INFINI-GRAM MINI: Exact n-gram Search at the Internet Scale with FM-Index

Hao Xu[♡] **Jiacheng Liu**[♡♠] **Yejin Choi**[♢] **Noah A. Smith**[♡♠] **Hannaneh Hajishirzi**[♡♠]
[♡]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♠]Allen Institute for AI [♢]Stanford University

Abstract

Language models are trained mainly on massive text data from the Internet, and it becomes increasingly important to understand this data source. Exact-match search engines enable searching in large text corpora – counting string appearances and retrieving the enclosing documents – yet the high storage overhead hinders their application on Internet-scale data. We present **INFINI-GRAM MINI**, an efficient and scalable system that can make petabyte-level text corpora searchable. Based on the FM-index data structure (Ferragina and Manzini, 2000), which simultaneously indexes and compresses text, our system creates indexes with size only 44% of the corpus. INFINI-GRAM MINI greatly improves upon the best existing implementation of FM-index in terms of indexing speed (18×) and memory use during both indexing (3.2× reduction) and querying (down to a negligible amount). We index 83TB of Internet text in 99 days with a single CPU node with 128 vCPUs (or 19 hours if using 137 such nodes). We show one important use case of INFINI-GRAM MINI in a large-scale analysis of benchmark contamination. We find several core LM evaluation benchmarks to be heavily contaminated in Internet crawls (up to 74.2% in GSM8K), which could lead to overestimating the capabilities of language models if trained on such data. We host a benchmark contamination bulletin to share the contamination rate of many core and community-contributed benchmarks. We also release a web interface and an API endpoint to serve general search queries on INFINI-GRAM MINI indexes.

🏠	**Project Home**	`infini-gram-mini.io`
🤗	**Web Interface**	`infini-gram-mini.io/demo`
aws	**API Endpoint**	`api.infini-gram-mini.io`
🌐	**Documentation**	`infini-gram-mini.io/docs`
	Source Code	`infini-gram-mini.io/code`
🤗	**Contam Bulletin**	`infini-gram-mini.io/bulletin`

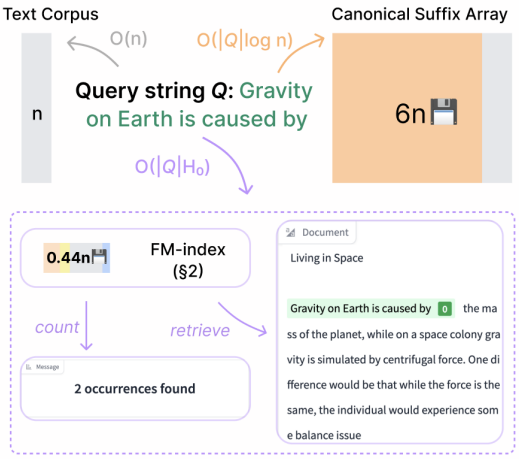


Figure 1: Overview of INFINI-GRAM MINI. Based on the FM-index data structure, INFINI-GRAM MINI supports efficient exact-match search in massive text corpora ($n \simeq 10^{15}$ bytes) while reducing the index size down to 7% compared to a canonical suffix array index. Searching naively in the corpus would have time complexity of $O(n)$ and is thus impractical; with INFINI-GRAM MINI, the search time complexity is independent of n. $|Q|$ is the length of query string and can be arbitrarily long, and $H_0 \approx 2.1$ is the zeroth-order entropy of the text corpus.

1 Introduction

Modern language models (LMs) are trained mainly on text datasets downsampled from massive, petabyte-level text corpora like Common Crawl (Common Crawl Foundation, 2025). As these LMs are deployed more broadly, it becomes more pressing to understand the training data and its effects on model behavior (Liu et al., 2025; Han and Tsvetkov, 2022). As a starting point, we want to make these text corpora *searchable*; in particular, searching for *exact* matches of *long* sequences has gained increasing interest (Elazar et al., 2024; Merrill et al., 2024; Lu et al., 2024). The size of these corpora makes this problem extremely challenging, creat-

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 24944–24969
November 4-9, 2025 ©2025 Association for Computational Linguistics

ing demand for more efficient indexing techniques.

Prior systems for exact-match search build indexes several times as large as the text corpora. Merrill et al. (2024) index 1.3TB of text with a suffix automaton (storage multiplier 29×); Liu et al. (2024) index 12TB of text with a suffix array (storage multiplier 6×); Elazar et al. (2024) index 35TB of text with the proprietary ElasticSearch engine (storage multiplier about 2×). The size of these indexes renders it impractical to apply them to petabyte-level corpora.

To address this challenge, we index text corpora with **FM-index** (Ferragina and Manzini, 2000), a compact data structure frequently used in bioinformatics (Guo, 2025; Depuydt et al., 2023; Wang et al., 2018; Li, 2014), but not yet used for natural language data at scale. Prior work only applies this data structure to 13.4 GB dataset (Bevilacqua et al., 2022). We explain FM-index in detail in §2. For natural text, the size of FM-index can be made as small as 0.26× of the corpus; practically, to ensure low query latency, we build FM-index with a storage multiplier of **0.44**×, or 7% compared to the canonical suffix array.

INFINI-GRAM MINI is the system we developed for efficiently constructing the FM-index at scale and answering search queries. For indexing, we extend and combine components from Liu et al. (2024), Gog et al. (2014), and Labeit et al. (2017) into a highly parallelized program, which achieves a **18**× speedup and uses only **32%** as much RAM compared to the best existing implementation by SDSL (Gog et al., 2014).[1] We use INFINI-GRAM MINI to construct FM-index for two LM pretraining datasets – the Pile and DCLM-baseline – and 7 recent Common Crawl dumps from January to July 2025. Altogether, we indexed **83TB** of text in 99 days with a single CPU node with 128 vCPUs, and this could have been done in 19 hours if embarrassingly parallelized across 137 such nodes. We estimate that indexing the full Common Crawl would take 1,200 node-days, or 19 hours if parallelized across 1,500 nodes. For answering queries, we extend SDSL to work with on-disk index, reducing the RAM requirement to a negligible amount. Our inference engine supports counting the occurrences of a query string and retrieving documents that contain the query string, both within seconds when working on the above corpora.

We apply INFINI-GRAM MINI to **analyzing contamination of benchmarks** widely used in state-of-art LM evaluations (§4). INFINI-GRAM MINI allows us to do the analysis on larger corpora than prior works and on new benchmarks uploaded. This would be much more expensive if using other indexing methods. We find several core benchmarks to be heavily contaminated (§4.2). INFINI-GRAM MINI detects exact overlap for question of contaminated entries in text corpora, among which a large majority of questions appear together with the correct answer. **This reveals a dire evaluation crisis:** as benchmarks get increasingly contaminated by Internet crawls and consequently LM training data, evaluation results may give inflated estimates of true model capabilities. As such, we host a benchmark contamination bulletin to continually monitor contamination of core and community-contributed benchmarks on new Internet snapshots, and we call for more community attention on this matter.

Beyond contamination analysis, INFINI-GRAM MINI opens up more impactful use cases such as task-specific dataset construction and pretraining data curation. We release a **web interface** and **API endpoint** of INFINI-GRAM MINI, so that everyone can search in the text corpora that we have indexed. We plan to continue indexing new corpora and share regular updates. We also release our source code. We hope this tool can enable more insightful analysis and better use of massive text corpora.

2 Background: FM-index

The FM-index (Ferragina and Manzini, 2000) is a full-text index that supports efficient pattern matching, counting, and text retrieval on a highly compressed representation of the text corpus. Compared with a canonical suffix array, FM-index stores a compact variation of a suffix array and the text corpus, greatly reducing storage overhead.

2.1 Data Structures and Implementation

On a high level, FM-index has two core components: (1) a sampled suffix array and its inverse, and (2) the Burrows-Wheeler transform represented using a Huffman-shaped wavelet tree. Below we describe each component as applied to a single string (appropriate for our application; see §3). Figure 2 shows a toy example illustrating the data structure.

Suffix array and sampling. The suffix array SA of a string T of length n is an array of integers

[1] https://github.com/simongog/sdsl-lite

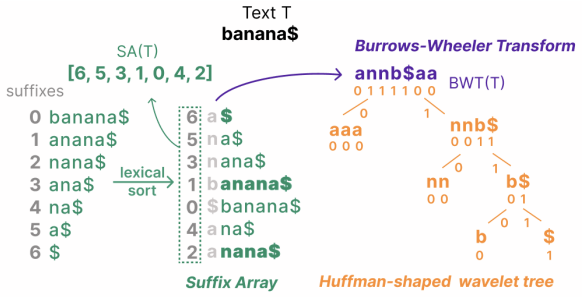


Figure 2: The FM-index data structure (§2.1) used in INFINI-GRAM MINI, shown for a toy string with length $n = 7$. The suffix array is sampled with a sampling rate $a = 3$ and only elements corresponding to **bolded** suffixes are stored. The BWT can be derived from the SA, and is stored in compressed form as a Huffman-shaped wavelet tree.

representing the starting positions of all suffixes of T in lexicographic order. Formally, $SA[i] = j$ if the suffix $T[j...n]$ ranks ith among all suffixes in lexicographical ordering. The suffix array enables quickly locating the positions where a pattern appears in the string. Canonically, storing the whole suffix array would take $O(n \log n)$ space, or $5n$ bytes for strings with length up to one trillion (Liu et al., 2024). To reduce storage overhead, FM-index samples SA at regular intervals, storing only every ath entry where a is a parameter. When querying the index, if we need to access an unsampled SA entry, we can recover its value by referencing the BWT data structure, which is introduced below.

Burrows-Wheeler transform (BWT). BWT (Burrows et al., 1994) rearranges the string T (with termination symbol \$) into a reversible permutation L that clusters repeated symbols. The BWT is defined as a string L where $L[i] = T[SA[i] - 1]$ if $SA[i] > 0$, or $L[i] =$ \$ otherwise. Intuitively, L is the concatenation of the symbol preceding each suffix when the suffixes are sorted in the SA order (see Figure 2).

A key property of BWT is the Last-to-First (LF) mapping, which maps the ith occurrence of a symbol c in L to the ith occurrence of it in F, where $F[i] = T[SA[i]]$. The LF mapping is defined as $LF(i) = C[c] + \text{rank}(c, i)$, where $C[c]$ is the number of symbols in L lexicographically smaller than c, and $\text{rank}(c, i)$ counts the occurrences of c in $L[0...i]$. This property allows us to traverse the string in reverse order, which is essential for finding patterns in and reconstructing (part of) the string T from the index.

Huffman-shaped wavelet trees. Given the BWT L and the sampled SA, we can reconstruct the original string T from these data structures so we don't need to store T. However, L still takes as much storage as T. To further compress L, a wavelet tree is used to represent L by hierarchically partitioning the alphabet. In the wavelet tree, each leaf node represents a symbol in the alphabet, and each non-leaf node stores a bitvector marking whether the symbol at each position of L belongs to the left or right subtree. A Huffman-shaped wavelet tree (Mäkinen and Navarro, 2005) optimizes the hierarchy by grouping and coding symbols based on their frequencies in L. Storing it costs $(nH_0 + 2\sigma \log n)$ bits, where σ is the alphabet size, and H_0 is the zeroth-order entropy[2] of L and $H_0 \approx 0.26 \log_2 \sigma$ in our experiments with natural language text; this is smaller than storing L directly, which is $n \log_2 \sigma$ bits. Besides compression, the tree can efficiently support two operations crucial to LF mapping: $\text{rank}(c, \ell)$ counts the number of occurrences of a character c in $L[0...\ell - 1]$, and $\text{select}(c, m)$ finds the position of the mth occurrence of c in L. Both operations have a time complexity of $O(H_0)$.

Inverse suffix array. To allow reconstructing part of the original string T from any given position, FM-index also stores a sampled version of the inverse suffix array ISA. ISA is the inverse of the permutation specified in SA and is defined as $ISA[j] = i$ if $SA[i] = j$. When reconstructing T from a given position, ISA is used to identify the corresponding rank in the suffix array to start reconstruction. In FM-index, we can use a different sampling rate for ISA, storing only every bth entry, where b can be larger than a as ISA is used less.

2.2 Querying the FM-index

There are three basic operations supported by FM-index: *find*, *locate*, and *reconstruct*. Counting a pattern can be done with a find operation, and retrieving segments containing the pattern from the original string can be done with a combination of all three operations. See App. §A for more details.

3 INFINI-GRAM MINI

We develop INFINI-GRAM MINI as a scalable and efficient implementation of FM-index on natural

[2] zeroth-order entropy measures the unigram probability distribution of individual symbols (i.e. UTF-8 characters).

language data. For indexing, INFINI-GRAM MINI achieves 18× speedup and reduces RAM usage by 3.2× compared to the previous best existing implementation in SDSL. This paves way for indexing petabyte-level text corpora with reasonable time and compute. For querying, we extended the implementation in SDSL to work with indexes stored on disk, allowing us to query large indexes on machines with limited RAM.

