nothing from 2.4× more parameters (a 62.7M variant with embedding 384, hidden 768 and 6 layers) or from three epochs over the same 10M pages.

Model	Input	Pages	Context	Docs/s/chip	F1
FastText	HTML	3.35M (80 files)	—	361 / core	0.789
FastText	HTML	6.66M (160 files)	—	362 / core	0.798
FastText	HTML	6.00M (256 files)	—	357 / core	0.792
FastText	HTML	5.63M (640 files)	—	364 / core	0.796
FastText	text	5.63M (640 files)	—	4,876 / core	0.810
ModernBERT-base	HTML	1M	8k	21.5	0.834
ModernBERT-base	HTML	10M	8k	21.5	0.860
ModernBERT-base	text	1M	8k	21.5	0.859
ModernBERT-base	text	10M	8k	21.5	0.874
ModernBERT-base	text	10M	2k	90.3	0.873
Ettin-68M	HTML	1M	8k	36.4	0.830
Ettin-68M	HTML	10M	8k	36.4	0.856
Ettin-68M	text	1M	8k	36.4	0.856
Ettin-68M	text	10M	8k	36.4	0.874
Ettin-68M	text	10M	4k	48.4	0.873
Ettin-68M	text	10M	2k	141.5	0.872
Ettin-68M	text	10M	1k	244.4	0.871
Pooled transformer	HTML	1M	8k	3,932.6	0.778
Pooled transformer	HTML	10M	8k	3,932.6	0.813
Pooled transformer	HTML	10M, 3 epochs	8k	3,932.6	0.806
Pooled transformer, 62.7M	HTML	10M	8k	3,591.3	0.814
Pooled transformer	HTML	90M	8k	3,932.6	0.827
Pooled transformer	text	1M	8k	3,932.6	0.813
Pooled transformer	text	10M	8k	3,932.6	0.833
Pooled transformer	text	90M	8k	3,932.6	0.842

Table 7: Every configuration of the selected models in Figure 3. "Pages" is the number of training pages (10M is 10,002,709 and 90M is 88,839,133); for FastText it is the number of training lines after subsampling, with the number of source WARC files in parentheses. "Context" is the number of tokens the model reads at scoring time. Text-input FastText is faster per core because extracted text is about 14× shorter than the HTML.

F Data Mixing Details

Domains. Each corpus is partitioned into $24 \times 5 = 120$ cells, crossing the 24 WebOrganizer topics (Wettig et al., 2025) with five bins of the calibrated quality score of Held (2026b) (cut at $0.2, 0.4, 0.6, 0.8$). Each cell is a mixing domain, and every pipeline is mixed independently over its own cells.

Proxy swarm. For each corpus we train $K = 363$ proxy models, one per candidate mixture. Mixtures are drawn from a Dirichlet centered on the corpus's natural token distribution with concentration log-spaced in $[1, 20]$, and weights below 0.01 are clipped to zero. $K = 3(m+1)$ satisfies Olmix's recommended over-determination for $m = 120$ domains, and we hold K fixed across corpora so that proxy compute does not confound the comparison. Each proxy is the 157M model from our ladder trained at 3×10^{18} FLOPs (≈ 4.3B tokens).

Objective. Every proxy is evaluated on the 51-task Olmix suite of bits-per-byte scores over QA, code, and math benchmarks, including the four MMLU categories (Hendrycks et al., 2021), the

seven Minerva MATH subjects (Lewkowycz et al., 2022), HumanEval (Chen et al., 2021), and MBPP (Austin et al., 2021) in 17 languages.[2] For the domain-targeted mixtures, every proxy is instead scored on all 57 MMLU subjects 5-shot, and the objective is the mean choice log-probability over the target category's subjects (18 STEM, 13 Humanities, 12 Social Science). This contrastive signal scores the distractors along with the gold answer and remains informative at 157M, where accuracy sits at chance.

