Beyond a Single Extractor: Re-thinking HTML-to-Text Extraction for LLM Pretraining

Jeffrey Li[3*] Josh Gardner[1○] Doug Kang[1] Fangping Shi[1]
Karanjeet Singh[1] Chun-Liang Li[1] Herumb Shandilya[2] David Hall[2]
Oncel Tuzel[1] Percy Liang[2] Ludwig Schmidt[2] Hadi Pour Ansari[1] Fartash Faghri[1]

[1]Apple [2]Stanford [3]University of Washington

jwl2162@cs.washington.edu, fartash@apple.com

[*]Work done during an internship at Apple. [○]Work done while at Apple.

Abstract

One of the first pre-processing steps for constructing web-scale LLM pretraining datasets involves extracting text from HTML. Despite the immense diversity of web content, existing open-source datasets predominantly apply a single fixed extractor to all webpages. In this work, we investigate whether this practice leads to suboptimal coverage and utilization of Internet data. We first show that while different extractors may lead to similar model performance on standard language understanding tasks, the pages surviving a fixed filtering pipeline can differ substantially. This suggests a simple intervention: by taking a *Union* over different extractors, we can increase the token yield of DCLM-Baseline by up to 71% while maintaining benchmark performance. We further show that for structured content such as tables and code blocks, extractor choice can significantly impact downstream task performance, with differences of up to 10 percentage points (p.p.) on WikiTQ and 3 p.p. on HumanEval.

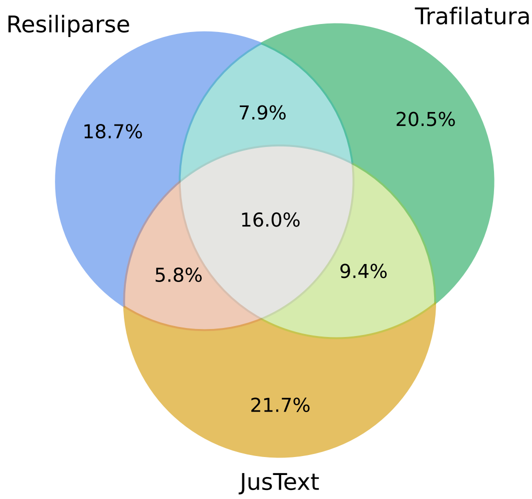


Figure 1: **Different extractors lead to different final pages.** The Venn diagram shows the overlap in resulting pages that come from applying the DCLM-Baseline pipeline to the outputs of three different initial extractors. 61% of pages are uniquely kept for just one extractor.

1 Introduction

Large language models (LLMs) are primarily pretrained on webpages crawled from the Internet, such as from Common Crawl (Common Crawl, 2007). A key early step for building such web-scale training datasets is to convert each page's HTML contents into plaintext. This is crucial not only because we wish to interact with models via natural language but also because HTML can contain a lot of unhelpful and auxiliary boilerplate such as navigation bars, hidden elements, and visual styling.

Yet, while most existing research for LLM pretraining datasets has focused on filtering, cleaning, and deduplicating already extracted text, relatively little attention has been placed on extraction itself. Indeed, the standard practice for leading open-source datasets has been to select an existing rule-based extraction package and apply it uniformly to all of Common Crawl (Common Crawl, 2007). For instance, DCLM-Baseline (Li et al., 2024) uses `resiliparse` (Bevendorff et al., 2018, 2021), FinedWeb-Edu (Lozhkov et al., 2024a) uses `trafilatura` (Barbaresi, 2021), and Nemotron-CC (Su et al., 2024) uses `jusText` (Pomikálek, 2011; Zamazal, 2024). These choices were motivated by qualitative inspection and findings that extractors achieved comparable downstream performance on standard benchmarks, differing mainly in other aspects such as runtime efficiency (Li et al., 2024) or token yield (Su et al., 2024).

In this work, we investigate whether this practice of applying a single extractor to all webpages leads to sub-optimal usage of Common Crawl data. We demonstrate that while different extractors may result in similar aggregate performance on standard language understanding benchmarks, this similarity masks two critical issues: (1) extractors can capture complementary subsets of the web; (2) they

Findings of the Association for Computational Linguistics: EACL 2026, pages 5836–5861

exhibit larger differences in their ability to handle structured elements such as tables and code blocks.

First, we show that different extractors can lead to capturing surprisingly distinct portions of the web. When applying a fixed post-extraction curation pipeline from DCLM-BASELINE, we find that `resiliparse`, `trafilatura`, and `jusText` yield substantially different sets of pages, with only 39% surviving across more than one extractor (see Figure 1). This complementarity suggests a straightforward practical intervention: by taking the union of surviving pages across multiple extractors, we can increase the token yield from the DCLM-BASELINE pipeline by 71% (and 58% when running further deduplication) while maintaining performance on standard benchmarks. Further, this increased yield translates to improved performance in data-constrained settings which simulate the practical finiteness of the Internet.

Second, we examine webpage elements that require specialized formatting—tables and code blocks—finding that extractor choice has substantial downstream impact. When isolating portions of Common Crawl containing such elements, model performance on relevant tasks depends heavily on the extractor used. For tables, a 7B model trained on `resiliparse` outputs outperforms those coming from `jusText` and `trafilatura` by 10.3 and 8.2 p.p. (a 3.2× improvement) respectively on WikiTableQuestions (Pasupat and Liang, 2016). For code, `jusText` significantly underperforms both alternatives, with degradations up to 3.6 p.p. on HumanEval (Chen et al., 2021).

Overall, our findings challenge current extraction practices when curating general web data for pretraining. While prior work has largely found popular extractors to be interchangeable when measured by common downstream model evaluations, we show that extraction remains an under-explored axis for improving pretraining datasets. Through more systematic evaluation of three widely-used extractors (`resiliparse`, `trafilatura`, and `jusText`) on Common Crawl, we provide actionable guidance for practitioners: using multiple extractors in parallel can substantially improve data yield, and extractor choice can significantly impact performance based on content type. Our analysis demonstrates that more thoughtful use of existing extraction tools can already yield meaningful improvements in dataset curation.

2 Related work

In this section, we discuss existing extractors, their usage in pretraining datasets, and filtering pipelines for tables and code blocks. We focus on the most relevant works here and defer an expanded discussion to Appendix A.

Text extraction approaches. Earlier approaches like `jusText` (Pomikálek, 2011; Zamazal, 2024) work by segmenting HTML into blocks and then classifying each as boilerplate or main content using heuristics based on features such as length, link density, and stopword frequency. Introduced around the same time, `boilerpipe` (Kohlschütter et al., 2010) similarly utilizes shallow text features but in combination with learned decision trees. More recently, `trafilatura` (Barbaresi, 2021) emphasizes a more balanced approach between precision and recall. It uses rule-based heuristics that can fall back to other extractors like `jusText` when needed, but differs in design philosophy by capturing diverse content types (lists, tables, comments) rather than focusing narrowly on full English sentences (as `jusText` does). Meanwhile, `resiliparse` (Bevendorff et al., 2018, 2021) prioritizes computational efficiency, using simple tag and regex-based rules implemented in optimized C++ code. In contrast, another line of approaches develops deep learning models as extraction classifiers, including `web2text` (Vogels et al., 2018), `boilernet` (Leonhardt et al., 2020), and `neuscraper` (Xu et al., 2024). While such efforts may be promising, we limit our investigation to rule-based extractors, which are easier to scale and have been used by the largest open datasets.

Choosing text extractors for LLM pretraining. Creators of the largest public pretraining datasets have either used the extractions provided by Common Crawl (i.e., WET files) or existing rule-based extractors. Datasets such as C4 (Raffel et al., 2020; Dodge et al., 2021), RPJ (Together Computer, 2023; Computer, 2023), and Dolma (Soldaini et al., 2024) used WET files. Meanwhile, The Pile (Gao et al., 2021) and more recent state-of-the-art efforts run other extractors to remove more boilerplate. The FineWeb (Lozhkov et al., 2024a) and RefinedWeb (Penedo et al., 2023) datasets use `trafilatura`, while DCLM (Li et al., 2024) prefers `resiliparse` due to its speed. In contrast, Nemotron-CC (Su et al., 2024) uses `jusText` over `trafilatura` as it yielded more tokens after running the FineWeb-Edu filtering classifier. Likely due to poorer scalability, neural-based extractors have been tested minimally for large-scale pretraining, though Xu et al. (2024) show promising results for $\leq$ 410M parameter models. Notably, extractors were also traditionally evaluated against human extractions, but these metrics don't always predict how well LLMs can be trained on extractor outputs. For instance, Bevendorff et al. (2023) finds `trafilatura` to significantly outperform `jusText` but the two perform similarly in Su et al. (2024).

Filtering for tables and code blocks. For tables, the most relevant work is The WDC Web Table Corpora (Lehmberg et al., 2016; Eberius et al., 2015). This work filters `<table>` elements from two 2012 and 2015 Common Crawl dumps using various heuristics and classifies them based upon their purpose (e.g., whether the `<table>` contains true relational data or exists just for layout purposes). For code blocks, Redstone-Code (Chang et al., 2024) finds `<code>` elements, filters for genuine code via regular expressions, and then extracts the parent pages using the same approach as WET files. A more recent concurrent work is Nemotron-CC-Math (Mahabadi et al., 2025), which takes URLs from leading mathematics datasets and re-extracts them using the `lynx` browser in combination with Phi-4 (Abdin et al., 2024). While they did not directly target code, they find that their dataset contains many pages with code snippets and improves their extractions, leading to better code generation capabilities. Overall, our work uses similar filtering methods for constructing datasets to study the role of extraction (rather than to compete with these approaches). We defer more detailed comparisons to Sections 4 and 5 as well as Appendix A.

3 Different extractors lead to different high-quality pages

We first examine the impacts of extractor choice on standard English benchmarks, given a fixed (high-quality) post-extraction data pipeline.

3.1 Experiment setup

We start with the original Common Crawl WARCs that correspond to the `1B-1x`, `7B-1x`, and `7B-2x`[1] versions of DCLM-RefinedWeb (Li et al., 2024) and re-extract these pages using three different extractors. While DCLM originally starts from `resiliparse` extractions (with `main_content` set to `True`), we now also try `trafilatura` and `jusText` (both with default arguments) and then in parallel repeat DCLM-Baseline's filtering and deduplication procedures.

Given these three different versions of DCLM-Baseline, we then assess the overlaps in page IDs[2] when taking *Unions* where we combine all surviving pages across versions and remove duplicate IDs (i.e., for pages where multiple extractions survive, we keep only one version). When deduplicating page IDs, we try two strategies for picking which version to keep: *Random* chooses one surviving extraction arbitrarily while *Manual* defines a hard-coded preference ordering favoring larger token yield: i.e., first `resiliparse`, second `trafilatura`, and third `jusText`.

3.2 Results

We start by comparing existing extractors in our setup. Then, we show that the resulting datasets contain many non-overlapping pages, allowing us to significantly improve data yield by combining them in our Unions. Finally, we examine the benefits of this approach in data-constrained settings.

