DRIPPER: TOKEN-EFFICIENT MAIN HTML EXTRACTION WITH A LIGHTWEIGHT LM

Anonymous authors
Paper under double-blind review

ABSTRACT

Accurately and efficiently extracting main content from general web pages is of great significance for obtaining training data for large models. Using well-pretrained decoder-only generative language models offers excellent document comprehension capabilities, thereby effectively enhancing parsing quality. However, it remains constrained by issues such as context window length, inference cost, and format hallucination. We present Dripper, an efficient HTML main content extraction framework powered by lightweight language models, which addresses these challenges through four key innovations: (1) We design a specialized HTML simplification algorithm that reduces input token count to 22% compared to raw HTML while preserving critical structural information; (2) We reformulate main content extraction as a semantic block sequence classification task, significantly reducing inference cost; (3) We introduce a controlled decoding mechanism that strictly constrains the output space through logits processors, effectively eliminating hallucination issues common in small-scale models; (4) We propose MainWebBench, an evaluation dataset containing over 7,800 web pages with meticulously human-annotated main content extraction labels. Experimental results demonstrate that using only a 0.6B parameter model, Dripper achieves state-of-the-art performance across all evaluation benchmarks and outperforms all baseline methods, attaining an ROUGE-N F1 score of 81.58%(83.13% with fall-back strategy) on our proposed MainWebBench dataset.

1 INTRODUCTION

The World Wide Web forms the foundational data repository for modern AI, serving as the primary source for training corpora like C4(Raffel et al., 2020) and for building the knowledge graphs that power large-scale applications(Wang et al., 2019). The sheer scale of this resource is immense, with web archiving projects like Common Crawl(Common Crawl Foundation) preserving billions of new pages each month. This massive volume presents a fundamental challenge for data utilization: the raw, unstructured HTML must first be converted into high-quality, structured data. Accordingly, the development of robust and accurate content extraction techniques has become a critical prerequisite for a wide range of downstream information processing tasks(Vogels et al., 2018b).

The primary obstacle lies in the failure of traditional extraction methods to handle the web's inherent complexity. While HTML standards provide semantic tags with clear intended uses—such as `<article>` for main content or `<aside>` for sidebars—their adoption in practice is highly inconsistentWang et al. (2022), rendering simple tag-based rules unreliable. Similarly, heuristic methods based on statistical properties like text or link density often falter. Even pages built from the same template can exhibit vast statistical variations simply due to differences in their core content, undermining the stability of these metrics. Furthermore, vision-based approaches like diffbot(Diffbot, 2025) are often rendered ineffective in large-scale offline processing scenarios. Web archives like Common Crawl typically store only raw HTML, lacking the corresponding CSS files required to render a page as its developer originally intended. These fundamental challenges explain why established tag-based, heuristic, and vision-based methods struggle to achieve both high accuracy and robust generalization. While the semantic understanding of well-trained decoder-only language models offers a promising theoretical solution(Wang et al., 2025), their direct application is thwarted by a distinct set of severe practical barriers. First, **excessive context length** makes processing raw HTML infeasible at scale.Our analysis of 14,000 Common Crawl files shows 29.3% of pages ex-

ceeded 32k tokens and 21.0% surpassed 128k tokens, lengths that far exceed the context windows of most SLMs. Second, the **structural complexity** of HTML presents a critical trade-off. While stripping all tags is an effective way to significantly reduce input length, this action simultaneously destroys the vital structural information they contain. Without these cues, an algorithm cannot reliably distinguish main content from noise and perform accurate extraction. Finally, LLMs are prone to **output hallucination** (Ji et al., 2023), a tendency to generate content not present in the source document, which constitutes a critical failure for an extraction task that demands high fidelity.

To address these challenges, we introduce **Dripper**, a novel framework that reframes web content extraction as an efficient Sequential Block Classification task, specifically designed for Small Language Models (SLMs). Our three-stage pipeline begins with a pre-processing step that simplifies the raw HTML, making it tractable for a compact model. We then employ a 0.6B parameter SLM, **Dripper-0.6B**, to perform a localized binary classification on each semantic block of the simplified document. To ensure perfect output fidelity and eliminate hallucinations, we guide the SLM's decoding with a custom logits processor, forcing it to produce a structured sequence of labels. Finally, a post-processing step uses these high-confidence labels to precisely extract the corresponding content blocks from the original HTML structure. The text from these selected blocks is then evaluated against the ground truth using ROUGE-N F1 as the primary metric. This approach circumvents the context length and hallucination issues inherent in holistic generative methods.

Our main contributions are summarized as follows:

(1) We introduce **a novel HTML simplification algorithm** that strips redundant information while preserving critical structural markers, compressing the average document size by 22% and making processing feasible for SLMs.

(2) The HTML document is represented as **a sequence of semantic blocks**, which transforms the task into a series of localized binary classifications. This approach dramatically reduces the problem's complexity while retaining essential hierarchical and contextual relationships.

(3) We design a **constrained decoding mechanism** using a custom logits processor. This converts the task from open-ended generation to producing a fixed, structured output, thereby systematically eliminating hallucinations and ensuring high-fidelity results.

(4) To facilitate rigorous and comprehensive evaluation, we construct and will publicly release **MainWebBench**, a new large-scale benchmark with over 7,800 meticulously annotated samples, making it seven times larger than any existing public alternative. Our experiments demonstrate that Dripper, using only a 0.6B parameter model, achieves state-of-the-art performance, outperforming all baselines on MainWebBench with a leading F1 score of **81.58%**, which increases to **83.13%** when augmented with a fallback strategy. Our trained model weights[1],code[2] and the MainWebBench benchmark[3] are publicly available.

2 Related Work

Main text extraction aims to extract main content from raw HTML while filtering out boilerplate elements such as navigation and advertisements, a critical technique for building high-quality web corpora. The methods for accomplishing this task have evolved through several distinct paradigms, each addressing the limitations of its predecessor.