Fit and solve. For every task i we fit a log-linear law $\hat{f}_i(\mathbf{p}) = c_i + \exp(\mathbf{t}_i^\top \mathbf{p})$ from mixture $\mathbf{p}$ to score, and solve

$$\min_{\mathbf{x}\in\Delta^m} \frac{1}{n}\sum_{i=1}^{n} \exp(\mathbf{t}_i^\top \mathbf{x}) \;+\; \lambda\,\mathrm{KL}(\mathbf{x}\,\|\,\mathbf{p}_{\text{nat}}) \quad \text{s.t.} \quad x_j \le kN_j/R,$$

where N_j is the token count of cell j, R the target training budget, and k the maximum number of epochs any cell may be repeated. Cells that no swarm mixture sampled (29 of 120 for Generative Curation, 38 for the Cascade, holding $< 1\%$ of tokens) have no identifiable coefficient and are held at their natural share. We solve once per corpus and reuse the mixture at every point on the compute ladder.

Settings. The general mixture uses $\lambda = 0.01$, $R = 30$B tokens, and $k = 20$. The domain-targeted mixtures use $\lambda = 0.05$ and $R = 100$B tokens with $k = 20$ unchanged; since the per-cell cap is kN_j/R, the larger R tightens it by a further $3.3\times$ on top of the stronger KL penalty.

G AdamH Hyperparameter Configuration

All pretraining runs in Section 5 use the *AdamH* optimizer and the closed-form hyperparameter recipe from the Delphi scaling suite (Held, 2026a). Rather than tune hyperparameters per run, every setting is extrapolated from a single calibrated reference point along the scaling laws in Table 8. This lets us train every model×budget cell with no per-cell tuning while keeping the comparison across curation methods fair.

The optimizer. AdamH (Wen et al., 2025) is a norm-preserving variant of Adam. For every linear weight matrix W it replaces standard weight decay with a projection that holds the Frobenius norm $\|W\|$ fixed at its initialization value: after each Adam step the matrix is rescaled back to its original norm. Consequently AdamH has *no* weight-decay term; its learning rate η plays the role of $\sqrt{\eta_{\text{Adam}} \cdot \lambda}$ (the peak LR $\times$ weight decay) in a conventional AdamW setup. All other parameters (embeddings, normalization gains, and biases) are excluded from the projection and updated by ordinary Adam with a separate learning rate η_{Adam}. The recipe therefore carries *two* learning rates that scale by different laws.

Reference point. The anchor is the empirically optimal AdamH configuration for a $\sim$130M Qwen3 model on a Nemotron mixture: batch B_0=64, tokens D_0=2.5×10^9, hidden size 512, sequence length 4096, with η_0=6.30×10^{-3}, $\eta_{\text{Adam},0}$=6.56×10^{-4}, ϵ_0=1.85×10^{-8}, and $\beta_{2,0}$=0.9999.

Deriving a run. Each model is fixed by its hidden size $d \in \{512, 1536, 2432\}$, which sets the remaining architecture: $d/128$ attention heads, d_{ff}=$4d$, and L=$\mathrm{round}\big(d/(55 + 4\log_2 d)\big)$ layers. Given a FLOP budget C, the token count is $D = C/(3F)$, where F is the model's forward FLOPs per token including attention (i.e. $6ND$ with attention counted). We choose the batch size to land near a fixed S=2^{16} training steps, rounding *up* to the nearest power of two: $B = \lceil D/(S \cdot L_{\text{seq}})\rceil_2$. All remaining hyperparameters then follow from B and D by the laws in Table 8. If a law would leave its range (a learning rate at or above 10^{-2}, or β_2 at its floor of 0.9), the batch is halved and steps extended until all values are in range. When D is too small for even B=8 to reach S steps (the smallest budget or two for each model), we hold B=8 and train for fewer steps.

[2] OLMix reports 52 metrics; we drop the `basic_skills` parent aggregate, which duplicates its six subskill entries in the flat mean.