In contrast to the original infini-gram which tokenizes the text, we construct the index directly on raw UTF-8 bytes, which saves tokenization time and allows more flexibility when querying. Tokenization is a technique to reduce index size, but since FM-index is already a compressed index, tokenization would not be helpful here.

3.1 Index Construction

Optimizing the indexing steps. We use a parallelized implementation of each indexing step. For building SA and BWT, we adapt from the parallel implementation in Liu et al. (2024). For building the wavelet tree, we use the parallel implementation in Labeit et al. (2017). For sampling SA and creating a sampled version of ISA, we parallelized the implementation in SDSL. All these steps were single-threaded in SDSL, and by parallelizing them we achieve significant speedup.

We measured the time and peak RAM usage for indexing an 8.7 GB corpus (a single file from DCLM-baseline) using implementation of SDSL and INFINI-GRAM MINI. Their implementation required 5,847 seconds and 74,807 MB of peak RAM, whereas INFINI-GRAM MINI completed indexing in 324 seconds (18× speedup) and peak RAM of 23,742 MB (3.2× reduction).

Preprocessing. FM-index is designed to work on a single string. To index a text corpora, which consists of a collection of documents, we encode all documents with UTF-8 and concatenate all these byte arrays with the `\xff` byte (not used by UTF-8) to mark document boundaries. We then construct FM-index for this big byte string. We work on UTF-8 bytes rather than characters to keep the alphabet size small ($\sigma = 256$). Along with this *text index*, we also store a *text offset* file that records the starting position of each document, which is useful for retrieving whole document and metadata. Aside from the actual text, we also index the metadata of the documents to make the metadata searchable while storing it in a similar compressed form.

Partitioning. For large corpora, we partition it into multiple shards and index each shard independently. Searching across multiple shards produces the same result as searching a single, larger index, and this allows us to build indexes with limited RAM per node as well as embarrassingly parallelize across multiple nodes.

Indexed corpora. We have built the index for the following corpora: the Pile (1.3TB training set, 1.4GB validation set; Gao et al., 2020), DCLM-baseline (17TB; Li et al., 2024), and the Common Crawl between January and July 2025 ("CC-2025-05" to "CC-2025-30" ; 7 crawls and 65TB total; Common Crawl Foundation, 2025).[3]

Indexing time. We use CPU nodes with 128 vCPUs and 2TiB RAM to construct indexes, and under this constraint, each shard can be as large as 700GB which can be indexed in 12–19 hours. Table 1 reports the time for indexing the above corpora. Indexing time shows a slightly super-linear increase with respect to shard size, and we report detailed stepwise breakdown in App. §B. If we were to index the full Common Crawl dataset (about 1PB), we would split it into 1,500 shards, which can be indexed in 1,200 node-days.

Index size. We choose the sampling rate empirically to balance storage savings and query latency. In our implementation, we sample every $a = 32$ entries of SA and every $b = 64$ entries of ISA. This yields indexes with 0.44× the size of the corpora (Table 1). Conceptually, if we set a and b to be very big, then the SA and ISA would have negligible size, and the index can be as small as 0.26× the size of the corpus.

3.2 Querying with INFINI-GRAM MINI

Similar to other exact-match search engines, INFINI-GRAM MINI supports two types of query: *counting* the number of occurrences of a string, and *retrieving documents* that contain a string. At query time, INFINI-GRAM MINI keeps all index files on disk as memory-mapped files, thus requiring only minimal CPU and RAM (loading all indexes uses only $\sim$ 30 MB if RAM and 1 vCPU in our setting).

Querying in INFINI-GRAM MINI is slower and more complex than in canonical suffix arrays due to the compressive nature of FM-index. The SA is subsampled, and the original text is shuffled and

[3]We extract text from the CC crawls with resiliparse, following Li et al. (2024).

Dataset	Original Size (TB)	Indexing Time (CPU node-days)	Index Size (TB)	Num. Shards
Pile-validation	0.0001345	0.00057	0.000602 (0.45×)	1
Pile-train	1.308	1.31	0.588 (0.45×)	2
DCLM-baseline	16.666	12.6	7.523 (0.45×)	25
CC-2025-05	9.079	11.8	3.972 (0.44×)	15
CC-2025-08	8.163	10.6	3.574 (0.44×)	15
CC-2025-13	10.393	13.6	4.563 (0.44×)	17
CC-2025-18	10.498	13.7	4.664 (0.44×)	17
CC-2025-21	9.221	12.1	4.302 (0.46×)	15
CC-2025-26	8.724	11.4	3.835 (0.44×)	15
CC-2025-30	9.006	11.7	3.952 (0.44×)	15
Total	**83.059**	**98.8**	**36.884 (0.44×)**	137

Table 1: Text corpora we indexed with INFINI-GRAM MINI, along with indexing time and index size. The reported numbers only include actual document content and do not include metadata.

compressed, both requiring additional random disk reads to recover. The latency of queries is largely determined by the disk I/O performance.

Counting a string. We use FM-index's *find* operation to count the number of occurrences of a query string. We parallelize this operation across all k shards, and report the sum of counts across all shards. The total number of random disk reads is $O(k|Q|H_0)$, where $|Q|$ is the length of the query string. If the disk's I/O throughput is high enough, the query latency can be as low as $O(|Q|H_0)$.

Retrieving documents. We first use the *find* operation to get the range in the SA that corresponds to the query string. Each element in this range indicates one appearance of the query string in the text corpus. If the index has multiple shards, we will have one range for each shard. For each element in this range, we can use the *locate* operation to find the position of the match in the original text T, then get the boundaries of the enclosing document with a binary search in the text offset file, and finally use the *reconstruct* operation to get the document text. The number of random disk reads for *locate* is $O(aH_0)$, and for *reconstruct* is $O((b+d)H_0)$ where d is the document length. We parallelize *reconstruct* operation by dividing document text into up to $t = 10$ chunks, or chunks of length 100 if $d < 1000$. Retrieving a single document is parallelized across t threads, and it can be further parallelized for retrieving multiple documents.

Query latency. INFINI-GRAM MINI presents second-level latency on both types of query. We benchmark the query latency with the index files stored on Google Cloud Platform (GCP) SSD disks with 80,000 IOPS and 1200 MB/s throughput. We measure the average latency over 100 queries for each query type and setting. For counting, short queries ($|Q| \leq 10$) can be handled within 0.4 seconds for all corpora we indexed, while longer queries ($|Q| = 1000$) are handled in 8 seconds for CC-2025-05 and 25 seconds for DCLM-baseline. The difference in query latency is caused by the different number of shards in the corpora. For document retrieval, retrieving a snippet of $d = 100$ bytes can be done within 2 seconds for all corpora, and retrieving $d = 3000$ bytes takes up to 4.5 seconds. See App. §C for benchmarking details.

3.3 Web Interface and API Endpoint

We host a Hugging Face interface for easy access to counting and document retrieval with 83 TB corpora (App. §E). We also release an API endpoint that allows programmatically submitting requests.

4 Analyzing and Monitoring Benchmark Contamination with INFINI-GRAM MINI

In this section, we showcase how INFINI-GRAM MINI can be used to analyze benchmark contamination at scale. INFINI-GRAM MINI allows us to search in the largest body of text ever in the open-source community with minimal storage overhead, enabling large-scale benchmark contamination analysis at low cost. Our work also supports analyzing new benchmarks uploaded later.

By identifying lexical overlap between 24 widely-used evaluation benchmarks and three ma-

jor corpora, we find non-trivial contamination across 12 benchmarks (§4.2). We then retrieve documents accounting for contamination for further analysis (§4.3). Powered by INFINI-GRAM MINI, we develop a monitoring system with a benchmark contamination bulletin, where we continuously monitor new crawls and report benchmarks' contamination status over time (§4.4). We also invite the community to contribute by suggesting additional benchmarks or uploading new ones for analysis.

INFINI-GRAM MINI has further potential applications where exact-match search is needed, including (1) task-specific dataset construction, where creation of those datasets requires retrieving documents containing specific terms or phrases, and (2) data curation, where INFINI-GRAM MINI can assist in identifying and removing duplicate, low-quality, or sensitive text and documents. We leave these directions for future exploration.

4.1 Setup

Contamination detection method. We check contamination of test examples by measuring lexical overlap with the text corpus, following standard practice in the literature (Brown et al., 2020; Llama Team, 2024). Given a text entry, we extract all 50-character substrings S with a stride of one word, and determine entry contamination based on the proportion of substrings that appear at least once in the corpus:

$$\eta = \frac{\sum_{s \in S} \mathbb{I}\left[\texttt{count}(s) > 0\right]}{|S|}.$$

We classify an entry into three contamination levels using the same thresholds with different namings as Touvron et al. (2023):

- Clean, if $\eta < 20\%$
- Suspicious, if $20\% \leq \eta < 80\%$
- Dirty, if $\eta \geq 80\%$

Benchmarks. To comprehensively evaluate contamination, we analyze a broad range of benchmarks widely used to evaluate state-of-the-art LMs. We categorize benchmarks into five groups based on their primary focus: (1) **knowledge and reasoning**, i.e., understanding and reasoning over factual and conceptual knowledge; (2) **math**, i.e., mathematical reasoning ability; (3) **code** generation and modification abilities; (4) **commonsense understanding**, i.e., commonsense knowledge and reasoning abilities; and (5) **reading comprehension** of textual context. In total, we analyze 24 benchmarks; see App. §F for the full list and citations.

For benchmarks with question-answering format, we specifically check the question entries for contamination. Questions are usually longer and contain sufficient contextual information to uniquely identify the benchmark example, whereas answers can be short, such as a multiple choice option or a number. For language understanding tasks that involve reading a context or paragraph before answering, we check the context entries for contamination.

We evaluate on only the test set of benchmarks. For benchmarks with large test set, we downsample to 1,000 entries for efficiency. For benchmarks with multiple subtasks, we sample proportionally from each subtask to maintain representative distribution.

4.2 Results

Table 2 reports the percentage of dirty entries in benchmarks against the Pile (knowledge cutoff in 2020), DCLM-baseline (knowledge cutoff in 2022), and seven CC crawls (knowledge cutoff in January-July 2025). The detailed count of dirty and suspicious entries can be found in App. §G.

Many widely-used benchmarks are highly contaminated. When checking against DCLM-baseline, MMLU has 27.70% dirty entries, MMLU-Pro has 16.20%, ARC-Challenge has 32.6%, and ARC-Easy has 32.3%. These benchmarks, especially MMLU, have been used to evaluate virtually every new LM in recent years. We believe that the observed contamination levels are a strong signal that many recently reported results may overestimate language model abilities on truly new, unseen evaluation items.

Larger and newer corpora show greater contamination. Compared with Pile-train, most benchmarks show a higher dirty rate on DCLM-baseline. For example, ARC-Challenge has 17× more and MMLU has 1.1× more dirty entries. However, CC-2025-05 has lower contamination rate than DCLM-baseline on most benchmarks, which is likely because DCLM-baseline corpus is larger and is a high-quality subset of crawls spanning a decade, while CC-2025-05 is an unfiltered single crawl.

Contamination level varies by domain. Benchmarks in historically-significant domains, such as commonsense understanding (e.g., ARC,

OpenbookQA) and reading comprehension (e.g., SQuAD, CoQA), tend to show higher dirty rates on all corpora, while those in emerging domains like math (e.g., GSM8K, MATH-500) and code (e.g., HumanEval, LiveCodeBench) remain relatively clean at this writing.

New benchmarks are gradually getting contaminated in recent corpora. Newer benchmarks are initially clean on earlier corpora. For example, AIME-2024 is uncontaminated on Pile-train and DCLM-baseline, but show 40.00% dirty rate on the more recent CC-2025-21; similarly, GPQA shows 2.70% dirty rate on CC-2025-26. GSM8K is very clean on Pile-train, DCLM-baseline, and earlier CC-2025 crawls, but has 74.2% dirty entries on CC-2025-21.

4.3 Analysis

With INFINI-GRAM MINI, we can retrieve documents from the corpus that contain contaminated examples for further analysis. We use LLM-as-a-judge to categorize all dirty instances in Pile-train, DCLM-baseline, and CC-2025-05 into following scenarios (see App. §I for more details), with examples shown in Table 3:

- **Type 1: Question and answer appear as-is in corpus.** The question and correct answer appear in the corpus in the exact format as in benchmark entry. The model may memorize entirely from these data instead of performing capabilities examined in benchmarks. This accounts for 72.5% of all dirty entries in Pile-train, 82.6% on DCLM-baseline, and 58% on CC-2025-05.
- **Type 2: Question appeared, answer in natural language.** The answer is expressed in natural language rather than the exact benchmark format, and model may directly infer the correct answer from it. This accounts for 4.5% of all dirty entries in Pile-train, 2.1% on DCLM-baseline, and 6.2% on CC-2025-05.
- **Type 3: Question appeared, no corresponding answer.** The question (and maybe the multiple-choice options) appeared in the corpus, but correct answer is missing. This accounts for 18.1% of all dirty entries in Pile-train, 10.9% on DCLM-baseline, and 30.2% on CC-2025-05.
- **Type 4: False positives.** There are documents that superficially match the question but are unrelated to the benchmark example. This happens on entries with a very short question. This accounts for 3.1% of all dirty entries in Pile-train, 1% on DCLM-baseline, and 3% on CC-2025-05.