Heuristic and rule-based Methods. Early approaches predominantly relied on manually engineered heuristics to distinguish main content from boilerplate. These methods operate on the observation that content-rich regions differ structurally from noisy elements, using features like text-to-tag ratios (CETR)(Weninger et al., 2010), visual cues from the rendered page (VIPS)(Cai et al., 2003), or a combination of heuristics such as link and stop-word density (Readability(Mozilla, 2015), jusText(Pomikálek, 2011)). While computationally efficient, these methods are often brittle and require continuous maintenance to adapt to evolving web design patterns.

[1] https://huggingface.co/anonymous-s2wrvq/Dripper

[2] https://anonymous.4open.science/r/dripper-1825

[3] https://huggingface.co/datasets/anonymous-s2wrvq/MainWebBench

Supervised Learning Methods. To move beyond handcrafted rules, subsequent work approached body text extraction as a supervised machine learning problem. This paradigm shift began with classic methods like Boilerpipe(Kohlschütter et al., 2010), Dragnet(Peters & Lecocq, 2013), which treated the task as a classification problem using manually designed features. The advent of deep learning marked a further evolution from feature engineering to representation learning. (Vogels et al., 2018a; Leonhardt et al., 2020; Zhou et al., 2021). To better leverage the hierarchical structure of HTML, subsequent research introduced Graph Neural Networks (GNNs)(Zhou et al., 2021) and Transformer-based architectures like WebFormer(Endrédy & Novák, 2013), which improved extraction accuracy by capturing complex relationships between nodes. While achieving higher accuracy, these models often require substantial labeled data, and their complex architectures incur significant computational overhead.

Hybrid Systems and Production Tools. In parallel with academic advancements, a suite of powerful open-source tools has emerged, often blending multiple techniques for practical application. Trafilatura(Barbaresi, 2021) has become a strong baseline by integrating a sophisticated cascade of rules with established algorithms like jusText(Pomikálek, 2011) and Readability(Mozilla, 2015) as fallbacks. Other tools, such as magic-html(opendatalab, 2024), focus on simplifying complex HTML structures before extraction, often as part of larger document AI ecosystems. More recently, frameworks such as crawl4ai(UncleCode, 2024) have adopted an explicitly hybrid architecture, combining rule-based selectors, traditional machine learning, and Large Language Models (LLMs) to provide versatile solutions for AI data pipelines.

Generative-Language-based Methods. Recent months have seen rapid progress in decoder-only large language models. Base models pre-trained on massive, high-quality, and highly-diverse corpora have become the de-facto starting point for most NLP tasks. The most representative work in this line is ReaderLM-v2(Wang et al., 2025), which frames main-content extraction as an HTML-to-Markdown translation problem. Starting from a 1.5 B-parameter Qwen2.5 checkpoint, the authors first extend the context window to 512 k tokens through continual pre-training, then fine-tune with supervised fine-tuning (SFT) and direct-preference optimization (DPO) to produce clean Markdown. This pipeline reuses the open-source model zoo and inference-acceleration stacks already available in the LLM community. Nevertheless, even the official best-practice implementation [4] still expects the full, un-pruned HTML page as input and generates the complete body text in one pass. This incurs heavy computational overhead and, during long-sequence generation, often produces unwanted artifacts such as repetitions or un-escaped HTML tags. Consequently, the potential of SLMs for extraction remains largely untapped.

3 METHODOLOGY

In this section, we detail the methodology of our Dripper framework. We begin in §3.1 with an overview of the system's three-stage architecture. Next, in §3.2, we elaborate on the core pre-processing and post-processing modules that enable efficient extraction. We then formally define the task as a sequence labeling problem in §3.3. Finally, in §3.4, we introduce our constrained decoding mechanism, which uses a custom logits processor to eliminate hallucinations.

3.1 SYSTEM ARCHITECTURE OVERVIEW

The Dripper framework operates through a three-stage pipeline: pre-processing, SLM-based extraction, and post-processing. As illustrated in Figure1, the system takes a raw HTML document as input and transforms it into a clean, structured Markdown output.

The process begins with the pre-processing module, which takes a raw HTML document and generates two distinct representations. The first is a `Simplified HTML`, which is simplified and chunked. The second is a `Mapping HTML`, which is only chunked but otherwise unmodified. This parallel representation is crucial for ensuring the final extracted content remains a valid subtree of the original Document Object Model (DOM). The `Simplified HTML` is then passed to Dripper-0.6B, which identifies and labels the main content blocks. The decoding process is constrained by a custom logits processor to guarantee the structural integrity and correctness of the output format. Finally, in the post-processing stage, the Dripper-0.6B's classification output is used to prune

[4] https://huggingface.co/jinaai/ReaderLM-v2

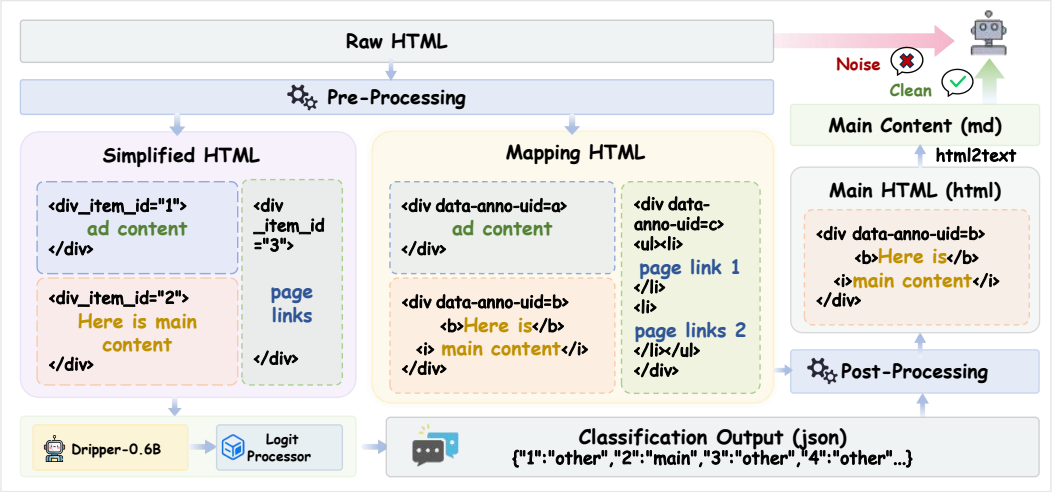

Figure 1: An overview of the Dripper framework, which operates as a three-stage pipeline. (1) Pre-processing: A raw HTML document is converted into two parallel representations: **Simplified HTML** for model input and **Mapping HTML** for final reconstruction. (2) Dripper-0.6B Extraction: Dripper-0.6B performs sequential block classification on the simplified input, guided by a custom logits processor to output a structured sequence. (3) Post-processing: The labels are used to select the corresponding blocks from **Mapping HTML** to construct the final, clean **Main Content**.

the `Mapping HTML`, yielding the final `Main HTML`. For downstream usability, `Main HTML` is converted into Markdown format using the html2text[5] library.

