HtmlRAG: HTML is Better Than Plain Text for Modeling Retrieved Knowledge in RAG Systems

Jiejun Tan*
Zhicheng Dou†
zstanjj@ruc.edu.cn
dou@ruc.edu.cn
Gaoling School of Artificial Intelligence
Renmin University of China
Beijing, China

Wen Wang
Mang Wang
Weipeng Chen
Baichuan Intelligent Technology
Beijing, China

Ji-Rong Wen
Gaoling School of Artificial Intelligence
Renmin University of China
Beijing, China
jrwen@ruc.edu.cn

Abstract

Retrieval-Augmented Generation (RAG) has been shown to improve knowledge capabilities and alleviate the hallucination problem of LLMs. The Web is a major source of external knowledge used in RAG systems, and many commercial RAG systems have used Web search engines as their major retrieval systems. Typically, such RAG systems retrieve search results, download HTML sources of the results, and then extract plain texts from the HTML sources. Plain text documents or chunks are fed into the LLMs to augment the generation. However, much of the structural and semantic information inherent in HTML, such as headings and table structures, is lost during this plain-text-based RAG process. To alleviate this problem, we propose HtmlRAG, which uses HTML instead of plain text as the format of retrieved knowledge in RAG. We believe HTML is better than plain text in modeling knowledge in external documents, and most LLMs possess robust capacities to understand HTML. However, utilizing HTML presents new challenges. HTML contains additional content such as tags, JavaScript, and CSS specifications, which bring extra input tokens and noise to the RAG system. To address this issue, we propose HTML cleaning, compression, and a two-step block-tree-based pruning strategy, to shorten the HTML while minimizing the loss of information. Experiments on six QA datasets confirm the superiority of using HTML in RAG systems. Our code and datasets are available at https://github.com/plageon/HtmlRAG.

CCS Concepts

• **Information systems** → **Web search engines**.

Keywords

HTML, Retrieval-Augmented Generation, Large Language Model

*This work was done when Jiejun Tan was doing an internship at Baichuan.
†Corresponding author.

WWW '25, April 28–May 2, 2025, Sydney, NSW, Australia.

ACM ISBN 979-8-4007-1274-6/25/04
https://doi.org/10.1145/3696410.3714546

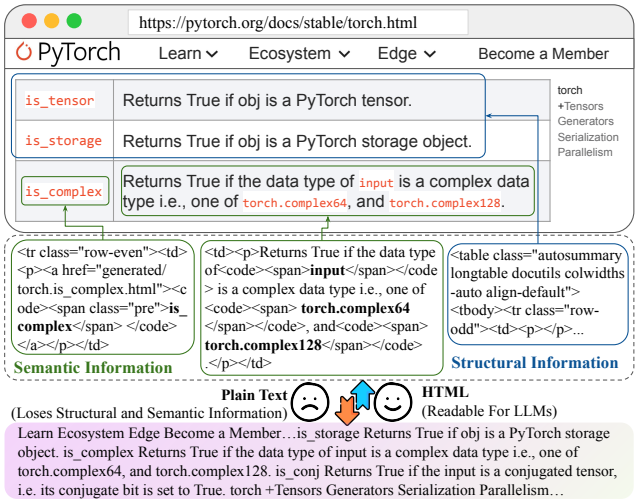

Figure 1: Information loss in HTML to plain text conversion.

ACM Reference Format:
Jiejun Tan, Zhicheng Dou, Wen Wang, Mang Wang, Weipeng Chen, and Ji-Rong Wen. 2025. HtmlRAG: HTML is Better Than Plain Text for Modeling Retrieved Knowledge in RAG Systems. In *Proceedings of the ACM Web Conference 2025 (WWW '25), April 28–May 2, 2025, Sydney, NSW, Australia.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3696410.3714546

1 Introduction

Large Language Models (LLMs) have been proven to have powerful capabilities in various natural language processing tasks [51, 53, 55]. However, at the same time, LLMs show deficiencies such as forgetting long-tailed knowledge [32], offering outdated knowledge [3], and hallucination [46, 47, 83]. Retrieval-augmented generation (RAG) utilizes a retrieval system to fetch external knowledge and augment the LLM. It has proved effective in mitigating hallucinations of LLMs [50, 85]. Many RAG systems, such as Perlexity [56] and SearchGPT [52], have been developed, and they commonly use Web search engines as the underlying retrieval systems.

Traditional RAG pipelines typically use plain text as the format for retrieved knowledge [24, 72]. HTML documents from the Web are often converted into plain text and concatenated with the user's query before being fed into the LLM. We found that converting

HTML to plain text leads to the significant loss of structural and semantic information. Figure 1 illustrates that a web page containing tabular form becomes disordered when converted to plain text. Even worse, original HTML tags, such as "<code>" and "<a>", denoting important information, are discarded during conversion. Thus, in this paper, we tend to investigate an intuitive idea: *Can we take HTML as the format of external knowledge in RAG systems to preserve the information in HTML documents to a larger extent?*

Taking HTML as the format of external knowledge offers several advantages beyond preserving the information inherent in HTML documents. During pre-training, LLMs have encountered HTML documents [6, 18, 20], which means that they inherently possess the ability to understand HTML without requiring further fine-tuning [30, 82]. Recently, both proprietary and open source LLMs have begun to support increasingly longer input windows, making it feasible to input more extensive HTML documents [14, 78, 81]. Furthermore, documents in Latex, PDF, and Word formats can be converted to HTML with minimal loss, expanding the potential application of HTML as the format of external knowledge [7, 70, 73].

However, employing HTML as the knowledge format for LLMs also presents the challenge of handling longer input sequences and noisy contexts. Our preliminary experiments show that a real HTML document from the Web contains over 80K tokens on average, among which over 90% of the tokens are CSS styles, JavaScript, Comments, or other meaningless tokens. Compared to the common maximum context window of current LLMs, which ranges from 32K to 128K, an individual document length of 80K is unacceptable. The aforementioned meaningless tokens in HTML documents can also affect the generation quality of LLMs. To solve this problem, in this paper, we devise a **HTML Cleaning** module to remove semantically irrelevant content in HTML documents, while keeping the main content intact. We also adjust the HTML tree structure without losing semantic information, for example, merging multiple layers of single nested HTML tags and removing empty tags. These processes reduce the length of the HTML to 6% of its original size.

Even after cleaning, HTML documents remain relatively long (over 4K each) to LLMs. To shorten the input context and remove the noise contained in the original retrieved documents, existing RAG systems have utilized different types of post-retrieval result refiners [22, 26, 75, 84]. These refiners extract the relevant text chunks or key sentences from the documents, regarding the user's query and LLMs' preference, and discard other content. These plain-text-based refiners cannot be directly applied to HTML because simply chunking HTML without considering its structure may generate unreasonable chunks. Hence, we further design an **HTML Pruning** module, which functions upon the intrinsic tree structure of HTML. The pruning process is comprised of the following steps:

(1) **Building a Block Tree**. Each HTML document can be parsed into a DOM tree [67]. We do not simply prune HTML on the DOM tree because it is too finely-grained [19, 71], which brings much computational cost. Instead, we propose to build a corresponding block tree, in which the original DOM tree nodes are merged into hierarchical blocks. The granularity of the block tree can be adjusted by the degree of merging.

(2) **Pruning Blocks based on Text Embedding**. We then prune the block tree using an on-the-shelf embedding model, because it is a simple but effective way to calculate the block's relevance scores with the user's query based on their embedding similarity. We apply a greedy pruning algorithm that removes blocks with lower similarity scores, and gets a pruned block tree. However, we observe that the embedding model may fail to work well with the fine-grained blocks because embeddings learned for these small blocks are usually vague and inaccurate, so this pruning step is limited to coarse-grained block trees.

(3) **Generative Fine-grained Block Pruning**. To prune the block tree further, we expand the leaf nodes of the pruned block tree and build a finer-grained block tree. Since the generative model has a longer context window, it can model the block tree globally and is not limited to modeling one block at a time. Thus we further develop a generative model to prune HTML over the fine-grained blocks. The generative model is supposed to calculate the score for each block, which is given by the generation probability of a unique sequence indicating the block. The sequence is given by the path of HTML tags, starting from the root tag and walking down to the block's tag and text (e.g., "<html><body> <p>block content..."). Finally, according to the block scores, we apply a similar greedy pruning algorithm to get the final pruned HTML.

We conduct extensive experiments on six datasets including ambiguous QA, natural QA, multi-hop QA, and long-form QA. Experimental results confirm the superiority of HTML as the format of external knowledge over plain text.

Our contributions are threefold: (1) We propose to take HTML as the format of knowledge in RAG systems, which retains information of the original HTML; (2) We propose a simple but effective HTML cleaning algorithm; (3) We propose a two-stage HTML pruning algorithm. This can be applied to most RAG systems and strikes a balance between efficiency and effectiveness.

2 Related Works

2.1 Retrieval-Augmented Generation (RAG)

RAG systems augment LLM with external knowledge. A typical RAG pipeline includes components such as a query rewriter [64], a retriever [11, 37, 48, 62], a reranker [62, 72], a refiner [22, 26, 75], and a reader [5, 86]. This typical pipeline is widely used by mainstream RAG frameworks, such as LangChain [8] and LlamaIndex [43]. Many works aim to optimize components in the pipeline, and previous works also manage to enhance the performance of RAG in other ways. Some methods devise new RAG frameworks, like retrieving external knowledge actively when internal knowledge is missing [5, 23, 64], or letting the LLM plan the retrieval process in a straight line or a tree structure [31, 61]. However, most existing RAG systems take plain text as the format of external knowledge [12, 13, 25, 38]. Instead, we propose to take HTML as a new format, and we believe using HTML can keep richer semantics in retrieved results.

2.2 Post-Retrieval Process of RAG

RAG systems usually apply post-retrieval processes (i.e., result refiners) to extract only the useful content to shorten the input context sent to LLMs. The chunking-based refiner is a widely used solution, which first chunks the text according to certain rules, and then uses a reranking model to select top chunks with high

relevance [29, 36, 49]. Another solution is abstractive refiner, which utilizes a text-to-text language model to generate abstracts of results [17, 22, 75]. Some works use off-the-shelf abstractive models [79, 80] or fine-tuned abstractive models [22] to summarize retrieved results in a segmented and hierarchical manner. Others leverage the logits of language models to determine the importance of words within documents [41, 45].