How do existing extractors perform on standard tasks? Across our `1B-1x` and `7B-1x` results in Table 1, we observe no consistent winner among extractors. On Core_V2, `trafilatura` and `jusText` perform similarly while `resiliparse` shows lower a score at `1B-1x` but better performance at `7B-1x`. However, based on a preliminary study of run-to-run variation at `1B-1x` scale (see Table 6), we find that these differences are likely within the noise of single runs—when averaged across three runs, the means (and standard deviations) of `resiliparse` and `trafilatura` become 28.3 ± 0.7 and 28.6 ± 1.0 respectively. Meanwhile, MMLU is close to random chance at `1B-1x` and more variable at `7B-1x` due to insufficient training.

Does combining extractors lead to better performance? As evidenced by the increases in token counts (Table 1) and overlap statistics (Figure 1) for Union datasets, different extractors yield substantially different pages. Further, these differences can be systematic: Figure 2 (left) shows that on a non-trivial portion of web domains, the vast majority of surviving pages come from a single extractor. Overall, our Union datasets maintain performance on

[1] As in DCLM, the scale names refer to the model size and the number of training tokens, given as a multiplier of the Chinchilla optimal amount (Hoffmann et al., 2022).

[2] This is determined by the concatenation of the `WARC-Record-ID` and `WARC-Date` metadata keys.

Table 1: **Using multiple extractors increases token yield while maintaining performance.** We compare the token yields and CORE_V2 performance of using different extractors in the DCLM-BASELINE curation pipeline. “Union” refers to combining the resulting pages from the individual extractors. The “Random” and “Manual” variants refer to whether the pages in the intersection are selected randomly or according to a manually selected preference ordering: i.e., `resiliparse`, then `trafilatura`, then `jusText`. This is also the ordering of the percentiles for Union methods in the “`fastText` Threshold” column. Note that while we show MMLU performance at all scales for completeness, we advise caution when drawing conclusions from it at `1B-1x` and `7B-1x` scales due to potential noise.

Training Scale	Extractor	`fastText` Thresholds	Token Yield	CORE_V2	MMLU
`1B-1x` (29B Tokens)	`resiliparse`	0.11	39B	28.5	24.7
	`trafilatura`	0.11	27B	**29.6**	25.7
	`jusText`	0.11	25B	29.3	**25.8**
	Union (Random)	(0.11, 0.11, 0.11)	55B	28.1	25.1
	Union (Manual)	(0.11, 0.11, 0.11)	**57B**	29.2	24.1
`7B-1x` (138B Tokens)	`resiliparse`	0.11	193B	42.6	39.5
	`trafilatura`	0.11	135B	41.9	37.7
	`jusText`	0.11	124B	41.2	34.5
	Union (Random)	(0.11, 0.11, 0.11)	273B	41.3	33.6
	Union (Manual)	(0.11, 0.11, 0.11)	**283B**	**43.1**	**43.5**
`7B-2x` (276B Tokens)	`resiliparse`	0.11	386B	47.4	**52.9**
	`resiliparse`	0.15	540B	47.6	51.7
	`resiliparse`	0.18	650B	47.1	47.4
	Union (Random)	(0.11, 0.11, 0.11)	546B	47.8	52.3
	Union (Manual)	(0.11, 0.11, 0.11)	565B	47.9	51.4
	Union (Random)	(0.11, 0.15, 0.15)	639B	**48.0**	51.0
	Union (Manual)	(0.11, 0.15, 0.15)	**662B**	47.6	50.8

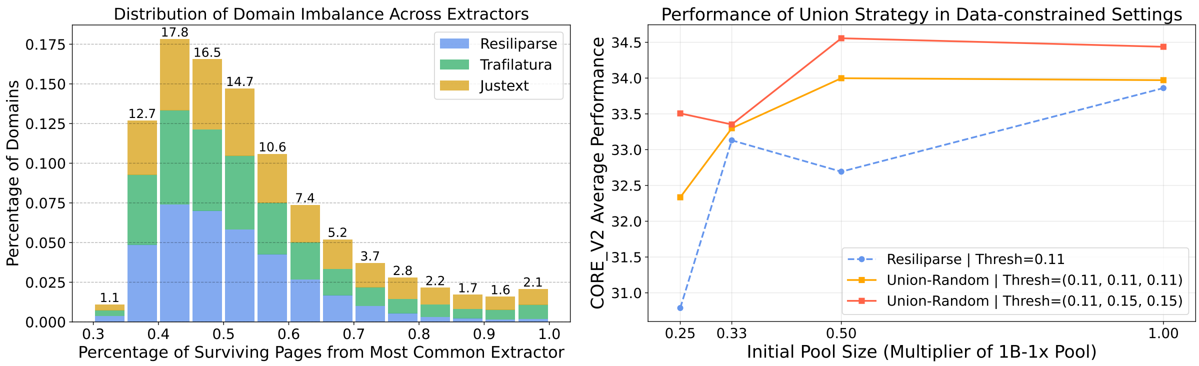


Figure 2: **(Left) Analysis of extractor imbalance across domains.** We group pages from all three extractors (Top 11%) by domain. For every domain with at least 50 pages, we compute the maximum ratio represented by any one extractor, plotting the distribution above. For 26.7% and 7.6% of domains respectively, at least 60% and 80% of surviving pages come from just one extractor. **(Right) Does higher yield lead to better performance when data-constrained?** We train `1B-5x` models on smaller subsamples of data curated from the `1B-1x` raw pool. The Union datasets that yield more tokens are able to achieve better performance. See Table 7 for extended results.

DCLM’s CORE_V2 evaluations while increasing token yield by up to 71% when we relax filtering thresholds for `trafilatura` and `resiliparse` to top-15% (bringing their individual yields closer to `resiliparse`’s). Meanwhile, we also see that performing a Union over high-quality pages from each extractor beats simply loosening the filtering threshold from the original `resiliparse`-based dataset when targeting a similar token count. That is, we are able to *recover* high-quality pages better when applying stricter filters across different extractors.

We note that our results do not prescribe an *optimal number or set* of existing extractors to combine. We chose to study three approaches that are each battle-tested at scale but follow different heuristics and design choices. Adding more extractors may yield diminishing returns if they follow similar designs, or could perhaps significantly increase coverage if they are markedly different (e.g., neural-based or domain-specific). That said, our results remain a strong proof-of-concept that existing single-extractor approaches can be improved.

Does the Union operation simply reintroduce duplicates? Despite deduplicating by page IDs, it is possible that the Union operation could reintroduce some fuzzy duplicates that were previously removed when running deduplication separately for each extractor. More concretely, suppose pages x_1 and x_2 are fuzzy duplicates both when we use extractors e_1 and e_2. It could be that the individual deduplication runs arbitrarily end up removing $e_1(x_1)$ and $e_2(x_2)$, leading to the Union containing a pair of fuzzy duplicates in $e_1(x_2)$ and $e_2(x_1)$. Whereas, if the same page was kept from both extractors, e.g., $e_1(x_1)$ and $e_2(x_1)$, the deduplication by page IDs would keep only one. To test whether this occurs, we run a second round of DCLM's fuzzy deduplication on our Union datasets and find that this decreases token count by 7-8%. While not insignificant, the Union datasets still retain up to 58% higher yield compared to DCLM-BASELINE.

Meanwhile, we see from Table 2 that rededuplication can have varying impacts on CORE_V2. At `1B-1x` scale, the best result with rededuplication is 1.0 p.p. lower than the best result without it. Meanwhile, at `7B-1x` scale, rededuplication yields our best result of 44.1 which is 1.0 p.p. higher. It remains unclear whether these differences stem fully from variance or from more complex interactions between scale, union type, and rededuplication—for instance, whether systematic format differences in cross-extractor duplicates may benefit language modeling or whether Union (Random) tends to combine better with rededuplication.

Does the Union operation lead to better performance under data constraints? We explore training models in settings where data is more constrained by the initial amount of "raw" Internet data. This simulates the modern reality (albeit at a much smaller scale) that the largest frontier model runs may be approaching the limits of available Internet data. Specifically, we train `1B-5x` models on progressively smaller random subsamples of our `1B-1x` datasets. This means the Union datasets which yield more tokens will allow for fewer effective repetitions during training (see Table 7). As shown in Figure 2, this translates to larger performance gains when data constraints are more severe.

4 Extraction of tables

Given the proliferation of tabular data in many real-world applications, table understanding is an important skill for LLMs that has received growing attention (Ashury-Tahan et al., 2025a; Gardner et al., 2024; van Breugel and van der Schaar, 2024). However, such capabilities are not often tested by those proposing data curation techniques. Meanwhile, Common Crawl contains a large amount of HTML tables (Eggert et al., 2023) that could presumably be used to learn such skills but which have also been under-emphasized during curation. In this section, we examine how different extractors behave on such pages and how these differences impact downstream models using the WikiTQ tabular understanding task (Pasupat and Liang, 2016).

Table 2: **Effect of reduplicating Union datasets.** We compare the token yields and CORE_V2 performance of our Union datasets with and without reduplication. "Rededup" refers to running another round of deduplication on the Union dataset (now across the pages from different extractors). Here we always take a Union over the (0.11, 0.11, 0.11) thresholds. All other nomenclature follows from Table 1.

Scale	Union Type	Token Yield	CORE_V2
`1B-1x`	Random	55B	28.1
	Manual	**57B**	**29.2**
	Random w/ Rededup	51B	28.2
	Manual w/ Rededup	53B	27.4
`7B-1x`	Random	273B	41.3
	Manual	**283B**	43.1
	Random w/ Rededup	253B	**44.1**
	Manual w/ Rededup	265B	41.8

4.1 CC-Tables: Filtering Common Crawl for data tables.

We first filter down Common Crawl to a set of pages that contain `<table>` elements which are likely to be data tables, referring to the resulting set of pages as *CC-Tables*. Inspired by the WDC Web Table Corpora (Lehmberg et al., 2016), we also use a combination of structural (e.g., based on row/column counts and row consistency) and model-based heuristics while simplifying the implementations of both (see Appendix A). Our dataset and classifier model also differ in that they were: (1) built using all pre-2023 Common Crawl dumps as opposed to only two from 2012 and 2015; (2) we focus on extracting whole pages rather than just segments surrounding tables.

We found that our structural heuristics still left in many tables that were unlikely to be helpful, such as sizing charts for clothing products, landing pages for forums, and empty calendars. We train a `fastText` classifier with such tables as negative ex-

Table 3: **resiliparse is the most effective extractor for CC-Tables pages.** We compare the performance of `resiliparse`, `trafilatura`, and `jusText` on pages from our CC-Tables dataset. For each extracted version of CC-Tables, we mix it with DCLM-BASELINE in a 20%-80% mix, train models, and evaluate on WikiTableQuestions (WikiTQ). "No CC-Tables" refers to training on only DCLM-BASELINE.