3.2 Pre-processing and Post-processing

`Raw HTML` is primarily designed for visual rendering, not for semantic interpretation by language models. Naively including all tags and attributes results in excessively long input sequences. Our pre-processing module is therefore guided by a multi-faceted strategy for simplification and chunking. The process begins with the **(1) preemptive removal of non-content tags**, such as `<style>`, `<script>`, `<header>`, and `<aside>`. Concurrently, we perform **(2) attribute simplification**, pruning all attributes except for class and id, which often carry the most valuable semantic cues for distinguishing content blocks. Following this, the document undergoes **(3) block-level chunking**, where it is segmented at elements that typically induce a line break in rendering. This strategy treats cohesive units like tables (`<table>`) and lists (`<ul>`) as indivisible blocks to preserve their integrity. To handle the common misuse of tables for page layout, we apply heuristic rules to permit splitting within them when necessary. Finally, to manage excessively long individual blocks, such as a table with many cells, a list with numerous items, or an overly long paragraph, we employ **(4) partial content truncation**. For instance, we may retain only a subset of table cells or the initial 200 characters of a long paragraph, as we empirically find this partial data is sufficient for accurate classification while significantly reducing input length.

This pre-processing pipeline transforms `Raw HTML` into a sequence of simplified blocks ready for Dripper-0.6B. To ensure the final output is a valid DOM subtree, the Mapping HTML is generated in parallel by applying only the block-level chunking to the original, unmodified HTML. The post-processing module then uses the Dripper-0.6B's output to select the corresponding content-bearing blocks from this `Mapping HTML` to construct the final result.

3.3 Task Formulation

The system architecture detailed above effectively transforms the content extraction task into a well-defined **sequence labeling problem**. Formally, our pre-processing module converts an HTML doc-

[5]https://pypi.org/project/html2text/

ument into a sequence of n simplified blocks, $X = [x_1, x_2, \ldots, x_n]$. Each block x_i has a corresponding ground-truth label $y_i \in \{0, 1\}$, where 1 indicates main content and 0 indicates boilerplate. The core task is to train a model f_θ that takes the sequence X as input and produces a predicted label sequence, $Y_{pred} = f_\theta(X)$, where $Y_{pred} = [y'_1, y'_2, \ldots, y'_n]$. This predicted sequence is then used by the post-processing module to select the corresponding blocks from the `Mapping HTML` and construct the final `Main HTML`.

This sequence labeling formulation is highly efficient and reliable. By simplifying and chunking the input, the token load on the model is substantially reduced. Furthermore, framing the task as a classification of discrete blocks constrains the output to a simple sequence of binary labels. This design minimizes the required output length and, by avoiding free-form text generation, inherently eliminates the risk of hallucination, guaranteeing that the extracted content is a faithful subset of the original document.

3.4 Constrained Decoding via a Custom Logits Processor

To eliminate hallucination and guarantee a valid output format, we implement a custom logits processor that functions as a deterministic finite state machine (FSM) during decoding. The FSM precisely controls the generation of the JSON-like output structure (e.g., {"1": "main", ...}) by deterministically managing all syntactic tokens, such as braces, quotes, and numeric keys. At each decoding step, it masks the SLM's logits, permitting the model to make a probabilistic choice only at the single critical juncture of classifying a block. At this point, the vocabulary is restricted to just 'main' and 'other', effectively converting the task into a series of high-confidence binary classifications. This method guarantees syntactically perfect output, fundamentally removing the risk of format errors or extraneous content, and enables even a small 0.6B model to perform this structured prediction task with perfect fidelity.

4 Dataset and Benchmark

In this section, we detail the construction of our large-scale training dataset (Section 4.1) and our new evaluation benchmark, MainWebBench (Section 4.2), along with its evaluation metrics.

4.1 Training Data Construction

To train our model effectively, we construct a large-scale, multi-faceted training dataset engineered to capture the diversity of the modern web. The dataset is curated through a three-stage sampling and filtering pipeline, ensuring variety in page layout, language, and document format.

Stage 1: Layout-Diverse Sampling. The initial stage focuses on capturing structural diversity. We begin by grouping pages by domain across 107 dumps of the Common Crawl dataset. For each domain, we featurize the DOM tree structure of its pages (capped at 10,000 randomly sampled pages for larger domains) and computed their pairwise cosine similarity. We then apply the DBSCAN algorithm to these feature vectors to identify distinct layout clusters. From this process, we sample one representative webpage from each of approximately 40 million unique clusters, yielding a candidate pool of 40 million structurally diverse pages.

Stage 2: Multilingual and Format-Aware Filtering. From this candidate pool, the second stage filtered for linguistic and format diversity. We first extract the main content of each page using `Trafilatura` and then employ the `Fasttext` lid-176[6] model for language identification. This step produced a balanced 10-million-page subset (4.75M English, 4.75M Chinese, 0.5M other languages). To further enhance diversity, we categorize these pages using the format classifier proposed by Wettig et al. (2025). A final balanced sampling across these identified formats results in a set of approximately 1 million pages (485k English, 487k Chinese, 50k other) for the final annotation stage.

Stage 3: Final Annotation. In the final stage, we process these 1 million pages through our simplification algorithm (detailed in Section 3.2). The resulting `Simplified HTML` is then provided to

[6]https://fasttext.cc/docs/en/language-identification.html

the Deepseek-chat API with a carefully crafted prompt (see Appendix Figure 5) to generate block-level labels. This automated pipeline yields approximately 1 million pages with high-quality, block-level annotations. After a final filtering step to remove samples containing no main content (i.e., all blocks were labeled as 'other'), we obtain our final training dataset of 870,945 samples.