We show 7 contamination examples in App. §H and their source of contamination. We found that contamination is caused by (1) the benchmark is sourced from the Internet (e.g., AIME-2024, Figure 6; SWE-bench, Figure 8), (2) there are other benchmarks or online quizzes that are sourced from the benchmark (e.g, GSM8K, Figure 10), (3) blogs and papers cite a benchmark entry as example (e.g, GPQA, Figure 9; BigBenchHard, Figure 11; OpenbookQA, Figure 12), and (4) benchmark entry coincide with online source (e.g., MMLU, Figure 7).

Contamination could cause LLM to overperform on evaluation benchmarks by enabling models to retrieve memorized answers from training data rather than performing task-specific reasoning. Our finding shows that a large majority of dirty entries contain exact matches of both question and answer, though even subtler forms of contamination such as question-only matches and paraphrased answers could inflate benchmark scores. As training corpora grow, the risk of benchmark contamination and the potential for rote memorization increase, making it more important to decontaminate training corpora and construct contamination-free benchmarks to avoid overestimating model capabilities.

4.4 Benchmark Contamination Bulletin

Using INFINI-GRAM MINI, we implement a benchmark contamination monitoring system that tracks benchmark contamination. In the future, we will keep indexing the latest crawl in Common Crawl and update contamination results to track benchmark contamination as corpora evolve. The system also allows anyone to add or upload new benchmarks to be monitored, fostering collaboration in benchmark monitoring. See App. §J for system interface.

5 Related Work

Exact-match search in large text corpora. Prior work has used different techniques to enable exact-match full-text search in large text corpora, including suffix arrays, suffix automata, and proprietary search engines. Below we survey the largest-scale implementation of each technique known to us. Merrill et al. (2024) apply a suffix automaton to 1.3TB of text. Liu et al. (2024) apply a suffix array to 12TB of text. Elazar et al. (2024) use the proprietary ElasticSearch to index and analyze 35TB of text. All these methods have significant storage

	Test Size	Pile train	DCLM baseline	CC 2025-05	CC 2025-08	CC 2025-13	CC 2025-18	CC 2025-21	CC 2025-26
Knowledge and Reasoning									
MMLU	1000	13.20	28.40	13.50	9.00	12.10	11.50	11.70	9.20
MMLU-Pro	1000	5.50	16.20	7.10	5.40	6.00	6.30	7.40	6.90
BigBenchHard	1000	0.00	0.10	1.40	1.40	3.20	2.30	1.80	1.70
AGIEval	1000	0.80	3.10	2.70	3.60	3.00	7.00	9.40	4.60
GPQA	448	0.00	0.00	0.90	2.00	1.30	0.70	0.90	2.70
HLE	881	0.00	0.30	0.10	0.00	0.10	0.00	0.00	0.00
Math									
AIME-2024	30	0.00	0.00	10.00	3.30	6.70	40.00	40.00	13.30
GSM8K	1000	0.00	5.00	5.00	0.80	6.90	0.70	74.20	7.30
MATH-500	500	0.60	3.20	0.60	7.80	0.80	0.80	0.80	8.20
MGSM	250	0.00	0.00	5.60	1.60	35.60	0.80	72.80	6.00
Code									
HumanEval	164	0.00	0.00	0.00	0.60	0.60	0.60	0.00	0.00
HumanEval+	164	0.00	0.00	0.00	0.60	0.60	0.60	0.00	0.00
LiveCodeBench	880	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
SWE-bench	500	0.00	0.00	0.20	0.20	0.00	0.00	0.00	0.00
MBPP	500	0.00	0.40	1.00	1.40	1.20	1.80	1.00	1.40
Commonsense Understanding									
ARC-Challenge	1000	1.80	34.10	11.90	4.00	3.10	3.80	4.20	4.80
ARC-Easy	1000	1.30	31.70	5.40	9.50	5.50	5.50	6.10	6.20
CSQA	1000	0.10	1.00	0.10	0.10	0.20	0.10	0.00	0.10
HellaSwag	1000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.10
OpenbookQA	500	10.80	15.60	14.60	30.20	13.20	13.40	13.20	12.20
Social IQa	1000	0.00	0.50	0.20	4.40	0.20	0.30	0.20	0.10
WinoGrande	1000	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
Reading Comprehension									
CoQA	500	8.00	18.40	7.40	8.80	8.60	7.20	7.60	8.80
SQuAD	1000	2.80	40.10	2.70	33.00	10.10	1.50	2.00	8.50

Table 2: Dirty rates for benchmarks across the Pile, DCLM-baseline, and Common Crawl from January to July, 2025. Full result is reported in Table 7. For benchmark entries with over 1000 entries, we report dirty rate on the downsampled subset. Cell background color indicates benchmark cleaniness, with more redness representing increasing levels of contamination.

multiplier wrt the size of text indexed, ranging from ElasticSearch's 2× to suffix automata's 29×. In contrast, our FM-index-based index has a storage multiplier as small as 0.26×, allowing us to index the largest body of text ever in the open-source community.

Benchmark contamination. Benchmark contamination appeared as a critical concern in LLM evaluations in recent studies. Prior works has quantified benchmark contamination on open-sourced models using various matching strategies, including n-gram or token overlap (Soldaini et al., 2024; Llama Team, 2024; Soldaini et al., 2024; OLMo et al., 2025), longest substring match (Singh et al., 2024), skipgram match (Touvron et al., 2023), and full-text exact match (Elazar et al., 2024). Sainz et al. (2024) reports data contamination from multiple sources through shared efforts. However, indexing large-scale pretraining corpora and performing exhaustive searches is computationally expensive, and prior studies on open corpora are limited in their scale (up to RedPajama-1T and Dolma, 12TB). To the best of our knowledge, our work conducts contamination analysis on the largest open corpora to date.

6 Conclusion

We introduce INFINI-GRAM MINI, an efficient system for indexing text with 0.44× their original size, enabling efficient counting and searching in mas-

	Test Question	Document Snippet
Type 1	**Question:** Increasing the credit period from 30 to 60 days, in response to a similar action taken by all of our competitors, would likely result in 0) an increase in the average collection period. 1) a decrease in bad debt losses. 2) an increase in sales. 3) higher profits. *(benchmark: MMLU)*	Increasing the credit period from 30 to 60 days, in response to a similar action taken by all of our competitors, would likely result in: A) an increase in the average collection period. B) a decrease in bad debt losses. C) an increase in sales. D) higher profits. Ans: A *(corpus: CC-2025-05; source: website page)*
Type 2	**Question:** A red-tailed hawk is searching for prey. It is most likely to swoop down on A. an eagle B. a cow C. a gecko D. a deer *(benchmark: OpenbookQA)*	**Generated Hypothesis**: H: A red-tailed hawk is searching for prey. It is most likely to swoop down on a gecko. **Retrieved Fact from OpenBook:** F: hawks eat lizards *(corpus: Pile; source: ArXiv)*
Type 3	**Question:** To express the distance between the Milky Way galaxy and other galaxies, the most appropriate unit of measurement is the A. meter. B. kilometer. C. light-year. D. astronomical unit. *(benchmark: ARC-Easy)*	To express the distance between the Milky Way galaxy and other galaxies, the most appropriate unit of measurement is the (A) meter. (B) kilometer. (C) light-year. (D) astronomical unit. *(corpus: DCLM-baseline; source: CC-MAIN-2018-47)*
Type 4	**Question:** Where do you buy vegetables? A. supermarket B.fridge C.refrigerator D.cashier E.vegetable garden *(benchmark: CSQA)*	- What makes you think Suu will be late? - I am your friend. - Where do you buy vegetables? - I am not exaggerating. I'm just being honest about my emotions. *(corpus: DCLM-baseline; source: CC-MAIN-2019-18)*

Figure 3: Examples of four contamination types. **Violet** text is the text overlap between benchmark entry and corpus. **Magenta** text is the mapping of answers.

sive text corpus, and we show its scalability to a petabyte-scale corpus. We showcase INFINI-GRAM MINI's application on benchmark contamination analysis at scale.

Limitations

Although INFINI-GRAM MINI shows significant improvements on text compression rate, document retrieval latency remains higher than a canonical suffix array. For example, reconstructing a 3000-character document takes 1.8 seconds on Pile and 4.5 seconds on DCLM-baseline, compared to millisecond-level retrieval in system like Infini-gram (Liu et al., 2024). This is a trade-off with INFINI-GRAM MINI's compression rate: INFINI-GRAM MINI does not store the original text in a contiguous block. To retrieve a document, we need to reconstruct it character-by-character, leading to a large number of reads in random addresses. These process takes long time since the entire index is kept on disk at inference time. The high latency can potentially be reduced using techniques like disk page prefetching.

Identifying co-occurrences of multiple patterns is inefficient with the FM-index, since mapping every match from the suffix array range back to its position in original text takes very long time thus making this operation impractical. In contrast, the original infini-gram supports this operation efficiently.

INFINI-GRAM MINI only supports exact-match searches. As a result, our benchmark contamination analysis is limited to case-sensitive exact matching, which may fail to detect contamination of instances with minor textual discrepancies.

As the text corpora may contain biased or toxic content, document retrieval output may contain content that can be perceived as ethically problematic, and may contain sensitive information. The output of INFINI-GRAM MINI does not reflect authors' views.

Acknowledgments

We thank Christina Boucher (University of Florida) for telling us about the FM-index data structure. We would like to thank members of H2lab and AllenNLP for sharing their valuable feedback on this project. This work was funded in part by NSF IIS-2044660 and IIS-2113530.

References

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, and 1 others. 2021. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*.

Michele Bevilacqua, Giuseppe Ottaviano, Patrick Lewis, Wen tau Yih, Sebastian Riedel, and Fabio Petroni. 2022. Autoregressive search engines: Generating substrings as document identifiers. *Preprint*, arXiv:2204.10628.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Michael Burrows, D J Wheeler D I G I T A L, Robert W. Taylor, David J. Wheeler, and David Wheeler. 1994. A block-sorting lossless data compression algorithm.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, and 39 others. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. 2018. Think you have solved question answering? try arc, the ai2 reasoning challenge. *arXiv:1803.05457v1*.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Common Crawl Foundation. 2025. Common crawl.

Lore Depuydt, Luca Renders, Thomas Abeel, and Jan Fostier. 2023. Pan-genome de bruijn graph using the bidirectional fm-index. *BMC Bioinformatics*, 24(1).

Yanai Elazar, Akshita Bhagia, Ian Magnusson, Abhilasha Ravichander, Dustin Schwenk, Alane Suhr, Pete Walsh, Dirk Groeneveld, Luca Soldaini, Sameer Singh, Hanna Hajishirzi, Noah A. Smith, and Jesse Dodge. 2024. What's in my big data? *Preprint*, arXiv:2310.20707.

P. Ferragina and G. Manzini. 2000. Opportunistic data structures with applications. In *Proceedings 41st Annual Symposium on Foundations of Computer Science*, pages 390–398.

Leo Gao, Stella Biderman, Sid Black, Laurence Golding, Travis Hoppe, Charles Foster, Jason Phang, Horace He, Anish Thite, Noa Nabeshima, Shawn Presser, and Connor Leahy. 2020. The Pile: An 800gb dataset of diverse text for language modeling. *arXiv preprint arXiv:2101.00027*.

Simon Gog, Timo Beller, Alistair Moffat, and Matthias Petri. 2014. From theory to practice: Plug and play with succinct data structures. In *13th International Symposium on Experimental Algorithms, (SEA 2014)*, pages 326–337.

Li-Lu Guo. 2025. Effloc: An efficient locating algorithm for mass-occurrence biological patterns with fm-index. *Journal of computational biology : a journal of computational molecular cell biology*.

Xiaochuang Han and Yulia Tsvetkov. 2022. Orca: Interpreting prompted language models via locating supporting data evidence in the ocean of pretraining data. *ArXiv*, abs/2205.12600.

Dan Hendrycks, Collin Burns, Steven Basart, Andy Zou, Mantas Mazeika, Dawn Song, and Jacob Steinhardt. 2021. Measuring massive multitask language understanding. *Preprint*, arXiv:2009.03300.