Hyperparameter	Scaling law	Ref. value
Learning rate η	$\eta_0 \sqrt{B/B_0}\,(D_0/D)^{0.3}$	6.30×10^{-3}
Adam LR η_{Adam}	$\eta_{\text{Adam},0}\sqrt{r/r_0}$	6.56×10^{-4}
Epsilon ϵ	$\epsilon_0\sqrt{r_0/r}$	1.85×10^{-8}
β_2	$\text{clip}\big(\beta_{2,0}^{B/B_0}, 0.9, 0.9999\big)$	0.9999
β_1	fixed	0.9
Batch B	$\lceil D/(S\,L_{\text{seq}})\rceil_2$, $S{=}2^{16}$	64
Grad clip	fixed	0.1
LR schedule	10% linear warmup, constant, linear decay to 0 over the final 20%	
z-loss	fixed	1×10^{-7}

Table 8: AdamH hyperparameter recipe. B/D are the run's batch size and token count; B_0/D_0 the reference values; $r/r_0 = (B\,D_0)/(B_0\,D)$. Learning rates are capped at 10^{-2}. The bottom block lists settings that are not scaled; both learning rates follow the same schedule.

Independence from the training data. The recipe is a function of model size and FLOP budget *only*; the dataset enters nowhere in the optimizer. At a fixed model and budget, every curated corpus and mixture therefore trains with *identical* optimizer settings, and any difference in loss is attributable to the data rather than to hyperparameters.

H Additional Results

Figure 7 repeats Figure 4 on Uncheatable Eval, and Figure 8 repeats Figure 5 at the 157M scale. Both show the same trends as the main results.

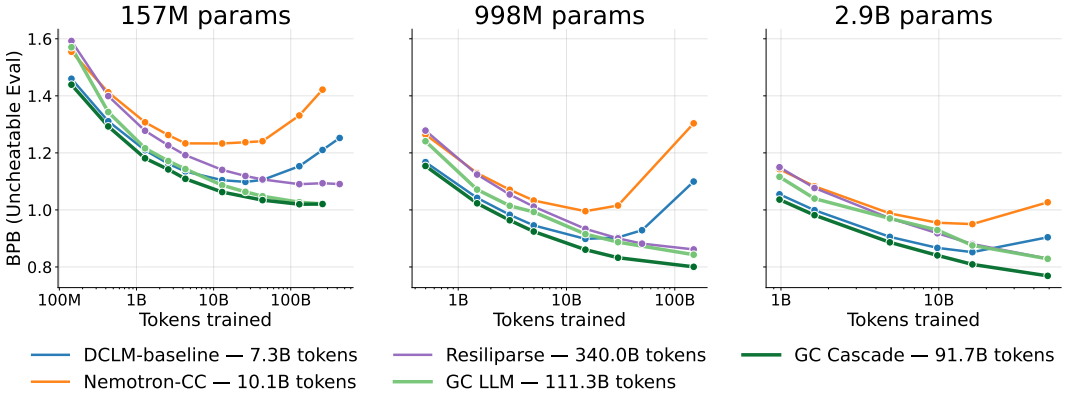

Figure 7: Uncheatable eval bpb on data mixes tuned for Olmix evals. All data from DCLM-400M-1x pool. Across all model scales and token budgets, our GC-Cascade achieves the lowest bpb of any curation strategy. Additionally, methods that curate fewer tokens degrade when trained on more tokens (due to epoching), but GC-LLM, GC-Cascade, and Resiliparse push the data frontier.

I An Example of Single-Spec Curation

Before settling on the two-spec design of Section 3.2, we built a **Single-Spec** system and ran it over the same 10,364-WARC DCLM-400M-1x pool. A single Qwen3-8B call, with reasoning, reads a page's raw HTML and either discards it or emits training text. We include the resulting corpus, Single-Spec (High Quality), as an example of single-spec curation next to our two-spec systems. It is not a controlled ablation. Besides the change in structure, it uses a different and much stricter spec: the page must be in English, contain roughly 300 or more words of prose, and meet a high bar for educational value, with a target of keeping about one page in ten. Our final filter spec is permissive by design (Section 3.3).