4.2 MainWebBench: A New Benchmark for Content Extraction

To facilitate a more rigorous and fine-grained evaluation of web content extraction, we construct **MainWebBench**, a new benchmark comprising 7,887 meticulously annotated samples. Each sample contains four keys: 'html'(the raw html document); 'main_html'(the ground-truth as a valid html subtree identified by human annotators); 'convert_main_content'(a Markdown representation, generated from the ground-truth); and 'meta'(a rich set of annotations). MainWebBench is designed to serve as a gold-standard resource for evaluating extraction accuracy and enabling multi-dimensional performance analysis. An example data entry is shown in Appendix Figure 4.

4.2.1 Benchmark Construction

MainWebBench is constructed using a hybrid sampling strategy to ensure broad representation: 90% of pages are randomly sampled from Common Crawl to cover the long-tail of the web, while 10% are drawn from a list of top-ranking websites (Chinaz Alexa[7]) to include popular, well-designed pages. To address the ambiguity in defining "main content," we establish annotation rules based on two principles: **Contextual Integrity**, which includes content integral to the primary article (e.g., abstracts, references) and excludes peripheral elements (e.g., related-articles sidebars); and **Human-Generated Content**, which focuses on substantive material like article bodies and comments while filtering out auto-generated metadata (e.g., timestamps). Each page is meticulously annotated through a rigorous multi-stage process by using a custom-built tool(see Appendix Figure 3). Furthermore, we enrich the benchmark with rich metadata annotations—including language, style, a quantitative difficulty level, and rich content tags—enabling fine-grained analysis. More details of benchmark construction can be found in Appendix A.5

4.2.2 Evaluation Metrics

To accommodate the two primary output formats of extraction tools—(1) raw Markdown text and (2) `Main HTML` document—we establish a standardized evaluation protocol. For the latter case, all `Main HTML` outputs are first converted to a canonical Markdown representation using the `html2text` library to ensure a fair and consistent comparison. The primary evaluation metric is the ROUGE-N F1 score, computed between the predicted Markdown and the ground-truth. We use the jieba tokenizer for all computations and set N=5. We specifically choose ROUGE-N instead of ROUGE-L, as the latter's Longest Common Subsequence (LCS) algorithm has prohibitive computational complexity on the long documents in our benchmark, making ROUGE-N a more scalable and practical choice for evaluation.

5 Experiments

5.1 Experimental Setup

Supervised fine-tuning. We employ the Qwen3-0.6B((Team, 2025)) model as our base model, which is the smallest model in the Qwen3 series, featuring a 32K context window and support for over 100 languages. Supervised fine-tuning is performed using the Llama-Factory((Zheng et al., 2024)) framework, training on the full set of 870K samples for a fixed total of 4 epochs. We use the last checkpoint as **Dripper-0.6B**.

Baseline Methods. To comprehensively evaluate Dripper, we compare it against a diverse set of establish and state-of-the-art content extraction systems. Our comparison spans a wide spectrum of approaches, including classic heuristic and rule-based systems, supervised learning methods, production-grade hybrid tools, and recent large language model-based extractors. A detailed list and description of each baseline method is provided in Appendix, Table 4.

[7] https://malexa.chinaz.com/

Evaluation Modes. To ensure a fair comparison across tools with diverse output capabilities, we established a clear evaluation protocol. We test every applicable output format for each tool and use a consistent suffix to denote the mode: `-HTML+MD` for tools that output an intermediate HTML which we convert to Markdown; `-MD` for tools that natively output Markdown; and `-TEXT` for tools that natively output plain text. Because Dripper cannot process inputs that exceed its context-length limit, we assign a score of 0 to such inputs. Following the practice of `Trafilatura`, which uses a fallback algorithm for parsing failures, we also test a version of our method, Dripper_fallback, which invokes `Trafilatura` for oversized inputs.

5.2 Result of overhead reduction

The computational cost of a decoder-only language model is primarily determined by the input and output sequence lengths, with its complexity approximated by Equation 1.

$$\text{Cost} \approx \left(\mathrm{L\,d\,(N^2 + M\,N + M^2) + L\,d^2\,(N + M)}\right) \text{flops} \tag{1}$$

where L is the number of attention layers, d is the hidden-state dimension, N is the number of input tokens, and M is the number of output tokens. For Qwen3-0.6B we set $L = 28$ and $d = 1024$.

To quantify the efficiency gains of our approach, we compare its cost against a naive generative baseline. The baseline cost is estimated by using `Raw HTML` as input to generate the full Markdown content. For our method, we use `Simplified HTML` as input and the structured JSON classification as output. We measure the token lengths for both scenarios on the MainWebBench, and the results are detailed in Table 1.

Table 1: Token-length and cost comparison.

Pre-process	Input length (tokens)		Output length (tokens)		Cost estimate (flops)	
	mean	median	mean	median	mean	median
Without	44705.9	31987.0	2303.7	675.0	1.102×10^{14}	3.206×10^{13}
With	5734.5	3109.0	383.4	187.0	5.702×10^{12}	5.254×10^{11}
Ratio	12.83%	9.72%	16.64%	27.70%	5.18%	1.64%

The results reveal a substantial reduction in computational overhead. Our pre-processing pipeline dramatically shortens the input, reducing the mean token count to just 12.83% of `Raw HTML`, which is crucial for fitting within the model's context window. Simultaneously, reframing the task to output a compact JSON classification reduces the mean output length to 16.64% of the full content. These two synergistic effects culminate in a remarkable reduction in computational load, lowering the mean inference cost to just 5.18% of the naive approach. This makes SLM-based content extraction not only feasible but also highly efficient and controllable.

5.3 Results on MainWebBench

We present the main performance comparison on our MainWebBench benchmark in Table 2. The results are broken down by various tracks, including difficulty levels and the presence of rich content.

