Cleaner Pretraining Corpus Curation with Neural Web Scraping

Zhipeng Xu[1], **Zhenghao Liu**[1*], **Yukun Yan**[2], **Zhiyuan Liu**[2], **Ge Yu**[1] **and Chenyan Xiong**[3]
[1]Department of Computer Science and Technology, Northeastern University, China
[2]Department of Computer Science and Technology, Institute for AI, Tsinghua University, China
Beijing National Research Center for Information Science and Technology, China
[3]Language Technologies Institute, Carnegie Mellon University, United States

Abstract

The web contains large-scale, diverse, and abundant information to satisfy the information-seeking needs of humans. Through meticulous data collection, preprocessing, and curation, webpages can be used as a fundamental data resource for language model pretraining. However, when confronted with the progressively revolutionized and intricate nature of webpages, rule-based/feature-based web scrapers are becoming increasingly inadequate. This paper presents a simple, fast, and effective **Neu**ral web **Scraper** (`NeuScraper`) to help extract primary and clean text contents from webpages. Experimental results show that `NeuScraper` surpasses the baseline scrapers by achieving more than a 20% improvement, demonstrating its potential in extracting higher-quality data to facilitate the language model pretraining. All of the code is available at `https://github.com/OpenMatch/NeuScraper`.

1 Introduction

Large Language Models (LLMs) have shown impressive performance in various NLP tasks as the size of models scaling up (Chowdhery et al., 2023; Touvron et al., 2023; Achiam et al., 2023; Zhao et al., 2023). However, recent findings in scaling laws indicate that both model size and training data should be scaled proportionally (Hoffmann et al., 2022), posing a significant challenge in acquiring sufficiently large pretraining datasets or even raising concerns about data scarcity (Penedo et al., 2024; Villalobos et al., 2022).

To curate more data for pretraining, researchers pay more attention to collecting more valuable data from the Web. The web-crawled datasets, such as CommonCrawl, have been widely used for pretraining, facilitating the development of language models (Wenzek et al., 2020; Radford et al., 2019; Raffel et al., 2020; Penedo et al., 2024). Nevertheless, prior research has demonstrated that, even after aggressive cleaning, the quality of pre-extracted text provided by CommonCrawl still fails to reach the expected (Raffel et al., 2020; Gao et al., 2021; Penedo et al., 2024). The reason lies in that advertisements, banners, hyperlinks, and other harmful content are usually mixed within the primary content of the page, thereby only extracting these primary contents brings lots of noise to pretraining (Gibson et al., 2005; Vogels et al., 2018).

Web scrapers provide opportunities to extract valuable content from the raw HTML pages (Barbaresi, 2021). However, rule-based and heuristic scrapers have notable limitations. On the one hand, web pages are becoming increasingly sophisticated, requiring more intricate underlying code to deal with the page layout (Butkiewicz et al., 2011). In this case, maintaining the scraper rules is time-consuming and requires much human effort. On the other hand, HTML functions as a markup language, enabling web designers to customize web pages according to individual preferences. Consequently, web pages frequently lack complete standardization, which may mislead the rule-based web scrapers (Hantke and Stock, 2022).

In this paper, we present a simple, fast, and effective Neural Web Scraper (`NeuScraper`) designed to extract primary content from webpages. `NeuScraper` employs a shallow neural architecture and integrates layout information for efficient scraping. Our experiments demonstrate that `NeuScraper` surpasses baseline scrapers, achieving a 20% improvement in performance and generating a higher-quality corpus for language model pretraining. Notably, `NeuScraper` shows the potential of high processing speeds when utilized on GPU. The easy-to-use and open-source tool, `NeuScraper`, can facilitate the creation of large-scale corpora for pretraining.

* indicates corresponding author.

Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pages 802–812

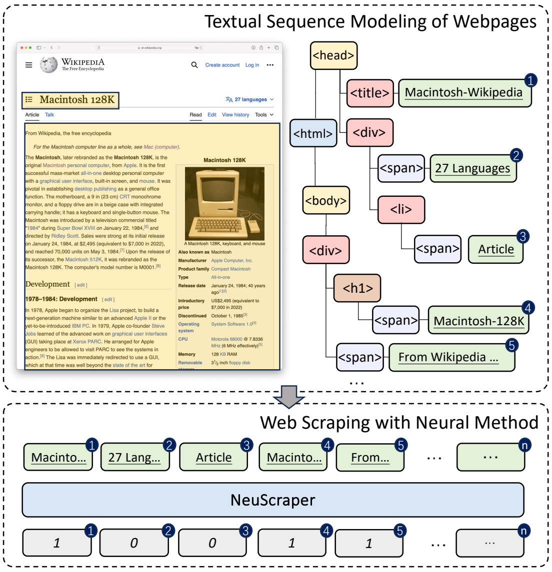


Figure 1: The Pipeline of Primary Content Extraction Using `NeuScraper` (Neural Web Scraper).

2 Related Work

Leveraging web scrapers for extraction provides a promising way to extract high-quality content from webpages. Such a web scraping task is usually defined as text extraction, boilerplate removal, template removal, or generic web extraction in different webpage processing pipelines (Finn et al., 2001; Rahman et al., 2001; Vieira et al., 2006), which is distinguished from the web information extraction task that extracts the entities from webpages (Li et al., 2022; Wang et al., 2022). The web scrapers can be divided into rule-based and feature-based methods.

Rule-based web scrapers start from web wrappers, which often need manual designs or a wrapper induction system for producing (Muslea et al., 1999; Crescenzi et al., 2001). The web wrappers usually need to be tailored for each webpage, which is not feasible to process large-scale webpages (Guo et al., 2010). A more conventional approach is to create a Document Object Model (DOM) tree, which assists in building a rule-based scraper (Gupta et al., 2003; Guo et al., 2010) or help the comparison of webpages (Yi et al., 2003). Additionally, the work also incorporates tag cumulative distributions (Finn et al., 2001), text density (Sun et al., 2011), and tag ratios (Weninger et al., 2010) to benefit the content extraction from webpages.