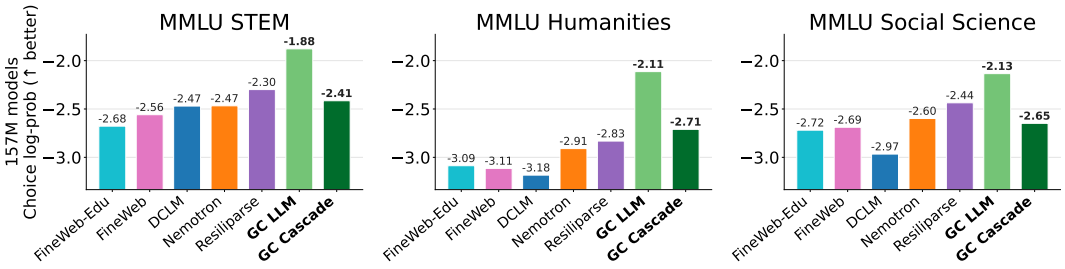

Figure 8: Highest MMLU choice log-probability per target domain at the 157M scale (higher is better) swept from 1e17 to 2e20 FLOPs. Each plot represents training on a different data mix optimized for that domain. GC-LLM or GC-Cascade methods dominate the top spot in every domain.

Context length. A Single-Spec call must fit the raw HTML, the reasoning trace, and the extracted text within the model's 32,768-token context, with 6,144 tokens reserved for output. Pages whose raw HTML was too long to fit were dropped before reaching the model, and so were never judged. The two-spec design avoids this limit: the filter reads a plain-text rendering of the page, which is far more compact than HTML, and extraction runs on 18k-token chunks, so pages of any length can be kept.

Results. Figure 9 shows Olmix eval bpb for Single-Spec (High Quality), GC-LLM, and GC-Cascade, using the same data-mixing procedure as Figure 4. At small and moderate token budgets, Single-Spec (High Quality) performs on par with GC-Cascade and reaches lower bpb than GC-LLM at every model scale. It yields 21.3B tokens, against 111.3B for GC-LLM and 91.7B for GC-Cascade. At the largest budgets, its bpb flattens or rises as training repeats its data, while GC-Cascade continues to improve.

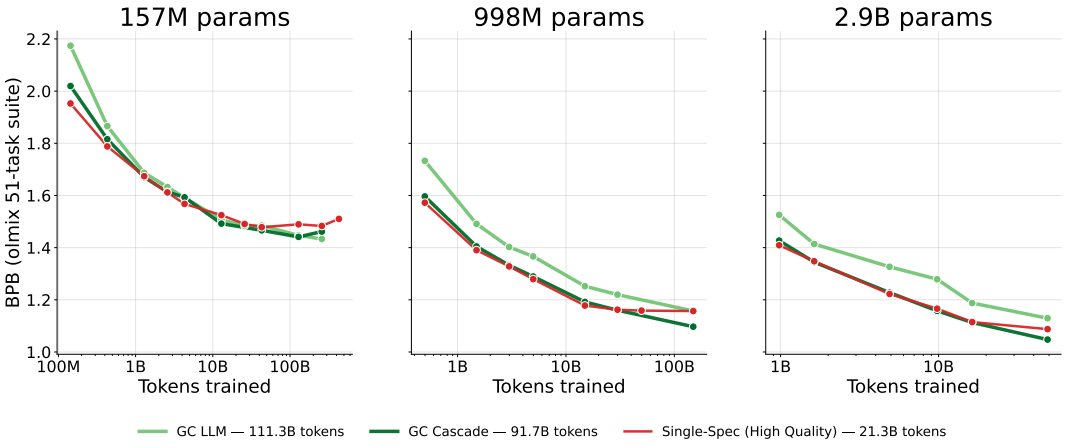

Figure 9: Olmix eval bpb on data mixes tuned for Olmix evals, for an example single-spec corpus, Single-Spec (High Quality), and our two-spec systems, GC-LLM and GC-Cascade. All data from the DCLM-400M-1x pool. Single-Spec (High Quality) also uses a much stricter spec, so this is not a controlled comparison of the two designs.