CC-Tables Extraction	Model Scale	WikiTQ-Avg.
No CC-Tables	7B-2x	1.1
resiliparse	7B-1x	**11.9**
trafilatura	7B-1x	3.7
jusText	7B-1x	1.6

amples and tables from higher quality URLs (e.g., English Wikipedia and .gov domains) as positives. We then use this model to score all tables and filter out pages where no table is classified as positive. Notably, this filtering pipeline for CC-Tables does not yet *apply any downstream extractor*, allowing us to then compare different extractions of these pages while limiting potential bias. More details are provided about CC-Tables in Appendix D.2.

4.2 Qualitative extractor differences

From our sample of pages provided in Appendix D.1, we observe `jusText` generally removes tables altogether (also noted by Gao et al. (2021)), while `resiliparse` and `trafilatura` format them in different ways. The former uses whitespace delimiting but can sometimes improperly merge together columns, while `trafilatura` tries to convert tables into a markdown format but can fail to keep cell contents (see Figure 7). Given these differences, a natural question is how this affects performance under different *test-time* serializations. The ToRR benchmark (Ashury-Tahan et al., 2025b) allows us to study this question as it implements running evaluations over seven different serializations, including both space-delimited (referred to as "concat") and markdown.

4.3 Empirical Results

Training setup. We train various 8K-context models on mixes of 80% general web data (i.e., DCLM-Baseline) and 20% CC-Tables after applying different extraction algorithms. To train longer context models efficiently, we use the Dataset Decomposition technique from (Pouransari et al., 2024). The $4\times$ increase in context length compared to standard DCLM models is important for our evaluations as some serializations (e.g. HTML and JSON) in ToRR lead to examples exceeding a context length of 2048. For evaluation, we use a modified version of ToRR's (Ashury-Tahan et al., 2025b) implementation of WikiTableQuestions (WikiTQ) (Pasupat and Liang, 2016). Specifically, we modify the set of post-processors that ToRR uses to avoid inflating the scores of some shorter generations (see Appendix D.3 for more details).

Which extractors are best for CC-Tables? From our results in Table 3, we see that the qualitative differences between extractors indeed lead to significant quantitative differences on WikiTQ. Unsurprisingly, `jusText` performs the worst, while both `resiliparse` and `trafilatura` lead to non-trivial improvements over training on more tokens of DCLM-Baseline alone. Interestingly, `resiliparse` performs markedly better than `trafilatura`. We see from Figure 3 that while `resiliparse` produces only space-delimited tables and favors "Concat" at test-time, it still leads to the best performance on all test formats. Meanwhile, `trafilatura` leads to favoring markdown, but suffers worse performance overall.

These results perhaps echo a similar finding from Meta AI (2024) that markdown markers can be harmful to models mostly trained on webcrawled data. We also observe from Table 8 in Appendix D.4 that our results hold even if we modify the mixture weights and use the `trafilatura` version of DCLM-Baseline (i.e., it is not the result of conflicting extractors between the different mixture components). However, we cannot rule out the role of other extraction differences besides formatting (e.g., how surrounding text is captured or whether all data cells are actually retained).

Can CC-Tables patch tabular performance for DCLM-7B-8k? Finally, we see in Table 4 whether CC-Tables-`resiliparse` can help close the sizable gap between DCLM-7B-8k and LLama-3-8B on WikiTQ. While the original DCLM paper shows that the two models are comparable on standard language evaluations, we see here the latter is far better at tabular understanding. To bridge this gap, we find that replacing the final DD cooldown of DCLM-7B-8k with a 80%-20% mix of DCLM-Baseline and CC-Tables-`resiliparse` can close 73% of this gap. While the improvement on WikiTQ initially comes at the cost of standard metrics, a simple WiSE-FT (Wortsman et al., 2022) weight interpolation can help mitigate this, with the weight $\alpha = 0.85$ providing a reasonable balance.

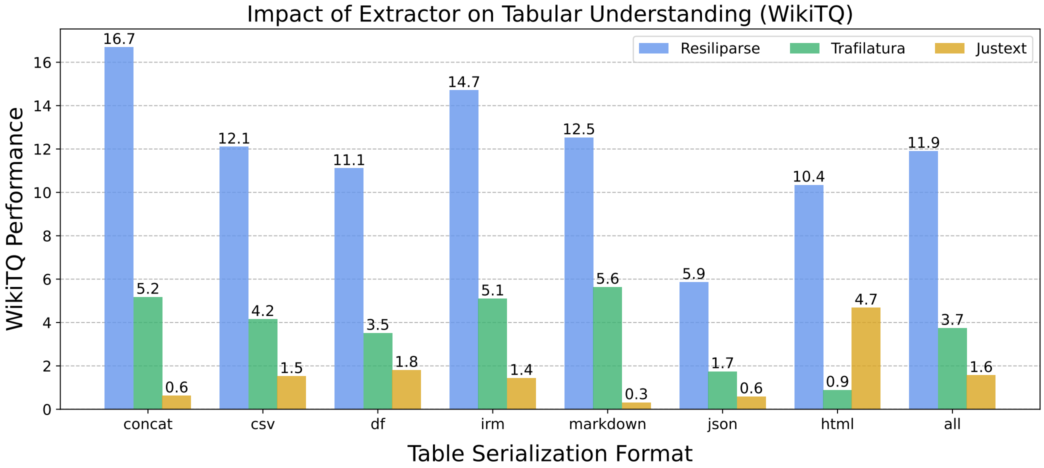


Figure 3: **Extractor performance remains consistent across serialization formats.** We plot the WikiTQ performance stratified across different test-time table serializations. Despite `resiliparse` and `trafilatura` producing tables that most closely match "concat" and "markdown" formats respectively, we observe a surprising degree of generalization across most serializations (with the exception of "json").

Table 4: **CC-Tables-`resiliparse` closes 72% of the gap between the original DCLM-7B-8k model and Llama-3-8B.** The top rows show results for existing models while the bottom rows are new models obtained by replacing the original DD-style cooldown for DCLM-7B-8K with one that mixes CC-Tables-`resiliparse` and DCLM-BASELINE and performs WiSE-FT (Wortsman et al., 2022). We try different interpolation weights $\alpha \in [0.8, 1.0]$ with $\alpha = 0.85$ providing a good balance between tabular and standard evaluations.

Model / Cooldown Dataset	Model Size	Tokens	WikiTQ-Avg.	CORE_V2	MMLU
LLama-3-8B	8B	15T	**29.3**	56.5	66.2
DCLM-7B-8k	7B	2.8T	5.0	56.0	63.7
DCLM-Bas. + CC-Tables ($\alpha = 1.00$)	7B	2.7T	**22.7**	53.7	62.3
DCLM-Bas. + CC-Tables ($\alpha = 0.90$)	7B	2.7T	22.8	54.7	62.6
DCLM-Bas. + CC-Tables ($\alpha = 0.85$)	7B	2.7T	22.4	55.7	62.7
DCLM-Bas. + CC-Tables ($\alpha = 0.80$)	7B	2.7T	18.8	**56.1**	**63.0**

5 Extraction of code blocks

We conduct similar experiments for code blocks as we do for tables, isolating pages that likely contain genuine code blocks and then exploring the impacts of different extractors on downstream models. Here, we evaluate on HumanEval (Chen et al., 2021) using the BigCode evaluation harness (Ben Allal et al., 2022) to measure the ability of models to write code completions based on provided function signatures and docstrings.

5.1 CC-Code: filtering Common Crawl for code blocks

Similar to Redstone-Code (Chang et al., 2024), we aim to identify genuine code blocks that are interleaved with natural text from the general web (e.g., tutorials, documentation, and forums) in an effort to complement traditional "source-code" datasets (e.g., created from GitHub) that are commonly used during pretraining. Our approach differs from that of Chang et al. (2024) in that we (1) focus on directly classifying `<pre>` elements instead of relying on the presence of child `<code>` elements; (2) train a model for said classification as opposed to using regular expressions. We do not claim nor evaluate whether our approach is better for curating a more performant dataset. Rather, our goal is to explore the impact of different extractors given a reasonable filtered set of code-related pages.

We limit our exploration to `<pre>` HTML elements, as the `<pre>` tag is commonly used to maintain pre-formatted text (i.e., preserving spacing such as tabs and line breaks). However, based upon qualitative inspection, we found that `<pre>` elements are also used often for other purposes such as to represent poems, lyrics, and emails. As such, we train a `fastText` classifier to filter out non-code usages of `<pre>`. We construct the training set for this classifier primarily by manually inspecting and labeling the 160 most frequent domains containing

Table 5: **CC-Code extraction and code generation performance.** We observe that using `jusText` leads to the additional CC-Code pages harming HumanEval performance. Meanwhile, `resiliparse` and `trafilatura` perform comparably. In each mix we train on 25% StackV2, 25% CC-Code (when present), and fill in the remainder with DCLM-BASELINE. The top rows are for cooldown runs on top of the DCLM-7B (Li et al., 2024) checkpoint prior to their original cooldown at 2T tokens. The bottom rows correspond to training `7B-2x` models from scratch. All code generation metrics are reported as Pass@1 computed from 20 samples per problem.

Dataset	Python	Java	JS	C++	Go	HE-Avg.	CORE_V2	MMLU
Alternative 200B token cooldown for DCLM-7B								
DCLM-BASELINE (Top-7%) + Stack v2 + CC-Code-`res.`	26.7	**29.9**	24.0	**26.6**	**17.7**	**25.0**	54.7	60.8
DCLM-BASELINE (Top-7%) + Stack v2 + CC-Code-`traf.`	28.0	27.0	**25.3**	23.0	16.9	24.0	54.3	60.5
DCLM-BASELINE (Top-7%) + Stack v2 + CC-Code-`jusT.`	25.0	24.1	21.7	20.3	15.8	21.4	54.0	58.9
DCLM-BASELINE (Top-7%) + Stack v2	**28.1**	23.6	24.6	20.9	15.2	22.5	**55.4**	**62.4**
Trained from scratch, DCLM `7B-2x` scale								
DCLM-BASELINE (Original) + Stack v2 + CC-Code-`res.`	20.5	20.1	18.5	**19.8**	11.7	18.1	44.0	41.8
DCLM-BASELINE (Original) + Stack v2 + CC-Code-`traf.`	**23.2**	20.8	18.6	16.6	**13.8**	18.6	43.0	41.1
DCLM-BASELINE (Original) + Stack v2 + CC-Code-`jusT.`	19.4	16.4	16.6	16.4	11.6	16.1	40.4	33.2
DCLM-BASELINE (Original) + Stack v2	21.9	**20.8**	**19.6**	19.5	13.0	**18.9**	**47.4**	**50.7**

`<pre>` as either code or non-code related (ignoring a domain if ambiguous). After applying our classifier to score `<pre>` elements, we retain pages that contain at least one that is high-scoring. We refer to the surviving set of pages as CC-Code and then evaluate different extractors by applying them to CC-Code, before also using DCLM's English and `fastText` filters to improve overall quality. More details about our filtering pipeline for CC-Code are provided in Appendix E.2.