Except for these rule-based methods, some scrapers use feature-based approaches to better extract the primary contents from webpages. Specifically, they divide the webpage into several blocks using rules built based on the HTML tags or DOM tree. Then they extract dozens to hundreds of handcrafted features from these blocks, such as markup, text/document features (Spousta et al., 2008), linguistic, structural & visual features (Bauer et al., 2007) and DOM tree-based features (Vogels et al., 2018). These features can be fed into SVM (Bauer et al., 2007; Kohlschütter et al., 2010), conditional random fields (Spousta et al., 2008), logistic regressions (Peters and Lecocq, 2013) or convolutional neural network (Vogels et al., 2018) to classify whether the texts in the block are the primary content of the webpages.

3 Neural Web Scraper

This section introduces our Neural Web Scraper (`NeuScraper`) to extract primary contents from webpages. We first introduce the sequence modeling method of webpages (Sec. 3.1) and then describe our neural-based web scraper (Sec. 3.2).

3.1 Textual Sequence Modeling of Webpages

As shown in Figure 1, the primary content extraction task aims to extract the content from the highlighted areas, which consists of clean text and represents the main information of the webpage. To facilitate the web scraping with `NeuScraper`, we convert the HTML code into textual sequences.

Previous work (Bauer et al., 2007) has demonstrated the effectiveness of both structural and visual features in helping to identify primary contents. Thus, to preserve webpage layout information, we rely on the DOM tree structure to transform webpages into textual sequences. Specifically, we employ the BeautifulSoup4[1] toolkit to build the DOM tree for each webpage, conduct the depth-first traversal on the tree and regard the visited order as additional location information to represent the nodes. During this process, only the nodes that contain plant texts, table nodes (tagged with *<table>*), and list nodes (tagged with *<ol>*, *<ul>* or *<dl>*) are reserved to produce the final textual sequences $X = \{x_1, x_2, ..., x_n\}$, where n denotes the number of the reserved DOM nodes. After processing, the web scraping task primarily involves determining whether the node x_i contains the primary content of the webpage for evaluation.

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

3.2 Web Scraping with the Neural Method

In this subsection, we introduce our neural modeling method to build the web scraper. To process the textual sequences $X = \{x_1, x_2, ..., x_n\}$, we build a hierarchical architecture for node-level prediction.

Specifically, to guarantee the efficiency of `NeuScraper`, we use the first layer of the XLM-Roberta (Conneau et al., 2020) model to encode the text representation x_i of the i-th DOM node as the 768-dimensional node representation h_i:

$$h_i = \text{XLMRoberta-Layer}^1(x_i), \tag{1}$$

where h_i is the representation of the "[CLS]" token. Then we feed these node representations $H = \{h_1, h_2, ..., h_n\}$ into a 3-layer transformer model (Vaswani et al., 2017) with 8 attention heads to get the encoded node representations:

$$\hat{h}_i = \text{Transformer}(\text{Linear}(h_i)), \tag{2}$$

where the linear layer projects h_i to 256-dimensional embeddings for efficient modeling. Following previous work (Overwijk et al., 2022), the DOM nodes can be categorized into six kinds of labels y^k, including primary content, heading, title, paragraph, table, and list. Then we calculate the label prediction probability $P(y_i^k = 1|x_i)$ of the k-th category label y_i^k of the i-th node:

$$P(y_i^k = 1|x_i) = \text{Sigmoid}(\text{MLP}(\hat{h}_i)) \tag{3}$$

Finally, `NeuScraper` is trained using the loss L:

$$L = \sum_{k=1}^{6} \sum_{i=1}^{n} \text{CrossEntropy}(P(y_i^k|x_i), \mathcal{Y}_i^k), \tag{4}$$

where $\mathcal{Y}_i^k$ is the ground truth label. $\mathcal{Y}_i^k$ is a binary label and $\mathcal{Y}_i^k = 1$ indicates that the i-th DOM node belongs to the k-th label category. During inference, we only consider the primary content label to extract the texts from webpages.

4 Experimental Methodology

In this section, we describe the datasets, baselines, evaluation metrics and implementation details.

Dataset. We use ClueWeb22 (Overwijk et al., 2022) dataset in experiments. The content extraction labels of ClueWeb22 were generated from the production system of a commercial search engine. The labels are not available for general web scraping tools, because they are annotated with more expensive signals of page rendering and visualization. More details are shown in Appendix A.2.

Method	Evaluation Metrics				Latency
	Acc.	Prec.	Rec.	F1	(ms)
htmlparser	40.73	40.65	98.95	57.63	19.01
bs4	41.29	40.96	**99.94**	58.10	12.65
html2text	40.44	39.35	85.40	53.88	15.85
boilerpipe	66.48	66.79	35.27	46.16	11.05
jusText	62.58	72.62	13.08	22.17	10.91
lxml	64.62	61.48	35.22	44.78	10.96
inscriptis	45.35	42.48	96.43	58.98	14.99
readability	68.47	72.84	36.04	48.22	12.36
trafilatura	70.70	66.57	56.42	61.08	11.95
NeuScraper	**86.35**	**80.77**	87.29	**83.90**	11.39

Table 1: Overall Performance. We use ClueWeb22 to evaluate the content extraction effectiveness of different web scrapers. More details are shown in Appendix A.2.