J Better Domain-Specific Data with GC-LLM

In Section 5 we used GC-LLM to control *which* web content entered the training corpus. Here we use the same tool to control the *format* of that content once it has been selected. As a case study, we continue pretraining Qwen3 base models on URL pools drawn from math, coding, and medical domains, and compare data curated by GC-LLM against data extracted by Resiliparse. We show that applying a small amount of LLM filtering and rephrasing, rather than training on Resiliparse plain text, leads to significant improvements.

Domain	Benchmark	Tokens G	Tokens R	Qwen3-0.6B-Base Base	Qwen3-0.6B-Base Δ G↑	Qwen3-0.6B-Base Δ R↑	Qwen3-8B-Base Base	Qwen3-8B-Base Δ G↑	Qwen3-8B-Base Δ R↑	Qwen3-14B-Base Base	Qwen3-14B-Base Δ G↑	Qwen3-14B-Base Δ R↑
Code	HumanEval	260M	793M	30.5	**+3.0**	+0.6	64.6	**+3.7**	-7.9	54.3	**+23.1**	+15.2
Math	Minerva Math	253M	490M	26.7	**+3.1**	-3.8	49.4	**+3.2**	+0.8	53.6	**+7.8**	+1.2
Medical	MMLU Clinical	560M	2.43B	62.3	-0.4	-0.4	80.4	**+0.8**	+0.4	83.0	**+1.1**	+0.4

Table 9: Continued pretraining of Qwen3 base models, by curation method and scale. GC-LLM (G) matches or beats Resiliparse (R) in every (scale, domain) cell despite training on 2–4× fewer tokens. Δ values are gains over base.

Experimental setting. For each domain we hold the input source URLs constant and vary only the curation step. Resiliparse converts HTML to plain text. GC-LLM, using our Qwen3-8B distill (§3), takes the same HTML and a single domain-specific spec that asks it to reformat content into a question/reasoning/answer structure where applicable (mostly forum sites). The pools differ in size and character: code draws from 47 hosts spanning Q&A communities, language and framework documentation, and tutorial sites; math is restricted to three Q&A forums drawn from sources flagged by Nemotron-CC-Math (Mahabadi et al., 2026) as rich in mathematical content; medical draws from 13 sources, comprising six Q&A forums, six clinical-reference sites, and PubMed/NCBI for biomedical literature. The spec content is in Appendix D.

Evaluation. For each domain we define a validation set (for hyperparameter selection) and a test set (the headline metric in Table 9). For math we use Minerva Math 4-shot (Lewkowycz et al., 2022) as the test set and GSM8K Platinum 8-shot CoT (Vendrow et al., 2025) for validation. For code we use HumanEval 0-shot pass@1 (Chen et al., 2021) as the test set and Mostly Basic Python Puzzles (MBPP) 3-shot pass@1 (Austin et al., 2021) for validation. For medicine we use 5-shot MMLU (Hendrycks et al., 2021) scored by log-likelihood, since the task is multiple choice rather than generative. We use `mmlu_college_medicine` for validation and `mmlu_clinical_knowledge` for test.

Training. We continue pretraining Qwen3 base models at 0.6B, 8B, and 14B. Each model trains for one epoch over the curated corpus, so Resiliparse runs take more optimizer steps (Resiliparse yields 2–4× more tokens than GC-LLM). For each (model, domain, curation method) cell we sweep learning rate and batch size, with weight decay and warmup tuned for the largest sweeps. At 8B and 14B we run a fresh sweep on code and transfer three configurations to math and medical rather than sweeping per domain. We hold the cosine schedule, maximum sequence length of 4096, and single epoch fixed for all runs, and report the test score of the cell with the highest validation accuracy.