The results clearly demonstrate that Dripper achieves state-of-the-art performance, significantly outperforming all baseline methods across every track. The standalone Dripper model achieves an overall score of 0.8182, surpassing the best baseline, magic-html (0.7091), by a large margin. Notably, Dripper shows exceptional strength on challenging content types where traditional methods falter, such as pages with tables, equations, and especially conversational layouts (0.8028 vs. 0.5766 for the best baseline). This highlights the robustness of our semantic, block-based classification approach.

Additionally, due to limitations in preprocessing capacity and model generalization, Dripper occasionally fails to extract meaningful content from certain pages. We note that since Dripper follows a fundamentally different technical approach compared to rule-based systems like `Trafilatura`, its failures tend to be orthogonal to those of such systems. This allows for a straightforward fallback strategy: when Dripper returns no valid output, we use `Trafilatura` as a backup. With this

Table 2: Mean ROUGE-N F1 on MainWebBench with different tracks

name	mode	all	simple	mid	hard	table	code	equation	conversational
magic-html(opendatalab, 2024)	Html+MD	0.7091	0.7811	0.7095	0.6367	0.6681	0.8471	0.8470	0.4678
Readability(Mozilla, 2015)	Html+MD	0.6491	0.7370	0.6525	0.5570	0.5896	0.7774	0.7800	0.4608
Trafilatura(Barbaresi, 2021)	Html+MD	0.6358	0.7277	0.6391	0.5396	0.5505	0.6006	0.7327	0.5750
Trafilatura	MD	0.6237	0.7115	0.6279	0.5305	0.5400	0.5741	0.7168	0.5766
Trafilatura	TEXT	0.6049	0.6900	0.6088	0.5149	0.5271	0.5566	0.6955	0.5681
html2text(Swartz et al., 2025)	MD	0.5977	0.7499	0.5812	0.4678	0.5937	0.7729	0.7129	0.5494
BoilerPy3(Riebold et al., 2023)	TEXT	0.5413	0.6347	0.5448	0.4434	0.4380	0.4833	0.6590	0.4695
GNE(Kingname et al., 2024)	Html+MD	0.5148	0.6477	0.4942	0.4098	0.4129	0.5495	0.6160	0.3296
news-please(Hamborg et al., 2017)	TEXT	0.5012	0.5399	0.5250	0.4307	0.4193	0.5118	0.6701	0.4073
jusText(Pomikálek, 2011)	TEXT	0.4770	0.5132	0.5070	0.4010	0.3962	0.3779	0.6652	0.5222
BoilerPy3	Html+MD	0.4766	0.6443	0.4706	0.3174	0.3783	0.5532	0.6157	0.4103
Goose3(Lababidi et al., 2025)	TEXT	0.4354	0.4514	0.4645	0.3808	0.3589	0.2900	0.6376	0.3064
ReaderLM-v2(Wang et al., 2025)	MD	0.2264	0.3374	0.2078	0.1403	0.1801	0.2431	0.2927	0.1537
Dripper	Html+MD	**0.8182**	**0.8837**	**0.8178**	**0.7536**	**0.7693**	**0.8368**	**0.8889**	**0.7671**
Dripper_fallback	Html+MD	**0.8399**	**0.9010**	**0.8392**	**0.7799**	**0.7964**	**0.8673**	**0.9067**	**0.8028**

mechanism, the combined system (Dripper_fallback) achieves an overall F1 score of 0.8399. This result indicates that our semantic approach not only establishes a new state-of-the-art on its own but can also be effectively combined with existing methods to improve robustness and coverage.

5.4 Ablation Study

To analyze the data efficiency of our approach, we fine-tune the Qwen3-0.6B model on training subsets of increasing size: 2k, 5k, 10k, 100k, and 870k. We evaluate each resulting checkpoint on MainWebBench, from which we excluded samples whose simplified HTML exceeded our 32k token context window, as the standard Dripper model is designed to score 0 on such oversized inputs. This results in a performance gap of about 1.9% (0.818 for the full bench and 0.834 for the filtered bench).

To isolate the impact of our constrained decoding mechanism, we compare the performance of models trained with and without the custom logits processor. As shown in Figure 2, the logits processor provides a consistent performance improvement across nearly all data scales. The most significant gain (+2.3%) is observed at the 2k data scale, indicating that the FSM provides a strong structural prior that helps the model learn the task more efficiently in low-data regimes. As the training set grows, the model begins to learn the output format implicitly, and the performance gap narrows. Nevertheless, the logits processor provides an absolute guarantee of a syntactically perfect, hallucination-free output. This ensures the output is always stable and machine-readable, preventing format errors that would otherwise disrupt downstream tasks and making the processor a critical component for production-level reliability.

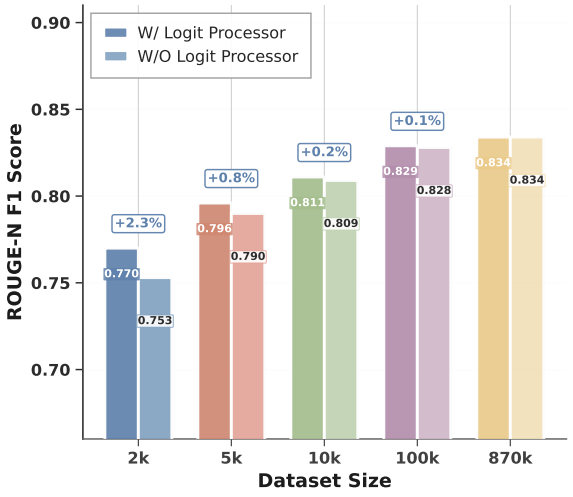

Figure 2: Impact of the logits processor on performance across various training data scales.

5.5 Performance on WCEB

To assess the generalization capabilities of Dripper, we evaluate it on the Web Content Extraction Benchmark (WCEB, (Bevendorff et al., 2023)) , a comprehensive and unified benchmark. WCEB addresses inconsistencies prevalent in many legacy datasets—such as plain-text-only ground truths, file encoding errors, and corrupted content from script injections—by providing a filtered and standardized collection. Since the ground truths in this consolidated benchmark are in plain text, we

adapt our evaluation protocol by using the `html-text`[8] library for the final conversion, a configuration we denote as `Html+TEXT`. To enable a more granular analysis, we also apply our difficulty stratification scheme to this dataset. A detailed description of the benchmark can be found in Appendix, Table 5.