Baseline. The scraping baselines consist of nine open-sourced web scrapers, including basic HTML manipulators (`html2text` and `inscriptis` (Weichselbraun, 2021)), generic webpage parsers (`beautifulsoup4`, `lxml` and `htmlparser`), rule-based scrapers (`jusText` and `readability`) and machine learning-based scraper (`boilerpipe` (Kohlschütter et al., 2010)). `trafilatura` (Barbaresi, 2021) is our main baseline, which combines different rules and heuristic methods.

Evaluation Metrics. The accuracy, precision, recall, and F1 score, are used to evaluate the effectiveness in extracting primary contents. Furthermore, we use different scrapers to produce the web corpus and pretrain language models. The quality of scraping can be demonstrated by the results of standard downstream tasks.

Implementation Details. `NeuScraper` is trained for 30 epochs using the AdamW optimizer with a batch size of 1024. Learning rate adjustments followed the cosine decay schedule, with a warm-up phase spanning the initial 5% of iterations and the peak rate fixed at 6e-4. To accommodate memory and computational speed limitations, the maximum length of node sequences was chunked to 384.

5 Evaluation Result

In this section, we first show the effectiveness of different scrapers in extracting primary contents from the raw webpages. Subsequently, we evaluate the quality of the extracted data and utilize it to pretrain language models of varying scales.

Size	Method	BLIMP	ARC-e	ARC-c	SWAG	WinoG	SciQ	Lambada	LogiQA	AVG
ClueWeb22										
160M	htmlparser	70.87	41.16	17.23	32.24	49.88	66.10	16.96	22.58	39.63
	trafilatura	73.46	42.46	18.25	34.08	48.61	**69.20**	**18.10**	**22.11**	40.78
	NeuScraper	**74.01**	**42.84**	**18.43**	**34.14**	**51.46**	69.00	17.58	21.50	**41.12**
410M	htmlparser	74.24	42.63	18.77	34.45	49.80	70.80	22.35	22.42	41.93
	trafilatura	**77.84**	45.28	**20.56**	**37.29**	**52.32**	72.90	23.77	21.96	43.99
	NeuScraper	76.71	**47.34**	20.47	37.00	50.74	**74.40**	**26.76**	**24.42**	**44.73**
CommonCrawl										
160M	htmlparser	58.38	29.71	**18.77**	28.85	**50.27**	38.60	5.16	19.66	31.17
	trafilatura	**69.72**	34.72	18.51	32.04	49.56	56.90	11.70	**23.96**	37.13
	NeuScraper	69.27	**36.15**	18.43	**32.61**	**51.77**	**60.50**	**15.48**	20.73	**38.12**
410M	htmlparser	61.30	28.28	17.23	29.36	50.35	41.00	6.50	20.73	31.84
	trafilatura	72.66	36.74	**20.13**	33.91	**51.30**	55.40	16.08	21.35	38.44
	NeuScraper	**74.42**	**39.30**	18.60	**34.77**	50.03	**61.40**	**20.66**	**21.81**	**40.12**

Table 2: Effectiveness of Pythia Pretraining Using the Extracted Data from Different Scrapers. We pretrained Pythia models of different sizes on ClueWeb22 and CommonCrawl respectively. More details are shown in Appendix A.3.

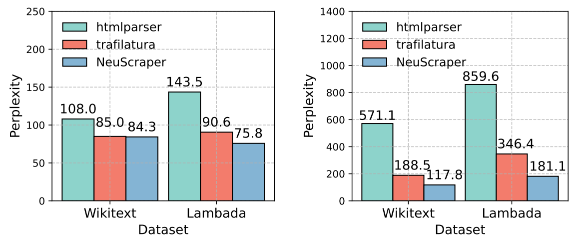
5.1 Overall Performance

The effectiveness of baseline scrapers and our `NeuScraper` in extracting primary contents from the raw webpages is shown in Table 1. Among all baseline scrapers, the `trafilatura` exhibits the highest performance, showcasing its effectiveness in content extraction through its cascade of rule-based filters and content heuristic methods. Our `NeuScraper` surpasses all traditional web scrapers and achieves over a 20% improvement. It illustrates the effectiveness of our `NeuScraper` in learning the schemes of the primary contents, generalizing its advances to handle various layouts of pages and extracting high-quality texts from them. Notably, with the GPU support and distributed computation, `NeuScraper` achieves competitive scraping latency.

5.2 Effectiveness of the Cleaned Web Data in Language Model Pretraining

This part evaluates the effectiveness of language models pretrained on the web data.

As shown in Table 2, we utilize different scrapers to handle the webpages sourced from ClueWeb22 and CommonCrawl, and leverage the extracted data to pretrain Pythia models (Biderman et al., 2023). The evaluation results demonstrate that employing the `NeuScraper` for webpage processing enhances the performance of language models in downstream tasks. It is noteworthy that the `NeuScraper` represents a data-driven scraping approach, circumventing the need for building sophisticated rules and conducting intricate feature engineering to deal with the continuously evolving HTML layouts.



(a) ClueWeb22. (b) CommonCrawl.

Figure 2: The Effectiveness of Language Models Trained on Web Data to Reproduce the Target Corpora. Lower perplexity indicates more proficiency in language models for reproducing.

5.3 Evaluation on the Quality of Extracted Data Using NeuScraper

In this subsection, we aim to estimate the quality of extracted data using `NeuScraper`. The evaluation results are shown in Figure 2.

It is apparent that if two corpora are of comparable quality, their n-gram distributions should exhibit similarity. Thus, we use the language models pretrained on web data (the same as Sec. 5.2) to ask these language models to reproduce the target corpora, such as Wikitext (Merity et al., 2017) and Lambada (Radford et al., 2019). The perplexity is used to evaluate the effectiveness of the language models pretrained on web data in replicating the target corpora. The lower perplexity indicates the language model is more proficient to the target corpora, showing the pretrained data and target data have more overlaps and are more similar.

