CRAW4LLM: Efficient Web Crawling for LLM Pretraining

Anonymous ACL submission

Abstract

Web crawl is a main source of large language models' (LLMs) pretraining data, but the majority of crawled web pages are discarded in pretraining due to low data quality. This paper presents CRAW4LLM, an efficient web crawling method that explores the web graph based on the preference of LLM pretraining. Specifically, it leverages the influence of a webpage in LLM pretraining as the priority score of the web crawler's scheduler, replacing the standard graph-connectivity-based priority. Our experiments on a web graph containing 900 million webpages from a commercial search engine's index demonstrate the efficiency of CRAW4LLM in obtaining high-quality pretraining data. With just 21% URLs crawled, LLMs pretrained on CRAW4LLM data reach the same downstream performances of previous crawls, significantly reducing the crawling waste and alleviating the burdens on websites. We will make our code publicly available.

1 Introduction

Massive in size and diverse in topics, web data usually serve as the primary source of pretraining data for large language models (LLMs), providing an extensive and heterogeneous corpus that captures a wide spectrum of human knowledge and real-world information (Baack, 2024; Dubey et al., 2024; Penedo et al., 2024). Pretraining datasets are typically built from large-scale web crawls such as Common Crawl (CommonCrawl, 2007), which may contain TBs of data spanning billions of webpages (Penedo et al., 2024; Weber et al., 2024).

Despite their vast scale, most of the data collected from web crawls are not used in the pretraining of LLMs. Existing work often discards over 90% of the raw data collected from the web (Li et al., 2024; Penedo et al., 2024; Tang et al., 2024), highlighting the *inefficiency* of current web crawlers in collecting LLM pretraining data. Common web crawlers like Common Crawl prioritize pages based on graph connectivity metrics like PageRank (Page et al., 1999; Cho et al., 1998) or harmonic centrality (Boldi and Vigna, 2014; Baack, 2024), which favor documents with a high number of inlinks (indegree) (Fortunato et al., 2008) rather than those most relevant for pretraining. This misalignment not only leads to waste in computational resources during excessive data processing for LLM developers, but also incentivizes overcrawling, which burdens website operators with redundant traffic and increases ethical and legal risks related to fair use of data and copyright (Longpre et al., 2024; New York Times, 2023).

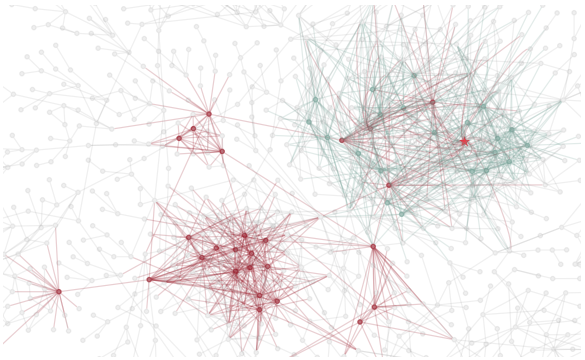
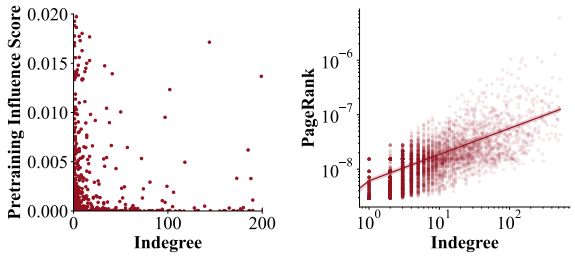


Figure 1: Graph traverse process of a traditional graph-connectivity-based crawler (green) and CRAW4LLM (red) starting from a same seed URL (star).

To bridge this gap, we propose Web **Craw**ling for **LLM** Pretraining (CRAW4LLM). Instead of relying on traditional graph-connectivity-based signals, CRAW4LLM improves crawling efficiency by prioritizing webpages based on their influence on LLM pretraining. Specifically, during each crawling iteration, all newly discovered documents are scored with a pretraining influence scorer derived from data-filtering pipelines for pretraining (Li et al., 2024; Penedo et al., 2024), and documents with the highest scores are used to discover new documents. By prioritizing webpages with high influence scores, as illustrated in Figure 1,



(a) Pretraining (-0.11). (b) PageRank (0.88).