5.2 Qualitative extractor differences

We present some examples from CC-Code in Appendix E.1. Likely due to its dependence on stopwords, we find that `jusText` often ends up excluding code blocks. Meanwhile, we find that `trafilatura` and `resiliparse` both keep some code blocks but the former fails to preserve proper formatting (e.g., collapsing newlines and removing indentation). It is a natural question whether or not these misformattings might hinder code generation ability, or if additionally including traditional source-code (which is assumed to be properly formatted) in training is enough alleviate these issues.

5.3 Empirical results

Training setup. We train models in two different setups. First, we pretrain models from scratch using the hyperparamters from the DCLM `7B-2x` track (Table 5, bottom). Second, we start from DCLM-7B's pre-cooldown checkpoint (trained on 2T tokens) and modify the original 200B cooldown dataset (Table 5, top). Originally in DCLM, the cooldown dataset was a mix of more strictly filtered DCLM-BASELINE (i.e., Top-7%) mixed with 30% math data. Here, since we care about code performance, we train on mixes that are at least 25% code from Stack v2 (Wei et al., 2024) and optionally 25% CC-Code. The remainder is then made up of DCLM-BASELINE.

What extractors are best for CC-Code? As expected from our inspection, we observe from Table 5 that `jusText` leads to sub-optimal extraction of CC-Code pages. In the "cooldown" setup, it performs 2.6 p.p. worse than `resiliparse` and `trafilatura`, and even 1.0 p.p. worse than replacing CC-Code with general web documents. Meanwhile, better utilization of CC-Code pages can be achieved with `trafilatura` and `resiliparse`, which are able to improve code performance, particularly for Java and C++. Between the two, `resiliparse` offers slightly better performance, perhaps suggesting that its improved whitespace preservation helps languages where line breaks and indentation matter (i.e., JavaScript is a notable exception as it is more robust to whitespace removals). However, we do note that the absolute performance metrics are still close overall.

In the "from-scratch" setup, `jusText` still leads to the worst performance but CC-Code is not as useful overall as dropping CC-Code altogether is slightly better than including it. We speculate that this may be due to source-code being more relevant to HumanEval-style evaluations than interleaved code, as the prompts are already in the form of partially written code. Therefore, when starting from scratch, it may be more important to build up a base of knowledge from seeing more source-code before interleaved data becomes additionally useful. In the "cooldown" setting, the checkpoint we start from has already been trained on a mix containing Stack v2, perhaps allowing for better usage of CC-Code. For both setups, we observe that (perhaps as expected), replacing DCLM-BASELINE tokens with CC-Code tokens decreases standard evaluation performance, with bigger drops in the from-scratch setting.

6 Conclusions

We demonstrate that text extraction is an underexplored aspect of LLM pretraining data curation. While standard practice applies a single extractor to all of Common Crawl, we show this leads to suboptimal data yield—using multiple extractors in parallel can significantly increase yield while maintaining performance. We further show that extractor choice can significantly impact model performance depending on content type, with larger differences on tables and code tasks compared to standard language evaluations. We hope this work encourages practitioners to revisit extraction as a first-class component of their curation pipelines.

Limitations

While we systematically study three popular extractors, we do not develop new extraction methods or sophisticated strategies for selecting between them based on page characteristics. Relatedly, these libraries remain under active development, so newer releases may patch specific failure modes and yield better results. We believe these are all promising directions for future work. Another limitation is that we did not extensively explore alternative filtering strategies for CC-Tables and CC-Code. Our primary aim was to develop reasonable pipelines for isolating pages with genuine tables and code blocks to study the impact of extractor choice. While we believe our conclusions about extractor performance on structured content should generalize across filtering approaches, improved filtering could yield higher absolute scores on downstream evaluations, such as those obtained by Nemotron-CC-Math (Mahabadi et al., 2025). Finally, due to computational constraints, we were unable to fully assess run-to-run variation, particularly for our 7B results.

Ethical considerations

Better extraction can lead to more effective coverage of Internet data, which can come with both benefits and risks. Exposing models to more diverse content could reduce potential blind spots in their capabilities. However, better extraction may also increase exposure to harmful, toxic, or inappropriate content. For instance, it may inadvertently increase the capture of personal information or copyrighted material that a given single extractor might miss. While we do not release datasets or models from our experiments, we recommend that practitioners using these techniques continue to implement robust content filtering and respect data usage rights. Finally, we note that larger datasets enable longer training runs with greater computational and environmental costs that should be weighed against their benefits.

Acknowledgements

We would like to thank Rick Chang, Cem Koc, Stephen Pulman, Denise Hui, Elle Barnes, Merhawie Woldezion, and Pranay Pattani for valuable feedback, guidance, and support. We'd also like to thank Shir Ashury-Tahan, Yifan Mai, and Elron Bandel for helpful discussions related to setting up and using ToRR.

References

Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, and 8 others. 2024. Phi-4 technical report. *Preprint*, arXiv:2412.08905.

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, Joshua Lochner, Caleb Fahlgren, Xuan-Son Nguyen, Clémentine Fourrier, Ben Burtenshaw, Hugo Larcher, Haojun Zhao, Cyril Zakka, Mathieu Morlon, and 3 others. 2025. Smollm2: When smol goes big – data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*.

Shir Ashury-Tahan, Yifan Mai, Rajmohan C, Ariel Gera, Yotam Perlitz, Asaf Yehudai, Elron Bandel, Leshem Choshen, Eyal Shnarch, Percy Liang, and Michal Shmueli-Scheuer. 2025a. The mighty torr: A benchmark for table reasoning and robustness. *arXiv preprint arXiv:2502.19412*.

Shir Ashury-Tahan, Yifan Mai, Rajmohan C, Ariel Gera, Yotam Perlitz, Asaf Yehudai, Elron Bandel, Leshem Choshen, Eyal Shnarch, Percy Liang, and Michal Shmueli-Scheuer. 2025b. The mighty torr: A benchmark for table reasoning and robustness. *arXiv preprint arXiv:2502.19412*.

Elron Bandel, Yotam Perlitz, Elad Venezian, Roni Friedman, Ofir Arviv, Matan Orbach, Shachar Don-Yehiya, Dafna Sheinwald, Ariel Gera, Leshem Choshen, Michal Shmueli-Scheuer, and Yoav Katz. 2024. Unitxt: Flexible, shareable and reusable data preparation and evaluation for generative AI. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 3: System Demonstrations)*, pages 207–215, Mexico City, Mexico. Association for Computational Linguistics.

Adrien Barbaresi. 2021. Trafilatura: A web scraping library and command-line tool for text discovery and extraction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 122–131, Online. Association for Computational Linguistics.

Loubna Ben Allal, Niklas Muennighoff, Logesh Kumar Umapathi, Ben Lipkin, and Leandro von Werra. 2022. A framework for the evaluation of code generation models. `https://github.com/bigcode-project/bigcode-evaluation-harness`.

Janek Bevendorff, Sanket Gupta, Johannes Kiesel, and Benno Stein. 2023. An empirical comparison of web content extraction algorithms. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '23, page 2594–2603, New York, NY, USA. Association for Computing Machinery.

Janek Bevendorff, Martin Potthast, and Benno Stein. 2021. FastWARC: Optimizing Large-Scale Web Archive Analytics. In *International Symposium on Open Search Technology (OSSYM)*. `https://github.com/chatnoir-eu/chatnoir-resiliparse`.

Janek Bevendorff, Benno Stein, Matthias Hagen, and Martin Potthast. 2018. Elastic ChatNoir: Search Engine for the ClueWeb and the Common Crawl. In *European Conference on Information Retrieval Research (ECIR)*. `https://github.com/chatnoir-eu/chatnoir-resiliparse`.

Yaoyao Chang, Lei Cui, Li Dong, Shaohan Huang, Yangyu Huang, Yupan Huang, Scarlett Li, Tengchao Lv, Shuming Ma, Qinzheng Sun, and 1 others. 2024. RedStone: Curating general, code, math, and QA data for large language models. *arXiv preprint arXiv:2412.03398*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde, Jared Kaplan, Harrison Edwards, Yura Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 34 others. 2021. Evaluating large language models trained on code. *ArXiv preprint*, abs/2107.03374.

Common Crawl. 2007. Common Crawl. `https://commoncrawl.org`.

Together Computer. 2023. Redpajama: an open dataset for training large language models.

Jesse Dodge, Maarten Sap, Ana Marasović, William Agnew, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. 2021. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 1286–1305, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Julian Eberius, Katrin Braunschweig, Markus Hentsch, Maik Thiele, Ahmad Ahmadov, and Wolfgang Lehner. 2015. Building the dresden web table corpus: A classification approach. In *2015 IEEE/ACM 2nd International Symposium on Big Data Computing (BDC)*, pages 41–50.

Gus Eggert, Kevin Huo, Mike Biven, and Justin Waugh. 2023. Tablib: A dataset of 627m tables with context. *arXiv preprint arXiv:2310.07875*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn

Presser, and Connor Leahy. 2021. The Pile: An 800gb dataset of diverse text for language modeling. *ArXiv preprint*, abs/2101.00027.

Joshua P Gardner, Juan Carlos Perdomo, and Ludwig Schmidt. 2024. Large scale transfer learning for tabular data via language modeling. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Suchin Gururangan, Mitchell Wortsman, Samir Yitzhak Gadre, Achal Dave, Maciej Kilian, Weijia Shi, Jean Mercat, Georgios Smyrnis, Gabriel Ilharco, Matt Jordan, Reinhard Heckel, Alex Dimakis, Ali Farhadi, Vaishaal Shankar, and Ludwig Schmidt. 2023. OpenLM: a minimal but performative language modeling (lm) repository. `https://github.com/mlfoundations/open_lm`.

Felix Hamborg, Norman Meuschke, Corinna Breitinger, and Bela Gipp. 2017. news-please: A Generic News Crawler and Extractor. In *Proceedings of the 15th International Symposium of Information Science*.

Xiaotian Han, Yiren Jian, Xuefeng Hu, Haogeng Liu, Yiqi Wang, Qihang Fan, Yuang Ai, Huaibo Huang, Ran He, Zhenheng Yang, and Quanzeng You. 2025. Infimm-webmath-40b: Advancing multimodal pre-training for enhanced mathematical reasoning.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. In *9th International Conference on Learning Representations, ICLR 2021, Virtual Event, Austria, May 3-7, 2021*. OpenReview.net.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, and 1 others. 2022. Training compute-optimal large language models. In *Advances in Neural Information Processing Systems (NeurIPS)*. `https://arxiv.org/abs/2203.15556`.

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. 2010. Boilerplate detection using shallow text features. In *Proceedings of the Third ACM International Conference on Web Search and Data Mining*, WSDM '10, page 441–450, New York, NY, USA. Association for Computing Machinery.

Oliver Lehmberg, Dominique Ritze, Robert Meusel, and Christian Bizer. 2016. A large public corpus of web tables containing time and context metadata. In *Proceedings of the 25th International Conference Companion on World Wide Web*, WWW '16 Companion, page 75–76, Republic and Canton of Geneva, CHE. International World Wide Web Conferences Steering Committee.