The evaluation results reveal that the utilization

Method	Evaluation Metrics				Latency
	Acc.	Prec.	Rec.	F1	(ms)
CPU	86.35	80.77	87.29	83.90	55.25
+ `qint8`	86.37	80.70	87.48	83.95	42.22
+ `quint8`	86.39	80.68	87.56	83.98	41.48
GPU	86.35	80.77	87.29	83.90	11.39

Table 3: Quantization Performance of `NeuScraper` on ClueWeb22. We further quantized `NeuScraper` to accelerate its inference on the CPU.

of extracted content from some simple scrapers, such as `htmlparser`, significantly impacts the effectiveness of language models, which causes an increase of more than 20 points in perplexity due to the noise derived from webpages. Compared with the `trafilatura`, `NeuScraper` decreases the perplexity by over ten points, showing its capability to yield higher-quality data for pretraining through learning to extract primary content.

5.4 Model Quantization for NeuScraper

In this subsection, we quantize the model of `NeuScraper` via `onnxruntime`[2] to evaluate its efficiency in resource-constrained scenarios.

As shown in Table 3, we utilize `qint8` and `quint8` to quantize our `NeuScraper`. The `qint8` quantizes model parameters or layer outputs to signed 8-bit integers, while `quint8` quantizes them to unsigned 8-bit integers, reducing model size and improving computational efficiency. Benefiting from quantization, `NeuScraper` accelerates by 25% with no loss of performance compared to the original model. While processing is still 4-5x slower compared to GPUs, it also provides a potential way to scrap in low-resource scenarios via `NeuScraper`.

6 Conlusion

This paper proposes `NeuScraper`, which employs a shallow neural architecture to clean the webpages. The experimental results show the effectiveness of `NeuScraper`. The open-sourced and easy-used web scraper may facilitate the research on language model pretraining.

Limitation

To guarantee efficiency, `NeuScraper` needs the powerful parallelism of GPUs to achieve high-speed web scraping. In addition, for large-scale pretraining corpus processing, a high throughput storage medium is required to ensure inference efficiency due to the frequent data swapping between the storage medium and GPU.

[2] `https://onnxruntime.ai`

Acknowledgments

This work is partly supported by the Natural Science Foundation of China under Grant (No. 62206042, No. 62137001, and No. 62272093), the Joint Funds of Natural Science Foundation of Liaoning Province (No. 2023-MSBA-081), and the Fundamental Research Funds for the Central Universities under Grant (No. N2416012).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *ArXiv preprint*, abs/2303.08774.

Adrien Barbaresi. 2021. Trafilatura: A web scraping library and command-line tool for text discovery and extraction. In *Proceedings of ACL*, pages 122–131, Online.

Daniel Bauer, Judith Degen, Xiaoye Deng, Priska Herger, Jan Gasthaus, Eugenie Giesbrecht, Lina Jansen, Christin Kalina, Thorben Kräger, Robert Märtin, et al. 2007. Fiasco: Filtering the internet by automatic subtree classification, osnabruck. In *Proceedings of WAC3*, pages 111–121.

Stella Biderman, Hailey Schoelkopf, Quentin Gregory Anthony, Herbie Bradley, Kyle O'Brien, Eric Hallahan, Mohammad Aflah Khan, Shivanshu Purohit, USVSN Sai Prashanth, Edward Raff, et al. 2023. Pythia: A suite for analyzing large language models across training and scaling. In *Proceedings of ICML*, pages 2397–2430.

Michael Butkiewicz, Harsha V Madhyastha, and Vyas Sekar. 2011. Understanding website complexity: measurements, metrics, and implications. In *Proceedings of IMC*, pages 313–328.

Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, et al. 2023. Palm: Scaling language modeling with pathways. *JMLR*, (240):1–113.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2020. Unsupervised cross-lingual representation learning at scale. In *Proceedings of ACL*, pages 8440–8451, Online.

Valter Crescenzi, Giansalvatore Mecca, Paolo Merialdo, et al. 2001. Roadrunner: Towards automatic data extraction from large web sites. In *Proceedings of VLDB*, pages 109–118.

Aidan Finn, Nicholas Kushmerick, and Barry Smyth. 2001. Fact or fiction: Content classification for digital libraries. In *Proceedings of DELOS*.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, et al. 2021. The pile: An 800gb dataset of diverse text for language modeling. *ArXiv preprint*, abs/2101.00027.

David Gibson, Kunal Punera, and Andrew Tomkins. 2005. The volume and evolution of web page templates. In *Proceedings of WWW*, pages 830–839.

Yan Guo, Huifeng Tang, Linhai Song, Yu Wang, and Guodong Ding. 2010. Econ: an approach to extract content from web news page. In *Proceedings of APWEB*, pages 314–320.

Suhit Gupta, Gail E. Kaiser, David Neistadt, and Peter Grimm. 2003. Dom-based content extraction of HTML documents. In *Proceedings of WWW*, pages 207–214.

Florian Hantke and Ben Stock. 2022. Html violations and where to find them: a longitudinal analysis of specification violations in html. In *Proceedings of IMC*, pages 358–373.

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. *ArXiv preprint*, abs/2203.15556.

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. 2010. Boilerplate detection using shallow text features. In *Proceedings of WSDM*, pages 441–450.

Junlong Li, Yiheng Xu, Lei Cui, and Furu Wei. 2022. MarkupLM: Pre-training of text and markup language for visually rich document understanding. In *Proceedings of ACL*, pages 6078–6087, Dublin, Ireland.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. 2017. Pointer sentinel mixture models. In *Proceedings of ICLR*.

Ion Muslea, Steve Minton, and Craig Knoblock. 1999. A hierarchical approach to wrapper induction. In *Proceedings of AGENTS*, pages 190–197.