The aforementioned post-retrieval result refiners are all based on plain text. The existing chunking-based methods cannot be directly applied to HTML because simply chunking HTML without considering its structure may generate unreasonable chunks. Furthermore, the abstractive refiners may have problems such as difficulty in dealing with excessively long HTML, high computational cost, or limited understanding of HTML. To alleviate these problems, in this paper, we propose to prune HTML based on its DOM structure.

2.3 Structured Data Understanding

Previous works have demonstrated that structured data such as HTML [9, 77] and Excel tables [35, 66, 68] contain richer information compared to plain text. These works design specialized tasks [34, 68] over structured data or fine-tune language models to understand structured data [4, 71]. Our research is not limited to understanding a certain format of data but recommends using a richer data format in the general RAG systems. To the best of our knowledge, we are the first to propose using HTML as the input for RAG systems.

3 Methodology

In this paper, we propose HtmlRAG, which uses HTML instead of plain text as the format of retrieved knowledge in RAG systems, aiming to keep richer semantic and structured information that is missing in plain text. We emphasize that HTML is a popular data format for documents in a knowledge base and other document formats can be easily converted into HTML.

Taking HTML as the format of external knowledge presents a new challenge of excessively long context. Hence, in HtmlRAG, we propose to prune the original HTML documents into shorter ones progressively. We first apply an HTML cleaning module (§3.2) to remove useless elements and tags. We then propose a two-step structure-aware pruning method to further refine the resulting HTML (§3.4). More specifically, we delete less important HTML blocks with low embedding similarities with the input query (§3.4.1), and then conduct a finer block pruning with a generative model (§3.4.2). The overview of our method is shown in Figure 2.

3.1 Problem Definition

In the RAG pipeline, a retriever retrieves a collection of HTML documents D from the Web, with a total length of L. Meanwhile, we have an LLM M as the reader, which generates an answer a. The LLM has a maximum length of context window l, considering both efficiency and quality. Our HTML compression algorithms map D to a shorter HTML document d. Its length can fit into the LLM's context window, namely the length of d must be less than or equal to l. Our goal is to optimize the compression algorithm to find the best mapping from D to d so that the answer a output by the LLM has the highest quality.

3.2 HTML Cleaning

Since the original HTML documents are excessively long (over 80K each), model-based methods are inappropriate at this step. Thus, we first design a rule-based HTML cleaning, which pre-processes the HTML without considering the user's query. This cleaning process removes irrelevant content and compresses redundant structures, retaining all semantic information in the original HTML. The compressed HTML after HTML cleaning is suitable for RAG systems equipped with long-context LLMs that are not willing to lose any information before generation. The cleaned HTML also serves as the basis for the following HTML pruning.

3.2.1 HTML Content Cleaning. The HTML documents retrieved from the Web contain a large amount of extra content that is invisible to human users, such as HTML tags, CSS, JavaScript, etc. Most of the HTML tags provide rich structural information that helps the LLM understand the HTML, while CSS and JavaScript content provide limited assistance. So the specific cleaning steps, which are almost lossless, are as follows: (1) We remove CSS styles, Comments, and JavaScript; (2) We clear lengthy HTML tag attributes.

3.2.2 Lossless Structural Compression. We find that in most HTML documents, their original HTML structure contains redundancies. We can conduct the following compression to the HTML structure without losing semantic information: (1) We merge multiple layers of single-nested tags. For example, we simplify " some text " to " some text "; (2) We removed empty tags, such as " ".

3.3 Granularity-Adjustable Block Tree Building

To prune all retrieved HTML documents as a whole, we first concatenate all retrieved HTML documents together, and use Beautiful Soup [59] to parse the concatenated HTML document to a single DOM tree. Pruning HTML using the DOM tree is the most natural way, but the DOM tree is so finely-grained that numerous nodes and the deep tree structure bring huge computational costs.

Considering the above problem, we propose an optimized tree structure that models HTML, which is not so fine-grained. Ideally, the granularity of the tree structure can be adjusted for different pruning requirements. We term it as a "block tree", and we set the maximum number of words per block, $maxWords$ to control the granularity of the block tree. In terms of block tree construction, we start from a DOM tree, and we merge fragmented child nodes into their parent and treat them as a block. We can recursively merge blocks or child nodes into their parent to form a bigger block under the condition that the number of words in a block does not exceed $maxWords$. After merging, original leaf nodes that are unable to be merged are also regarded as blocks. Algorithm details are demonstrated in Appendix F.

3.4 Block-Tree-Based HTML Pruning

The block-tree-based HTML pruning consists of two steps, both of which are conducted on the block tree structure. The first pruning step uses an embedding model to prune the result output by the HTML cleaning module, while the second step uses a generative model to prune the result output by the first pruning step.

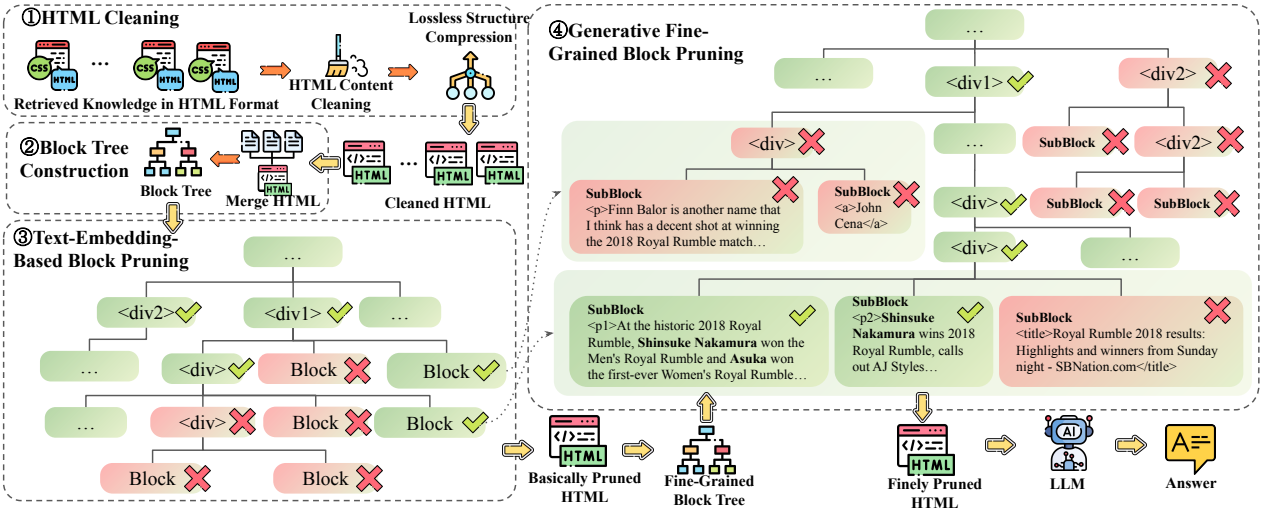

Figure 2: Overview of the HtmlRAG pipeline.

3.4.1 Pruning Blocks based on Text Embedding. The refining process is expected to shorten the retrieval results while preserving key information as much as possible. A straightforward idea is to extract plain text in the block and calculate a similarity score with the user's query using text embeddings. Then we use a greedy algorithm to prune the block tree by deleting low-similarity blocks and retraining higher ones. In practice, we keep deleting the block with the lowest relevance until the total length of the HTML documents satisfies the context window we set. After block deleting, redundant HTML structures will re-appear, so we re-adjust the HTML structure, meaning multiple layers of single-nested tags are merged and empty tags are removed. The detailed pruning algorithm is demonstrated in Appendix F.

The embedding-based HTML pruning algorithm is lightweight but effective. It adapts to the HTML format better compared to plain-text-based refiners. However, it still has limitations, mainly reflected in the following aspects: (1) The embedding model's context window is limited to the scope of text within the block each time. It does not directly compare candidate blocks in a single inference. Thus the embedding model lacks a global view of the document information; (2) The embedding model cannot handle block trees with finer granularity, because the text within most blocks is not long enough for the embedding model to obtain semantic features.

3.4.2 Generative Fine-Grained Block Pruning. To further prune blocks with a finer granularity, we expand the leaf nodes of the pruned block tree and get a finer-grained block tree. Given the limitations of the embedding-model-based block pruning, we propose to use a generative model because it has a long context to cover the whole block tree and is not limited to modeling one block at a time. Yet processing the cleaned HTML directly with a generative model is inappropriate because the cleaned HTML is long (60K on average), which brings much computational cost. Similarly, the generative model is supposed to calculate scores for blocks. Inspired by CFIC [57] and Generative Retrieval [39, 40], which takes the text chunk's sequence generation probability as the score for that chunk, we propose to use a sequence of tags to identify a block. Specifically, the sequence consists of tags starting from the root tag and walking down to the block's tag, and we term this sequence as "block path". In the inference phase, the generative model follows the structure of the block tree and calculates the scores of blocks in the block tree. The scores of blocks are derived from the token logits, as displayed in Figure 3. At last, we use the same block pruning operation as we mention in §3.4.1 to obtain the refined HTML document.

The details of the generative fine-grained block pruning module are introduced in the remaining section.

(1) Training a Path-aware Generative Model. Long-context LLMs are capable of modeling a long-context input containing HTML format and following instructions [10, 44]. Considering the computational cost, we employ an existing lightweight long-context LLM as the foundation model. The model input is the concatenation of an HTML, the query, and an instruction, as demonstrated in Figure 4. The instruction is specially designed to help the LLM understand this path generation task, but we find that the unfine-tuned LLM does not meet our requirements. We attribute this to the fact that existing LLMs have not encountered similar tasks or instructions in either pre-training data or instruction fine-tuning data, because the path generation task is proposed for the first time.

Thus we fine-tune the generative model to align with the target of generating the path for the most relevant block. So we design the output format as shown in Figure 4: the block path, followed by the block content. The block content is appended to provide an extra supervising signal that helps the generative model learn the features of the most relevant block. Additionally, to discriminate between children with the same tag name, we append a number to the end of the original tag name. For example, two children with the same " " tag are renamed as "<div1>" and "<div2>".