Figure 2: Correlations between pretraining influence scores from DCLM fastText (Li et al., 2024) and PageRank to indegrees, on randomly sampled ClueWeb22-B documents (Overwijk et al., 2022). Spearman correlation coefficients are reported in parentheses.

CRAW4LLM explores the web graph in a fundamentally different manner from traditional graph-connectivity-based crawlers, uncovering a distinct subset of the web more useful for pretraining.

We conduct large-scale crawling simulations on ClueWeb22-A (Overwijk et al., 2022), a snapshot of the web containing 900 million English webpages obtained from the central index of a commercial search engine. Results show that, by crawling only 1× of the pretraining dataset size, CRAW4LLM can outperform traditional crawlers which collect 1×, 2×, and 4× more data followed by data selection. Compared to the baseline crawler that achieves the same performance, CRAW4LLM crawls only 21% of the webpages. Further analysis reveals that during crawling, CRAW4LLM quickly discovers documents that align with the oracle selection, which crawls the full web graph. As a result, it achieves 95% of the oracle performance while crawling only 2.2% of the data.

2 Methodology

In this section, we introduce Web Data **Craw**ling for **LLM** Pretraining (CRAW4LLM), an efficient crawling method that integrates LLM pretraining preference into the crawler. The algorithm of CRAW4LLM is presented in Algorithm 1.

Similar to traditional crawlers (Cho et al., 1998), CRAW4LLM starts with a set of seed URLs. For each unvisited outlink of them, CRAW4LLM assigns a score using a pretraining-oriented URL scoring function $\textsc{Score\_URL}(\cdot; \mathcal{M})$, where $\mathcal{M}$ is a pretraining influence scorer which rates a document's influence for pretraining. $\mathcal{M}$ can be derived from data classification models for pretraining data, which have been used to decide whether a document should be retained in or filtered out from the raw dataset (Li et al., 2024; Penedo et al., 2024). Formally, given a pretraining influence scorer $\mathcal{M}$, the score s of a URL u is calculated as

$$s \leftarrow \textsc{Score\_URL}(u; \mathcal{M}) = \mathcal{M}(\textsc{FetchPage}(u)), \quad (1)$$

where $\textsc{FetchPage}(u)$ gets the page content of u and $\mathcal{M}(\cdot)$ returns the score. Once all outlinks have been scored, following the standard procedures of existing crawlers, they are inserted into a priority queue, which automatically orders them based on their scores. The top n highest-scoring URLs are then dequeued for pretraining and serve as the sources for the next round of crawling. This process repeats until N documents have been collected, forming the final pretraining dataset $\mathcal{P}$.

In contrast, traditional crawlers typically rely on graph connectivity metrics, such as PageRank (Cho et al., 1998) and harmonic centrality (Baack, 2024), which basically assign higher priority to pages with higher indegrees (Fortunato et al., 2008). As shown in Figure 2(a), the indegrees of webpages exhibit a poor correlation with the scores assigned by the DCLM fastText classifier, a pretraining influence scorer for identifying high-quality pretraining data (Li et al., 2024). This confirms that graph connectivity-based crawlers are inefficient in crawling pretraining data.

By incorporating a pretraining influence scorer, CRAW4LLM traverses the web graph in a way that prioritizes high-quality pretraining documents. This makes the crawling more efficient and enables the discovery of documents dramatically different with connectivity-based crawlers.