Naman Jain, King Han, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, and Ion Stoica. 2024. Livecodebench: Holistic and contamination free evaluation of large language models for code. *Preprint*, arXiv:2403.07974.

Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik Narasimhan. 2024. Swe-bench: Can language models resolve real-world github issues? *Preprint*, arXiv:2310.06770.

Julian Labeit, Julian Shun, and Guy E. Blelloch. 2017. Parallel lightweight wavelet tree, suffix array and fm-index construction. *Journal of Discrete Algorithms*, 43:2–17.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. 2022. Deduplicating training data makes language models better. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 8424–8445, Dublin, Ireland. Association for Computational Linguistics.

Heng Li. 2014. Fast construction of fm-index for long sequence reads. *Bioinformatics*, 30(22):3274–3275.

Jeffrey Li, Alex Fang, Georgios Smyrnis, Maor Ivgi, Matt Jordan, Samir Gadre, Hritik Bansal, Etash Guha, Sedrick Keh, Kushal Arora, Saurabh Garg, Rui Xin, Niklas Muennighoff, Reinhard Heckel, Jean Mercat, Mayee Chen, Suchin Gururangan, Mitchell Wortsman, Alon Albalak, and 40 others. 2024. Datacomp-lm: In search of the next generation of training sets for language models. *Preprint*, arXiv:2406.11794.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *Preprint*, arXiv:2305.20050.

Jiacheng Liu, Taylor Blanton, Yanai Elazar, Sewon Min, YenSung Chen, Arnavi Chheda-Kothary, Huy Tran, Byron Bischoff, Eric Stuart Marsh, Michael Schmitz, Cassidy Trier, Aaron Sarnat, Jenna James, Jon Borchardt, Bailey Kuehl, Evie Cheng, Karen Farley, Sruthi Sreeram, Taira Anderson, and 12 others. 2025. Olmotrace: Tracing language model outputs back to trillions of training tokens.

Jiacheng Liu, Sewon Min, Luke Zettlemoyer, Yejin Choi, and Hannaneh Hajishirzi. 2024. Infini-gram: Scaling unbounded n-gram language models to a trillion tokens. *arXiv preprint arXiv:2401.17377*.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. 2023. Is your code generated by chatGPT really correct? rigorous evaluation of large language models for code generation. In *Thirty-seventh Conference on Neural Information Processing Systems*.

AI @ Meta Llama Team. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Ximing Lu, Melanie Sclar, Skyler Hallinan, Niloofar Mireshghallah, Jiacheng Liu, Seungju Han, Allyson Ettinger, Liwei Jiang, Khyathi Chandu, Nouha Dziri, and Yejin Choi. 2024. Ai as humanity's salieri: Quantifying linguistic creativity of language models via systematic attribution of machine text against web text. *ArXiv*, abs/2410.04265.

Veli Mäkinen and Gonzalo Navarro. 2005. Succinct suffix arrays based on run-length encoding. In *Combinatorial Pattern Matching*, pages 45–56, Berlin, Heidelberg. Springer Berlin Heidelberg.

William Merrill, Noah A. Smith, and Yanai Elazar. 2024. Evaluating n-gram novelty of language models using rusty-dawg. *ArXiv*, abs/2406.13069.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. 2018. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*.

Art of Problem Solving. 2024. Aime problems and solutions.

Team OLMo, Pete Walsh, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Shane Arora, Akshita Bhagia, Yuling Gu, Shengyi Huang, Matt Jordan, Nathan Lambert, Dustin Schwenk, Oyvind Tafjord, Taira Anderson, David Atkinson, Faeze Brahman, Christopher Clark, Pradeep Dasigi, Nouha Dziri, and 21 others. 2025. 2 olmo 2 furious. *Preprint*, arXiv:2501.00656.

Long Phan, Alice Gatti, Ziwen Han, Nathaniel Li, Josephina Hu, Hugh Zhang, Chen Bo Calvin Zhang, Mohamed Shaaban, John Ling, and Sean Shi et al. 2025. Humanity's last exam. *Preprint*, arXiv:2501.14249.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Siva Reddy, Danqi Chen, and Christopher D. Manning. 2019. CoQA: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics*, 7:249–266.

David Rein, Betty Li Hou, Asa Cooper Stickland, Jackson Petty, Richard Yuanzhe Pang, Julien Dirani, Julian Michael, and Samuel R. Bowman. 2023. Gpqa: A graduate-level google-proof q&a benchmark. *Preprint*, arXiv:2311.12022.

Oscar Sainz, Iker García-Ferrero, Alon Jacovi, Jon Ander Campos, Yanai Elazar, Eneko Agirre, Yoav Goldberg, Wei-Lin Chen, Jenny Chim, Leshem Choshen, Luca D'Amico-Wong, Melissa Dell, Run-Ze Fan, Shahriar Golchin, Yucheng Li, Pengfei Liu, Bhavish Pahwa, Ameya Prabhu, Suryansh Sharma, and 9 others. 2024. Data contamination report from the 2024 conda shared task. *Preprint*, arXiv:2407.21530.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. 2019. Winogrande: An adversarial winograd schema challenge at scale. *Preprint*, arXiv:1907.10641.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan LeBras, and Yejin Choi. 2019. Socialiqa: Commonsense reasoning about social interactions. *Preprint*, arXiv:1904.09728.

Freda Shi, Mirac Suzgun, Markus Freitag, Xuezhi Wang, Suraj Srivats, Soroush Vosoughi, Hyung Won Chung, Yi Tay, Sebastian Ruder, Denny Zhou, Dipanjan Das, and Jason Wei. 2022. Language models are multilingual chain-of-thought reasoners. *Preprint*, arXiv:2210.03057.

Aaditya K. Singh, Muhammed Yusuf Kocyigit, Andrew Poulton, David Esiobu, Maria Lomeli, Gergely Szilvasy, and Dieuwke Hupkes. 2024. Evaluation data contamination in llms: how do we measure it and (when) does it matter? *Preprint*, arXiv:2411.03923.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Ben Bogin, Khyathi Chandu, Jennifer Dumas, Yanai Elazar, Valentin Hofmann, Ananya Jha, Sachin Kumar, Li Lucy, Xinxi Lyu, Nathan Lambert, Ian Magnusson, Jacob Morrison, Niklas Muennighoff, and 17 others. 2024. Dolma: an open corpus of three trillion tokens for language model pretraining research. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 15725–15788, Bangkok, Thailand. Association for Computational Linguistics.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, , and Jason Wei. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

Alon Talmor, Jonathan Herzig, Nicholas Lourie, and Jonathan Berant. 2019. CommonsenseQA: A question answering challenge targeting commonsense knowledge. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4149–4158, Minneapolis, Minnesota. Association for Computational Linguistics.

Hugo Touvron, Louis Martin, Kevin R. Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, Daniel M. Bikel, Lukas Blecher, Cristian Cantón Ferrer, Moya Chen, Guillem Cucurull, David Esiobu, Jude Fernandes, Jeremy Fu, Wenyin Fu, and 49 others. 2023. Llama 2: Open foundation and fine-tuned chat models. *ArXiv*, abs/2307.09288.

Yuanrong Wang, Xueqi Li, Dawei Zang, Guangming Tan, and Ninghui Sun. 2018. Accelerating fm-index search for genomic data processing. In *Proceedings of the 47th International Conference on Parallel Processing*, ICPP '18, New York, NY, USA. Association for Computing Machinery.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, Tianle Li, Max Ku, Kai Wang, Alex Zhuang, Rongqi Fan, Xiang Yue, and Wenhu Chen. 2024. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Preprint*, arXiv:2406.01574.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. 2019. Hellaswag: Can a machine really finish your sentence? In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*.

Wanjun Zhong, Ruixiang Cui, Yiduo Guo, Yaobo Liang, Shuai Lu, Yanlin Wang, Amin Saied, Weizhu Chen, and Nan Duan. 2023. Agieval: A human-centric benchmark for evaluating foundation models. *Preprint*, arXiv:2304.06364.

A Querying the FM-Index

This section introduces in details three operations supported by FM-index: *find*, *locate*, and *reconstruct*.

Find. Each occurrence of a pattern string Q in the haystack string T corresponds to an element in the SA, and all these occurrences live in a consecutive range in the SA. The *find* operation computes this range with backward search: starting with the full range, it iterates through the symbols in Q in reverse order, with each iteration narrowing the range using the character table C and the rank operation of the BWT's wavelet tree. The length of the final range is the count of Q in T. Figure 4 (left) illustrates finding pattern "ana" in string "banana". The time complexity of *find* is $O(|Q|H_0)$, where $|Q|$ is the length of the pattern.

Locate. The *locate* operation maps any position in the SA to its corresponding position in T. Since the SA is sampled at interval a, for an unsampled position i, the algorithm applies LF mapping at most a times to locate the nearest sampled SA entry. Figure 4 (middle) illustrates locating the two occurrences of "ana" in "banana". The time complexity of locating each position is $O(t_{SA})$, where t_{SA} is the complexity of accessing an SA entry, which is $O(1)$ for sampled entries and $O(aH_0)$ for unsampled entries.

Reconstruct. After getting the position of a pattern occurrence in T, we can *reconstruct* a substring of T enclosing that occurrence with some additional context. Starting at the end position of the desired substring, we apply the LF mapping to traverse through the BWT and use ISA to recover the symbol, and reconstruct symbols in reverse order until reaching the start position of the desired substring. Figure 4 (right) shows reconstructing the second occurrence of "ana" with context length of 1. The time complexity of *reconstruct* is $O(dH_0 + t_{ISA})$, where d is the length to reconstruct, and t_{ISA} is the complexity of accessing an ISA entry, which is $O(1)$ for sampled entries and $O(bH_0)$ for unsampled entries.

B Indexing Time

Table 3 presents indexing time for each shard of text corpora with stepwise breakdown. Suffix array construction scales super-linearly depending on the level of duplication (Lee et al., 2022). Constructing alphabet, wavelet tree, and sampling are linear operations. Indexing time also varies across similar-size shards originated from different text corpora, showing the time also depend on text corpora properties. For example, over 60% duplicate documents in the Pile (Elazar et al., 2024) causes SA construction to take significantly longer compared to DCLM-baseline.

C Benchmarking Query Latency

We benchmark the query latency of INFINI-GRAM MINI, and report results in Table 4. For *counting*, we experiment with query lengths $|Q| \in \{1, 2, 5, 10, 50, 100, 500, 1000\}$, and for each length, randomly sample the query strings from the corpus. For *retrieving documents*, we use queries of length 10 sampled from the corpus, and reconstruct text surrounding the query with total length of $d \in \{10, 50, 100, 500, 1000\}$. For each corpus and parameter setting, we repeat with 100 random queries and report the average latency. For benchmarking, we store the indexes on `pd-balanced` SSD disks on GCP which has a max IOPS of 80,000 and max throughput of 1200 MB/s, and we experiment on an `n2-highcpu-64` node where the max disk I/O performance can be achieved.

D Query Latency Comparison with Prior Works

Table 5 reports query latency on Pile-train corpus using our method compared to prior work by Liu et al. (2024).

E INFINI-GRAM MINI Web Interface

We host web interface for easy access to counting (Figure 5 left) and document retrieval (Figure 5 right) with the indexes we have built.

F Details for Benchmarks in Contamination Analysis

Table 6 shows the benchmarks we analyze under each category with citation and source.

G Detailed Benchmark Contamination Result

Table 7 reports the number of suspicious and dirty entries of each benchmark.

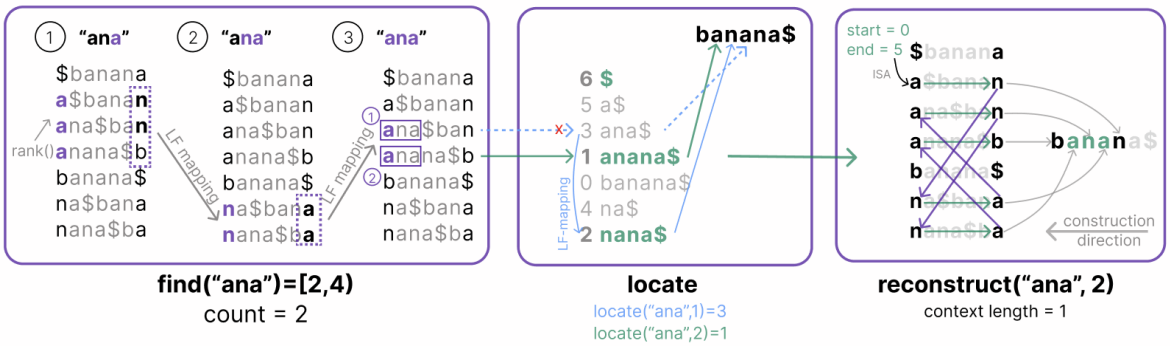


Figure 4: Illustration of operations on FM-index (§2.2, App. §A). **Left:** *find* operation computes the SA range corresponding to all occurrences of the pattern. **Middle:** *locate* operation computes the position of pattern occurrence in the original string for each position in the SA range. **Right:** *reconstruct* operation gets a substring of the original string enclosing the second pattern occurrence with a context length of 1. The occurrence ranking is based on its order in SA.