The results on this suite of nine established benchmarks, presented in Table 3, confirm Dripper's strong generalization capabilities. Our method again establishes a new state-of-the-art, with the standalone Dripper model (0.8002) outperforming the strongest prior method, `Trafilatura` (0.7833). Furthermore, echoing the findings on MainWebBench, the Dripper_fallback strategy again demonstrates the complementary nature of our SLM-based approach and traditional heuristics, boosting the score further to 0.8154. This strong performance across a diverse collection of legacy datasets highlights Dripper's robustness, setting a new state-of-the-art for general web content extraction.

Table 3: Results on WCEB

name	mode	all	simple	mid	hard
Trafilatura	TEXT	0.7833	0.8122	0.7785	0.7609
Trafilatura	Html+TEXT	0.7791	0.7896	0.7758	0.7731
Readability	Html+TEXT	0.7642	0.7744	0.7595	0.7601
magic-html	Html+TEXT	0.7506	0.7780	0.7573	0.7144
Goose3	TEXT	0.7272	0.7432	0.7312	0.7059
news-please	TEXT	0.7048	0.7051	0.7103	0.6970
justText	TEXT	0.6936	0.7445	0.6966	0.6389
BoilerPy3	TEXT	0.6221	0.6481	0.6468	0.5631
html2text	TEXT	0.6142	0.7273	0.6165	0.4982
BoilerPy3	Html+TEXT	0.6015	0.6532	0.6035	0.5474
GNE	Html+TEXT	0.5166	0.5138	0.5069	0.5323
ReaderLM-v2	TEXT	0.3077	0.3718	0.2928	0.2636
Dripper	Html+TEXT	**0.8002**	**0.8293**	**0.8005**	**0.7707**
Dripper_fallback	Html+TEXT	**0.8154**	**0.8363**	**0.8143**	**0.7959**

6 Conclusion

In this work, we introduce Dripper, a highly efficient and accurate framework for web content extraction. We demonstrate that our custom-trained 0.6B parameter Small Language Model, Dripper-0.6B, achieves state-of-the-art performance by reframing the extraction problem. Our approach's success is rooted in three key technical contributions. First, our HTML Simplification Algorithm intelligently strips redundant tags and attributes, drastically reducing the input token count while preserving essential structural cues. This simplified document is then processed through our novel Sequential Block Classification paradigm, which transforms the open-ended extraction task into a series of simple, localized binary classifications. Finally, to guarantee absolute fidelity, our Deterministic Logits Processor constrains the SLM's output during the decoding phase, which completely eliminates the risk of hallucination and ensures a syntactically perfect structured output. To rigorously validate our method, we also construct and release MainWebBench, a new large-scale benchmark of 7,887 samples, on which Dripper-0.6B proves its superiority over all baselines. Furthermore, by integrating a heuristic-based fallback for inputs that exceed its context window, our Dripper_fallback variant pushes performance even higher, demonstrating the robustness and complementary nature of our method.

7 Limitation and Future work

Despite careful web preprocessing development, 1.3% of Common Crawl pages still exceed Qwen3's content-window limit post-simplification and remain unprocessable. Additionally, extreme DOM structures in some pages break chunking/simplification algorithms, hindering effective main text extraction. Future fixes include enhancing preprocessing and extending the base model's context window via continued pre-training (to relax preprocessing's token budget). Moreover, while we use Qwen3's smallest 0.6B model to cut overhead, scaling to 100B-scale pages poses cost issues. A promising solution is tailoring data recipes for web parsing to pre-train small (0.01B–0.1B) dedicated base models from scratch, lowering inference costs.

[8] https://pypi.org/project/html-text/

8 Reproducibility Statement

We are committed to ensuring the full reproducibility of our research. The architecture of our proposed framework, **Dripper**, and its core components are detailed in the Methodology Section 3. The construction of our large-scale training dataset is described in Section 4.1, while the creation and structure of our new benchmark are detailed in the **MainWebBench** Section 4.2. Our complete experimental setup, including all baselines, evaluation protocols, and metrics, is presented in the Experiments Section 5. To facilitate direct verification and future work, we have made our resources publicly available: the full source code[9], the trained **Dripper** model weights[10], and the complete **MainWebBench** benchmark[11].

References

Adrien Barbaresi. 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*, pp. 122–131, 2021.

Janek Bevendorff, Sanket Gupta, Johannes Kiesel, and Benno Stein. An Empirical Comparison of Web Content Extraction Algorithms. In *46th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR 2023)*. ACM, 2023. doi: 10.1145/3539618.3591920. URL https://dl.acm.org/doi/10.1145/3539618.3591920.

Deng Cai, Shipeng Yu, Ji-Rong Wen, and Wei-Ying Ma. Vips: a vision-based page segmentation algorithm. 2003.

Common Crawl Foundation. Common crawl: Open-source web crawl data & infrastructure. https://commoncrawl.org/.

Diffbot. Extract API: Structured Data Extraction, 2025. URL https://www.diffbot.com/products/extract/.

István Endrédy and Attila Novák. More effective boilerplate removal-the goldminer algorithm. *Polibits*, (48):79–83, 2013.

Felix Hamborg, Norman Meuschke, Corinna Breitinger, and Bela Gipp. news-please: A generic news crawler and extractor. In *Proceedings of the 15th International Symposium of Information Science*, pp. 218–223, March 2017. doi: 10.5281/zenodo.4120316.

Ziwei Ji, Nayeon Lee, Rita Frieske, Tiezheng Yu, Dan Su, Yan Xu, Etsuko Ishii, Ye Jin Bang, Andrea Madotto, and Pascale Fung. Survey of hallucination in natural language generation. *ACM computing surveys*, 55(12):1–38, 2023.

Kingname et al. Generalnewsextractor, 2024. URL https://github.com/GeneralNewsExtractor/GeneralNewsExtractor.

Christian Kohlschütter, Peter Fankhauser, and Wolfgang Nejdl. Boilerplate detection using shallow text features. In *Proceedings of the third ACM international conference on Web search and data mining*, pp. 441–450, 2010.