Algorithm 1 CRAW4LLM Algorithm

Input: Seed URLs $\mathcal{U}_{\text{seed}}$, number of pages to be crawled N, number of pages to be crawled in each iteration n, pretraining influence scorer $\mathcal{M}(\cdot)$
Output: Crawled page set $\mathcal{P}$

Initialize **URL and score *priority* queue** $\mathcal{Q} \leftarrow \emptyset$
Initialize **crawled page set** $\mathcal{P} \leftarrow \emptyset$
Initialize **visited URL set** $\mathcal{V} \leftarrow \mathcal{U}_{\text{seed}}$
$\mathcal{U}_c \leftarrow \mathcal{U}_{\text{seed}}$
while $|\mathcal{P}| \leq N$ **do**
  $\mathcal{P}_c \leftarrow \textsc{FetchPages}(\mathcal{U}_c)$
  Merge $\mathcal{P}_c$ into $\mathcal{P}$
  $\mathcal{U}_{\text{out}} \leftarrow \textsc{ExtractURLs}(\mathcal{P}_c)$
  for all $v \in \mathcal{U}_{\text{out}}$ **do**
    if $v \notin \mathcal{V}$ **then**
      $\textsc{Enqueue}(\mathcal{Q}, v, \textsc{Score\_URL}(v; \mathcal{M}))$
      $\textsc{Add}(\mathcal{V}, v)$
    end if
  end for
  $\mathcal{U}_c \leftarrow \textsc{Dequeue}(\mathcal{Q}, n)$
end while
return $\mathcal{P}$

Crawling Method	Selection Pool Size	Commonsense Reasoning *(4 tasks)*	Language Understanding *(6 tasks)*	Reading Comprehension *(3 tasks)*	Symbolic Problem Solving *(5 tasks)*	World Knowledge *(5 tasks)*	Core *(23 tasks)*	% of Oracle
Oracle Selection (Upper Bound): Random sample from the top 10% rated data from ClueWeb22 using DCLM fastText for pretraining								
n.a.	45×	0.2438	0.2209	0.1483	0.2039	0.2403	0.2239	100%
Crawl-then-Select: Crawl 1× and 2× more data from ClueWeb22 and select top-rated 1× data using DCLM fastText for pretraining								
Random	1×	0.1906	0.1890	0.0244	0.1834	0.1930	0.1748	78.1%
	2×	0.1896	0.1967	**0.1260**	**0.2000**	0.2024	0.1964	87.7%
Indegree	1×	0.1730	0.1680	0.0326	0.1616	0.1668	0.1556	69.5%
	2×	0.1845	0.1856	0.0970	0.1958	0.1953	0.1865	83.3%
Ours: Crawl 1× data using CRAW4LLM for pretraining								
CRAW4LLM	1×	**0.2116**	**0.2311**	0.0826	0.1979	**0.2486**	**0.2133**	**95.3%**

Table 1: Downstream LLM performance. All models are pretrained on 1× data, which corresponds to 20M documents and 32.9B tokens. The evaluation metric is centered accuracy (0 = random guess) (Li et al., 2024). Best/2nd best in the last two groups are bolded/underlined. See Appendix C for detailed results.

3 Experimental Methodology

In this section, we introduce our experimental setup, with details on the crawler implementation and LLM training provided in Appendix A and B.

CRAW4LLM. To run experiments in our limited computational budget, we run a simulation of CRAW4LLM on the ClueWeb22 dataset (Overwijk et al., 2022), a snapshot of the web with graph information from a commercial crawler. We use the English subset of ClueWeb22-A, which is a web graph containing 900M webpages with links. We randomly sampled 10K URLs as our seed URLs. We set the number of total crawled documents N to 20M and crawled documents each iteration n to 10K. We use the DCLM fastText classifier (Li et al., 2024) as the pretraining influence scorer $\mathcal{M}(\cdot)$.

Baselines. We emulate traditional graph-connectivity-based crawlers by replacing the LLM-oriented URL scoring function (Eq. 1) with a function that returns the indegree for a given URL, since a node's indegree closely correlates with PageRank, a common graph connectivity metric, as shown in Figure 2(b) and previous findings (Fortunato et al., 2008). We also introduce a random crawling baseline, where the scorer assigns random scores. We run both of them in a *crawl-then-select* setting, first crawling 1× or 2× more documents and then selecting the top 1× (20M) documents based on scores assigned by the DCLM fastText classifier. This process mimics existing data-filtering pipelines, which begin with crawled documents and then apply filtering (Li et al., 2024; Penedo et al., 2024).