Jurek Leonhardt, Avishek Anand, and Megha Khosla. 2020. Boilerplate removal using a neural sequence labeling model. In *Companion Proceedings of the Web Conference 2020*, WWW '20, page 226–229, New York, NY, USA. Association for Computing Machinery.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, and 40 others. 2024. Datacomp-lm: In search of the next generation of training sets for language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 14200–14282. Curran Associates, Inc.

Anton Lozhkov, Loubna Ben Allal, Leandro von Werra, and Thomas Wolf. 2024a. Fineweb-edu.

Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, Tianyang Liu, Max Tian, Denis Kocetkov, Arthur Zucker, Younes Belkada, Zijian Wang, Qian Liu, Dmitry Abulkhanov, Indraneil Paul, and 47 others. 2024b. Starcoder 2 and the stack v2: The next generation. *ArXiv preprint*, abs/2402.19173.

Rabeeh Karimi Mahabadi, Sanjeev Satheesh, Shrimai Prabhumoye, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. 2025. Nemotron-cc-math: A 133 billion-token-scale high quality math pretraining dataset.

Meta AI. 2024. Introducing meta llama 3: The most capable openly available llm to date. `https://ai.meta.com/blog/meta-llama-3/`.

Keiran Paster, Marco Dos Santos, Zhangir Azerbayev, and Jimmy Ba. 2023. Openwebmath: An open dataset of high-quality mathematical web text. *ArXiv preprint*, abs/2310.06786.

Panupong Pasupat and Percy Liang. 2016. Inferring logical forms from denotations. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23–32, Berlin, Germany. Association for Computational Linguistics.

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Alessandro Cappelli, Hamza Alobeidli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2023. The RefinedWeb dataset for Falcon LLM: outperforming curated corpora with web data, and web data only. *ArXiv preprint*, abs/2306.01116.

Jan Pomikálek. 2011. *Removing boilerplate and duplicate content from web corpora*. Ph.D. thesis, Masaryk university, Faculty of informatics, Brno, Czech Republic.

Hadi Pouransari, Chun-Liang Li, Jen-Hao Rick Chang, Pavan Kumar Anasosalu Vasu, Cem Koc, Vaishaal Shankar, and Oncel Tuzel. 2024. Dataset decomposition: Faster llm training with variable sequence length curriculum. *arXiv preprint arXiv:2405.13226*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *J. Mach. Learn. Res.*, 21:140:1–140:67.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, and 1 others. 2024. Dolma: An open corpus of three trillion tokens for language model pretraining research. *ArXiv preprint*, abs/2402.00159.

Dan Su, Kezhi Kong, Ying Lin, Joseph Jennings, Brandon Norick, Markus Kliegl, Mostofa Patwary, Mohammad Shoeybi, and Bryan Catanzaro. 2024. Nemotron-cc: Transforming common crawl into a refined long-horizon pretraining dataset. *arXiv preprint arXiv:2412.02595*.

Together Computer. 2023. Redpajama: an open dataset for training large language models. https://github.com/togethercomputer/RedPajama-Data.

Boris van Breugel and Mihaela van der Schaar. 2024. Why tabular foundation models should be a research priority. *arXiv preprint arXiv:2405.01147*.

Thijs Vogels, Octavian-Eugen Ganea, and Carsten Eickhof. 2018. Web2text: Deep structured boilerplate removal. In *Advances in Information Retrieval, ECIR 2018*.

Yuxiang Wei, Federico Cassano, Jiawei Liu, Yifeng Ding, Naman Jain, Harm de Vries, Leandro von Werra, Arjun Guha, and Lingming Zhang. 2024. Starcoder2-instruct: Fully transparent and permissive self-alignment for code generation. https://huggingface.co/blog/sc2-instruct.

Mitchell Wortsman, Gabriel Ilharco, Jong Wook Kim, Mike Li, Simon Kornblith, Rebecca Roelofs, Raphael Gontijo Lopes, Hannaneh Hajishirzi, Ali Farhadi, Hongseok Namkoong, and Ludwig Schmidt. 2022. Robust fine-tuning of zero-shot models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 7959–7971.

Zhipeng Xu, Zhenghao Liu, Yukun Yan, Zhiyuan Liu, Chenyan Xiong, and Ge Yu. 2024. Cleaner pretraining corpus curation with neural web scraping. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*.

Pengcheng Yin, Graham Neubig, Wen tau Yih, and Sebastian Riedel. 2020. TaBERT: Pretraining for joint understanding of textual and tabular data. In *Annual Conference of the Association for Computational Linguistics (ACL)*.

Kryštof Zamazal. 2024. Evaluation of web page cleaning tool justext.

Shuo Zhang and Krisztian Balog. 2025. Web table extraction, retrieval and augmentation: A survey. *arXiv preprint arXiv:2002.00207*.

A Extended related work

Targeted extractors for specific domains. When extracting specific types of web pages, other works have sometimes developed custom approaches. For news articles, `newspaper3k`[3] and `news-please` (Hamborg et al., 2017) target cleaner extractions with better date and author parsing. For cooking recipes, `recipe-scrapers`[4] offers targeted parsing of ingredients, instructions, and cooking times. More recently, in the context of pretraining, custom extractors have been heavily emphasized as an important step for building mathematics datasets. As more general approaches were shown to inaccurately/inconsistently handle LaTeX equations, both OpenWebMath (Paster et al., 2023) and InfiMM-WebMath (Han et al., 2025) develop their own modifications to `resiliparse`, the former of which was also later used by FineMath (Allal et al., 2025). Nemotron-CC-Math (Mahabadi et al., 2025) offers state-of-the-art performance by re-extracting pages from the aforementioned datasets by making use of `lynx`, a text-based browser that more robustly handles math and code. They also clean these extractions with a fine-tuned LLM.

Datasets for web-crawled tables Compared to simply collecting existing tabular datasets, a promising complementary approach for improving tabular understanding is to curate large interleaved table/text datasets from the general web (Zhang and Balog, 2025). TabLib (Eggert et al., 2023) collects 627M (minimally filtered) tables with their contextual metadata (e.g., surrounding text) by crawling GitHub and one dump of Common Crawl (`CC-MAIN-2023-23`). Of their tables, 219M are HTML tables from Common Crawl. Tablib was then further filtered to create T4 (Gardner et al., 2024) via a set of table/row/column-level heuristics (e.g., based on counts, prevalence of NANs, and data homogeneity). The goal of this dataset was to improve downstream model performance for tabular prediction tasks (i.e., selecting and predicting a chosen target feature from the other feature columns). Another set of earlier but closely related efforts are the web tables collected by Web Data Commons (WDC) (Lehmberg et al., 2016; Eberius et al., 2015). These datasets are created from a pair of Common Crawl dumps (from 2012 and 2015) and involve a two step process where tables are filtered based on (1) hand-engineered features (e.g., sparseness, average attribute size, number of links); (2) classifying each table as either Relational, Entity, Matrix, or Layout via learned classifiers. The relational subset was used by Yin et al. (2020) to train TaBERT (along with an additional dataset based on just Wikipedia tables). In comparison, our approach for CC-Tables uses a smaller set of heuristic rules (i.e., based on header presence, row/columns counts, and row size consistency) and trains a weakly supervised `fastText` classifier directly on HTML content rather than a SVM or decision tree on hand-engineered features.

Datasets for interleaved code and natural text. For open code pretraining datasets, The Stack v2 and its filtered subsets from StarCoder (Lozhkov et al., 2024b) are commonly used directly to train models or as starting points for further curation. Mostly created from GitHub, it primarily consists of source code but also contains some with interleaved code/text (e.g., Issues/Commits/PRs, Jupyter notebooks, and code documentation). Meanwhile, RedStone-Code (Chang et al., 2024) directly sources interleaved documents by filtering pages from Common Crawl. They focus on `<code>` elements, classify blocks as code (or not) based upon regexes for common language keywords/patterns, and extract such pages using the same pipeline as Common Crawl WET files. In contrast, we train a `fastText` classifier instead of using regex rules and we do not rely on the presence of `<code>`, though we do use it as signal for sourcing positive training examples for our filtering model. Lastly, Nemotron-CC-Math (Mahabadi et al., 2025), as mentioned earlier, re-extracts content from popular math datasets (which incidentally contains significant amounts of code), finding that their re-extractions can improve performance on code evaluations.

[3] https://github.com/codelucas/newspaper

[4] https://github.com/hhursev/recipe-scrapers

B Assets and licenses

We used the following assets all for research purposes.

Datasets

- We make heavy use of the DCLM codebase and datasets (DCLM-POOL, DCLM-REFINEDWEB, DCLM-BASELINE), which are available by CC-BY-4 license.
- We also directly source raw data from Common Crawl WARCs, which are under their own ToU[5].

Extractor libraries

- `resiliparse` (`v0.14.5`) is made available via Apache 2.0 License. We use this version to match up with DCLM.
- `trafilatura` (`v2.0.0`) is made available via Apache 2.0 License
- `jusText` (`v3.0.2`) is made available via The BSD 2-Clause License

Evaluation harnesses

- `llm-foundry`[6] is used for DCLM's evaluations and is available under Apache 2.0 License.
- `unitxt`[7] (Bandel et al., 2024) implements ToRR (Ashury-Tahan et al., 2025b) and is available via Apache 2.0 License
- `bigcode-evaluation-harness`[8] (Ben Allal et al., 2022) is avaialble under Apache 2.0 License

[5] https://commoncrawl.org/terms-of-use
[6] https://github.com/mosaicml/llm-foundry
[7] https://github.com/IBM/unitxt
[8] https://github.com/bigcode-project/bigcode-evaluation-harness

C Additional results for union over extractors

Table 6: **Run-to-run variation at 1B-1x scale.** We report mean CORE_V2 performance and standard deviation across three runs for resiliparse and trafilatura. Note that results in Table 1 were all single-runs since we did not try multiple seeds for all methods.