Results. In every (scale, domain) cell, GC-LLM data improves over base by more than Resiliparse data (Table 9). The one exception is 0.6B medical, where both sources slightly decrease performance. At 14B the gap between GC-LLM and Resiliparse reaches 7.9 points on HumanEval and 6.6 on Minerva Math. The 14B HumanEval gain comes with a caveat: Qwen3-14B-Base wraps its outputs in markdown code fences that lm-evaluation-harness does not strip, breaking pass@1 grading. Re-grading the baseline samples with the fences stripped lifts pass@1 from 54.3 to 71.3, so ∼17 points of the raw +23.1 reflect format hygiene and only ∼6 reflect genuine capability improvement. The artifact does not appear at 0.6B or 8B, where base models rarely emit fences. These results show that GC-LLM delivers real gains for domain-specific data curation.

K Budget-Aware Generative Curation

Goyal et al. (2024) show that data filtering should be compute-budget aware: training with a larger compute budget benefits from less aggressive filtering. Mohri et al. (2026) extend this insight to LLMs and argue that at 10^{30} FLOPs it will be possible to drop data filtering altogether. Nemotron-CC argues that DCLM filters too aggressively and proposes relaxed filters (Su et al., 2025). Finally, Rathi & Radford (2026) argue for filtering at the token level rather than the document level, to preserve as much content as possible.

GC-LLM offers a single mechanism that covers each of these recommendations: by changing only the instructions in the spec, we can make curation less (or more) aggressive, down to deciding which

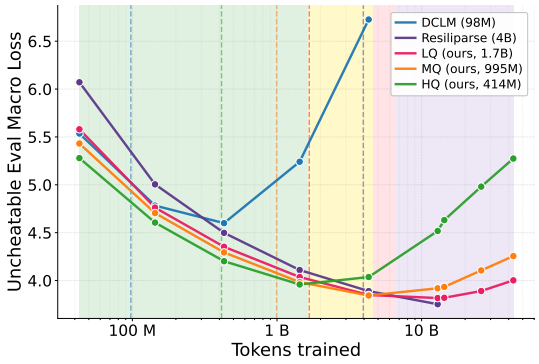

Figure 10: Uncheatable Eval macro loss for a 157M-parameter model trained on data curated from 100 WARC files with five curation strategies (unique tokens in the legend). Dashed vertical lines mark one epoch of each corpus, and shading marks the best strategy. HQ is best up to 1.6B tokens, MQ up to about 4.6B tokens, and LQ up to about 6.6B tokens, after which Resiliparse overtakes at the largest token budgets.

parts of a page to keep. In this appendix we ask whether choosing the spec according to the compute budget keeps GC-LLM on the Pareto frontier as the budget grows.

K.1 Setup

Specs. We write three specs that apply progressively weaker quality filters. **High Quality (HQ)** sets an extreme quality bar and keeps only pages that "teach the reader" with at least $\sim$300 words of carefully written prose. **Medium Quality (MQ)** keeps coherent prose that conveys information, ideas, narrative, or experience, and drops pages with fewer than five substantive sentences. **Low Quality (LQ)** is a minimal filter that removes only boilerplate, site machinery (login, error, and index pages), and pages with under $\sim$50 words of content. These specs predate the final spec in Appendix D. Each is a single spec that both filters and extracts in one call, and all three are run with our Qwen3-8B distill (§3). The full specs are in Appendix D.

Data. We extract subsets of $\{100, 500, 1\text{k}, 2\text{k}, 3\text{k}\}$ WARC files from the DCLM 400M-1x pool with each spec, and apply the same exact and fuzzy deduplication as in Section 5. Unlike Section 5, we train on each corpus as extracted, without Olmix re-weighting, and epoch whenever a budget requires more tokens than the corpus contains.