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

Guilherme Penedo, Quentin Malartic, Daniel Hesslow, Ruxandra Cojocaru, Hamza Alobeidli, Alessandro Cappelli, Baptiste Pannier, Ebtesam Almazrouei, and Julien Launay. 2024. The refinedweb dataset for falcon llm: Outperforming curated corpora with web data only. In *Proceedings of NeurIPS*.

Matthew E Peters and Dan Lecocq. 2013. Content extraction using diverse feature sets. In *Proceedings of WWW*, pages 89–90.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. Language models are unsupervised multitask learners.

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. *JMLR*, 21:140:1–140:67.

AFR Rahman, H Alam, R Hartono, et al. 2001. Content extraction from html documents. In *Proceedings of WDA*, pages 1–4.

Miroslav Spousta, Michal Marek, and Pavel Pecina. 2008. Victor: the web-page cleaning tool. In *Proceedings of LREC*, pages 12–17.

Fei Sun, Dandan Song, and Lejian Liao. 2011. DOM based content extraction via text density. In *Proceedings of SIGIR*, pages 245–254.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *ArXiv preprint*, abs/2302.13971.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of NeurIPS*, pages 5998–6008.

Karane Vieira, Altigran S Da Silva, Nick Pinto, Edleno S De Moura, Joao MB Cavalcanti, and Juliana Freire. 2006. A fast and robust method for web page template detection and removal. In *Proceedings of CIKM*, pages 258–267.

Pablo Villalobos, Jaime Sevilla, Lennart Heim, Tamay Besiroglu, Marius Hobbhahn, and Anson Ho. 2022. Will we run out of data? an analysis of the limits of scaling datasets in machine learning. *ArXiv preprint*, abs/2211.04325.

Thijs Vogels, Octavian-Eugen Ganea, and Carsten Eickhoff. 2018. Web2text: Deep structured boilerplate removal. In *Proceedings of ECIR*, pages 167–179.

Qifan Wang, Yi Fang, Anirudh Ravula, Fuli Feng, Xiaojun Quan, and Dongfang Liu. 2022. Webformer: The web-page transformer for structure information extraction. In *Proceedings of WWW*, pages 3124–3133.

Alex Warstadt, Alicia Parrish, Haokun Liu, Anhad Mohananey, Wei Peng, Sheng-Fu Wang, and Samuel R. Bowman. 2020. BLiMP: The benchmark of linguistic minimal pairs for English. *TACL*, 8:377–392.

Albert Weichselbraun. 2021. Inscriptis–a python-based html to text conversion library optimized for knowledge extraction from the web. *ArXiv preprint*, abs/2108.01454.

Tim Weninger, William H. Hsu, and Jiawei Han. 2010. CETR: content extraction via tag ratios. In *Proceedings of WWW*, pages 971–980.

Guillaume Wenzek, Marie-Anne Lachaux, Alexis Conneau, Vishrav Chaudhary, Francisco Guzmán, Armand Joulin, and Edouard Grave. 2020. CCNet: Extracting high quality monolingual datasets from web crawl data. In *Proceedings of LREC*, pages 4003–4012, Marseille, France.

Lan Yi, Bing Liu, and Xiaoli Li. 2003. Eliminating noisy information in web pages for data mining. In *Proceedings of SIGKDD*, pages 296–305.

Rowan Zellers, Yonatan Bisk, Roy Schwartz, and Yejin Choi. 2018. SWAG: A large-scale adversarial dataset for grounded commonsense inference. In *Proceedings of EMNLP*, pages 93–104, Brussels, Belgium.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *ArXiv preprint*, abs/2303.18223.

A Appendix

A.1 License

The terms of use for ClueWeb22 can be found on the Lemur Project website[3], while CommonCrawl provides its terms of use on its official website[4]. All of these licenses and agreements allow their data for academic use.

A.2 More Experimental Details of Overall Evaluation

In this subsection, we describe further details of the implementation of overall evaluation.

Dataset. We randomly selected about 8.28 million webpages from ClueWeb22-B English subset as the training set. To evaluate content extraction performance, we utilized a snapshot extracted from ClueWeb22-B, identified as `en0001-01`. This particular snapshot comprises 19,013 English webpages along with respective annotations. Notably, it's imperative to highlight that `en0001-01` was excluded from both the training, and validation datasets.

Metrics. In our experiments, we convert the web scanning task into a binary classification problem, so we can compute relevant metrics at the node level. However, some previous web scrapers would directly return the primary content without node information. Therefore, we directly check whether the reserved plain text contains the text spans of DOM tree nodes, which are annotated as ground truths in the benchmark.

Computing Platform. We conducted the training of `NeuScraper` on a server equipped with 8× NVIDIA A100-40G GPUs, with the training process spanning approximately 40 hours. For the evaluation of baseline scrapers, we utilized a setup comprising 2× Intel Xeon Gold-6348@2.60GHz CPUs with multiprocessing. In contrast, the evaluation of `NeuScraper` was carried out using 8× NVIDIA A100-40 GB GPUs, employing an inference batch size of 256 per GPU.

A.3 More Experimental Details on Using Cleaned Web Data for Language Model Pretraining

In this subsection, we describe additional details of the evaluation of the effectiveness of the cleaned web data in language model pretraining.

Pretraining Corpus. We utilize ClueWeb22-B and CommonCrawl `CC-MAIN-2023-50` as the source corpus for our pretraining endeavors. For ClueWeb22 , we employ various scrapers to acquire the corpus while ensuring an equivalent number of tokens, thereby pretraining the language model to mirror the performance of each scraper. For CommonCrawl, we used the pipeline from Pile-CC (Gao et al., 2021), but removed the language model filtering. For various sizes of Pythia models, the corpus from ClueWeb22 consistently contains 13 billion tokens, while the corpus from Common Crawl is fixed at 2.8 billion tokens.