Extractor	Run 1	Run 2	Run 3	Mean (Std)
resiliparse	28.5	27.5	28.8	28.3 (0.7)
trafilatura	29.6	27.7	28.5	28.6 (1.0)

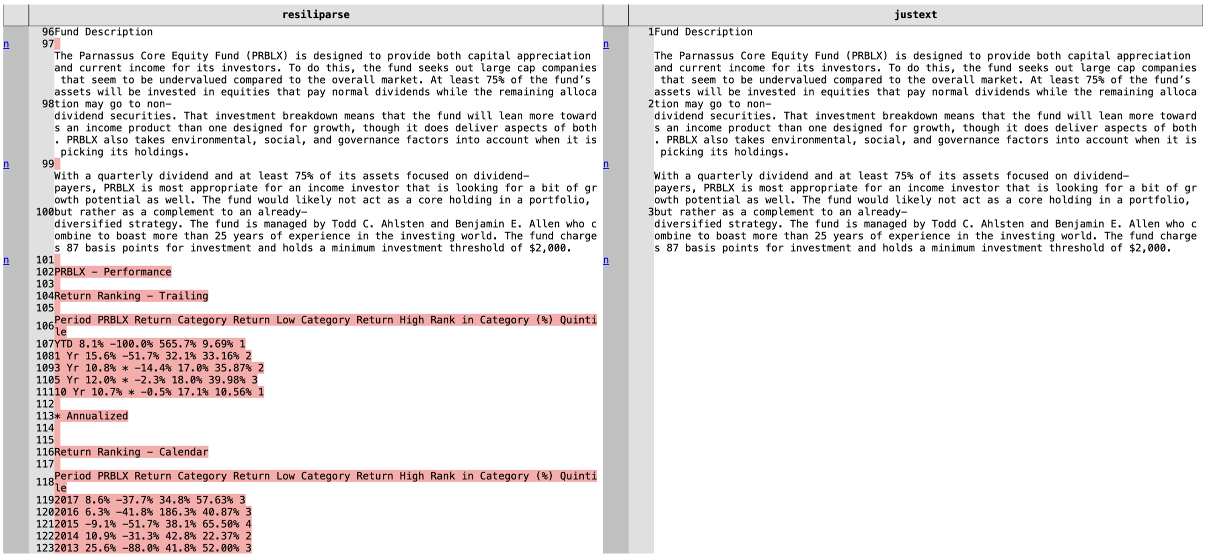
Table 7: **Using multiple extractors improves performance in settings which are data-constrained.** We train 1B-5x models on progressively smaller subsamples of data curated from the 1B-1x raw pool. More heavy subsampling leads to more repetition and thus worse performance, but this can be mitigated by having a curation pipeline that yields more tokens (e.g., our Union datasets). A subset of these results are plotted in Figure 2.

Initial Pool	Extractor	fastText Thresh.	Repeats	CORE_V2
1B-0.25x	resiliparse	0.11	15.0x	30.8
	Union (Random)	(0.11, 0.11, 0.11)	10.5x	32.3
	Union (Manual)	(0.11, 0.11, 0.11)	10.2x	32.4
	Union (Random)	(0.11, 0.15, 0.15)	9.0x	**33.5**
	Union (Manual)	(0.11, 0.15, 0.15)	8.7x	**33.5**
1B-0.33x	resiliparse	0.11	11.2x	33.1
	Union (Random)	(0.11, 0.11, 0.11)	7.9x	33.3
	Union (Manual)	(0.11, 0.11, 0.11)	7.7x	33.3
	Union (Random)	(0.11, 0.15, 0.15)	6.6x	33.4
	Union (Manual)	(0.11, 0.15, 0.15)	6.3x	**33.8**
1B-0.5x	resiliparse	0.11	7.2x	32.7
	Union (Random)	(0.11, 0.11, 0.11)	5.3x	34.0
	Union (Manual)	(0.11, 0.11, 0.11)	5.1x	34.3
	Union (Random)	(0.11, 0.15, 0.15)	4.4x	**34.6**
	Union (Manual)	(0.11, 0.15, 0.15)	4.2x	34.3
1B-1x	resiliparse	0.11	3.6x	33.9
	Union (Random)	(0.11, 0.11, 0.11)	2.6x	34.0
	Union (Manual)	(0.11, 0.11, 0.11)	2.5x	33.7
	Union (Random)	(0.11, 0.15, 0.15)	2.2x	**34.4**
	Union (Manual)	(0.11, 0.15, 0.15)	2.1x	33.8

D Table understanding experiments

D.1 Example documents

Example 1: http://mutualfunds.com/funds/prblx-parnassus-core-equity-investor/



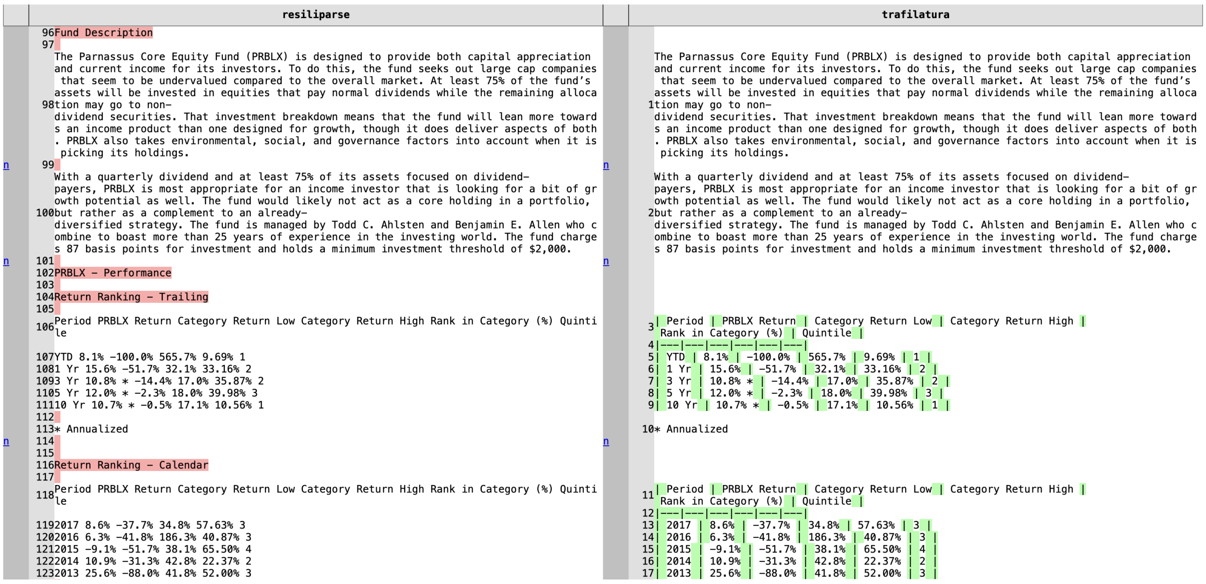


Figure 4: **Extraction comparison for mutual fund data.** We use `difflib` to visualize pairwise comparisons between `resiliparse` (left) and `jusText` (top) or `trafilatura` (bottom). For both tables in this page, `jusText` removes them while `trafilatura` applies markdown formatting.

Example 2: http://bmcinfectdis.biomedcentral.com/articles/10.1186/1471-2334-12-102

Figure 5: **Extraction comparison for an BioMed Central article.** We use `difflib` to visualize pairwise comparisons between `resiliparse` (left) and `jusText` (top) or `trafilatura` (bottom). Note that for the table shown, `jusText` removes it while `trafilatura` applies markdown formatting. Here, `resiliparse` ends up splitting entries across line breaks instead of single spaces.

Example 3: http://akron.prestosports.com/sports/msoc/2013-14/bios/caso%20clint%20yxrk

Figure 6: **Extraction comparison for soccer recruiting statistics.** We use `difflib` to visualize pairwise comparisons between `resiliparse` (left) and `jusText` (top) or `trafilatura` (bottom). Note that for both tables in this page, `jusText` removes them while `trafilatura` applies markdown formatting.

Example 4: https://bulbapedia.bulbagarden.net/wiki/Whiscash_(Pok%C3%A9mon)

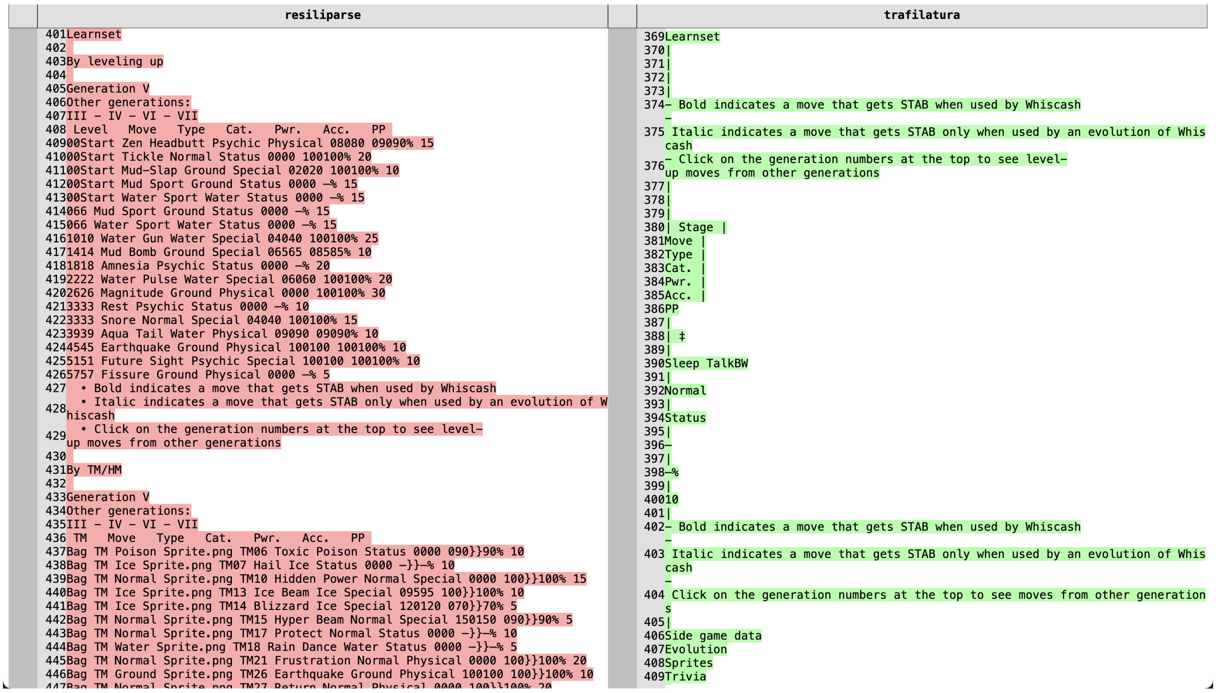


Figure 7: **Extraction comparison for a Pokémon learnset table on Bulbapedia.** We use `difflib` to visualize pairwise comparisons between `resiliparse` and `trafilatura`. Note that `jusText` removes the tables altogether so we do not show it. Meanwhile, `resiliparse` fails to cleanly split some columns while `trafilatura` fails to keep the actual contents.

D.2 Filtering for CC-Tables

We start from the raw data corresponding to the 7B-2x version of DCLM-REFINEDWEB. Here though, we prioritize filtering for pages with useful tables rather than for general quality, starting by considering only pages that contain <table> elements. Then we isolate these tables and filter them with a two-stage pipeline to try to keep genuine (relational) data tables. This is important as <table> is often used for other purposes such as to organize text in an aligned layout. We assume these "non-genuine" tables would not be useful for tabular understanding tasks.

Structure-based filtering. In this first stage, we filter out any table that

- Does not contain headers (i.e., <th> elements)
- Has fewer than 10 rows and 3 columns. These are more strict thresholds than those used by Lehmberg et al. (2016).
- Has an inconsistent number of columns per row.