Training. We follow the procedure of Section 5, using the AdamH hyperparameters derived for the Delphi suite (Held, 2026a) at each model size and compute budget, but additionally vary the size of the source pool. Starting from the 3,000-WARC pool, we scale the pool down to $\{2000, 1000, 500, 100\}$ WARCs and choose model sizes that keep the ratio of model size to pool size roughly fixed. The hyperparameter recipe depends only on model size and compute budget, so runs with the same model size and budget use identical optimizer settings at every pool size, and differences in loss are attributable to the data. We evaluate every run by macro-averaged loss on the same Uncheatable Eval split as Section 5 (Jellyfish042, 2024).

K.2 Results

Figure 10 shows a sweep of token budgets for a 157M-parameter model trained on data from 100 WARC files with five curation strategies: the three specs, DCLM, and Resiliparse. GC-LLM is best up to $\sim$6.6B tokens: the HQ spec wins below 1.6B tokens, MQ wins until about 4.6B tokens, and LQ wins until 6.6B tokens, beyond which unfiltered Resiliparse text is best. 6.6B tokens is roughly twice the Chinchilla compute-optimal budget for this model size ($20 \times 157\text{M} \approx 3.1\text{B}$ tokens). This example uses a small, fixed pool, which raises the question of how the choice of spec behaves at the scale of the full Common Crawl.

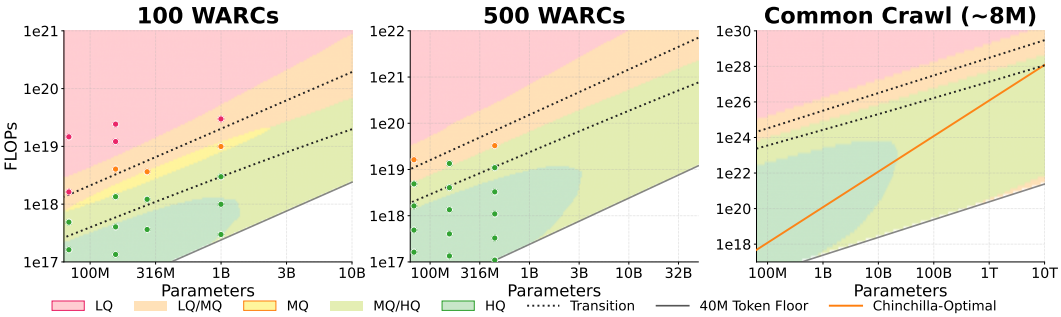

Figure 11: Predicted best spec as a function of parameter count and compute budget, from our modified version of the Sedova et al. (2026) scaling law (Appendix K.3). Pink (LQ), yellow (MQ), and green (HQ) regions mark where the fit prefers that spec in more than 95% of 100 bootstrap resamples. Orange (LQ/MQ) and lime (MQ/HQ) mark regions where the resamples do not confidently separate the two adjacent specs. Dotted lines mark the transitions under the point-estimate fit. Points are configurations we trained, colored by the spec with the lowest observed loss. Along the Chinchilla-optimal line (orange), HQ remains the best of the three specs up to 10T parameters.

To answer this, Figure 11 shows the spec predicted to be best for a given parameter count and compute budget, according to the scaling law fit in Appendix K.3. At the Chinchilla compute-optimal ratio of 20 tokens per parameter, the point-estimate fit prefers the HQ spec through 10T parameters, and HQ is preferred in more than 95% of bootstrap resamples through 10B parameters. When training well beyond compute-optimal, however, the fit predicts regimes in which a less aggressive spec is the better choice.

K.3 Scaling Law Fit

This section gives the full form of the modified scaling law behind Figure 11 and details how we fit it. We fit the law to all runs across the three specs (LQ, MQ, HQ) and the WARC scales $\{100, 500, 1\text{k}, 2\text{k}, 3\text{k}\}$, varying model size N and tokens trained D (299 runs in total). The fitted quantity is Uncheatable Eval macro loss.