Pretraining Details. Our pretraining framework extends from the `Lit-GPT`[5] and we evaluate the performance of pretrained models using the standard `lm-evaluation-harness` toolkit[6]. Specifically, for all Pythia models, we employed the AdamW optimizer with a peak learning rate in line with Biderman et al. (2023). The total batch size was set to 480 (with the batch size of 12 per GPU and gradient accumulation being set to 10). For ClueWeb22, the model undergoes training for just one epoch. For CommonCrawl, it is trained across three epochs due to the size of the corpus. All of the models were trained on 4× NVIDIA A100-40G GPUs.

Datasets for Evaluation. We choose 8 standard datasets to evaluate the performance of pretrained language models. Some of them are from the Pythia standard benchmark (Biderman et al., 2023), supplemented by SWAG (Zellers et al., 2018) and BLIMP (Warstadt et al., 2020).

Baselines. In this experiments, we chose to use `htmlparser`[7] and `trafilatura` (Barbaresi, 2021) as the main baselines for comparison. `htmlparser` serves as the text pre-extraction tool for CommonCrawl WET file, while `trafilatura` has become the state-of-the-art web scraper.

A.4 Performance on Multilingual Webpages

Thanks to the careful planning of ClueWeb22, which allows us to evaluate the performance of scrapers in different languages. Specifically, we tested on snapshots coded `0001-01` for each language, the results are shown in Table 4. Among all the baseline scrapers, `NeuScraper` demonstrated excellent performance, even though it was trained only on English data.

[3] https://lemurproject.org/clueweb22

[4] https://commoncrawl.org/terms-of-use

[5] https://github.com/Lightning-AI/lit-gpt

[6] https://github.com/EleutherAI/lm-evaluation-harness

[7] https://htmlparser.sourceforge.net

	English		German		Spanish		French		Italian	
	Acc	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
bs4	41.29	58.10	40.49	57.23	39.34	56.18	40.05	56.84	38.92	55.72
html2text	40.44	53.88	38.91	52.51	37.19	50.28	38.65	51.72	37.57	50.20
boilerpipe	66.48	46.16	66.38	43.63	70.04	51.74	67.83	46.56	69.85	50.56
jusText	62.58	22.17	65.84	42.98	61.25	2.13	60.79	2.63	61.56	0.53
lxml	64.62	44.78	63.47	43.07	67.45	48.82	65.32	45.44	67.12	48.61
inscriptis	45.35	58.98	43.82	57.27	42.74	56.30	42.99	56.19	43.42	56.49
readability	68.47	48.22	70.16	50.17	72.08	54.38	71.10	52.21	72.69	54.85
trafilatura	70.70	61.08	73.84	62.43	73.93	62.14	73.60	62.20	74.49	62.87
NeuScraper	86.35	83.90	79.10	73.02	78.89	71.90	76.58	68.12	78.76	71.33
	Chinese		**Japanese**		**Dutch**		**Portuguese**		**Polish**	
	Acc	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**	**Acc**	**F1**
bs4	49.10	65.33	49.95	65.75	36.86	53.51	40.39	57.24	36.95	53.60
html2text	48.29	63.94	50.00	64.74	35.44	48.96	38.57	52.09	36.16	49.26
boilerpipe	61.31	42.44	57.33	30.26	70.01	44.82	67.93	49.14	66.96	36.91
jusText	51.38	0.75	51.26	0.49	65.11	12.84	60.33	3.03	63.60	0.76
lxml	62.22	52.79	60.38	50.16	66.16	41.36	66.59	48.73	65.72	40.01
inscriptis	53.09	66.35	53.76	66.57	40.11	53.69	44.01	57.65	40.51	53.15
readability	67.45	56.61	64.64	50.14	71.54	47.03	70.60	53.26	66.81	42.38
trafilatura	68.57	63.29	71.82	67.08	74.06	59.88	72.64	61.67	71.58	53.02
NeuScraper	74.76	73.99	74.01	73.80	77.70	68.13	77.48	71.28	75.84	64.61

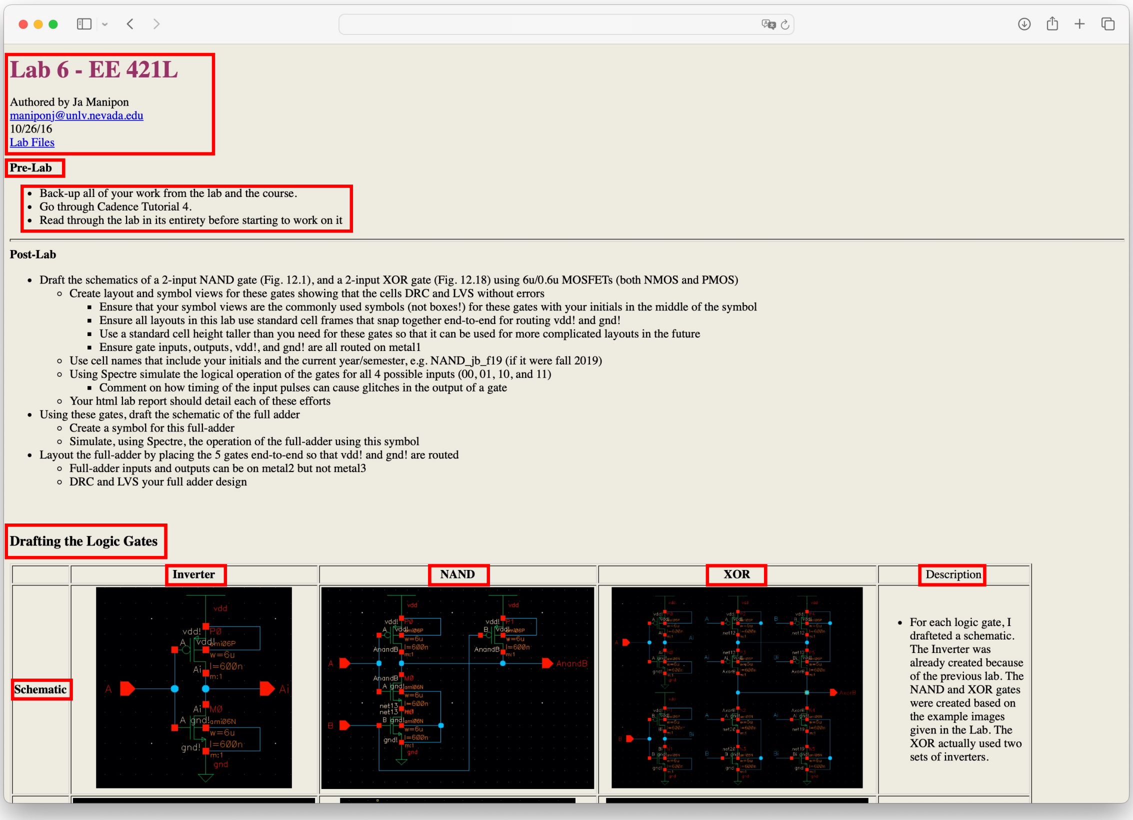
Table 4: Scarping Performance in Different Languages. We tested it on ClueWeb22 in different languages and NeuScraper showed significant improvements over the baseline scrapers.