Content-based filtering. After this first stage, we still found that many tables appeared that were unlikely to be useful (e.g. product lists, sizing charts, lists of forum threads). We also wished to prioritize tables that exist in knowledge-rich pages that interleave tables with natural text (e.g., Wikipedia articles). Based on these two goals, we train a `fastText` model to classify and further filter HTML tables. To do so, we first inspected the tables that survived structure-based filtering and created a weakly labeled training set. Specifically, we source positive and negative examples by inspecting tables and observing patterns from their source page's URLs[9]:

Negative examples: We focus on tables whose contents are either empty, esoteric, or unlikely to have supporting natural text

- *Product listings*: URLs containing one of the subwords: "`shop`", "`products`", "`cart`", "`items`", "`store`", "`promotion`", "`productdisplay`"
- *Site metrics*: URLs that end with "`metrics`"
- *Forum listings*: URLs containing either "`forum`" or "`forums`" as a subword
- *User profile pages*: URLs containing one of the subwords: "`users`", "`members-`"
- *Weather forecasts from Accuweather:* URLs from "`accuweathercom`"
- *Patent Listings:* URLs from Google Patents as they contain little to no associated natural text

Positive examples: We focus on genuine relational data tables that are likely to contain useful knowledge and be associated with natural text on their source page.

- *Wikipedia*: URLs from English Wikipedia
- *Government websites:* URLs from "`.gov`" domains
- *Dataset releases:* URLs containing one of the subwords: "`statistics`", "`database`", "`dataset`"
- *Article and Documentation:* URLs containing the subwords "`article`" or substring "`docs.`" to encourage interleaved text and tables
- Tables from T4: These are tables from TabLib (Eggert et al., 2023) that have been filtered as higher-quality by Gardner et al. (2024). While not all of these tables originate from Common Crawl (and were used by Gardner et al. (2024) as training data for tabular prediction tasks), we convert them to HTML serializations using `pandas`. We believe these tables are useful to include both for their contents but also the clean resulting HTML structure.

We then train a `fastText` model on the HTML versions of these tables, allowing the model to use both contents and markup. While by default `fastText` uses space-delimiting for constructing n-grams, we found it to be better to first tokenize each HTML table using the `o200k_base` encoder from `tiktoken`, allowing the model to instead train and make predictions on sequences of token indices. We use default hyperparameters with the exception of increasing the number of n-grams to 4 (from the default of 1). When applying our `fastText` model,

[9] Note that while we make use of URLs for labeling, we do not use them as input features to the model.

we consider any table to have a score above 0.75 to be useful. We then remove any page that does not contain a single table that meets this threshold. This left us with 62B tokens after `resiliparse` extraction, 59B tokens after `trafilatura`, and 21B tokens after `jusText`.

D.3 Modifications to ToRR's WikiTQ

In the original ToRR (Ashury-Tahan et al., 2025b) implementation of WikiTQ, the chosen preprocessors convert both the generated predictions and answers to the string representation of a list of words. This ends up rewarding short generations even if they contain a completely incorrect answer.

As an example, suppose `<prediction>` and `<answer>` are single-token words:

- They get converted to the strings "`['<prediction>']`" and "`['<answer>']`"
- The evaluation uses an F1 score that is applied over the tokenized sets of 5 tokens each:
  - Prediction: { `[`, `'`, `<prediction>`, `'`, `]`}
  - Answer: { `[`, `'`, `<answer>`, `'`, `]` }
- The brackets and single-quotes count as tokens and always appear in both the prediction and answer being compared. So even when `<prediction>` $\neq$ `<answer>`, we'd still get a score of 0.8.

To address this, we removed the list conversion (which is unnecessary since each true answer refers to a single entity) and added standard normalizers for leading/trailing whitespace and lowercasing. We now directly compute F1 over prediction and answer strings (tokenized with SpaCy).

D.4 Additional results

Table 8: **Conclusions about extractors hold regardless of mixing ratio or whether the DCLM-BASELINE and CC-Tables extractors are consistent (`7B-1x` scale).** We compare the performance of `resiliparse`, `trafilatura`, and `jusText` on pages from our CC-Tables dataset when it is mixed with different ratios and different extractions of DCLM-BASELINE (bottom section). Overall, we observe that there is a slight increase to WikiTQ performance for `resiliparse` as the ratio of CC-Tables is larger but that conclusions between the extractors are the same across all settings.

CC-Tables Extraction	DCLM-BASELINE Extraction	CC-Tables Mixing Ratio	WikiTQ-Avg.
`resiliparse`	`resiliparse`	0.1	**10.1**
`trafilatura`	`resiliparse`	0.1	2.2
`jusText`	`resiliparse`	0.1	1.9
`resiliparse`	`resiliparse`	0.2	**11.9**
`trafilatura`	`resiliparse`	0.2	3.7
`jusText`	`resiliparse`	0.2	1.6
`resiliparse`	`resiliparse`	0.5	**12.4**
`trafilatura`	`resiliparse`	0.5	3.4
`trafilatura`	`trafilatura`	0.5	2.7
`jusText`	`resiliparse`	0.5	1.5
`jusText`	`jusText`	0.5	1.7

E Code experiments

E.1 Example documents

Example 1: http://msdn.microsoft.com/en-us/library/cx9s2sy4(v=vs.100).aspx

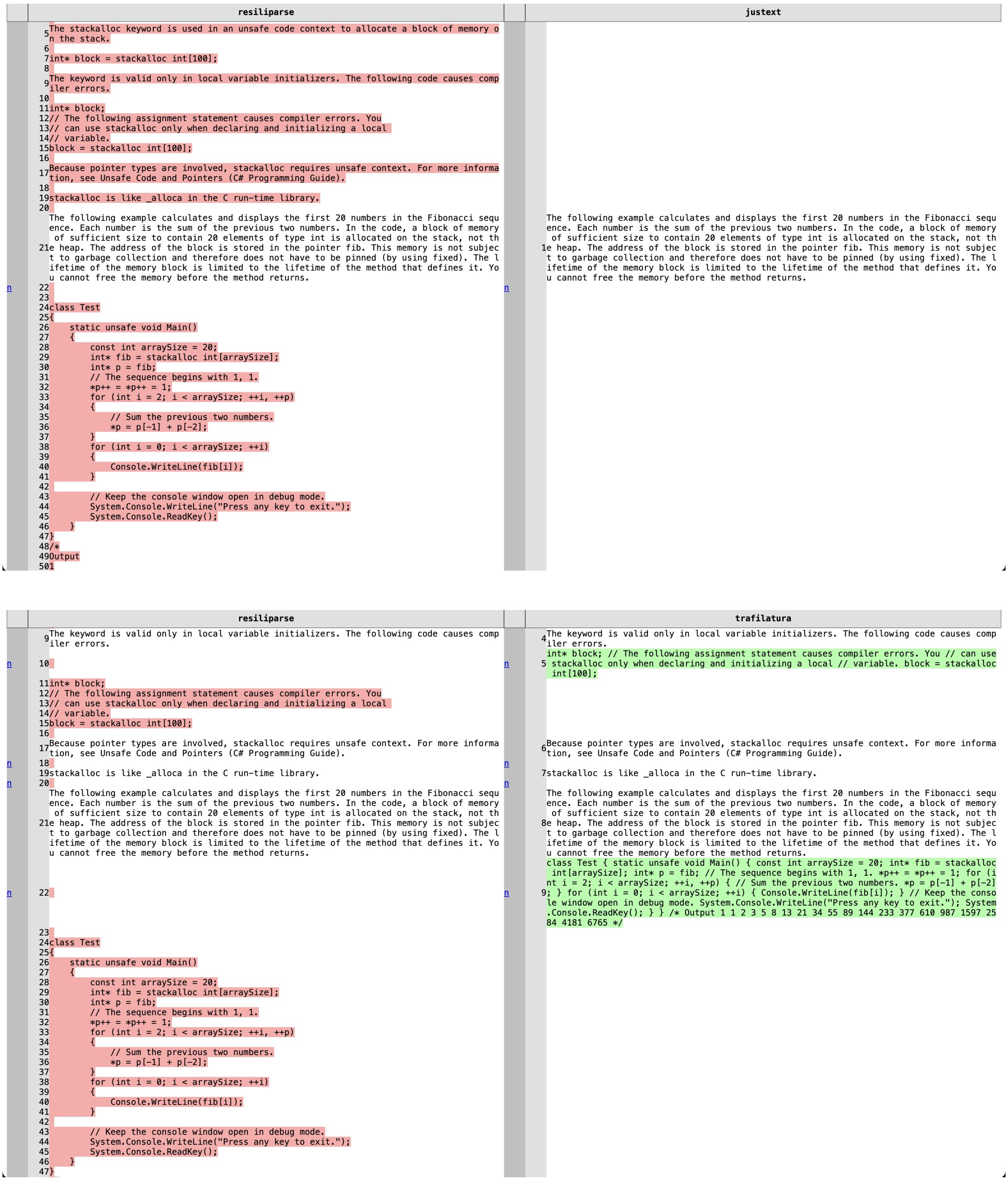
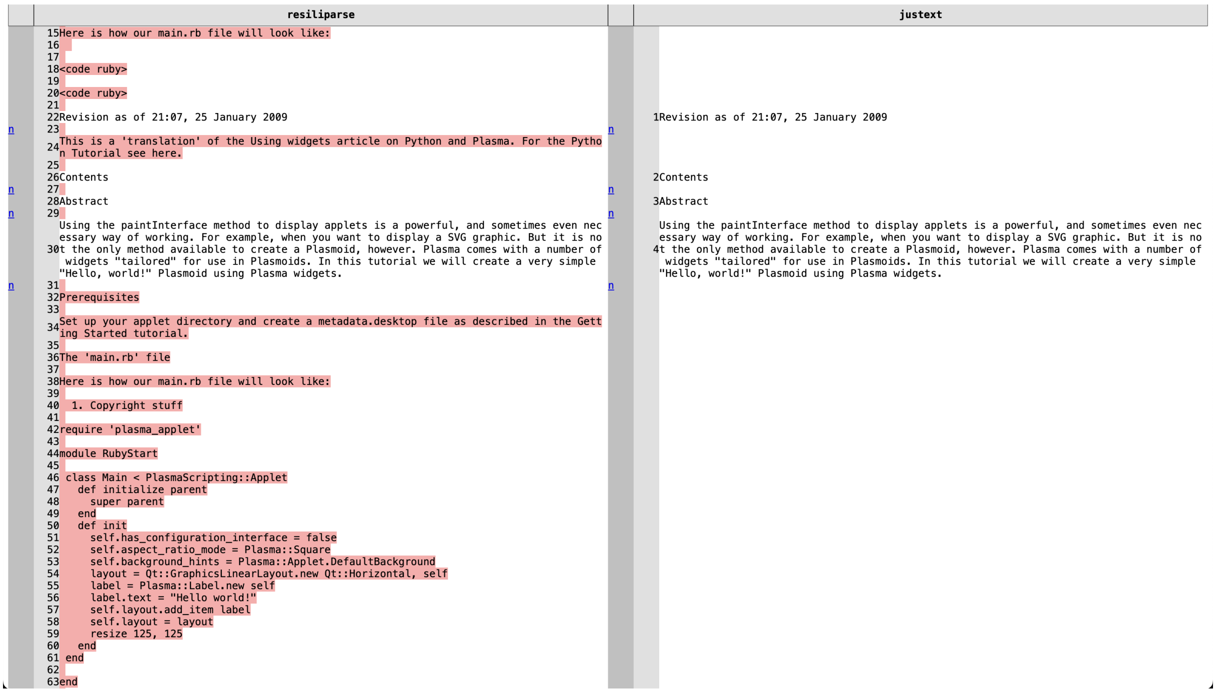


Figure 8: **Extraction comparison for a C# tutorial page.** We use `difflib` to visualize pairwise comparisons between `resiliparse` (left) and `jusText` (top) or `trafilatura` (bottom). Note that for code blocks in this page, `jusText` removes them while `trafilatura` collapses whitespace formatting.