Step	Pile-val (1.4 GB)	Pile-train (653 GB)	DCLM-baseline (667 GB)	CC-2025-05 (654 GB)
SA+BWT	29 s	41710 s	29543 s	55692 s
alphabet	4 s	2584 s	2895 s	2580 s
wavelet tree	13 s	5773 s	6257 s	5325 s
sample SA	1 s	2540 s	2232 s	2013 s
sample ISA	2 s	3975 s	2659 s	2313 s
Total	**49 s**	**15.7 h**	**12.1 h**	**18.9 h**

Table 3: Index construction time for **each shard** of the text corpora, with stepwise breakdown. The size of text in each shard is noted at the top. To get the indexing time of the full corpus on a single node, roughly multiply by the number of shards; though this can be embarrassingly parallelized across multiple nodes. Metadata size and metadata indexing time are excluded.

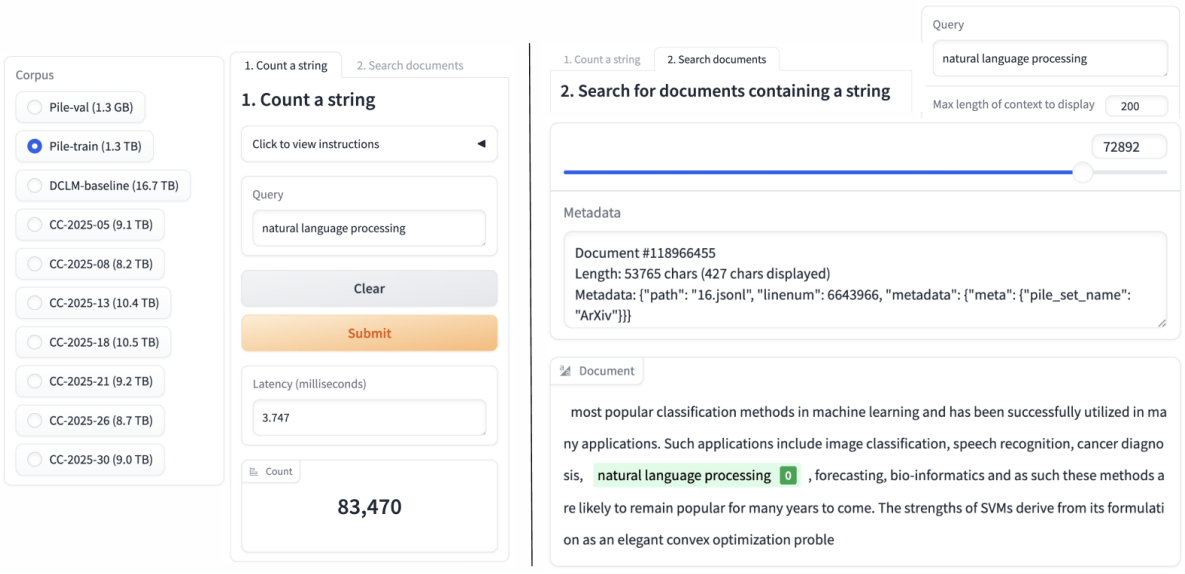


Figure 5: The web interface of INFINI-GRAM MINI. **Left:** counting a string. **Right:** retrieving documents.

	Pile-train (n = 1.3T) (S = 2)	**DCLM-baseline** (n = 16.7T) (S = 25)	**CC-2025-05** (n = 9.1T) (S = 15)	**Time Complexity**
Counting a query of length $\|Q\|$				$O(\|Q\|H_0)$
... ($\|Q\|$ = 1)	0.004 s	0.005 s	0.032 s	
... ($\|Q\|$ = 2)	0.015 s	0.017 s	0.094 s	
... ($\|Q\|$ = 5)	0.061 s	0.207 s	0.206 s	
... ($\|Q\|$ = 10)	0.106 s	0.402 s	0.350 s	
... ($\|Q\|$ = 20)	0.182 s	0.868 s	0.638 s	
... ($\|Q\|$ = 50)	0.393 s	1.743 s	1.063 s	
... ($\|Q\|$ = 100)	0.696 s	2.857 s	1.642 s	
... ($\|Q\|$ = 200)	1.281 s	5.699 s	2.753 s	
... ($\|Q\|$ = 500)	2.763 s	12.46 s	4.626 s	
... ($\|Q\|$ = 1000)	4.808 s	25.47 s	7.957 s	
Retrieving a text of length d				$O((a+b+d)H_0)$
... (d = 10)	0.426 s	0.895 s	1.101 s	
... (d = 50)	0.634 s	1.549 s	1.302 s	
... (d = 100)	0.734 s	1.991 s	1.326 s	
... (d = 500)	0.874 s	2.363 s	1.609 s	
... (d = 1000)	0.94 s	2.385 s	1.705 s	
... (d = 2000)	1.213 s	3.464 s	2.849 s	
... (d = 3000)	1.858 s	4.456 s	3.330 s	

Table 4: Inference time latency of INFINI-GRAM MINI. Average latency of each query is reported. Notations: n = number of bytes in the text corpus, S = number of shards for the index, $|Q|$ = length of query in bytes, d = length of text (in bytes) to reconstruct from the index, a = sampling rate of SA, b = sampling rate of ISA, H_0 = zeroth-order entropy of the corpus.

	$\|Q\| = 10$ bytes	$\|Q\| = 20$ bytes	$\|Q\| = 100$ bytes
infini-gram (Liu et al., 2024)	13 ms	14 ms	13 ms
INFINI-GRAM MINI	106 ms	182 ms	696 ms

Table 5: Text retrieving latency comparison between infini-gram and INFINI-GRAM MINI on Pile-train corpus.

Benchmark	Citation	Source
Knowledge and Reasoning		
MMLU	Hendrycks et al., 2021	https://huggingface.co/datasets/cais/mmlu
MMLU-Pro	Wang et al., 2024	https://huggingface.co/datasets/TIGER-Lab/MMLU-Pro
BigBenchHard	Suzgun et al., 2022	https://github.com/suzgunmirac/BIG-Bench-Hard/tree/main/bbh
AGIEval	Zhong et al., 2023	https://github.com/ruixiangcui/AGIEval/tree/main/data/v1_1
GPQA	Rein et al., 2023	https://huggingface.co/datasets/Idavidrein/gpqa
HLE	Phan et al., 2025	https://huggingface.co/datasets/cais/hle
Math		
AIME-2024	of Problem Solving, 2024	https://huggingface.co/datasets/Maxwell-Jia/AIME_2024
GSM8K	Cobbe et al., 2021	https://huggingface.co/datasets/openai/gsm8k
MATH-500	Lightman et al., 2023	https://huggingface.co/datasets/HuggingFaceH4/MATH-500
MGSM	Shi et al., 2022	https://huggingface.co/datasets/juletxara/mgsm
Code		
HumanEval	Chen et al., 2021	https://huggingface.co/datasets/openai/openai_humaneval
HumanEval+	Liu et al., 2023	https://huggingface.co/datasets/evalplus/humanevalplus
LiveCodeBench (code generation)	Jain et al., 2024	https://huggingface.co/datasets/livecodebench/code_generation
SWE-bench	Jimenez et al., 2024	https://huggingface.co/datasets/princeton-nlp/SWE-bench_Verified
MBPP	Austin et al., 2021	https://huggingface.co/datasets/google-research-datasets/mbpp
Commonsense Understanding		
ARC-Challenge	Clark et al., 2018	https://huggingface.co/datasets/allenai/ai2_arc
ARC-Easy	Clark et al., 2018	https://huggingface.co/datasets/allenai/ai2_arc
CSQA	Talmor et al., 2019	https://huggingface.co/datasets/tau/commonsense_qa
HellaSwag	Zellers et al., 2019	https://huggingface.co/datasets/Rowan/hellaswag
Openbook QA	Mihaylov et al., 2018	https://huggingface.co/datasets/allenai/openbookqa
Social IQa	Sap et al., 2019	https://huggingface.co/datasets/allenai/social_i_qa
WinoGrande	Sakaguchi et al., 2019	https://huggingface.co/datasets/allenai/winogrande
Reading Comprehension		
CoQA	Reddy et al., 2019	https://huggingface.co/datasets/stanfordnlp/coqa
SQuAD	Rajpurkar et al., 2016	https://huggingface.co/datasets/rajpurkar/squad

Table 6: Citation and source of each benchmark analyzed in this paper.

Benchmark	Test Set	Pile-train		DCLM-baseline		CC-2025-05		CC-202508		CC-202513		CC-202518		CC-202521		CC-202526	
	Size	Sus	Dirty	Sus	Dirty	Sus	Dirty	Sus	Dirty	Sus	Dirty	Sus	Dirty	Sus	Dirty	Sus	Dirty
Knowledge and Reasoning																	
MMLU	1000	57	133	142	277	122	135	139	90	121	122	115	120	117	125	92	124
MMLU-Pro	1000	35	59	77	162	31	81	55	54	60	52	63	58	74	60	69	40
BigBenchHard	1000	7	0	54	1	357	14	296	14	306	32	302	23	396	18	338	17
AGIEval	1000	45	8	155	31	107	27	92	36	115	30	109	70	95	94	104	46
GPQA	448	1	0	5	0	6	4	10	9	5	6	6	3	5	4	15	12
HLE	881	3	0	6	0	8	0	3	0	4	1	5	0	3	0	2	0
Math																	
AIME-2024	30	2	0	4	0	13	3	3	1	10	2	13	12	13	12	12	4
GSM8K	1000	1	0	5	4	33	76	8	8	37	69	10	7	210	742	20	73
MATH-500	500	9	3	48	16	26	3	46	39	25	4	27	4	26	4	41	41
MGSM	250	0	0	2	0	9	14	3	4	37	89	5	2	54	182	6	15
Code																	
HumanEval	164	1	0	1	0	14	0	60	1	53	1	55	1	65	0	77	0
HumanEval+	164	1	0	1	0	14	0	60	1	53	1	55	1	65	0	77	0
LiveCodeBench	880	2	0	2	0	188	0	241	0	221	0	274	0	167	0	168	0
SWE-bench	500	5	0	26	0	29	1	38	1	32	0	34	2	33	0	28	0
MBPP	500	16	0	64	2	87	5	58	7	89	6	75	9	76	5	84	7
Commonsense Understanding																	
ARC-Challenge	1000	6	14	4	341	18	119	29	40	27	31	26	38	28	42	29	48
ARC-Easy	1000	15	15	5	317	31	54	30	95	31	55	37	55	28	61	31	62
CSQA	1000	0	1	1	10	1	1	0	1	0	2	0	1	0	0	0	1
HellaSwag	1000	27	0	341	0	36	0	29	0	34	0	29	0	30	0	28	1
OpenbookQA	500	2	54	3	78	5	73	2	151	2	66	3	67	3	66	2	61
Social IQa	1000	0	0	2	5	2	2	1	44	2	2	2	3	3	2	0	1
WinoGrande	1000	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
Reading Comprehension																	
CoQA	500	189	40	188	92	121	37	113	44	123	43	134	36	123	38	125	44
SQuAD	1000	16	28	9	401	11	27	12	330	15	101	13	15	12	20	11	85

Table 7: Detailed benchmark contamination results. For test sets with more than 1000 entries, we randomly downsample to 1000.

H Contamination Examples

Figure 6 to Figure 12 shows example dirty entries in seven benchmarks and its contamination source retrieved from one of three corpus using INFINI-GRAM MINI. We also present the original webpage that is responsible for the contamination.