Oracle. We also introduce an oracle selection run in which we directly apply the DCLM fastText classifier to the entire ClueWeb22-A document set and select the top 10% documents for pretraining, serving as the upper bound.

LLM Training and Evaluation. For all runs, we use the final set of 20M crawled or selected documents to pretrain a 411M Transformer on 4× Chinchilla-optimal tokens (Hoffmann et al., 2022), totaling 32.9B tokens. The pretraining is conducted using the DCLM codebase (Li et al., 2024). To evaluate the pretrained LLMs, we follow the DCLM evaluation recipe, assessing performance on 23 (22 unique) *core* tasks.

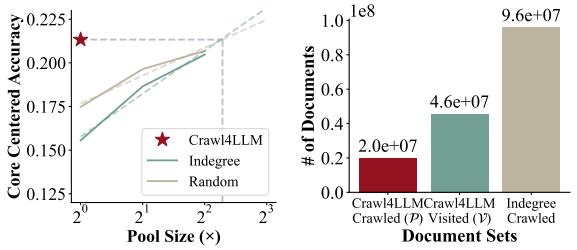
4 Evaluation Results

In this section, we first present the overall performance of CRAW4LLM (Sec. 4.1), followed by further analysis (Sec. 4.2).

4.1 Overall Performance

In this experiment, we compare the performance of CRAW4LLM with baseline crawlers by evaluating LLMs trained on their respective crawled data. As shown in Table 1, when all methods crawl the same amount of training data (1×), CRAW4LLM significantly outperforms random crawling and indegree crawling. In the crawl-then-select setting, where traditional crawlers are allowed to collect twice as much data (2×) for later selection, they still underperform compared to CRAW4LLM. This suggests that incorporating pretraining-oriented signals early in the crawling process is more beneficial than relying on post-selection. With only 1× of the data, CRAW4LLM retains 95% of the performance achieved by the oracle run, which directly selects from a substantially larger 45× data pool.

In Section 4.2, we further analyze the efficiency of CRAW4LLM compared to traditional crawlers and explore the reasons behind it.



(a) Extended crawling. (b) Visited documents.

Figure 3: Efficiency of crawlers. (a) shows the performance of LLMs trained on selected data crawled by CRAW4LLM and extended baseline crawlers. (b) presents the number of crawled ($\mathcal{P}$) and visited ($\mathcal{V}$) documents for CRAW4LLM, along with the estimated number of crawled documents required for indegree-based crawler to match CRAW4LLM's performance.

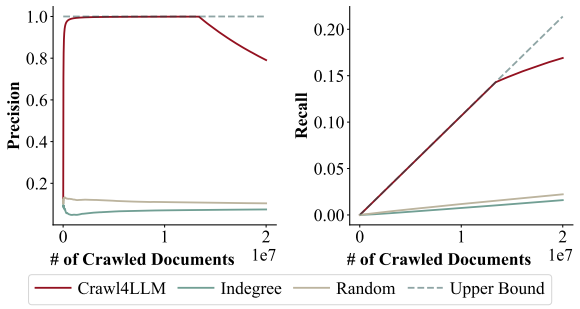
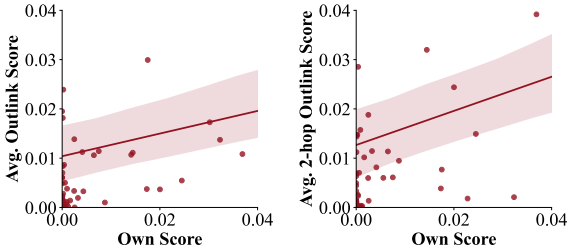


Figure 4: Precision (left) and recall (right) of the oracle documents among the documents crawled by CRAW4LLM, indegree, and random crawler. The upper bound represents always crawling the oracle documents.