We collect a small amount of supervised data to enhance the model's capability in block path generation. Following the typical SFT process [58], the steps for training data collecting, filtering, and constructing are as follows: First, we sample queries from the

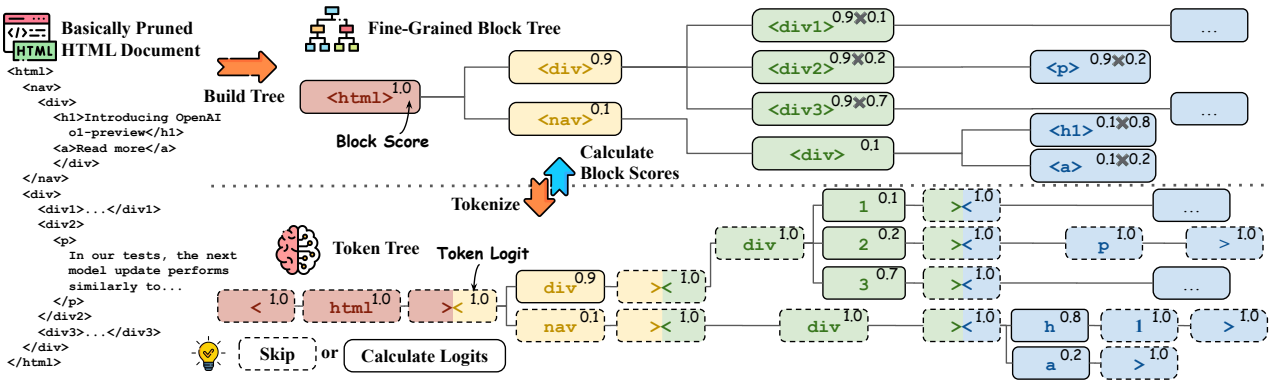

Figure 3: Block score calculation. The block tree is transformed into the token tree with a tokenizer, and corresponding HTML tags and tokens are marked with the same colors. Token generation probabilities are in the upper right corner, and tokens in dashed boxes do not require inference. In the upper right corner of the block tree, the block probabilities are displayed, which can be derived from the corresponding token probabilities.

```
Input:
**HTML**:  "{HTML}"
**Question**:  **{Question}**
Your task is to identify the most relevant text piece
to the given question in the HTML document. This text
piece could either be a direct paraphrase to the fact,
or a supporting evidence that can be used to infer the
fact. The overall length of the text piece should be
more than 20 words and less than 300 words. You should
provide the path to the text piece in the HTML document.
An example for the output is: <html1><body><div2><p>Some
key information...
Output:
<html1><body><div2><p>At the historic 2018 Royal Rumble,
Shinsuke Nakamura won the Men's Royal Rumble. . .
```

Figure 4: The prompt for the generative model.

training set of several open-source QA datasets. For each query, we retrieve a couple of related HTML documents using the online search engine Bing. Then we clean the retrieved HTML, and prune the HTML with the embedding model. By adjusting the output length in HTML pruning, we get pruned HTML documents of various lengths, ranging from 2K tokens to 32K tokens. After that, we build a block tree from each HTML document pruned by the embedding model, and calculate the exact match score for the content within blocks with the gold answer. To ensure the data quality, we discard samples in which no block's content exactly matches the gold answer, meaning highly relevant HTML documents are not retrieved. More training details are discussed in Appendix B.

(2) Efficient Tree-Based Inference with Dynamic Skipping. During inference, the generative model is supposed to calculate block scores, and the score for block b is Score(b). Each block has a block path, and we first tokenize it to tokens $\{t_1, t_2, \cdots, t_N)\}$, suppose it has N tokens in total (e.g., "<html> " is tokenized to {"<", "html", "><", "div", ">"}). Given the model's input sequence *input* and $n-1$ already generated tokens, the generative model GenModel calculates the logit of the n-th token t_n in the output sequence as below:

$$\text{Logits}(t_n) = \text{GenModel}(t_n | \{input, t_1, \cdots, t_{n-1}\}). \tag{1}$$

We propose an efficient tree-based inference, and the tree is termed as the "token tree", which has a one-to-one correspondence with the block tree, given a specific tokenizer. We merge tokenized block paths to get the block tree, as Figure 3 shows. For example, {"<", "html", "><", "nav", ">"} and {"<", "html", "><", "div", ">"} share the same prefix, {"<", "html", "><"}, and can be merged. Ultimately, the i-th token in the tokenized block path will appear at the i-th level of the token tree. After the token tree construction, we calculate the probabilities of tokens in the token tree. The calculation has the following conditions: (1) The probability of the root node is 1.0, which is often "<", depending on the tokenizer; (2) The probabilities of singleton child nodes, which have no siblings, are 1.0; (3) The probabilities of other nodes are calculated by the generative model *GenModel*. Suppose token t_n has K siblings, which are the n-th token in the output sequence, we get the logits of siblings $\{t_n^1, t_n^2, \cdots\}$ by Equation (1) and take the softmax of logits as probabilities. In summary, the probability of a token t_n^k (the n-th token in the tokenized block path, and the k-th sibling) is given by:

$$\text{P}(t_n^k) = \begin{cases} 1.0, & \text{if } n = 1 \text{ or } K = 1; \\ \frac{\exp(\text{Logits}(t_n^k))}{\sum_{i=1}^{K} \exp(\text{Logits}(t_n^i))}, & \text{overwise.} \end{cases} \tag{2}$$

In the first two conditions, it is needless to infer with the generative model, meaning many tokens can be skipped. This brings down the inference computational cost. Apart from token skipping, the order of token logit calculation also matters a lot in computational cost. We apply a depth-first algorithm to traverse the token tree and calculate token logits so that the tokens that are calculated sequentially share the longest prefix sequence. This strategy reuses the KV cache of prefix sequences at most. Algorithm details are displayed in Appendix F.

At last, we transform the generation probabilities from the token tree back to the block tree so that we can calculate block scores. To

Table 1: Results of HtmlRAG and baselines under the short-context setting. Hit@1 is the proportion of instances where at least one short answer matches. The best and second best results are in bold and underlined. The symbol † signifies that our model achieves superior results among baselines in a statistically significant manner (t-test, p-value < 0.05).

	Method	ASQA		Hotpot-QA	NQ		Trivia-QA		MuSiQue	ELI5	
		Hit@1	EM	EM	Hit@1	EM	Hit@1	EM	EM	ROUGE-L	BLEU
Llama3.1-8B-4K	BM25	45.00	19.84	36.25	40.75	30.66	84.75	26.17	5.75	15.90	**6.56**
	BGE	68.50	31.47	43.25	59.00	44.59	**92.25**	27.50	**10.00**	15.87	6.30
	E5-Mistral	62.50	28.51	38.50	56.50	41.73	90.00	27.05	9.00	15.77	5.85
	LongLLMLingua	59.25	26.34	40.75	55.25	41.82	90.00	27.02	9.00	**16.08**	6.45
	JinaAI Reader	53.50	23.14	34.00	47.25	34.41	84.75	24.83	6.75	15.80	5.65
	HtmlRAG	**71.75**†	**33.31**†	**43.75**†	**61.75**†	**45.90**†	91.75†	**27.82**†	8.75	15.51	5.84
Llama3.1-70B-4K	BM25	49.50	21.95	38.25	47.00	35.56	88.00	25.63	9.50	16.15	**6.99**
	BGE	68.00	**30.57**	41.75	59.50	45.05	93.00	27.04	12.50	16.20	6.64
	E5-Mistral	63.00	28.75	36.75	59.50	44.07	90.75	26.27	11.00	16.17	6.72
	LongLLMLingua	62.50	27.74	45.00	56.75	42.89	92.50	**27.23**	10.25	15.84	6.39
	JinaAI Reader	55.25	23.73	34.25	48.25	35.40	90.00	25.35	9.25	16.06	6.41
	HtmlRAG	**68.50**†	30.53†	**46.25**†	**60.50**†	**45.26**†	**93.50**†	27.03	**13.25**†	**16.33**†	6.77†

prevent precision overflow, we take the sum of the logarithm of token probabilities as the score of the block b:

$$\text{Score}(b) = \sum_{i=1}^{N} \log(\text{P}(t_i)). \tag{3}$$

After we get the block scores, we reuse the greedy block pruning algorithm introduced in §3.4.1 to get the finely pruned HTML.

4 Experiments

4.1 Datasets

We select six datasets, including: (1) ASQA [63]: a QA dataset consists of ambiguous questions that can be answered by multiple answers supported by different knowledge sources; (2) Hotpot-QA [76]: a QA dataset consists of multi-hop questions; (3) NQ [33]: A QA dataset containing real user's queries collected by Google; (4) Trivia-QA [28]: a QA dataset containing real user's questions; (5) MuSiQue [65]: A synthetic multi-hop QA dataset; (6) ELI5 [16]: A long-form QA dataset with questions collected from Reddit forum. We randomly sample 400 questions from the test set (if any) or validation set in the original datasets for our evaluation.

To simulate the real industrial web search environment, we require real web pages from the Web in HTML format as retrieved documents. However, the widely used Wikipedia search corpus mainly consists of pre-processed passages in plain text format. So, we apply Bing search API in the US-EN region to search for relevant web pages, and then we scrap static HTML documents through URLs in returned search results. We provide the URLs and corresponding HTML documents in our experiments for reproduction.

4.2 Evaluation Metrics

Our method aims to enhance the overall performance of RAG, so we evaluate the LLM's response as the end-to-end result. We choose different evaluation metrics for datasets according to their question-and-answer formats. For Hotpot-QA and MuSiQue, in which each question is annotated with a single short answer, we report Exact Match. For ASQA, NQ, and Trivia-QA, whose questions are annotated with several short answers, we report Exact Match and Hit@1. Hit@1 means at least one answer of the annotated answers finds the exact match in the LLM's response. ELI5 is annotated with long-form answers, and we report ROUGE-L [42] and BLEU [54].