A.5 Case Study

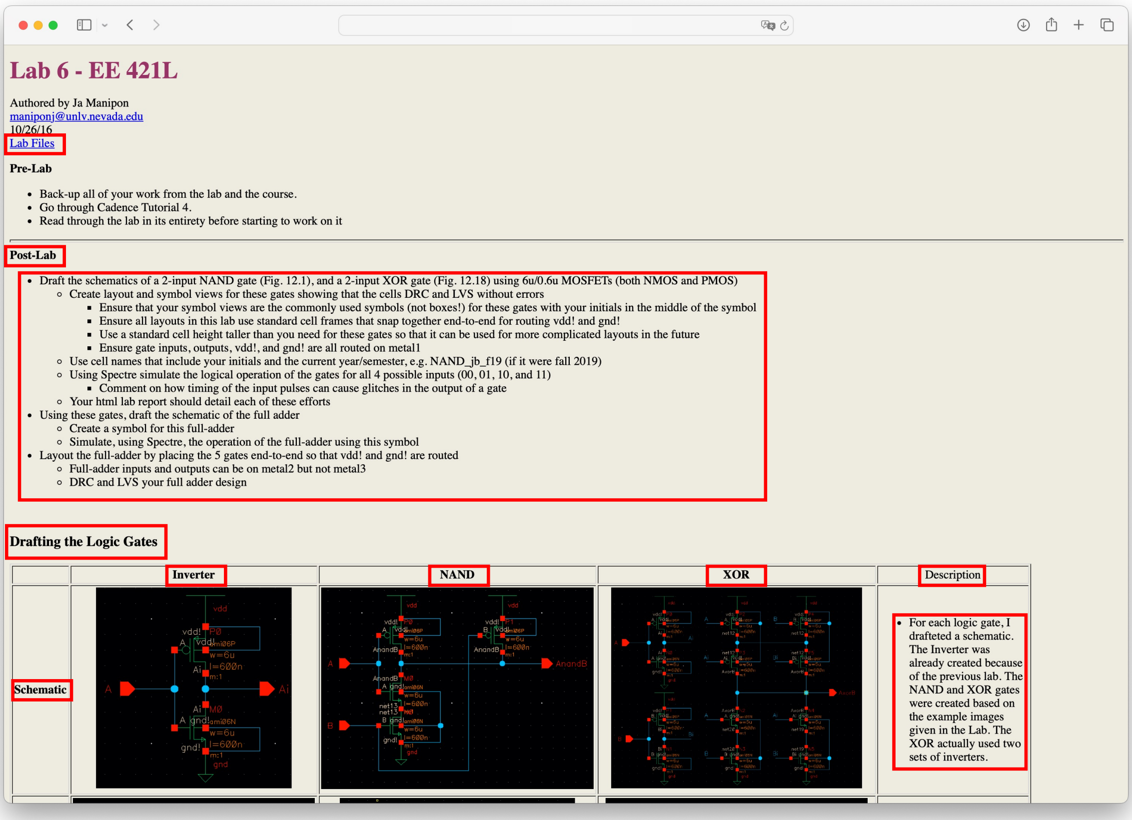
In this subsection, we show additional case studies of `NeuScraper` and `trafilatura`, our neural web scraper and a previously state-of-the-art web scraper.

We first analyze the case in Figure 3, where we use red boxes to indicate the content extracted by the scrapers. This is a college course page that contains some expertise in electrical engineering. When scraping this page, `trafilatura` loses a lot of textual content compared to our `NeuScraper`. By checking the raw HTML code, we found that there is an error caused by insufficient standardization of web pages: the paragraph tag “<*p*>” is used for headings on this page instead of the standard “<*h*>” tag. This page is readable for humans, but the HTML tag conveys an error that seriously affects the extraction performance of `trafilatura`. In contrast, our `NeuScraper` shows great adaptability. It not only extracts most of the paragraph content, but also removes useless information such as phone numbers, e-mails, dates, and so on.

Another typical case is interleaved boilerplate and body text, as shown in Figure 4. We use blue boxes to indicate the content extracted by the scraper. In this case, the boilerplate and body text are written in the same way. The boilerplate also uses “<*h*>” to identify headings and “<*p*>” for paragraphs, instead of the list surrounded by “<*li*>” in most cases. Recognizing it is difficult for `trafilatura`. `NeuScraper` leverages its ability to recognize latent semantic information to remove the boilerplate in such pages successfully.