4.2 Analysis

Crawling Efficiency. We evaluate the efficiency of CRAW4LLM by comparing the number of documents it crawls or visits against baseline crawlers. As shown in Figure 3(a), even when the baselines crawl 4× the required pretraining data for selection, they still underperform compared to CRAW4LLM. Extrapolation suggests that the indegree-based crawler would need to process 4.8× more documents (96M) to match CRAW4LLM's performance. Figure 3(b) further illustrates that CRAW4LLM achieves the same performance while crawling only 21% of the documents required by the indegree-based crawler, or 48% when considering all visited documents. These results highlight the efficiency of CRAW4LLM, demonstrating its potential to reduce website burdens and mitigate over-crawling.

Document Coverage. In this experiment, we plot the precision and recall of the oracle-selected documents among those crawled by CRAW4LLM and baseline crawlers throughout the crawling process. As shown in Figure 4, the precision quickly reaches 1.0, while the recall increases linearly, aligning with the theoretical upper bound. The saturated performance remains until 13 million documents have been crawled, after which the performance starts to decline, likely due to the lack of connectivity of the ClueWeb22 subgraph. In contrast, baseline crawlers exhibit minimal overlap with oracle-selected data, verifying that most of their crawled content is misaligned with pretraining needs and should be filtered (Li et al., 2024; Penedo et al., 2024). These results emphasize the importance of targeted crawling strategies for pretraining.



(a) 1-hop (0.61). (b) 2-hop (0.60).

Figure 5: Correlations between the pretraining influence scores of the documents themselves and the average scores of their 1- and 2-hop outlink documents. Spearman correlation coefficients are reported in parentheses.

Score Correlations Across Links. CRAW4LLM tracks the outlinks of the highest-scored documents in the current iteration to enrich the queue for future crawls. As shown in Figure 5, we plot the correlations between the pretraining influence scores of current documents and their 1- and 2-hop outlinks. The results indicate a correlation in influence scores across link hops, suggesting that highly-rated documents are interconnected and can be discovered through previously crawled documents.

5 Conclusion

This paper presents CRAW4LLM, a step toward more efficient and responsible web crawling for LLM pretraining. By prioritizing documents based on the pretraining needs, our method improves crawling efficiency and reduces unnecessary crawling, easing the burden on web hosts. While fair use of web data remains a critical challenge, we hope that CRAW4LLM can help mitigate these concerns and promote more compliant and sustainable practices in obtaining pretraining data for LLMs.

Limitations

Web crawling raises important concerns regarding copyright and the fair use of web data (Longpre et al., 2024), necessitating a better solution from the entire LLM community, such as sharing benefits with website owners. In this paper, we propose a more efficient crawling method that mitigates these challenges by reducing crawling, though it does not fully resolve them. Our experiments are conducted on a web graph dataset ClueWeb22 (Overwijk et al., 2022), thereby avoiding issues associated with actual web crawling. We hope that future advancements in web crawling will better align with ethical and legal standards.

While our crawling simulation is a sufficient research setup, further validation is required to assess the effectiveness of CRAW4LLM in real-world crawling scenarios. Our CRAW4LLM and baseline crawlers implement only the selection policy (Cho et al., 1998) of a crawler, which determines which pages to crawl. Although we try to mimic real-world crawling procedures used in systems like Apache Nutch[1], we do not implement other web crawling policies in industrial-level crawlers, such as the re-visit policy (Cho and Garcia-Molina, 2003a), politeness policy (Cho and Garcia-Molina, 2003b), and parallelization policy (Cho and Garcia-Molina, 2002). We leave the integration of CRAW4LLM into real-world crawling engines like Nutch and a comprehensive comparison between CRAW4LLM and traditional crawling methods in real-world crawling scenarios for future work.

References

Stefan Baack. 2024. A critical analysis of the largest source for generative AI training data: Common crawl. In *FAccT*, pages 2199–2208. ACM.

Paolo Boldi and Sebastiano Vigna. 2014. Axioms for centrality. *Internet Math.*, 10(3-4):222–262.

Junghoo Cho and Hector Garcia-Molina. 2002. Parallel crawlers. In *WWW*.

Junghoo Cho and Hector Garcia-Molina. 2003a. Effective page refresh policies for web crawlers. *ACM Trans. Database Syst.*, 28(4):390–426.