4.3 Baselines

Since to the best of our knowledge, we are the first to take HTML as the format of retrieved knowledge in RAG systems, we compare HtmlRAG to baselines that conduct post-retrieval processes. These baselines are mainly based on plain text or Markdown format. We select three chunking-based refiners and uniformly follow the chunking method in LangChain framework [8]. The reranking compartment is plug-and-play and we use three different rerank models: (1) BM25 [60]: A widely used sparse rerank model; (2) BGE [74]: An embedding model, BGE-Large-EN with encoder-only structure; (3) E5-Mistral [69]: A embedding model based on an LLM, Mistral-7B [21], with decoder-only structure. Besides we select two abstractive refiners: (1) LongLLMLingua [22]: An abstractive model using Llama7B to select useful context; (2) JinaAI Reader [27]: An end-to-end light-weight LLM with 1.5B parameters fine-tuned on an HTML to Markdown converting task dataset.

4.4 Experimental Settings

For a fair comparison, all end-to-end QA results are experimented with the latest open-source LLM, Llama-3.1-70B-Instruct and Llama-3.1-8B-Instruct [15] under a 4K context window. As for the implementation details of our method, we construct a block tree with a granularity of 256 words before pruning with the embedding model, and we construct a finer-grained block tree with a granularity of 128 words before pruning with the generative model. We choose BGE-Large-EN [74] as the embedding model for the HTML pruning. We choose a lightweight Phi-3.5-Mini-Instruct [1] with 3B parameters as the backbone for our generative model. The training

Table 2: Results of HtmlRAG without pruning and baselines under Llama-3.1-70B-Instruct-128K. Hit@1 is the proportion of instances where at least one short answer matches. The best and second best results are in bold and underlined. The symbol † signifies that our method achieves superior results among baselines in a statistically significant manner (t-test, p-value < 0.05).

Method	ASQA		Hotpot-QA	NQ		Trivia-QA		MuSiQue	ELI5	
	Hit@1	EM	EM	Hit@1	EM	Hit@1	EM	EM	ROUGE-L	BLEU
Vanilla HTML	44.00	17.52	28.00	46.75	36.06	81.50	22.58	3.25	15.69	5.16
Plain Text	**59.75**	<u>25.16</u>	<u>41.00</u>	**59.75**	**44.11**	<u>93.50</u>	<u>26.75</u>	<u>8.75</u>	<u>16.88</u>	**7.44**
Markdown	56.00	24.00	39.00	57.00	42.00	92.00	26.43	8.25	**16.91**	<u>6.74</u>
HtmlRAG w/o Prune	<u>58.75</u>†	**25.28**†	**42.25**†	<u>58.00</u>†	<u>43.65</u>†	**95.00**†	**27.21**†	**10.75**†	16.57	6.32

data used in fine-tuning the generative model contains 2635 automatically constructed training samples ranging from 2K to 32K in length. More implementation details can be found in Appendix B.

4.5 Experimental Results

Main experimental results are demonstrated in Table 1. Our method, HtmlRAG meets or exceeds the baselines across all metrics on the six datasets. This demonstrates the effectiveness of HTML pruning. Additionally, we make the following observations:

(1) For chunking-based refiners, we followed LangChain's [8] chunking rule, which chunks according to HTML tag headings (h1, h2, etc.). Although this chunking strategy considers certain HTML structures, it does not utilize the structural information as effectively as our method. Moreover, converting the final output to plain text still results in a loss of HTML structural and semantic information. Among the three rerankers we applied, the sparse retriever BM25 is inferior to two dense retrievers. Among two dense retrievers, the encoder-based BGE performs better than the decoder-based e5-mistral, despite the latter having more parameters.

(2) Among the abstractive refiners, LongLLMLingua is not optimized for HTML documents, so its extraction ability is affected when dealing with HTML. Additionally, the plain text output loses structural information, resulting in inferior performance compared to our method. The JinaAI-reader generates the refined Markdown given the HTML input. However, token-by-token decoding with long input and output lengths is not only challenging for end-to-end generative models, but also has high computational cost.

4.6 Further Analysis

4.6.1 The Effectiveness of HTML Cleaning. To validate the priority of HTML as the format of retrieved knowledge, we compare our HTML cleaning module, namely the results of HtmlRAG without pruning, with other rule-based cleaning strategies, including (1) Vanilla HTML; (2) Plain Text: The plain text extracted with an on-the-self package BeautifulSoup [59]; (3) Markdown: The Markdown converted by an on-the-self converter Markdownify [2]. Additional experiments on token count show that HTML-Clean drops over 94.07% tokens of the original HTML, while the number for plain text and Markdown conversion are 96.71% and 90.32% respectively.

The cleaned HTML is still long, so we conduct experiments under a long-context setting (128K), as shown in Table 2. When HTML is taken as the format of external knowledge, HtmlRAG without pruning meets or outperforms plain text and Markdown on most datasets, demonstrating its validity. Besides, we make the following observations: (1) Unprocessed HTML documents contain a large amount of irrelevant content, so all cleaning algorithms show improvements over vanilla HTML. (2) Under a limited context window, HTML format reference contains less documents and has lower exact match score due to extra HTML tags occupying tokens. Under such circumstances, HTML performs comparable or even better than plain text. This shows the positive effect of the rich structural information of HTML.

4.6.2 Ablation Study. We conduct ablation studies to demonstrate the effectiveness of each component in HtmlRAG, including block tree construction (Block Tree), HTML pruning with the embedding model (Prune-Embed), and HTML pruning with the generative model (Prune-Gen). From the results in in Table 3, we can see: (1) In the ablation study for block tree construction, we use the DOM tree instead of the block tree. Units in the DOM tree are so fragmented that the embedding model fails to capture sufficient semantic features, thus causing a drop in performance. The performance of the generative model is also affected due to the increase in the length of block paths. (2) In the ablation study for pruning with the embedding model, we only use the generative model to prune the cleaned HTML. Without the basically pruned HTML by the embedding model, the input to the generative model becomes very long (exceeds 32K), resulting in high computational costs and poor performance. (2) In the ablation study for pruning with the generative model, we only use the embedding model to prune the cleaned HTML. The result is inferior compared to the further pruned HTML using the generative model, because the embedding model's global understanding and ability to process finely-grained block trees are inferior to the generative model.

4.6.3 Impact of Block Tree Granularity. The most critical hyperparameter in HTML pruning is granularity. A coarse granularity reduces the flexibility of pruning, while a fine granularity makes it difficult to extract text embeddings for small blocks, and leads to overly long block paths for the generative model, so we need to find balancing points. In Figure 5, we experiment with HTML pruning under different granularity ranging from 64 to 512 words, and compare their result with a strong baseline. Prune-Embed stands for using the basically pruned HTML by the embedding model, and Prune-Gen stands for using the finely pruned HTML by the generative model. It can be observed that the generative model adapts to a finer granularity than the embedding model and generally outperforms the embedding model. This validates the rationality of our two-stage pruning method.

Table 3: Ablation studies for HtmlRAG.

Method	ASQA		Hotpot-QA	NQ		Trivia-QA		MuSiQue
	Hit@1	EM	EM	Hit@1	EM	Hit@1	EM	EM
HtmlRAG	**68.50**	**30.53**	**46.25**	**60.50**	**45.26**	**93.50**	**27.03**	**13.25**
w/o Block Tree	59.50 (9.00%↓)	25.50 (5.03%↓)	40.25 (6.00%↓)	56.25 (4.25%↓)	42.07 (3.19%↓)	92.00 (1.50%↓)	26.59 (0.44%↓)	8.00 (5.25%↓)
w/o Prune-Embed	56.75 (11.75%↓)	24.05 (6.48%↓)	37.50 (8.75%↓)	49.50 (11.00%↓)	37.27 (7.99%↓)	91.75 (1.75%↓)	26.02 (1.01%↓)	9.75 (3.50%↓)
w/o Prune-Gen	62.00 (6.50%↓)	26.74 (3.79%↓)	38.75 (7.50%↓)	57.75 (2.75%↓)	42.91 (2.35%↓)	89.50 (4.00%↓)	25.55 (1.48%↓)	7.00 (6.25%↓)

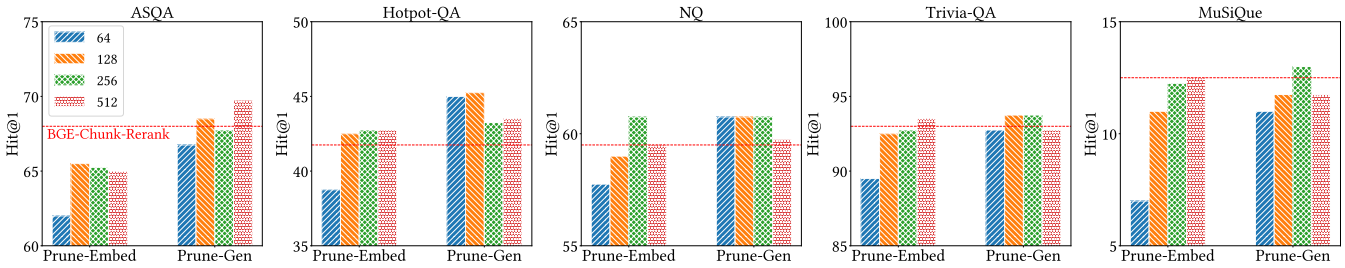

Figure 5: Experimental results for the impact of block tree granularity. The results of Prune-Embed and Prune-Gen are represented in a bar chart, with a red dashed horizontal line indicating the performance of the strong baseline method, chunking-based refiner with BGE (BGE-Chunk-Rerank).

Method	# Params	Storage	# In-Tok	# Out-Tok	Latency
BGE	200M	2.5G	93.54K	740.3	10.5ms
Prune-Emb	200M	2.5G	152.5K	2653	30.8ms
Prune-Gen	3B	7.2G	6750	28.70	6.15ms
LLM Chat	70B	131G	3661	182.9	915ms

Table 4: Analysis of inference cost on ELI5 dataset We compare the chunking-based refiner using BGE (BGE), the two HTML pruning steps basing on the text embedding (Prune-Embed) and the generative model (Prune-Gen) in HtmlRAG, and LLM chatting (LLM Chat) by model parameters, storage, average input tokens, and average output tokens.