The loss model. We build on the size-dependent, data-constrained scaling law of Sedova et al. (2026), which itself extends Muennighoff et al. (2025). It predicts loss from three quantities—model size N, total tokens trained D, and the number of *unique* tokens U—as

$$L(N, D, U) = E + C\,N^{-\beta} + B\,N^{\delta}\,D_{\text{eff}}(D, U)^{-\alpha}. \tag{1}$$

Here E is the irreducible loss, the term $C\,N^{-\beta}$ captures the gain from more parameters, and the term $B\,N^{\delta}\,D_{\text{eff}}^{-\alpha}$ captures the gain from data, where D_{eff} is the *effective* number of tokens: the raw count discounted for repetition. The exponent δ, the contribution of Sedova et al. (2026) over Muennighoff et al. (2025), couples the value of data to model size, encoding that larger models exhaust a fixed corpus faster and then become data-bottlenecked. The behavior of the law therefore hinges on the form of D_{eff}.

Effective data and over-epoching. Writing $\varepsilon = D/U$ for the number of epochs over the unique corpus, prior work models effective data as a monotonically increasing, saturating function of repetition,

$$D_{\text{eff}}^{\text{sat}}(D, U) = U\,R_D\left(1 - e^{-\varepsilon/R_D}\right), \tag{2}$$

which approaches a finite ceiling as ε grows: each additional epoch helps less than the last, but never hurts. The horizon R_D sets how quickly returns flatten. Empirically, however, past a corpus-dependent number of epochs, loss does not merely plateau but *rises* as the model overfits a small, heavily repeated corpus (e.g., DCLM and Nemotron in Figure 4, and Figure 10). No monotone D_{eff} such as equation 2 can express this upturn.

	LQ	MQ	HQ
Shared			
E		1.810	
C		1156.9	
β		0.349	
R_D		1.468	
R_{decay}		67.96	
Per spec			
B	52.5	67.4	82.8
δ	0.185	0.187	0.198
α	0.378	0.398	0.425
R^2	0.994	0.992	0.983

Table 10: Fitted parameters of the modified scaling law equation 1. $(E, C, \beta, R_D, R_{\text{decay}})$ are shared across specs; (B, δ, α) are fit per spec. Aggregate RMSE $= 0.085$ across 299 runs.

Adding a data-decay term. We therefore multiply the saturating term by a decaying factor, so that effective data peaks and then declines with excess repetition:

$$D_{\text{eff}}(D, U) = \underbrace{U\, R_D \left(1 - e^{-\varepsilon / R_D}\right)}_{\text{saturation}} \times \underbrace{e^{-\varepsilon / R_{\text{decay}}}}_{\text{over-epoching}}. \tag{3}$$

The first factor is the saturation of equation 2. The second is our addition: it discounts effective data once repetition becomes excessive, governed by an *independent* scale R_{decay} that sets when over-epoching begins. The two scales must be decoupled, since a single parameter cannot set both where the curve flattens and where it bends upward. Substituting equation 3 into equation 1 gives our complete model. Because D_{eff} now decreases for $\varepsilon \gg R_{\text{decay}}$, the data term $B\, N^{\delta} D_{\text{eff}}^{-\alpha}$ turns upward at high epoch counts, reproducing the observed over-epoching degradation while recovering the saturating behavior of prior work in the low-repetition regime.

Fitting. We fit equation 1 jointly across the three specs. The irreducible loss E, the parameter-scaling constants (C, β), and the two repetition scales (R_D, R_{decay}) are shared across specs, while the data-term parameters (B, δ, α) are fit per spec, for a total of $5 + 3{\times}3 = 14$ free parameters. We minimize the Huber loss between predicted and observed log-loss over all 299 runs, using basin-hopping with 100 random restarts of an L-BFGS-B inner optimizer to escape poor local minima. We fit on all runs, including those past the loss minimum: the over-epoching upturn is exactly the signal that identifies R_{decay}, so clipping it would remove the information the new term is meant to capture. Table 10 reports the fitted parameters. The fit attains an aggregate RMSE of 0.085 in loss units, with per-spec $R^2 \geq 0.98$.