I Dirty Entry Categorization using LLM-as-a-judge

We use gpt-4o-mini to categorize all dirty entries into one of four categories described in §4.3. For each dirty entry, we extract the 50-character substring that has least occurrence in the corpus. We then retrieve text snippet with context length of 400 characters (850 characters in total). We prompt gpt-4o-mini by providing both the dirty entry and the text snippet and let the model decide which category the dirty entry belongs. We use the following prompt:

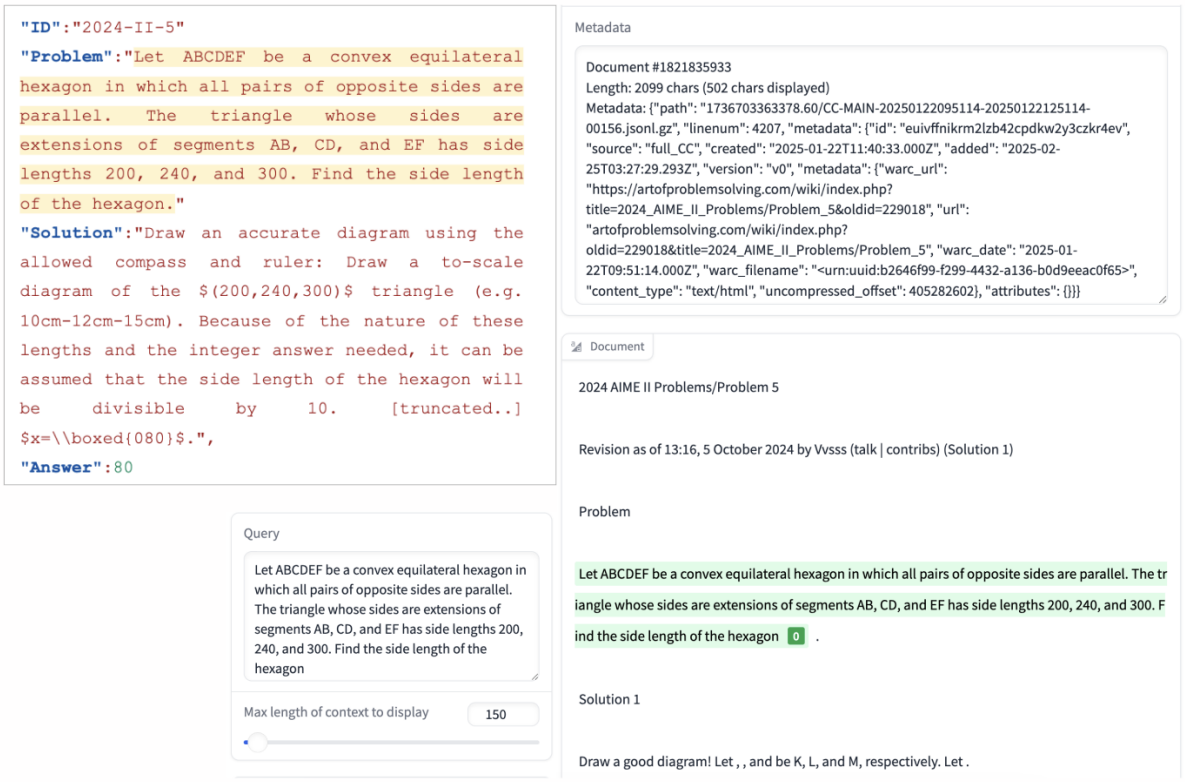
You are an expert in evaluating benchmark contamination. Given an entry and detected overlap in the corpus, categorize how the entry is contaminated in corpus into one of four categories:
1. You can find exact match of question stem and the correct answer (correct choice if multiple choices, or answer matching exactly the answer field) in corpus.
2. You can find exact match of question stem, and the correct answer appears, though not in exact match.
3. You can find exact match of question stem, but you cannot find the correct answer in any form.
4. False positive
Output only the category number.

J Interface of the Benchmark Contamination Monitoring System

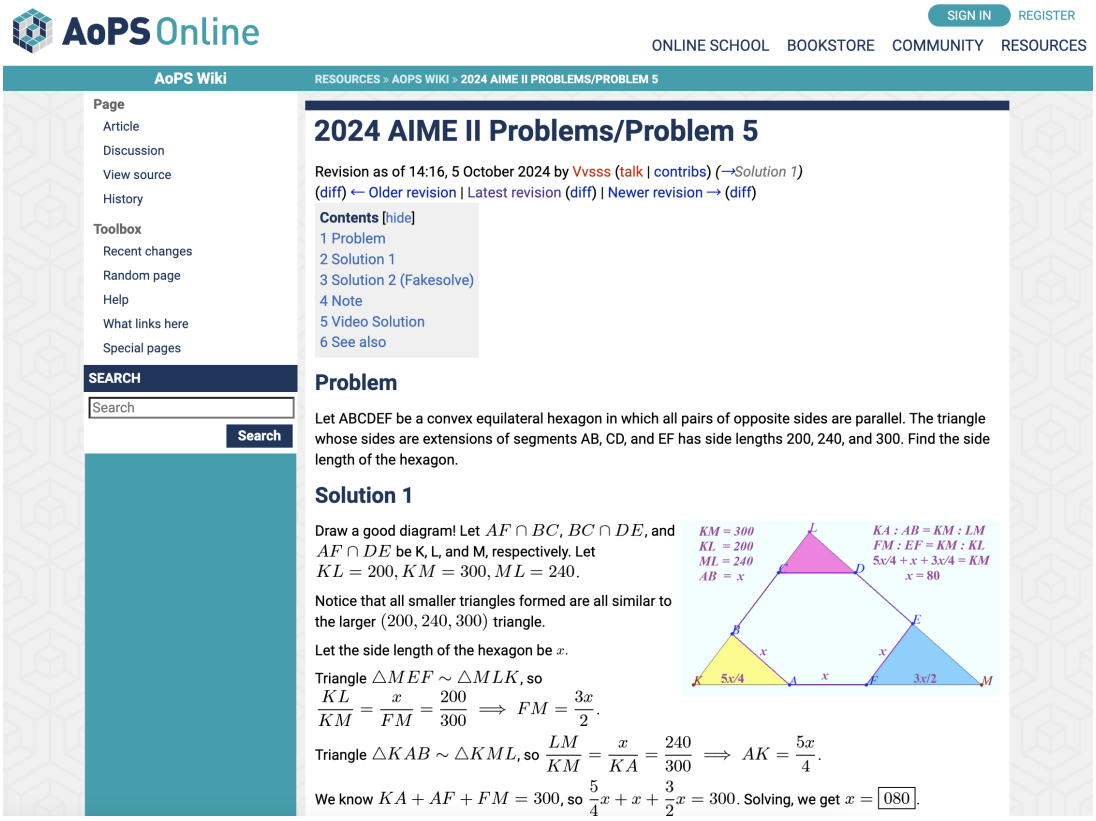
Figure 13 shows the interface of the system. It consists of two tabs: (1) Benchmark Contamination Bulletin, and (2) submission page for community to contribute. Benchmarks analyzed in this paper are reported in "core" table, and submitted benchmarks will be added to "community" table.

K License of Corpora Used in the Paper

Pile and DCLM are licensed under the MIT License. Common Crawl is licensed under its customized Limited License. We followed the listed intended use.

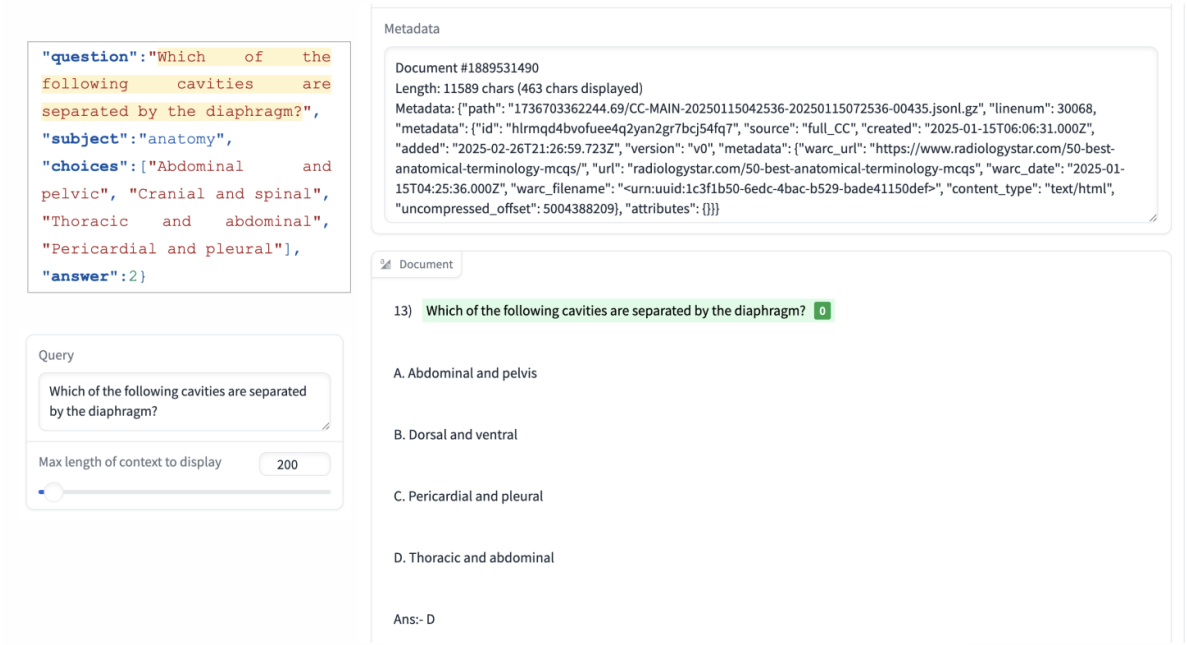


(a) **Left upper:** An entry in the AIME-2024 benchmark. **Right:** A document contaminating this entry, retrieved from CC-2025-05 by INFINI-GRAM MINI. This example belongs to Category 1, where the correct answer can be found in the document.

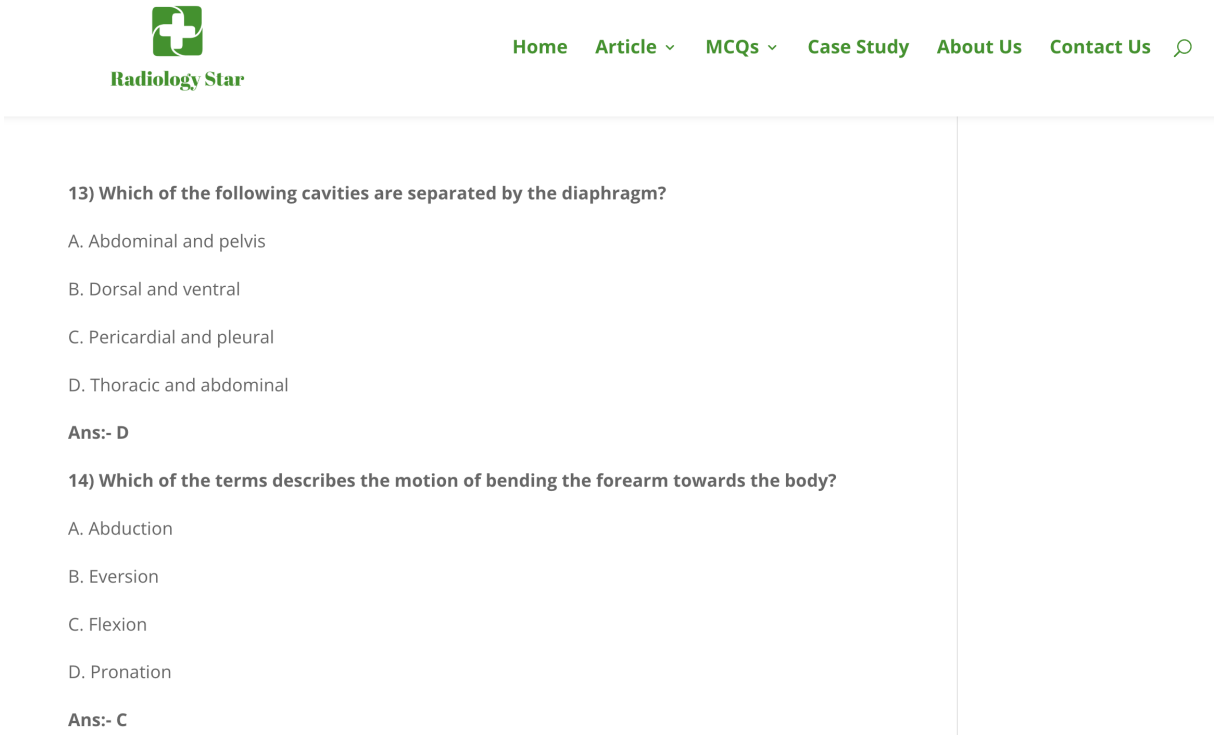


(b) The original webpage responsible for the contamination.

Figure 6: AIME dirty entry example in CC-2025-05. The contamination source is the official AOPS website, where AIME exams are published.