4.6.4 Light Weight HTML Pruning. To show that our HTML pruning method does not significantly increase the computational cost despite using an LLM with 3B parameters, we conduct an efficiency analysis. Table 4 shows the computational cost of our method compared to the baseline and the cost of the LLM's inference. We can see that the HTML pruning with the embedding model still maintains a similar computational cost to the chunking-based refiner. The computational cost of the generative model is a bit higher than the baseline but still much lower than the cost of the LLM for chatting. Additional experiments show that there are over 45% of nodes that can be skipped, explaining the little increase in the generative model's computational cost.

Analysis of token counts shows the average token count for all retrieved knowledge in HTML format is 1.6M, suppose we retrieve 20 HTML documents. HTML cleaning reduces the token count to 135K, HTML pruning based on text embedding reduces it to 8K, and generative HTML pruning reduces it to 4K. In typical RAG scenarios, since the computational cost of HTML pruning is much less than the inference cost of the LLM, we recommend using complete HTML pruning to achieve the best results. Meanwhile, in some resource-limited scenarios where the cost of HTML pruning is also a concern, we suggest using only the basically pruned HTML from the embedding model. Basically pruned HTML also yields performance that meets or surpasses the chunking-based refiner, as we can observe from Figure 5.

5 Conclusion and Future Works

In this work, we propose taking HTML as the format of external knowledge in RAG systems. To tackle the additional tokens brought by HTML, we design HTML cleaning and HTML pruning to shorten HTML while retaining key information. Experiments show that HtmlRAG outperforms existing post-retrieval processes based on plain text, and validates the priority of HTML as the format of retrieved knowledge. Moreover, this work opens up a new research direction and provides a simple and effective solution. We believe as LLMs become more powerful, HTML will be more suitable as the format of external knowledge. We also hope that future works will propose better solutions for processing HTML in RAG systems.

Acknowledgments

Zhicheng Dou is the corresponding author. This work was supported by the National Natural Science Foundation of China No. 62272467, Beijing Natural Science Foundation No. L233008, Beijing Municipal Science and Technology Project No. Z231100010323009, the fund for building world-class universities (disciplines) of Renmin University of China. The work was partially done at the Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE.

References

[1] Marah I Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat S. Behl, Alon Benhaim, Misha Bilenko, Johan Bjorck, Sébastien Bubeck, Martin Cai, Caio César Teodoro Mendes, Weizhu Chen, Vishrav Chaudhary, Parul Chopra, Allie Del Giorno, Gustavo de Rosa, Matthew Dixon, Ronen Eldan, Dan Iter, Amit Garg, Abhishek Goswami, Suriya Gunasekar, Emman Haider, Junheng Hao, Russell J. Hewett, Jamie Huynh, Mojan Javaheripi, Xin Jin, Piero Kauffmann, Nikos Karampatziakis, Dongwoo Kim, Mahoud Khademi, Lev Kurilenko, James R. Lee, Yin Tat Lee, Yuanzhi Li, Chen Liang, Weishung Liu, Eric Lin, Zeqi Lin, Piyush Madan, Arindam Mitra, Hardik Modi, Anh Nguyen, Brandon Norick, Barun Patra, Daniel Perez-Becker, Thomas Portet, Reid Pryzant, Heyang Qin, Marko Radmilac, Corby Rosset, Sambudha Roy, Olatunji Ruwase, Olli Saarikivi, Amin Saied, Adil Salim, Michael Santacroce, Shital Shah, Ning Shang, Hiteshi Sharma, Xia Song, Masahiro Tanaka, Xin Wang, Rachel Ward, Guanhua Wang, Philipp Witte, Michael Wyatt, Can Xu, Jiahang Xu, Sonali Yadav, Fan Yang, Ziyi Yang, Donghan Yu, Chengruidong Zhang, Cyril Zhang, Jianwen Zhang, Li Lyna Zhang, Yi Zhang, Yue Zhang, Yunan Zhang, and Xiren Zhou. 2024. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. *CoRR* abs/2404.14219 (2024). arXiv:2404.14219

[2] AlexVonB, Matthew Dapena-Tretter, and André van Delft. 2024. python-markdownify. https://github.com/matthewwithanm/python-markdownify

[3] Alfonso Amayuelas, Kyle Wong, Liangming Pan, Wenhu Chen, and William Yang Wang. 2024. Knowledge of Knowledge: Exploring Known-Unknowns Uncertainty with Large Language Models. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 6416–6432.

[4] Ryan Aponte, Ryan A. Rossi, Shunan Guo, Jane Hoffswell, Nedim Lipka, Chang Xiao, Gromit Yeuk-Yin Chan, Eunyee Koh, and Nesreen K. Ahmed. 2023. A ML-based Approach for HTML-based Style Recommendation. In *Companion Proceedings of the ACM Web Conference 2023, WWW 2023, Austin, TX, USA, 30 April 2023 - 4 May 2023*, Ying Ding, Jie Tang, Juan F. Sequeda, Lora Aroyo, Carlos Castillo, and Geert-Jan Houben (Eds.). ACM, 9–13.

[5] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2024. Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

[6] Stella Biderman, USVSN Sai Prashanth, Lintang Sutawika, Hailey Schoelkopf, Quentin Anthony, Shivanshu Purohit, and Edward Raff. 2023. Emergent and Predictable Memorization in Large Language Models. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.).

[7] Deyan Ginev Bruce R. Miller, mailto:bruce.miller@nist.gov. 2024. LaTeXML. https://github.com/brucemiller/LaTeXML

[8] Harrison Chase. 2022. *LangChain*. https://github.com/langchain-ai/langchain

[9] Jingye Chen, Tengchao Lv, Lei Cui, Cha Zhang, and Furu Wei. 2022. XDoc: Unified Pre-training for Cross-Format Document Understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, 1006–1016.

[10] Xingyu Chen, Zihan Zhao, Lu Chen, Jiabao Ji, Danyang Zhang, Ao Luo, Yuxuan Xiong, and Kai Yu. 2021. WebSRC: A Dataset for Web-Based Structural Reading Comprehension. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, EMNLP 2021, Virtual Event / Punta Cana, Dominican Republic, 7-11 November, 2021*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, 4173–4185.

[11] Yiruo Cheng, Kelong Mao, and Zhicheng Dou. 2024. Interpreting Conversational Dense Retrieval by Rewriting-Enhanced Inversion of Session Embedding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 2879–2893. https://doi.org/10.18653/V1/2024.ACL-LONG.159

[12] Yiruo Cheng, Kelong Mao, Ziliang Zhao, Guanting Dong, Hongjin Qian, Yongkang Wu, Tetsuya Sakai, Ji-Rong Wen, and Zhicheng Dou. 2024. CORAL: Benchmarking Multi-turn Conversational Retrieval-Augmentation Generation. arXiv:2410.23090 [cs.IR] https://arxiv.org/abs/2410.23090

[13] Guanting Dong, Xiaoshuai Song, Yutao Zhu, Runqi Qiao, Zhicheng Dou, and Ji-Rong Wen. 2024. Toward General Instruction-Following Alignment for Retrieval-Augmented Generation. *arXiv preprint arXiv:2410.09584* (2024).

[14] Zican Dong, Tianyi Tang, Junyi Li, and Wayne Xin Zhao. 2023. A Survey on Long Text Modeling with Transformers. *CoRR* abs/2302.14502 (2023). arXiv:2302.14502

[15] Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. 2024. The Llama 3 Herd of Models. *CoRR* abs/2407.21783 (2024). arXiv:2407.21783

[16] Angela Fan, Yacine Jernite, Ethan Perez, David Grangier, Jason Weston, and Michael Auli. 2019. ELI5: Long Form Question Answering. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 3558–3567.

[17] Henry Gilbert, Michael Sandborn, Douglas C. Schmidt, Jesse Spencer-Smith, and Jules White. 2023. Semantic Compression with Large Language Models. In *Tenth International Conference on Social Networks Analysis, Management and Security, SNAMS 2023, Abu Dhabi, United Arab Emirates, November 21-24, 2023*. IEEE, 1–8.

[18] Dirk Groeneveld, Iz Beltagy, Evan Pete Walsh, Akshita Bhagia, Rodney Kinney, Oyvind Tafjord, Ananya Harsh Jha, Hamish Ivison, Ian Magnusson, Yizhong Wang, Shane Arora, David Atkinson, Russell Authur, Khyathi Raghavi Chandu, Arman Cohan, Jennifer Dumas, Yanai Elazar, Yuling Gu, Jack Hessel, Tushar Khot, William Merrill, Jacob Morrison, Niklas Muennighoff, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Valentina Pyatkin, Abhilasha Ravichander, Dustin Schwenk, Saurabh Shah, Will Smith, Emma Strubell, Nishant Subramani, Mitchell Wortsman, Pradeep Dasigi, Nathan Lambert, Kyle Richardson, Luke Zettlemoyer, Jesse Dodge, Kyle Lo, Luca Soldaini, Noah A. Smith, and Hannaneh Hajishirzi. 2024. OLMo: Accelerating the Science of Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 15789–15809.

[19] Yu Guo, Zhengyi Ma, Jiaxin Mao, Hongjin Qian, Xinyu Zhang, Hao Jiang, Zhao Cao, and Zhicheng Dou. 2022. Webformer: Pre-training with Web Pages for Information Retrieval. In *SIGIR '22: The 45th International ACM SIGIR Conference on Research and Development in Information Retrieval, Madrid, Spain, July 11 - 15, 2022*, Enrique Amigó, Pablo Castells, Julio Gonzalo, Ben Carterette, J. Shane Culpepper, and Gabriella Kazai (Eds.). ACM, 1502–1512.

[20] Izzeddin Gur, Ofir Nachum, Yingjie Miao, Mustafa Safdari, Austin V. Huang, Aakanksha Chowdhery, Sharan Narang, Noah Fiedel, and Aleksandra Faust. 2023. Understanding HTML with Large Language Models. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 2803–2821.

[21] Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Lélio Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7B. *CoRR* abs/2310.06825 (2023). arXiv:2310.06825

[22] Huiqiang Jiang, Qianhui Wu, Xufang Luo, Dongsheng Li, Chin-Yew Lin, Yuqing Yang, and Lili Qiu. 2024. LongLLMLingua: Accelerating and Enhancing LLMs in Long Context Scenarios via Prompt Compression. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 1658–1677.