Mahmoud Lababidi et al. goose3, 2025. URL https://github.com/goose3/goose3.

Jurek Leonhardt, Avishek Anand, and Megha Khosla. Boilerplate removal using a neural sequence labeling model. In *Companion Proceedings of the Web Conference 2020*, pp. 226–229, 2020.

Mozilla. Readability.js, 2015. URL https://github.com/mozilla/readability.

opendatalab. magic-html. https://github.com/opendatalab/magic-html, 2024.

[9] https://anonymous.4open.science/r/dripper-1825

[10] https://huggingface.co/anonymous-s2wrvq/Dripper

[11] https://huggingface.co/datasets/anonymous-s2wrvq/MainWebBench

Matthew E Peters and Dan Lecocq. Content extraction using diverse feature sets. In *Proceedings of the 22nd international conference on world wide web*, pp. 89–90, 2013.

Jan Pomikálek. Removing boilerplate and duplicate content from web corpora. *Disertacnı práce, Masarykova univerzita, Fakulta informatiky*, 2011.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020. URL http://jmlr.org/papers/v21/20-074.html.

John Riebold et al. Boilerpy3, 2023. URL https://github.com/jmriebold/BoilerPy3.

Aaron Swartz, Charlie Tanksley, et al. html2text, 2025. URL https://pypi.org/project/html2text/.

Qwen Team. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

UncleCode. Crawl4ai: Open-source llm friendly web crawler & scraper. https://github.com/unclecode/crawl4ai, 2024.

Thijs Vogels, Octavian-Eugen Ganea, and Carsten Eickhoff. Web2text: Deep structured boilerplate removal. In *European Conference on Information Retrieval*, pp. 167–179. Springer, 2018a.

Thijs Vogels, Octavian-Eugen Ganea, and Carsten Eickhoff. Web2text: Deep structured boilerplate removal. *CoRR*, abs/1801.02607, 2018b. URL http://arxiv.org/abs/1801.02607.

Feng Wang, Zesheng Shi, Bo Wang, Nan Wang, and Han Xiao. Readerlm-v2: Small language model for html to markdown and json. *arXiv preprint arXiv:2503.01151*, 2025.

Peilu Wang, Hao Jiang, Jingfang Xu, and Qi Zhang. Knowledge graph construction and applications for web search and beyond. *Data Intelligence*, 1(4):333–349, 2019.

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

Tim Weninger, William H Hsu, and Jiawei Han. Cetr: content extraction via tag ratios. In *Proceedings of the 19th international conference on World wide web*, pp. 971–980, 2010.

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

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. Llamafactory: Unified efficient fine-tuning of 100+ language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand, 2024. Association for Computational Linguistics. URL http://arxiv.org/abs/2403.13372.

Yichao Zhou, Ying Sheng, Nguyen Vo, Nick Edmonds, and Sandeep Tata. Simplified dom trees for transferable attribute extraction from the web. *arXiv preprint arXiv:2101.02415*, 2021.

A Appendix

A.1 Baseline Methods for Web Content Extraction

Table 4: Baseline Methods for Web Content Extraction

Method	Description
Heuristic and Rule-Based Methods	
Readability	Reader view algorithm for removing distracting elements
jusText	Two-pass processing with block size, link density, and stopword heuristics
Goose3	Article extractor with hand-crafted rules
html2text	Simple HTML to markdown converter
GNE	Text and symbol density-based extraction using mathematical formulas
Supervised Learning Methods	
BoilerPy3	Python port of Boilerpipe, decision tree-based text block classification
Hybrid Systems and Production Tools	
Trafilatura	Sophisticated rule cascade with jusText and Readability as fallbacks
news-please	Meta-extractor combining multiple extractors for news articles
magic-html	HTML structure simplification for extraction pipelines
Pre-trained Language Models	
ReaderLM-v2	SLM-based content extraction with semantic understanding

A.2 Standard Benchmarks

Table 5: Details of Web Content Extraction Datasets

Dataset	Pages	Source & Characteristics
CleanEval	738	De-facto standard dataset from 2007 shared task combining development and evaluation sets of English web pages with basic structural markup ground truth
CleanPortalEval	71	Extension of CleanEval featuring multi-page samples from 4 major news domains
CETD	700	Created for density-based extractor evaluation across 6 domains
Dragnet	1,379	Combined sources from popular RSS feeds, 23 major news sites, 178 Technorati blogs, plus CETR and CleanEval conversions
L3S-GN1	621	Created by BoilerPipe authors with unique HTML annotation using span-wrapped CSS classes for 5-level content relevance
Google-Trends-2017	180	Dataset created for BoilerNet neural network training featuring binary CSS class annotations on DOM leaf nodes to distinguish content from boilerplate
Readability	115	Mozilla reader mode test suite with original and simplified HTML for evaluation
Scrapinghub	181	Created by Zyte for benchmarking proprietary extraction services

A.3 SCREENSHOT OF THE WEB PAGE ANNOTATION TOOL

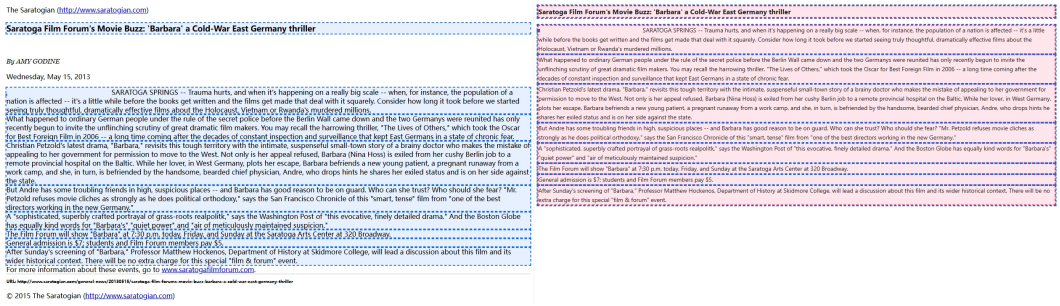

Figure 3: Screenshot of the web page annotation tool. The main content selection is highlighted in blue on the left, with a real-time preview on the right.