Example 2: http://techbase.kde.org/index.php?title=Development/Tutorials/Plasma/Ruby/Using_widgets&diff=37860&oldid=37859



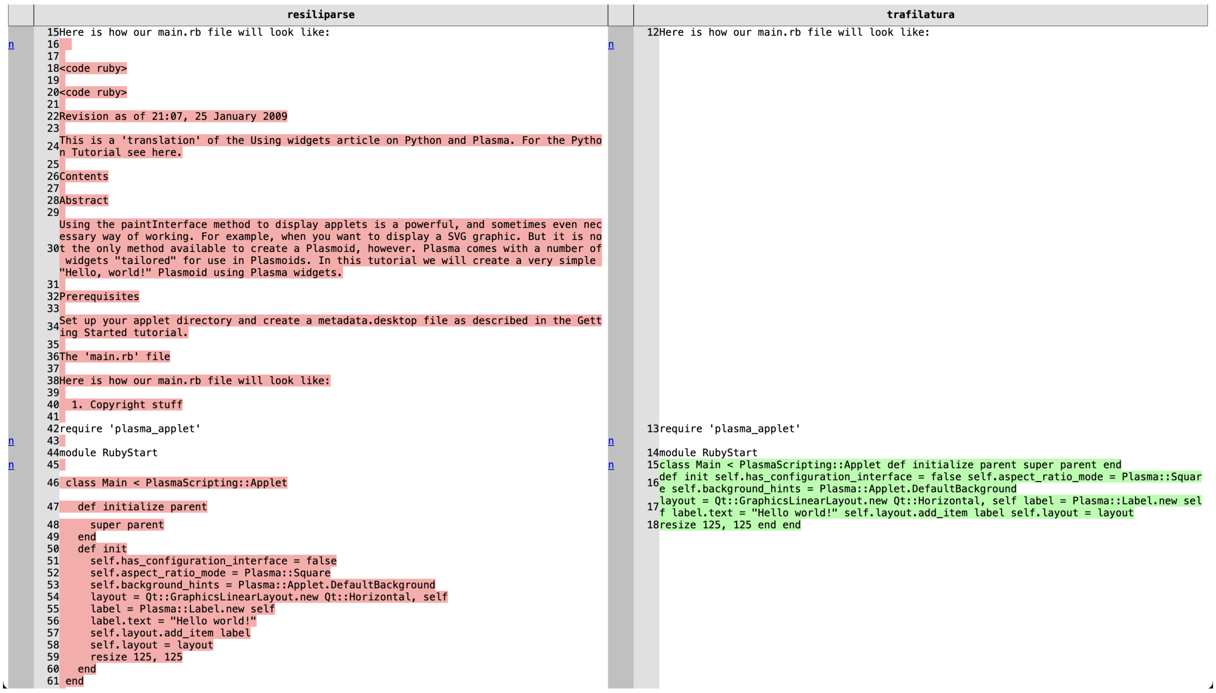
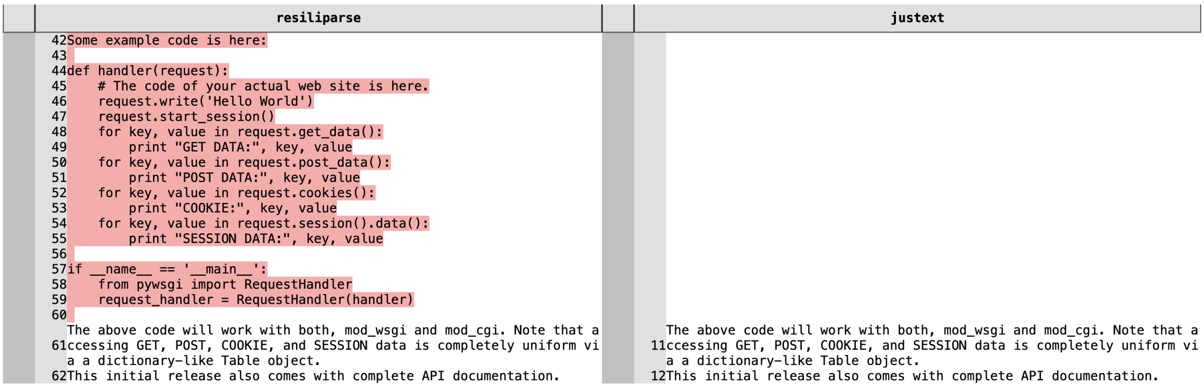


Figure 9: **Extraction comparison for a Ruby Plasma tutorial.** We use difflib to visualize pairwise comparisons between resiliparse (left) and jusText (top) or trafilatura (bottom). Note that for code blocks in this page, jusText removes them while trafilatura collapses whitespace formatting.

Example 3: http://www.advogato.org/person/knipknap/diary.html?start=82



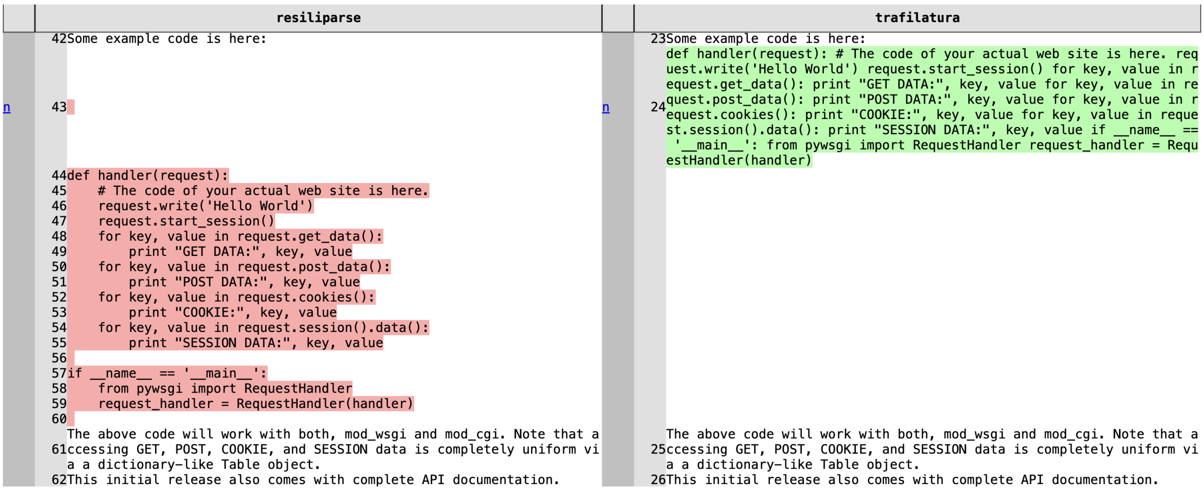


Figure 10: **Extraction comparison for release note blog post.** We use `difflib` to visualize pairwise comparisons between `resiliparse` (left) and `jusText` (top) or `resiliparse` (bottom). Note that for code blocks in this page, `jusText` removes them while `trafilatura` collapses whitespace formatting.

E.2 Filtering for CC-Code

We filter the same starting 7B-2x pool corresponding to DCLM-RefinedWeb. We first retain only pages that contain <pre> elements, focusing on <pre> since it is a common way to incorporate code blocks while preserving whitespace formatting (e.g., indentation). However, <pre> is also used for other types of pre-formatted content such as lyrics, poems, or mailing threads. We therefore train a fastText classifier to better identify genuine code blocks.

Content-based filtering for code blocks. We inspect the 160 domains that contain the most pages with <pre> elements, labeling each domain as either code or non-code related (when unambiguous). We use this categorization to label all <pre> elements from these source pages as positive and negative. For positives, we also include any <pre> elements that contain a child <code> element. As with the classifier we used for CC-Tables, we train and use a model based on *tokenized* HTML inputs. For code, however, we also ensemble this with an additional classifier that operates on an extracted version obtained using the get_text() function from BeautifulSoup[10] which keeps just the visible text. Based upon inspecting classifier scores, we averaged the scores from both and use 0.9 threshold. After this content-based filtering, we run our extractors and perform the same English filtering as is done by DCLM-Baseline but with a slightly looser threshold of 0.25 to account for the presence of code. We also use DCLM's fastText quality classifier to filter down to 55B tokens (to ensure sufficient size for no repeats in the cooldown setting).

F Training details

For our training runs, we generally follow the fixed training recipes (i.e., architectures and hyperparameters) specified by DCLM (Li et al., 2024) for their various competition scales and as implemented in OpenLM (Gururangan et al., 2023). We ran each 1B-1x experiment on 4 nodes of 8xH100 GPUs which, as also indicated in Table 1 from Li et al. (2024), uses 240 total GPU hours. For 7B-1x and 7B-2x experiments, we use 16 nodes and incur costs of 3.7K and 7.3K GPU hours respectively.

A set of experiments that we ran outside of these standard training configurations is the alternative cooldowns in Table 5 (top). For these, we still use the same number of concurrent GPUs as for 7B-1x and 7B-2x but the total cost now is 5.5K GPU hours due to the token count falling in between those scales. We also change the starting learning rate to resume from where it left off in the original DCLM-7B model's training run (close to 1e-3).

Another set of experiments where training details differ are for our table experiments which make use of the Dataset Decomposition (Pouransari et al., 2024) technique to train longer-context models. We train for the same number of tokens as indicated by the scale name. However, the training runs incur an extra overhead of about 40%. 1B-1x models are trained for 4 length curriculum cycles while 7B-1x models are trained for 8. Both otherwise retain hyperparameters from the standard runs. For trafilatura and jusText, we find it can help performance to use an intermediate checkpoint (saved after each cycle) but this does not meaningfully close the gap to resiliparse.

G Evaluation details

Given that we are evaluating pretrained language models, our test sets are separately curated English evaluations (as opposed to splits of our training data). Specifically, we use:

- DCLM's (Li et al., 2024) CORE_V2 evlauations include 22 different base model evaluations with varying test set sizes (please refer to Li et al. (2024) for more details).
- MMLU (Hendrycks et al., 2021) contains around 14K 4-way multiple choice questions covering 57 different subject categories
- For WikiTQ (Pasupat and Liang, 2016), we modify the implementation from Ashury-Tahan et al. (2025b) and evaluate each method on 100 instances $\times$ 7 serializations.
- For HumanEval (Chen et al., 2021), we use the implementation from Ben Allal et al. (2022) and evaluate each method on 164 examples $\times$ 5 languages.

[10] https://pypi.org/project/beautifulsoup4/