Junghoo Cho and Hector Garcia-Molina. 2003b. Estimating frequency of change. *ACM Trans. Internet Techn.*, 3(3):256–290.

Junghoo Cho, Hector Garcia-Molina, and Lawrence Page. 1998. Efficient crawling through URL ordering. *Comput. Networks*, 30(1-7):161–172.

CommonCrawl. 2007. Common crawl.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Santo Fortunato, Marián Boguñá, Alessandro Flammini, and Filippo Menczer. 2008. Approximating pagerank from in-degree. In *WAW*.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. 2022. Training compute-optimal large language models. In *NeurIPS*.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Yitzhak Gadre, Hritik Bansal, Etash Guha, Sedrick Scott 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 Abbas, Cheng-Yu Hsieh, Dhruba Ghosh, Josh Gardner, Maciej Kilian, Hanlin Zhang, Rulin Shao, Sarah M. Pratt, Sunny Sanyal, Gabriel Ilharco, Giannis Daras, Kalyani Marathe, Aaron Gokaslan, Jieyu Zhang, Khyathi Raghavi 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 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. 2024. Datacomp-lm: In search of the next generation of training sets for language models. In *NeurIPS*.

Shayne Longpre, Robert Mahari, Ariel Lee, Campbell Lund, Hamidah Oderinwale, William Brannon, Nayan Saxena, Naana Obeng-Marnu, Tobin South, Cole Hunter, Kevin Klyman, Christopher Klamm, Hailey Schoelkopf, Nikhil Singh, Manuel Cherep, Ahmad Anis, An Dinh, Caroline Shamiso Chitongo, Da Yin, Damien Sileo, Deividas Mataciunas, Diganta Misra, Emad A. Alghamdi, Enrico Shippole, Jianguo Zhang, Joanna Materzynska, Kun Qian, Kushagra Tiwary, Lester James V. Miranda, Manan Dey, Minnie Liang, Mohammed Hamdy, Niklas Muennighoff, Seonghyeon Ye, Seungone Kim, Shrestha Mohanty, Vipul Gupta, Vivek Sharma, Minh Chien Vu, Xuhui Zhou, Yizhi Li, Caiming Xiong, Luis Villa, Stella Biderman, Hanlin Li, Daphne Ippolito, Sara Hooker, Jad Kabbara, and Alex Pentland. 2024. Consent in crisis: The rapid decline of the AI data commons. In *NeurIPS*.

New York Times. 2023. Complaint, *the new york times company v. microsoft corporation, openai, inc., openai lp, openai gp, llc, openai llc, openai opco llc,*

[1] https://nutch.apache.org/

openai global llc, oai corporation, llc, and openai holdings, llc. Case 1:23-cv-11195, United States District Court, Southern District of New York.

Arnold Overwijk, Chenyan Xiong, Xiao Liu, Cameron VandenBerg, and Jamie Callan. 2022. Clueweb22: 10 billion web documents with visual and semantic information. *arXiv preprint arXiv:2211.15848*.

Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. 1999. The pagerank citation ranking : Bringing order to the web. In *The Web Conference*.

Guilherme Penedo, Hynek Kydlícek, Loubna Ben Allal, Anton Lozhkov, Margaret Mitchell, Colin A. Raffel, Leandro von Werra, and Thomas Wolf. 2024. The fineweb datasets: Decanting the web for the finest text data at scale. In *NeurIPS*.

Liping Tang, Nikhil Ranjan, Omkar Pangarkar, Xuezhi Liang, Zhen Wang, Li An, Bhaskar Rao, Linghao Jin, Huijuan Wang, Zhoujun Cheng, Suqi Sun, Cun Mu, Victor Miller, Xuezhe Ma, Yue Peng, Zhengzhong Liu, and Eric P. Xing. 2024. Txt360: A top-quality llm pre-training dataset requires the perfect blend.