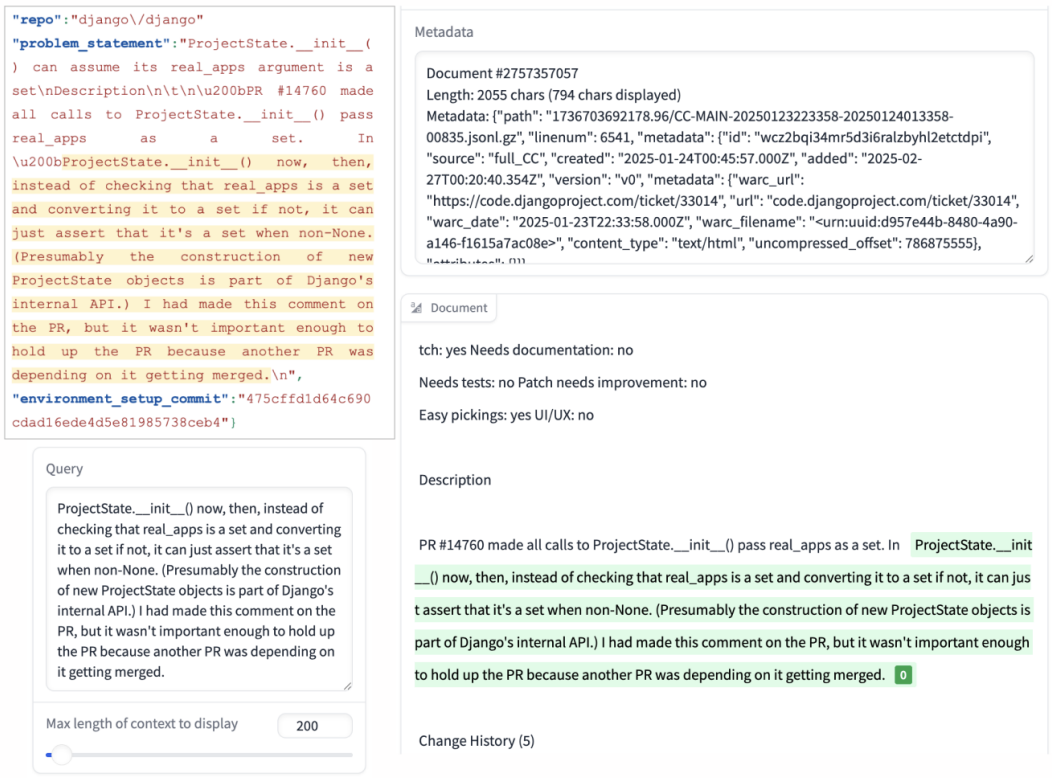


(a) **Left upper:** An entry in the MMLU benchmark. **Right:** A document contaminating this entry, retrieved from CC-2025-05 by INFINI-GRAM MINI. This example belongs to Category 1, where the correct answer presents (though choices are not in the exact same order).

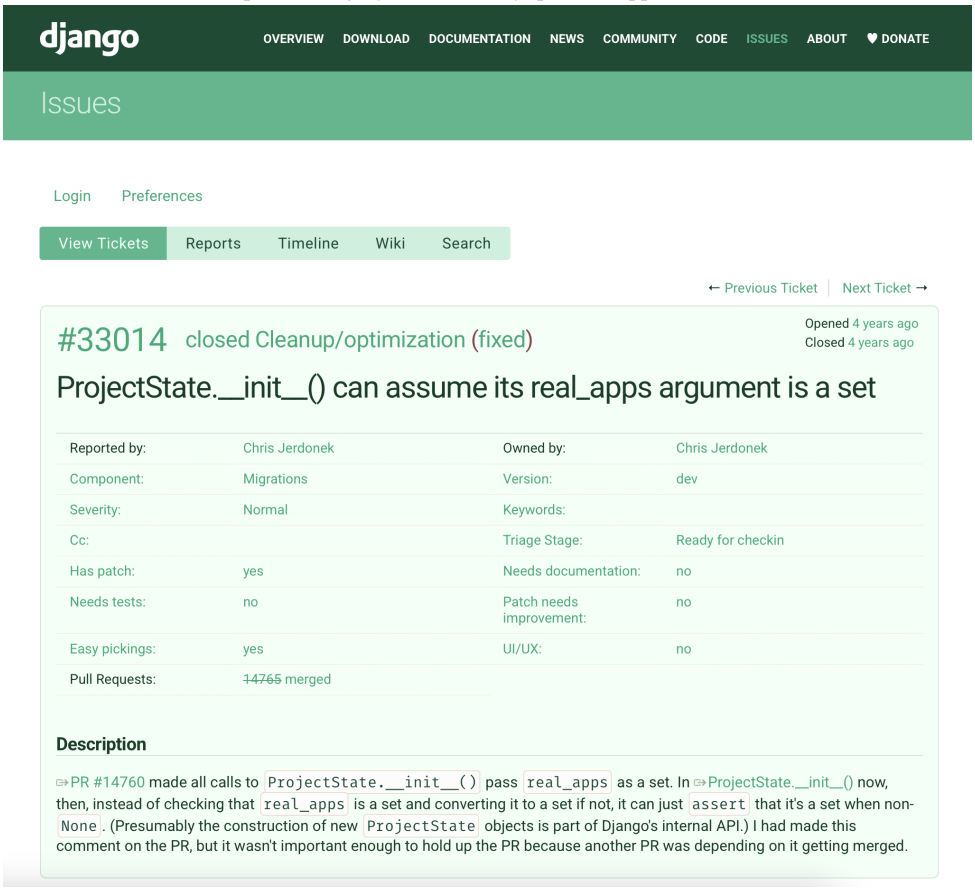


(b) The original webpage responsible for the contamination.

Figure 7: MMLU dirty entry example in CC-2025-05. The contamination source is a website containing multiple-choice question in related fields.

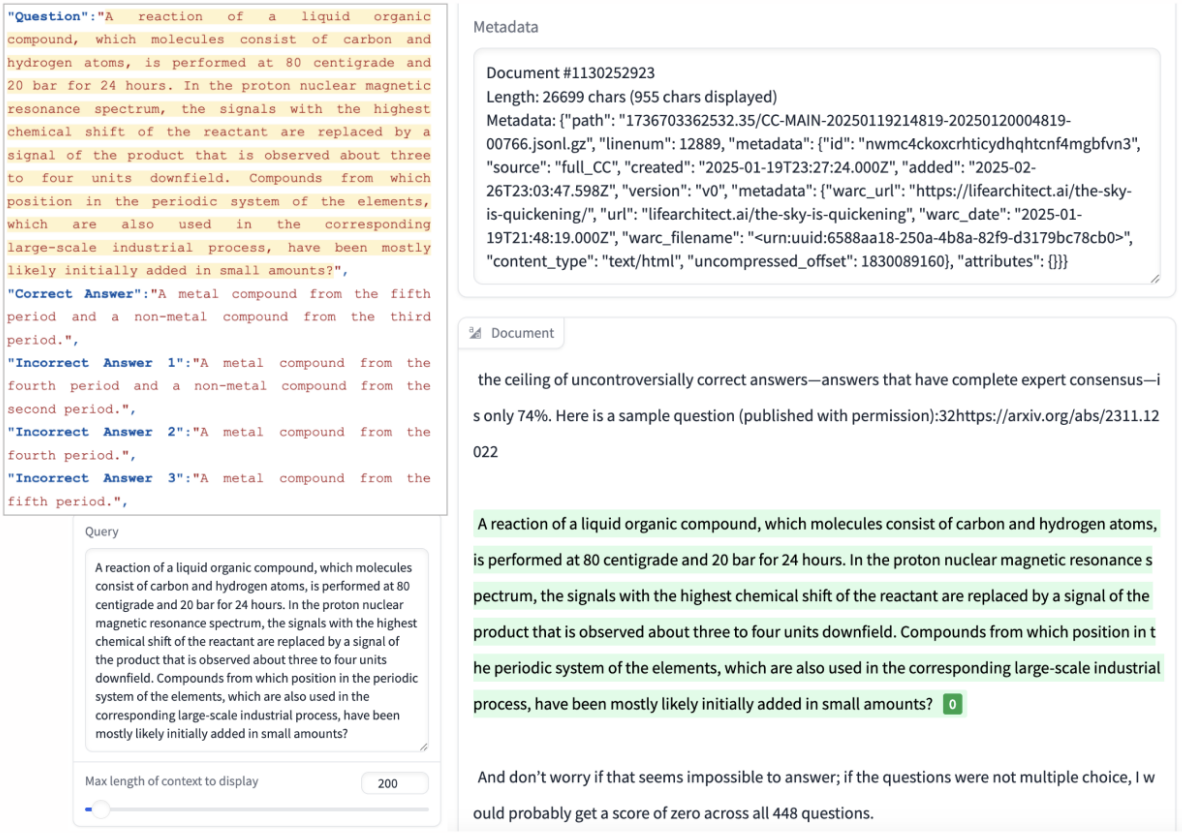


(a) **Left upper:** An entry in the SWE-bench benchmark. **Right:** A document contaminating this entry, retrieved from CC-2025-05 by INFINI-GRAM MINI. This example is Category 3, where only question appears but not the answer.

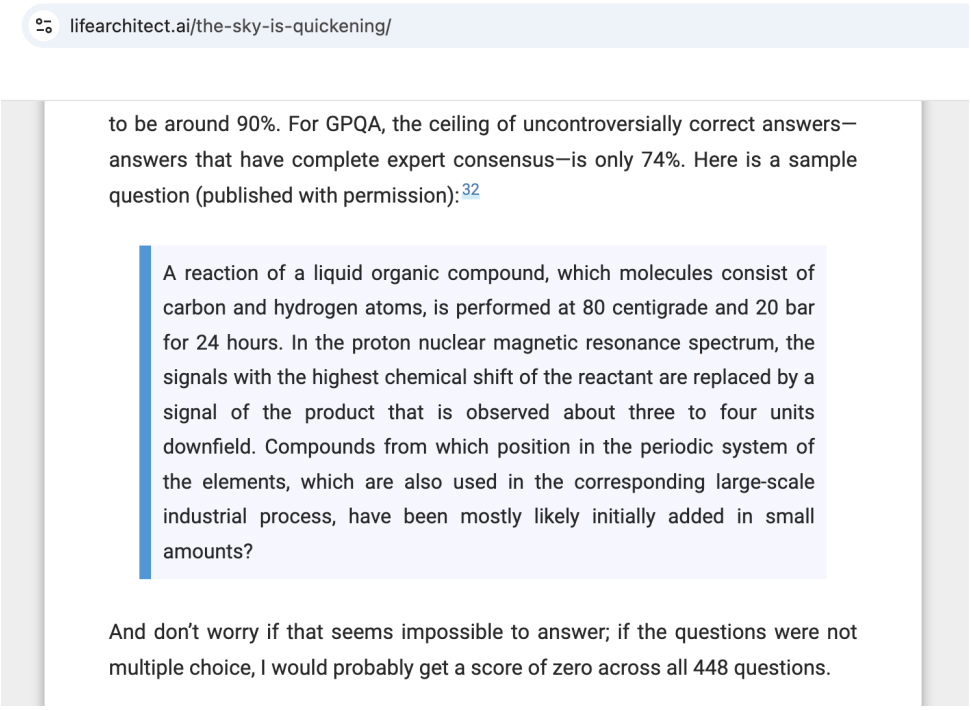


(b) The original webpage responsible for the contamination.

Figure 8: SWE-bench dirty entry example in CC-2025-05. The contamination source is a website recording pull requests for software developing.

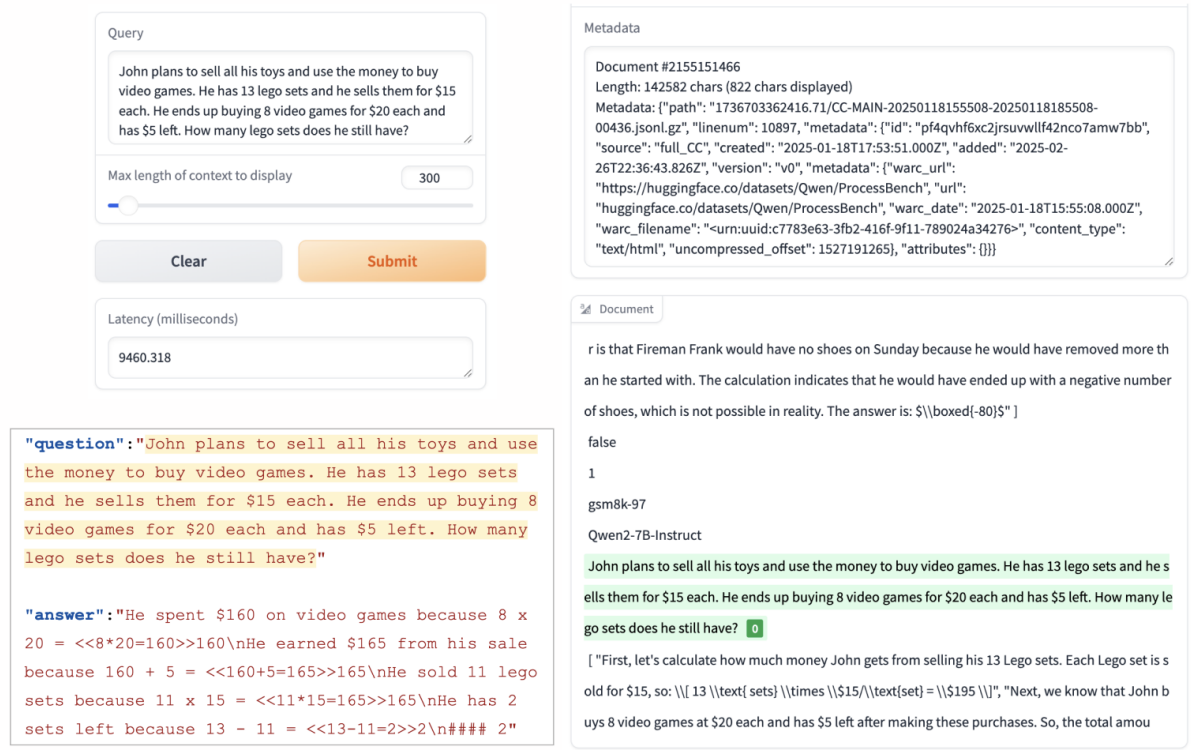


(a) **Left bottom:** An entry in the GPQA benchmark. **Right:** A document contaminating this entry, retrieved from CC-2025-05 by INFINI-GRAM MINI. This example belongs to Category 3, where only the question appears, not the answers.