[23] Zhengbao Jiang, Frank F. Xu, Luyu Gao, Zhiqing Sun, Qian Liu, Jane Dwivedi-Yu, Yiming Yang, Jamie Callan, and Graham Neubig. 2023. Active Retrieval Augmented Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 7969–7992.

[24] Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. 2024. FlashRAG: A Modular Toolkit for Efficient Retrieval-Augmented Generation Research. *CoRR* abs/2405.13576 (2024). arXiv:2405.13576

[25] Jiajie Jin, Yutao Zhu, Xinyu Yang, Chenghao Zhang, and Zhicheng Dou. 2024. FlashRAG: A Modular Toolkit for Efficient Retrieval-Augmented Generation

Research. *CoRR* abs/2405.13576 (2024). https://doi.org/10.48550/ARXIV.2405.13576 arXiv:2405.13576

[26] Jiajie Jin, Yutao Zhu, Yujia Zhou, and Zhicheng Dou. 2024. BIDER: Bridging Knowledge Inconsistency for Efficient Retrieval-Augmented LLMs via Key Supporting Evidence. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 750–761.

[27] JinaAI. 2024. Reader-LM: Small Language Models for Cleaning and Converting HTML to Markdown. https://jina.ai/news/reader-lm-small-language-models-for-cleaning-and-converting-html-to-markdown/. [Online; accessed 2024-10-05].

[28] Mandar Joshi, Eunsol Choi, Daniel S. Weld, and Luke Zettlemoyer. 2017. TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*, Regina Barzilay and Min-Yen Kan (Eds.). Association for Computational Linguistics, 1601–1611.

[29] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick S. H. Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, 6769–6781.

[30] Geunwoo Kim, Pierre Baldi, and Stephen McAleer. 2023. Language Models can Solve Computer Tasks. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*, Alice Oh, Tristan Naumann, Amir Globerson, Kate Saenko, Moritz Hardt, and Sergey Levine (Eds.).

[31] Gangwoo Kim, Sungdong Kim, Byeongguk Jeon, Joonsuk Park, and Jaewoo Kang. 2023. Tree of Clarifications: Answering Ambiguous Questions with Retrieval-Augmented Large Language Models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 996–1009.

[32] Suhas Kotha, Jacob Mitchell Springer, and Aditi Raghunathan. 2024. Understanding Catastrophic Forgetting in Language Models via Implicit Inference. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

[33] Tom Kwiatkowski, Jennimaria Palomaki, Olivia Redfield, Michael Collins, Ankur P. Parikh, Chris Alberti, Danielle Epstein, Illia Polosukhin, Jacob Devlin, Kenton Lee, Kristina Toutanova, Llion Jones, Matthew Kelcey, Ming-Wei Chang, Andrew M. Dai, Jakob Uszkoreit, Quoc Le, and Slav Petrov. 2019. Natural Questions: a Benchmark for Question Answering Research. *Trans. Assoc. Comput. Linguistics* 7 (2019), 452–466.

[34] Hanyu Lai, Xiao Liu, Iat Long Iong, Shuntian Yao, Yuxuan Chen, Pengbo Shen, Hao Yu, Hanchen Zhang, Xiaohan Zhang, Yuxiao Dong, and Jie Tang. 2024. AutoWebGLM: Bootstrap And Reinforce A Large Language Model-based Web Navigating Agent. *CoRR* abs/2404.03648 (2024). arXiv:2404.03648

[35] Chenliang Li, Bin Bi, Ming Yan, Wei Wang, Songfang Huang, Fei Huang, and Luo Si. 2021. StructuralLM: Structural Pre-training for Form Understanding. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, 6309–6318.

[36] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. 2025. Search-o1: Agentic Search-Enhanced Large Reasoning Models. *arXiv preprint arXiv:2501.05366* (2025).

[37] Xiaoxi Li, Zhicheng Dou, Yujia Zhou, and Fangchao Liu. 2024. CorpusLM: Towards a Unified Language Model on Corpus for Knowledge-Intensive Tasks. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2024, Washington DC, USA, July 14-18, 2024*, Grace Hui Yang, Hongning Wang, Sam Han, Claudia Hauff, Guido Zuccon, and Yi Zhang (Eds.). ACM, 26–37.

[38] Xiaoxi Li, Jiajie Jin, Yujia Zhou, Yongkang Wu, Zhonghua Li, Qi Ye, and Zhicheng Dou. 2024. RetroLLM: Empowering Large Language Models to Retrieve Fine-grained Evidence within Generation. *CoRR* abs/2412.11919 (2024). https://doi.org/10.48550/ARXIV.2412.11919 arXiv:2412.11919

[39] Xiaoxi Li, Jiajie Jin, Yujia Zhou, Yuyao Zhang, Peitian Zhang, Yutao Zhu, and Zhicheng Dou. 2024. From Matching to Generation: A Survey on Generative Information Retrieval. *CoRR* abs/2404.14851 (2024). https://doi.org/10.48550/ARXIV.2404.14851 arXiv:2404.14851

[40] Xiaoxi Li, Yujia Zhou, and Zhicheng Dou. 2024. UniGen: A Unified Generative Framework for Retrieval and Question Answering with Large Language Models. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (Eds.). AAAI Press, 8688–8696. https://doi.org/10.1609/AAAI.V38I8.28714

[41] Yucheng Li. 2023. Unlocking Context Constraints of LLMs: Enhancing Context Efficiency of LLMs with Self-Information-Based Content Filtering. *CoRR* abs/2304.12102 (2023). arXiv:2304.12102

[42] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81.

[43] Jerry Liu. 2022. *LlamaIndex*. https://github.com/jerryjliu/llama_index

[44] Junpeng Liu, Yifan Song, Bill Yuchen Lin, Wai Lam, Graham Neubig, Yuanzhi Li, and Xiang Yue. 2024. VisualWebBench: How Far Have Multimodal LLMs Evolved in Web Page Understanding and Grounding? *CoRR* abs/2404.05955 (2024). arXiv:2404.05955

[45] Yang Liu. 2019. Fine-tune BERT for Extractive Summarization. *CoRR* abs/1903.10318 (2019). arXiv:1903.10318

[46] Alex Mallen, Akari Asai, Victor Zhong, Rajarshi Das, Daniel Khashabi, and Hannaneh Hajishirzi. 2023. When Not to Trust Language Models: Investigating Effectiveness of Parametric and Non-Parametric Memories. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, Anna Rogers, Jordan L. Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, 9802–9822.

[47] Sewon Min, Kalpesh Krishna, Xinxi Lyu, Mike Lewis, Wen-tau Yih, Pang Wei Koh, Mohit Iyyer, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2023. FActScore: Fine-grained Atomic Evaluation of Factual Precision in Long Form Text Generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing, EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 12076–12100.

[48] Fengran Mo, Kelong Mao, Ziliang Zhao, Hongjin Qian, Haonan Chen, Yiruo Cheng, Xiaoxi Li, Yutao Zhu, Zhicheng Dou, and Jian-Yun Nie. 2024. A Survey of Conversational Search. arXiv:2410.15576 [cs.CL] https://arxiv.org/abs/2410.15576

[49] Joel Ruben Antony Moniz, Soundarya Krishnan, Melis Özyildirim, Prathamesh Saraf, Halim Cagri Ates, Yuan Zhang, and Hong Yu. 2024. ReALM: Reference Resolution as Language Modeling. In *Proceedings of the 25th Annual Meeting of the Special Interest Group on Discourse and Dialogue, SIGDIAL 2024, Kyoto, Japan, September 18 - 20, 2024*, Tatsuya Kawahara, Vera Demberg, Stefan Ultes, Koji Inoue, Shikib Mehri, David M. Howcroft, and Kazunori Komatani (Eds.). Association for Computational Linguistics, 51–65.

[50] Shiyu Ni, Keping Bi, Jiafeng Guo, and Xueqi Cheng. 2024. When Do LLMs Need Retrieval Augmentation? Mitigating LLMs' Overconfidence Helps Retrieval Augmentation. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 11375–11388.

[51] OpenAI. 2023. GPT-4 Technical Report. *CoRR* abs/2303.08774 (2023). arXiv:2303.08774

[52] OpenAI. 2024. SearchGPT Prototype. [Online; accessed 2024-10-14].

[53] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F. Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*, Sanmi Koyejo, S. Mohamed, A. Agarwal, Danielle Belgrave, K. Cho, and A. Oh (Eds.).

[54] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a Method for Automatic Evaluation of Machine Translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics, July 6-12, 2002, Philadelphia, PA, USA*. ACL, 311–318.

[55] Ajay Patel, Bryan Li, Mohammad Sadegh Rasooli, Noah Constant, Colin Raffel, and Chris Callison-Burch. 2023. Bidirectional Language Models Are Also Few-shot Learners. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

[56] PerplexityAI. 2024. Perplexity.

[57] Hongjin Qian, Zheng Liu, Kelong Mao, Yujia Zhou, and Zhicheng Dou. 2024. Grounding Language Model with Chunking-Free In-Context Retrieval. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 1298–1311.

[58] Yulei Qin, Yuncheng Yang, Pengcheng Guo, Gang Li, Hang Shao, Yuchen Shi, Zihan Xu, Yun Gu, Ke Li, and Xing Sun. 2024. Unleashing the Power of Data Tsunami: A Comprehensive Survey on Data Assessment and Selection for Instruction Tuning of Language Models. *CoRR* abs/2408.02085 (2024). arXiv:2408.02085

[59] Leonard Richardson. 2024. Beautiful Soup.

[60] Stephen E. Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (2009), 333–389.

[61] Zhihong Shao, Yeyun Gong, Yelong Shen, Minlie Huang, Nan Duan, and Weizhu Chen. 2023. Enhancing Retrieval-Augmented Large Language Models with Iterative Retrieval-Generation Synergy. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 9248–9274.

[62] Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Richard James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. 2024. REPLUG: Retrieval-Augmented Black-Box Language Models. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers), NAACL 2024, Mexico City, Mexico, June 16-21, 2024*, Kevin Duh, Helena Gómez-Adorno, and Steven Bethard (Eds.). Association for Computational Linguistics, 8371–8384.

[63] Ivan Stelmakh, Yi Luan, Bhuwan Dhingra, and Ming-Wei Chang. 2022. ASQA: Factoid Questions Meet Long-Form Answers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing, EMNLP 2022, Abu Dhabi, United Arab Emirates, December 7-11, 2022*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, 8273–8288.