Maurice Weber, Daniel Y. Fu, Quentin Anthony, Yonatan Oren, Shane Adams, Anton Alexandrov, Xiaozhong Lyu, Huu Nguyen, Xiaozhe Yao, Virginia Adams, Ben Athiwaratkun, Rahul Chalamala, Kezhen Chen, Max Ryabinin, Tri Dao, Percy Liang, Christopher Ré, Irina Rish, and Ce Zhang. 2024. Redpajama: an open dataset for training large language models. In *NeurIPS*.

A Details on Crawling

Our implementation of the indegree-based crawler employs a static URL scoring function, which directly returns the indegree of a given URL based on the full ClueWeb22 graph (Sec. 3). For real-world crawlers, as the true indegree value of a URL cannot be known in advance, a local graph must be maintained to track the *known* inlinks of discovered URLs. This local graph is updated iteratively as the discovered portion of the web expands during the crawling process (Cho et al., 1998).

Maintaining such a local graph during crawling introduces significant computational overhead. For simplicity, we instead implement the static simulation, where we directly return the global indegree of each URL. We believe that this simplified implementation does not underperform compared to real-world implementations, as our approach leverages global information from the entire graph, which should be better than the partial information from the local graph.

We run our simulated crawlers on a Linux server equipped with two Intel(R) Xeon(R) E5-2630 v3 CPUs (8 cores per socket, 16 cores in total, 1 thread per core), 125GiB of memory, and an SSD. A crawl of 20 million documents takes approximately one day to complete.

Hyper-parameter	Value
n_{layers}	24
n_{heads}	8
d_{model}	1,024
d_{head}	128
Warmup	2,000
Learning Rate	3e-3
Weight Decay	0.033
z-loss	1e-4
Global Batch Size	512
Sequence Length	2048

Table 2: Model and training hyper-parameters. n_{layers}, n_{layers}, d_{model}, and d_{head} denote the number of layers, attention heads, width, and width per attention head, respectively.

B Details on LLM Training and Evaluation

We pretrain a 411M-parameter[2] decoder-only Transformer model using the DCLM training recipe (Li et al., 2024)[3]. The hyper-parameters are presented in Tabel 2. To enhance training stability, we extend the original 411M-1x setting to 411M-4x, meaning the model is trained on 4 times the Chinchilla-optimal number of tokens (Hoffmann et al., 2022), which amounts to 32.9B tokens. The training process takes 1 day and 12 hours on 8 NVIDIA L40S GPUs. For further details, please refer to the DCLM paper (Li et al., 2024). Due to computational constraints, each pretraining experiment is conducted only once.

We use the DCLM evaluation recipe (Li et al., 2024) to evaluate model performance on 23 (22 unique) *core* tasks.

C Detailed Results

The raw (uncentered) accuracy of all evaluation tasks is presented in Table 3, 4, 5, 6, and 7. Please refer to Li et al. (2024) for more details on the evaluation tasks.

[2]Sometimes referred to as 400M in the DCLM paper (Li et al., 2024).

[3]https://github.com/mlfoundations/dclm

Crawling Method	Selection Pool Size	Commonsense Reasoning			
		CommonsenseQA	COPA	OpenBookQA	PIQA
Oracle Selection (Upper Bound)					
n.a.	45×	0.2850	0.7000	0.3300	0.6812
Crawl-then-Select					
Random	1×	0.2072	0.6700	0.2980	0.6746
Random	2×	0.2588	0.6200	0.3160	0.6785
Random	4×	0.2326	0.6400	0.3380	0.6757
Indegree	1×	0.3219	0.6000	0.2780	0.6513
Indegree	2×	0.1966	0.6600	0.3040	0.6752
Indegree	4×	0.2088	0.6400	0.3400	0.6817
Ours					
CRAW4LLM	1×	0.2277	0.6600	0.3300	0.6926

Table 3: Results for commonsense reasoning tasks.