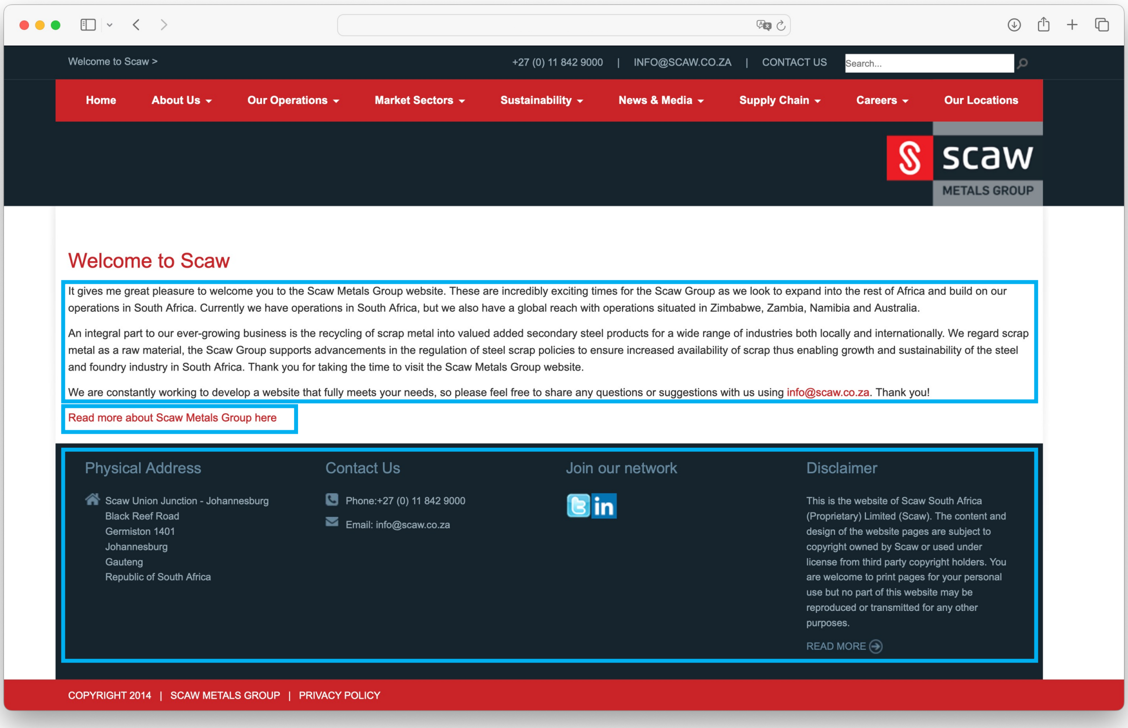


(a) Trafilatura.

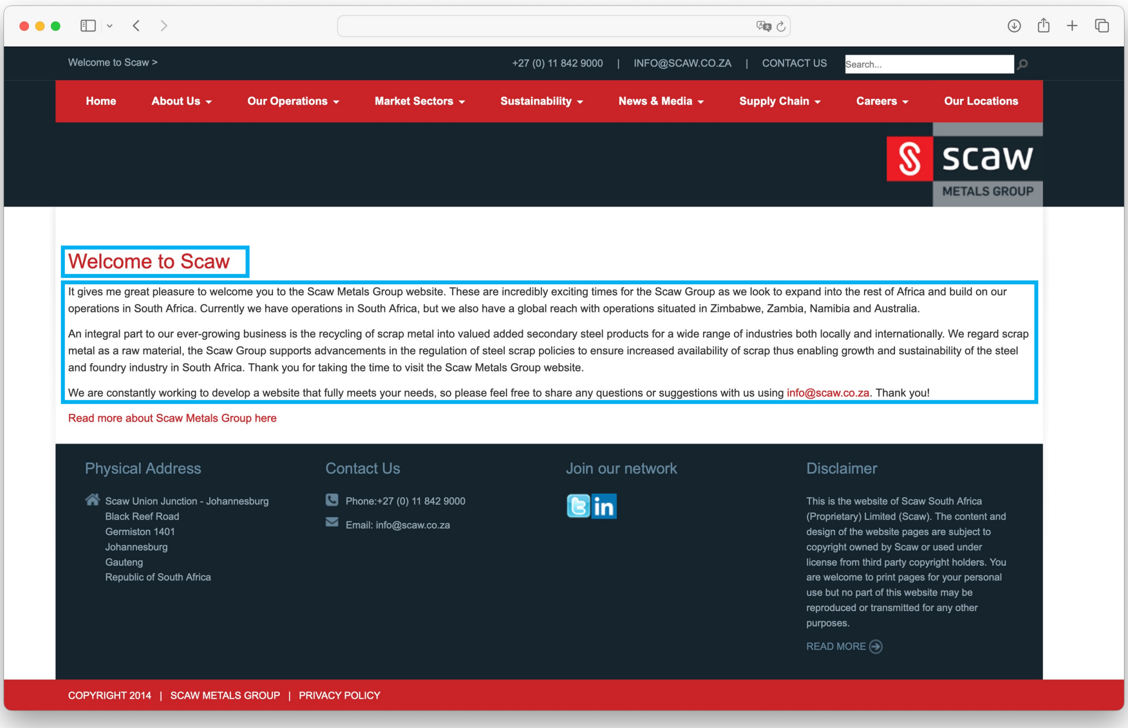


(b) NeuScraper.

Figure 3: Case#1 of the Primary Content Extraction Results Using Different Scrapers. The extracted parts are highlighted with red boxes.



(a) Trafilatura.



(b) NeuScraper.

Figure 4: Case#2 of the Primary Content Extraction Results Using Different Scrapers. The extracted parts are highlighted with blue boxes.