[64] Jiejun Tan, Zhicheng Dou, Yutao Zhu, Peidong Guo, Kun Fang, and Ji-Rong Wen. 2024. Small Models, Big Insights: Leveraging Slim Proxy Models To Decide When and What to Retrieve for LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 4420–4436.

[65] Harsh Trivedi, Niranjan Balasubramanian, Tushar Khot, and Ashish Sabharwal. 2022. MuSiQue: Multihop Questions via Single-hop Question Composition. *Trans. Assoc. Comput. Linguistics* 10 (2022), 539–554.

[66] Shin-Rong Tsai, Hsi-Yu Schive, and Matthew Turk. 2024. Libyt: A Tool for Parallel In Situ Analysis with yt, Python, and Jupyter. In *Proceedings of the Platform for Advanced Scientific Computing Conference, PASC 2024, Zurich, Switzerland, June 3-5, 2024*, Katherine Evans and Olaf Schenk (Eds.). ACM, 25:1–25:10.

[67] W3Schools. 2024. What is the HTML DOM? [Online; accessed 2024-10-14].

[68] Haochen Wang, Kai Hu, Haoyu Dong, and Liangcai Gao. 2024. DocTabQA: Answering Questions from Long Documents Using Tables. In *Document Analysis and Recognition - ICDAR 2024 - 18th International Conference, Athens, Greece, August 30 - September 4, 2024, Proceedings, Part I (Lecture Notes in Computer Science, Vol. 14804)*, Elisa H. Barney Smith, Marcus Liwicki, and Liangrui Peng (Eds.). Springer, 470–487.

[69] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024. Improving Text Embeddings with Large Language Models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 11897–11916.

[70] Lucy Lu Wang, Jonathan Bragg, and Daniel S. Weld. 2023. Paper to HTML: A Publicly Available Web Tool for Converting Scientific Pdfs into Accessible HTML. *SIGACCESS Access. Comput.* 134, Article 1 (Jan. 2023), 1 pages.

[71] Qifan Wang, Yi Fang, Anirudh Ravula, Fuli Feng, Xiaojun Quan, and Dongfang Liu. 2022. WebFormer: The Web-page Transformer for Structure Information Extraction. In *WWW '22: The ACM Web Conference 2022, Virtual Event, Lyon, France, April 25 - 29, 2022*, Frédérique Laforest, Raphaël Troncy, Elena Simperl, Deepak Agarwal, Aristides Gionis, Ivan Herman, and Lionel Médini (Eds.). ACM, 3124–3133.

[72] Shuting Wang, Xin Yu, Mang Wang, Weipeng Chen, Yutao Zhu, and Zhicheng Dou. 2024. RichRAG: Crafting Rich Responses for Multi-faceted Queries in Retrieval-Augmented Generation. *CoRR* abs/2406.12566 (2024). arXiv:2406.12566

[73] Michael Williamson, Jonathan Lehman, and Jacob Wang. 2024. mammoth.js. https://github.com/mwilliamson/mammoth.js

[74] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-Pack: Packaged Resources To Advance General Chinese Embedding. *CoRR* abs/2309.07597 (2023). arXiv:2309.07597

[75] Fangyuan Xu, Weijia Shi, and Eunsol Choi. 2024. RECOMP: Improving Retrieval-Augmented LMs with Context Compression and Selective Augmentation. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

[76] Zhilin Yang, Peng Qi, Saizheng Zhang, Yoshua Bengio, William W. Cohen, Ruslan Salakhutdinov, and Christopher D. Manning. 2018. HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, 2369–2380.

[77] Huaying Yuan, Zhicheng Dou, Yujia Zhou, Yu Guo, and Ji-Rong Wen. 2023. VILE: Block-Aware Visual Enhanced Document Retrieval. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management, CIKM 2023, Birmingham, United Kingdom, October 21-25, 2023*, Ingo Frommholz, Frank Hopfgartner, Mark Lee, Michael Oakes, Mounia Lalmas, Min Zhang, and Rodrygo L. T. Santos (Eds.). ACM, 3104–3113.

[78] Aohan Zeng, Bin Xu, Bowen Wang, Chenhui Zhang, Da Yin, Diego Rojas, Guanyu Feng, Hanlin Zhao, Hanyu Lai, Hao Yu, Hongning Wang, Jiadai Sun, Jiajie Zhang, Jiale Cheng, Jiayi Gui, Jie Tang, Jing Zhang, Juanzi Li, Lei Zhao, Lindong Wu, Lucen Zhong, Mingdao Liu, Minlie Huang, Peng Zhang, Qinkai Zheng, Rui Lu, Shuaiqi Duan, Shudan Zhang, Shulin Cao, Shuxun Yang, Weng Lam Tam, Wenyi Zhao, Xiao Liu, Xiao Xia, Xiaohan Zhang, Xiaotao Gu, Xin Lv, Xinghan Liu, Xinyi Liu, Xinyue Yang, Xixuan Song, Xunkai Zhang, Yifan An, Yifan Xu, Yilin Niu, Yuantao Yang, Yueyan Li, Yushi Bai, Yuxiao Dong, Zehan Qi, Zhaoyu Wang, Zhen Yang, Zhengxiao Du, Zhenyu Hou, and Zihan Wang. 2024. ChatGLM: A Family of Large Language Models from GLM-130B to GLM-4 All Tools. *CoRR* abs/2406.12793 (2024). arXiv:2406.12793

[79] Haopeng Zhang, Xiao Liu, and Jiawei Zhang. 2023. Extractive Summarization via ChatGPT for Faithful Summary Generation. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 3270–3278.

[80] Haopeng Zhang, Xiao Liu, and Jiawei Zhang. 2023. SummIt: Iterative Text Summarization via ChatGPT. In *Findings of the Association for Computational Linguistics: EMNLP 2023, Singapore, December 6-10, 2023*, Houda Bouamor, Juan Pino, and Kalika Bali (Eds.). Association for Computational Linguistics, 10644–10657.

[81] Xinrong Zhang, Yingfa Chen, Shengding Hu, Zihang Xu, Junhao Chen, Moo Khai Hao, Xu Han, Zhen Leng Thai, Shuo Wang, Zhiyuan Liu, and Maosong Sun. 2024. ınftyBench: Extending Long Context Evaluation Beyond 100K Tokens. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 15262–15277.

[82] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. 2024. GPT-4V(ision) is a Generalist Web Agent, if Grounded. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net.

[83] Lexin Zhou, Wout Schellaert, Fernando Martínez-Plumed, Yael Moros-Daval, Cèsar Ferri, and José Hernández-Orallo. 2024. Larger and more instructable language models become less reliable. *Nature* (2024), 1–8.

[84] Yujia Zhou, Yan Liu, Xiaoxi Li, Jiajie Jin, Hongjin Qian, Zheng Liu, Chaozhuo Li, Zhicheng Dou, Tsung-Yi Ho, and Philip S. Yu. 2024. Trustworthiness in Retrieval-Augmented Generation Systems: A Survey. arXiv:2409.10102 [cs.IR] https://arxiv.org/abs/2409.10102

[85] Yujia Zhou, Zheng Liu, Jiajie Jin, Jian-Yun Nie, and Zhicheng Dou. 2024. Metacognitive Retrieval-Augmented Large Language Models. In *Proceedings of the ACM on Web Conference 2024, WWW 2024, Singapore, May 13-17, 2024*, Tat-Seng Chua, Chong-Wah Ngo, Ravi Kumar, Hady W. Lauw, and Roy Ka-Wei Lee (Eds.). ACM, 1453–1463.

[86] Yutao Zhu, Peitian Zhang, Chenghao Zhang, Yifei Chen, Binyu Xie, Zheng Liu, Ji-Rong Wen, and Zhicheng Dou. 2024. INTERS: Unlocking the Power of Large Language Models in Search with Instruction Tuning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, 2782–2809.

Appendix

A Generalization to Rich Text Formats

The HTML cleaning and pruning process is worthwhile as long as the knowledge source is in HTML format or in other rich formats like PDF. Currently, major RAG frameworks like LangChain and LlamaIndex share the following workflow: "Retrieve HTML -> Convert to Plain Text -> Refine -> Generate Answer". We argue that the upper bound of the workflow above is limited because lots of information is lost during the early HTML to plain text conversion. Our proposed workflow goes like this: "Retrieve HTML -> Clean and Prune -> Convert to Other Formats (Optional) -> Generate Answer". Even if the LLM prefers other input formats or you'd like to save tokens, the conversion from HTML to other formats is optional and suggested to be done after HTML cleaning and pruning. This workflow has a higher upper bound because pruning is carried out before the information loss of format conversion.

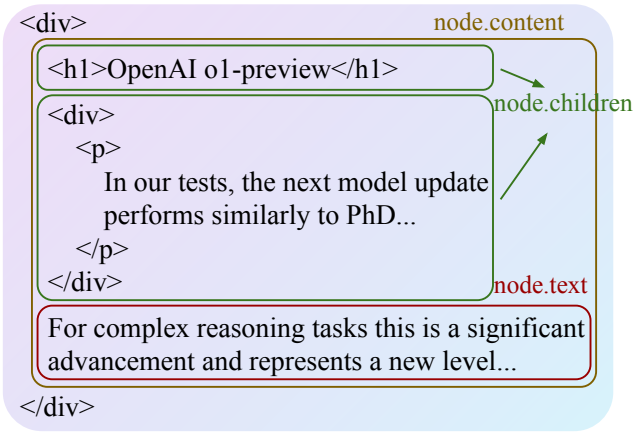

Figure 6: Node content explained

B Generative Model Training Details

Here we introduce several critical hyper-parameters that define the training process of the generative model. The model's max training context window is set to 35000 tokens. The model is trained for 3 epochs. The training is conducted on 4 computing nodes, with 32 Nvidia A800 GPUs, each having 80G memory. To manage memory usage and computational efficiency, *per_device_train_batch_size* is set to 1, while *gradient_accumulation_steps* is set to 8, effectively simulating a larger batch size during backpropagation.