Crawling Method	Selection Pool Size	Language Understanding					
		BIG-Bench Lang. Id.	HellaSwag (zero-shot)	HellaSwag	LAMBADA	Winograd	Winogrande
Oracle Selection (Upper Bound)							
n.a.	45×	0.2515	0.3856	0.3905	0.4432	0.6557	0.5130
Crawl-then-Select							
Random	1×	0.2490	0.3709	0.3716	0.3990	0.6044	0.5146
Random	2×	0.2468	0.3882	0.3925	0.4073	0.6007	0.5130
Random	4×	0.2521	0.4011	0.4019	0.4390	0.6154	0.5130
Indegree	1×	0.2566	0.3515	0.3519	0.3596	0.5971	0.5004
Indegree	2×	0.2547	0.3749	0.3771	0.3773	0.5861	0.5241
Indegree	4×	0.2562	0.3994	0.4008	0.4159	0.6190	0.5178
Ours							
CRAW4LLM	1×	0.2544	0.4035	0.4048	0.4196	0.6593	0.5288

Table 4: Results for language understanding tasks.

Crawling Method	Selection Pool Size	Reading Comprehension		
		BoolQ	CoQA	SQuAD
Oracle Selection (Upper Bound)				
n.a.	45×	0.5755	0.2479	0.3139
Crawl-then-Select				
Random	1×	0.5080	0.1799	0.1882
Random	2×	0.5807	0.2053	0.2759
Random	4×	0.5911	0.2361	0.2951
Indegree	1×	0.5324	0.1666	0.1616
Indegree	2×	0.5697	0.1843	0.2390
Indegree	4×	0.5765	0.2147	0.2736
Ours				
CRAW4LLM	1×	0.5440	0.2264	0.2215

Table 5: Results for reading comprehension tasks.

Crawling Method	Selection Pool Size	Symbolic Problem Solving				
		AGI Eval LSAT-AR	BIG-Bench CS Algorithms	BIG-Bench Dyck Lang.	BIG-Bench Operators	BIG-Bench Repeat Copy Logic
Oracle Selection (Upper Bound)						
n.a.	45×	0.2739	0.4341	0.2160	0.2143	0.0625
Crawl-then-Select						
Random	1×	0.2391	0.4568	0.1970	0.2143	0.0000
Random	2×	0.2696	0.4538	0.2520	0.1762	0.0313
Random	4×	0.1957	0.4568	0.2600	0.1857	0.0625
Indegree	1×	0.2304	0.4371	0.1900	0.1429	0.0000
Indegree	2×	0.2609	0.4235	0.2340	0.2143	0.0313
Indegree	4×	0.2174	0.4538	0.2530	0.1667	0.0938
Ours						
CRAW4LLM	1×	0.2696	0.4371	0.1620	0.2095	0.0938

Table 6: Results for symbolic problem solving tasks.

Crawling Method	Selection Pool Size	World Knowledge				
		ARC Easy	ARC Challenge	BIG-Bench-Bench QA Wikidata	Jeopardy	MMLU
Oracle Selection (Upper Bound)						
n.a.	45×	0.5951	0.3166	0.4945	0.1176	0.2805
Crawl-then-Select						
Random	1×	0.5152	0.2799	0.5186	0.0461	0.2552
Random	2×	0.5425	0.2807	0.5081	0.0648	0.2561
Random	4×	0.5577	0.2867	0.5126	0.0970	0.2543
Indegree	1×	0.4857	0.2509	0.4888	0.0138	0.2618
Indegree	2×	0.5248	0.2790	0.5205	0.0555	0.2464
Indegree	4×	0.5749	0.2935	0.5084	0.0959	0.2430
Ours						
CRAW4LLM	1×	0.6103	0.3208	0.5143	0.1323	0.2661

Table 7: Results for world knowledge tasks.

D The ClueWeb22 Dataset

ClueWeb22 (Overwijk et al., 2022) is distributed under a "TREC-style" license for research purpose. The dataset can be obtained by signing a data license agreement with Carnegie Mellon University[4]. We use ClueWeb22 only for research purpose.

E Use of AI Assistants

We use GitHub Copilot[5] to assist with coding and ChatGPT[6] (powered by GPT-4o) to enhance the writing of this paper.

[4] https://lemurproject.org/clueweb22/obtain.php
[5] https://github.com/features/copilot
[6] https://chatgpt.com/