A.4 EXAMPLE DATA FROM MAINWEBBENCH

```
{
  "track_id": "XXXX",
  "html": "<html><body><h1 cc-select=True>Hello
    ↪  world!</h1><aside>advertisement</aside></body></html>",
  "main_html": "<html><body><h1>Hello world!</h1></body></html>",
  "convert_main_content": "# Hello world!",
  "meta": {
    "language": "en",
    "style": "Normal",
    "level": "easy",
    "table": "without",
    "code": "without",
    "equation": "without"
  }
}
```

Figure 4: An example data from MainWebBench. It includes the raw source, the ground-truth main HTML, its Markdown conversion, and a rich set of metadata for fine-grained analysis.

A.5 BENCHMARK CONSTRUCTION

Data Sampling. MainWebBench is constructed using a hybrid sampling strategy to ensure both broad representation and relevance. 90% of the samples are randomly drawn from the Common Crawl dataset to cover the long-tail web, while the remaining 10% are sampled from a list of top-ranking websites (Chinaz Alexa[12]) to include popular, professionally designed pages. The final benchmark is highly diverse, containing pages from 5,434 unique top-level and 5,904 unique second-level domains.

Annotation Rules. To address the ambiguity in defining ”main content” for unconventional layouts, we establish two core annotation principles. First, **Contextual Integrity** dictates that content integral to the main article—such as a table of contents, abstract, or reference list—is included. Conversely, contextually independent elements like ”related articles” sidebars or copyright footers are excluded. Second, the main content is defined as **Human-Generated Content**, including article bodies, user comments, and Q&A posts, while associated auto-generated metadata like usernames and timestamps are excluded.

Annotation Process. The annotation for each page followed a rigorous three-stage process using a custom-built tool(see Appendix, Figure 3) that allowed for tag-level granularity. The process

[12]https://malexa.chinaz.com/

involved: (1) an initial pass by one annotator, (2) a review and correction pass by a second annotator, and (3) a final quality assurance check by a senior inspector, who made the final adjudication to resolve any discrepancies. Pages uninterpretable due to rendering issues were discarded.

Metadata Annotation. To enable detailed, fine-grained analysis, we annotate each page with a rich set of metadata. This includes **Language**, identified by GPT-5 and labeled as `en` (English) or `non_en` (other), and **Style**, classified by GPT-5 as `Conversational` for pages with user-generated content or `Normal` otherwise. We also develop a quantitative **Difficulty Level**, determined by an `overall_complexity_score` calculated for each page. To compute this score, we first measure four distinct metrics: *DOM structural complexity* (based on tree depth and width), *text distribution sparsity* (transitions between text/non-text nodes), *content-type diversity* (a count of rich content types), and *link density* (the ratio of hyperlinked text). These four values are individually normalized, and their weighted sum produces the final score. Based on the distribution of this `overall_complexity_score` across the benchmark, we then categorize pages into `simple`, `medium`, and `hard` using the 30th and 70th percentiles as dynamic thresholds. Finally, we add **Rich Content Tags** to identify the presence of tables (`<table>`), code blocks (`<code>`), and mathematical formulas (`<math>` or LaTeX patterns) using BeautifulSoup.

A.6 Prompt for data synthesis

A.7 Use of Large Language Models

A large language model is used as a writing assistant during the preparation of this manuscript. The primary use of the LLM is for improving grammar, clarity, and phrasing of the text. The LLM does not contribute to the core research ideas, experimental design, data analysis, or the formulation of our conclusions. The authors have reviewed and edited all text and take full responsibility for the final content of this paper.

```
f"""As a front-end engineering expert in HTML, your task is to analyze the given HTML structure and accurately classify elements with the {ITEM_ID_ATTR} attribute as either "main" (primary content) or "other" (supplementary content). Your goal is to precisely extract the primary content of the page, ensuring that only the most relevant information is labeled as "main" while excluding navigation, metadata, and other non-essential elements.
Guidelines for Classification:
Primary Content ("main")
Elements that constitute the core content of the page should be classified as "main". These typically include:
 For Articles, News, and Blogs:
The main text body of the article, blog post, or news content.
Images embedded within the main content that contribute to the article.
 For Forums & Discussion Threads:
The original post in the thread.
Replies and discussions that are part of the main conversation.
 For Q&A Websites:
The question itself posted by a user.
Answers to the question and replies to answers that contribute to the discussion.
 For Other Content-Based Pages:
Any rich text, paragraphs, or media that serve as the primary focus of the page.
Supplementary Content ("other")
Elements that do not contribute to the primary content but serve as navigation, metadata, or supporting information should be classified as "other". These include:
 Navigation & UI Elements:
Menus, sidebars, footers, breadcrumbs, and pagination links.
"Skip to content" links and accessibility-related text.
 Metadata & User Information:
Article titles, author names, timestamps, and view counts.
Like counts, vote counts, and other engagement metrics.
 Advertisements & Promotional Content:
Any section labeled as "Advertisement" or "Sponsored".
Social media sharing buttons, follow prompts, and external links.
 Related & Suggested Content:
"Read More", "Next Article", "Trending Topics", and similar sections.
Lists of related articles, tags, and additional recommendations.
Task Instructions:
You will be provided with a simplified HTML structure containing elements with an {ITEM_ID_ATTR} attribute. Your job is to analyze each element's function and determine whether it should be classified as "main" or "other".
Response Format:
Return a JSON object where each key is the {ITEM_ID_ATTR} value, and the corresponding value is either "main" or "other", as in the following example:
{{"1": "other","2": "main","3": "other"}}
Important Notes:
Do not include any explanations in the output, only return the JSON.
Ensure high accuracy by carefully distinguishing between primary content and supplementary content.
Err on the side of caution, if an element seems uncertain, classify it as "other" unless it clearly belongs to the main content.

Input HTML:
{html_str}

Output format should be a JSON-formatted string representing a dictionary where keys are item_id strings and values are either 'main' or 'other'. Make sure to include ALL item_ids from the input HTML
"""
```

Figure 5: Prompt template for Main HTML classification.