For parallelism, *seq_parallel_size* is set to 8, indicating that the model will distribute its computations across 8 devices if available. The *learning_rate* is set to 2e-5, striking a balance between rapid convergence and avoiding divergence. The learning rate scheduler (*lr_scheduler_type*) is set to 'constant', meaning the learning rate remains unchanged throughout the training unless manually adjusted. For optimization, the Adam optimizer parameters (*adam_beta1*, *adam_beta2*, and *adam_epsilon*) are chosen as 0.9, 0.98, and 1e-8 respectively, to ensure stable gradient updates. The *max_grad_norm* is set to 1.0 to prevent exploding gradients by clipping them if they exceed this norm. A weight decay (*weight_decay*) of 1e-4 is used to regularize the model and prevent overfitting. A *warmup_ratio* of 0.01 indicates that the learning rate will be gradually increased during the initial 1% of the training process before settling at the base learning rate. *gradient_checkpointing* is enabled to save memory at the cost of increased computation time.

DeepSpeed is configured for efficient distributed training. For ZeRO optimization (*zero_optimization*), stage 3 is selected, which represents the highest level of parameter partitioning and offloading. Gradient clipping (*gradient_clipping*) is set to 1.0, ensuring that the gradients do not grow too large, thus preventing potential issues like exploding gradients. The *wall_clock_breakdown* option is set to false, indicating that DeepSpeed will not provide a detailed breakdown of the training time spent on different components of the training loop, which can be useful for profiling but may add some overhead. Mixed precision training using bfloat16 is set to "auto", indicating that DeepSpeed will decide whether to use bfloat16 based on the capabilities of the system and the requirements of the model.

Table 7: Performance Comparison Between the Phi-3.8B-Pruner and Llama-1.5B Pruner.

Size	ASQA	HotpotQA	NQ	TriviaQA	MuSiQue	ELI5
	EM	EM	EM	EM	EM	ROUGE
3.8B	68.50	46.25	60.50	93.50	13.25	16.33
1.5B	66.50	45.00	60.75	93.00	10.00	16.25

C Analysis on Information Loss

We evaluate the information loss during pruning by the reference text's exact match scores, which are shown in Table 5. We list the score for reference text in some critical steps or baselines. Plain Text (128k), Markdown (128k), and HtmlRAG w/o Prune (128k) are long-context reference after rule-base cleaning (refer to Table 2 for end-to-end results). They are three lossless compression techniques. However, due to the 128k length limit, a small amount of information loss may still occur when the context is particularly long. Plain text retains the most information at the cost of maximally removing format - related tokens. In contrast, HTML has more format - related tokens, so it relatively loses more information. BM25 (4k), BGE (4k), E5-Mistral (4k), LongLLMLingua (4k), and JinaAI Reader (4k) are baselines (refer to Table 1 for end-to-end results). HtmlRAG (8k) is the coarsely pruned result using the embedding model and HtmlRAG (4k) is the final pruned result using both the embedding model and the generative model. In the final 4k results, HtmlRAG retains the most information, followed by the method that uses the HtmlRAG Prune approach combined with a high - performance embedding model.

D Analysis on the Pruners' Scaling

We have conducted very preliminary experiments on the scaling of the generative pruner. We compared the fine-tuned model of the Phi-3.5-mini-instruct (3.8B) and Llama-3.2-1B-Instruct (1.5B) using the same HTML pruning training data, and we get HTML-Pruner-Phi-3.8B and HTML-Pruner-Llama-1B. We found that the Phi-3.8B-Pruner won by a narrow margin.

Table 5: The information loss during pruning measured by the reference text's exact match (EM) scores. The information loss of baselines are also demonstrated for comparison.

Length	128k			8k	4k					
Dataset	Plain Text	Markdown	HtmlRAG w/o Prune	HtmlRAG w/o Gen	BM25	BGE	E5 - Mistral	LongLLMLingua	JinaAI Reader	HtmlRAG
ASQA	70.90	67.72	69.49	59.72	25.99	52.17	50.65	35.77	38.39	52.80
HQA	70.00	65.75	66.75	54.75	37.50	48.75	49.00	43.75	38.25	51.00

Table 6: Complementary results of HtmlRAG without pruning and baselines under Llama-3.1-8B-Instruct-128K. Hit@1 is the proportion of instances where at least one short answer matches. The best and second best results are in bold and underlined. The symbol † signifies that our method achieves superior results among baselines in a statistically significant manner (t-test, p-value < 0.05).

Method	**ASQA**		**Hotpot-QA**	**NQ**		**Trivia-QA**		**MuSiQue**	**ELI5**	
	Hit@1	EM	EM	Hit@1	EM	Hit@1	EM	EM	ROUGE-L	BLEU
Vanilla HTML	47.75	20.08	28.75	47.25	36.09	85.00	24.85	6.00	**16.13**	<u>6.28</u>
Plain Text	<u>61.50</u>	**27.82**	<u>39.25</u>	**59.25**	**44.31**	**94.00**	**28.23**	<u>7.75</u>	16.02	**6.35**
Markdown	**61.75**	<u>26.70</u>	37.50	57.50	42.85	91.50	26.67	7.50	<u>16.12</u>	5.91
HtmlRAG w/o Prune	61.00	<u>26.70</u>†	**39.50**†	<u>59.00</u>†	<u>43.46</u>†	<u>92.00</u>†	<u>27.50</u>†	**8.75**†	15.62	5.87

E Analysis on HTML-Cleaning

Following the settings in Table 2, we conduct complementary experiments with Llama-3.1-8B-Instruct, as shown in Table 6. On most datasets, a more capable LLM (70B) performs better than a less capable one (8B), while on several datasets, the 8B model counterintuitively performs better. We studied some counterintuitive cases, which show that 70B model gets distracted by those noise. We think this demonstrates the necessity to do HTML pruning.

F Key Algorithms

In this appendix section, we present all the algorithms mentioned in the main text using pseudo code, including the algorithm for constructing the block tree, the pruning algorithm using the embedding model, and the pruning algorithm using the generative model.

To make it clear, we first define elements under a certain node as follows: All sorts of elements under the node are referred to as *node.content*; Text wrapped by child tags is referred to as *node.children*; Text directly attached to the node is referred to as *node.text*. We show an example accordingly in Figure 6. To discriminate between children with the same HTML tag, we append a number to the end of the original tag name. For example, two children with the same " " tag are renamed as "<div1>" and "<div2>".

The block tree construction algorithm is demonstrated in Algorithm 1, which transforms a DOM Tree T into a Block Tree T'. In the block tree, a block is the smallest unit that be pruned in subsequent steps. We use a breadth-first algorithm to traverse all nodes in the DOM tree. Leaf nodes that are visited are directly considered as blocks. If the total number of tokens of all content under a node is less than the number we set (*maxWordss*), we merge all the content of the node and consider it as a block. Otherwise, we check the content of the node. The node's children are to be visited in subsequent steps. The node's text will be considered as a block. It is noteworthy that if there are only children but no text under the node, it will not be considered as a block. This algorithm merges fragmented nodes as a block, until the number of tokens exceeds *maxWords*.

Algorithm 1 Construct Block Tree T' from DOM Tree T

```
procedure ConstructBlockTree(T)
    Declare a queue nodeQueue
    R ← root node of T
    Enqueue R into nodeQueue
    while nodeQueue is not empty do
        node ← Dequeue from nodeQueue
        if node is a leaf node then
            node.block ← node.content
            node.isLeaf ← True
        else
            if node.content < maxTokens then
                Merge descendant nodes of node
                node.block ← node.content
                node.isLeaf ← True
            else
                Expand children of node
                for each child of node do
                    Enqueue child into nodeQueue
                end for
                if node.text is not empty then
                    node.block ← node.text
                    node.isLeaf ← False
                end if
            end if
        end if
    end while
    return T
end procedure
```

Algorithm 3 Token Probability Calculation

procedure TraverseTokenTree
  Declare a queue *nodeStack*
  $t_1 \leftarrow$ root node of T
  $t_1.score \leftarrow 1.0$ ▷ Set the score of t_1 as 1.0
  Push t_1 into *nodeStack*
  while *nodeStack* is not empty **do**
    $t_{n-1} \leftarrow$ Pop from *nodeStack*
    $children \leftarrow$ Expand children of node $p : (t_n^1, t_n^2, \cdots, t_n^K)$
    if $K = 0$ (p is a leaf node) **then**
      continue
    else if $K = 1$ **then**
      $t_n^1.score \leftarrow 1.0$
      Push the singleton child t_n^0 into *nodeStack*
    else
      $prefix \leftarrow \{input, t_1, \ldots, t_{n-1}\}$
      for each t_n^k in *children* **do**
        $t_n^k.score \leftarrow \frac{exp(\text{Logits}(t_n^k))}{\sum_{i=1}^{K} exp(\text{Logits}(t_n^i))}$
        Push t_n^k into *nodeStack*
      end for
    end if
  end while
end procedure

Another key algorithm is greedy block pruning, as demonstrated in Algorithm 2. We greedily delete the block with the lowest score until the length of the HTML document meets the context window we set. To elaborate, when deleting a block, if the block is a leaf node, we delete the block directly. Otherwise, if the block consists of directly attached text under a parent node, we delete only those text. After a block is deleted, the algorithm recursively checks if the parent node is empty. If the parent node is empty, it is to be deleted.

Algorithm 2 Greedy Block Pruning

procedure GreedyBlockTreePruning(**T**)
  nodes ← all nodes with blocks from **T**
  for each *node* in *nodes* **do**
    $node.score \leftarrow Rel(q, node.block)$ ▷ calculate semantic similarity between node *node* and user request
  end for
  Sort *nodes* by key *node.score* in ascending order
  while each *node* in *nodes* **do**
    node ← the node with the lowest score
    if $node.isLeaf$ **then**
      $parent \leftarrow node.parent$
      delete *node*
      while $parent.content$ is empty **do**
        $parent \leftarrow parent.parent$
        delete *parent*
      end while
    else
      delete $node.text$
    end if
  end while
end procedure

The last key algorithm is token probability calculation, as demonstrated in Algorithm 3. We use a depth-firth algorithm to traverse tokens in the token tree so that tokens visited sequentially share the longest prefix sequences. The probability of the root token and singleton child tokens are directly set to 1.0, and does not require calculation.