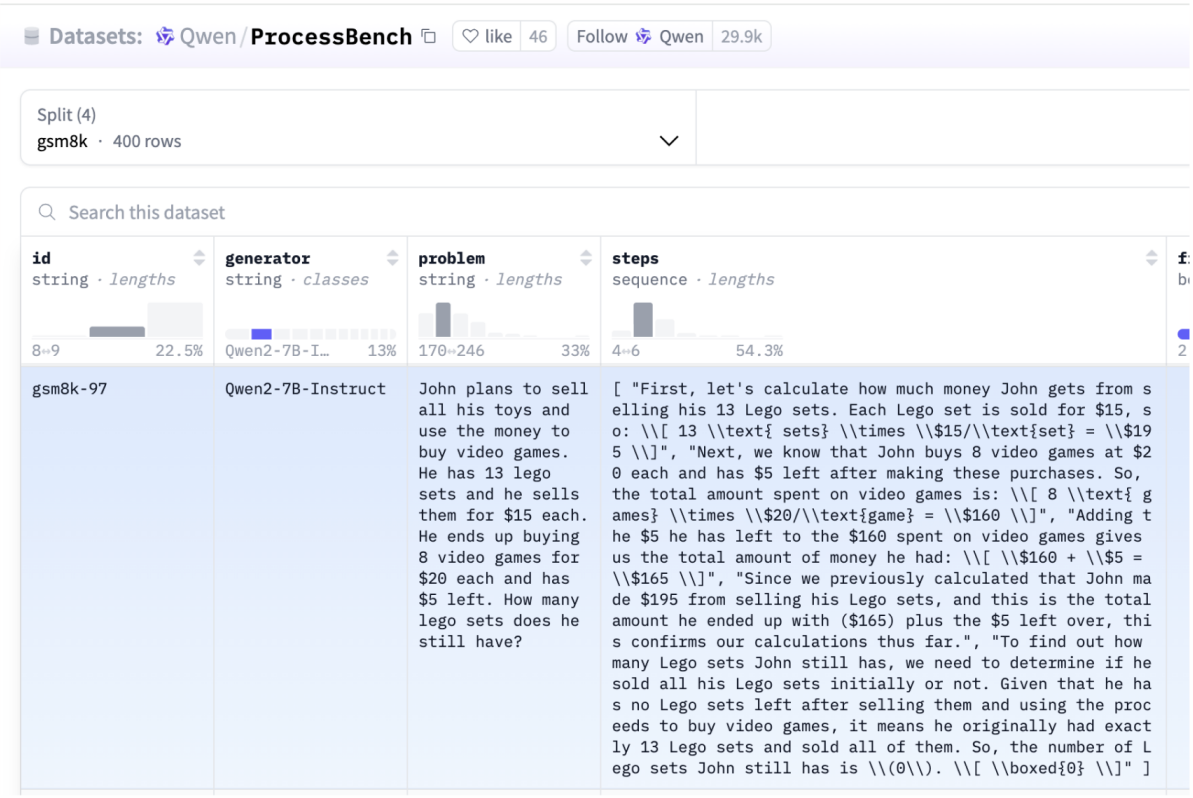


(b) The original webpage responsible for the contamination.

Figure 9: GPQA dirty entry example in CC-2025-05. The contamination source is a blog post citing a test set example.

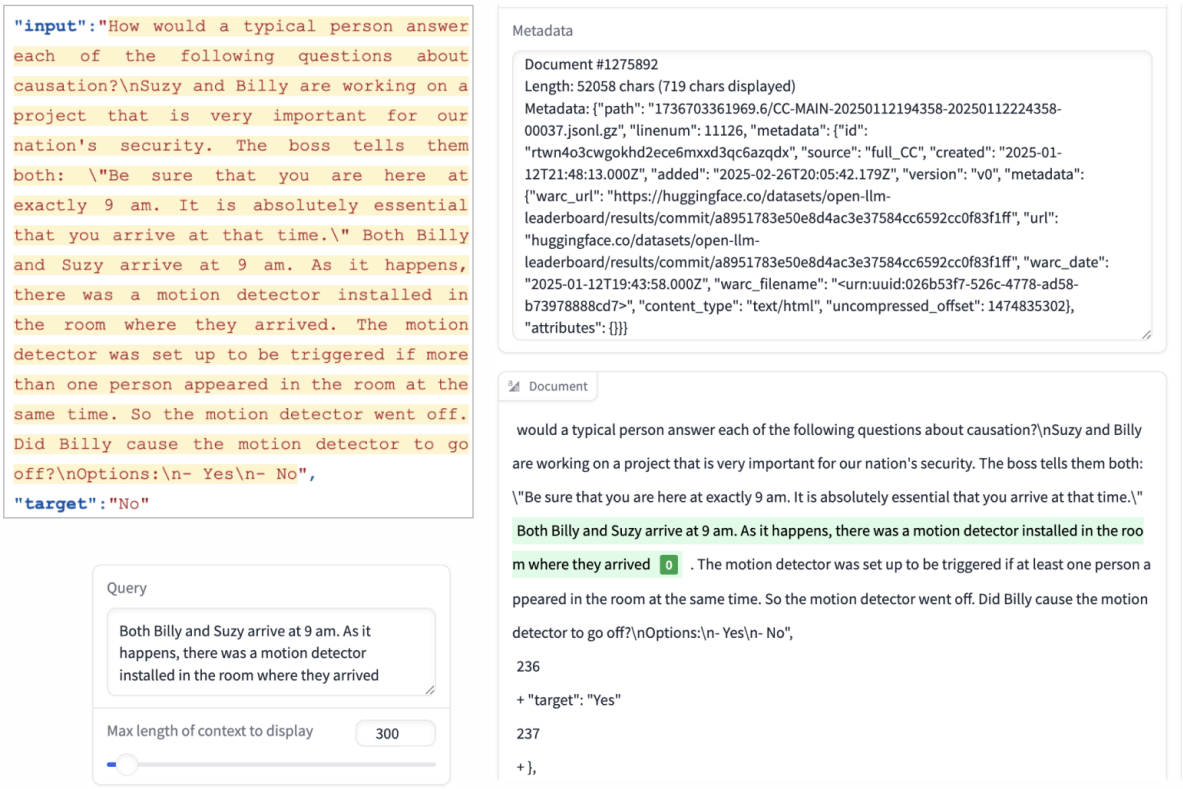


(a) **Left bottom:** An entry in the GSM8K benchmark. **Right:** A document contaminating this entry, retrieved from CC-2025-05 by INFINI-GRAM MINI. This example is Category 3 because the correct answer is not present.

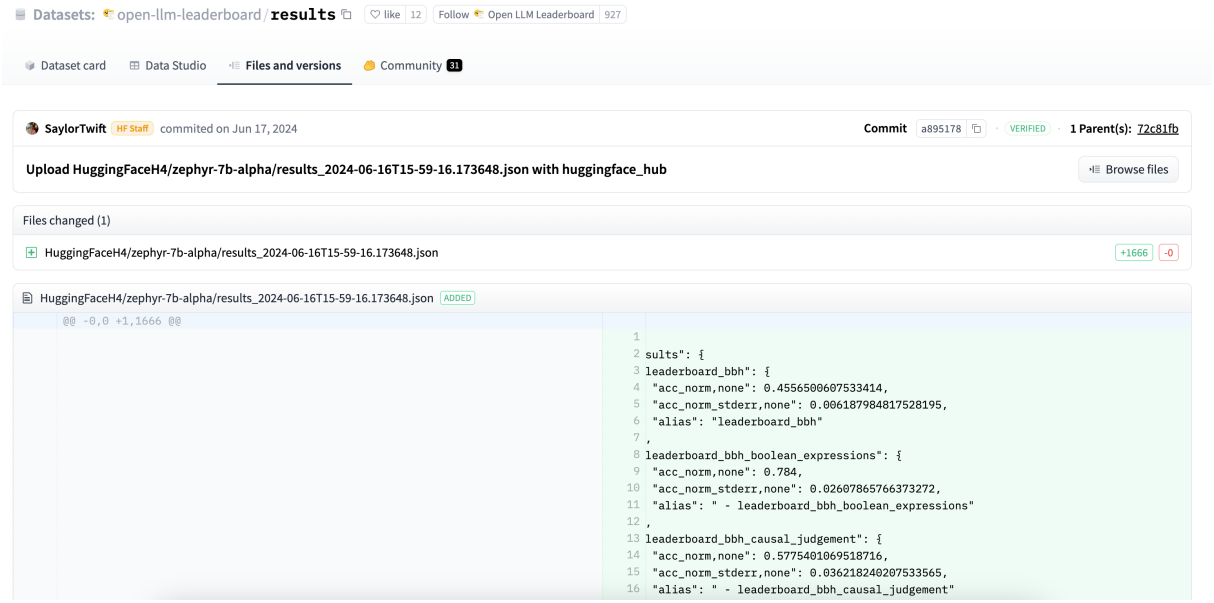


(b) The original webpage responsible for the contamination.

Figure 10: GSM8K dirty entry example in CC-2025-05. The contamination source is a HuggingFace dataset sourcing from GSM8K examples, and is the major source for GSM8K dirty entries. This new dataset contains erroneous "steps" field and final answer to examine LLM's ability to identify errors, so the correct answers do not appear.

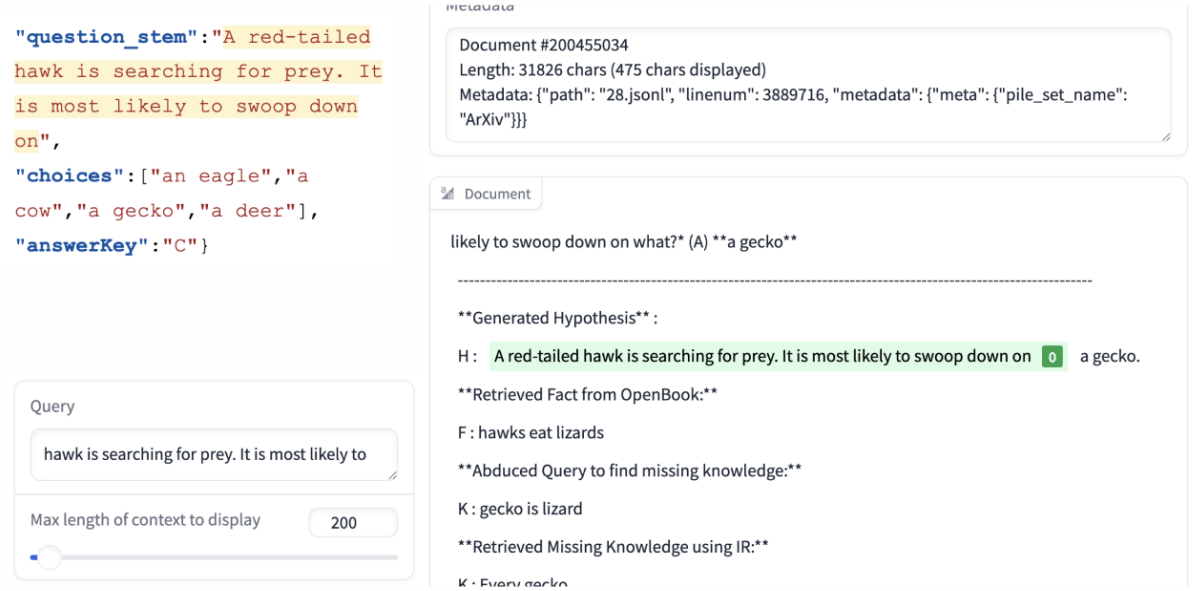


(a) **Left bottom:** An entry in the BigBenchHard benchmark. **Right:** A document contaminating this entry, retrieved from CC-2025-05 by INFINI-GRAM MINI. This example is Category 3 because the correct answer is not present.

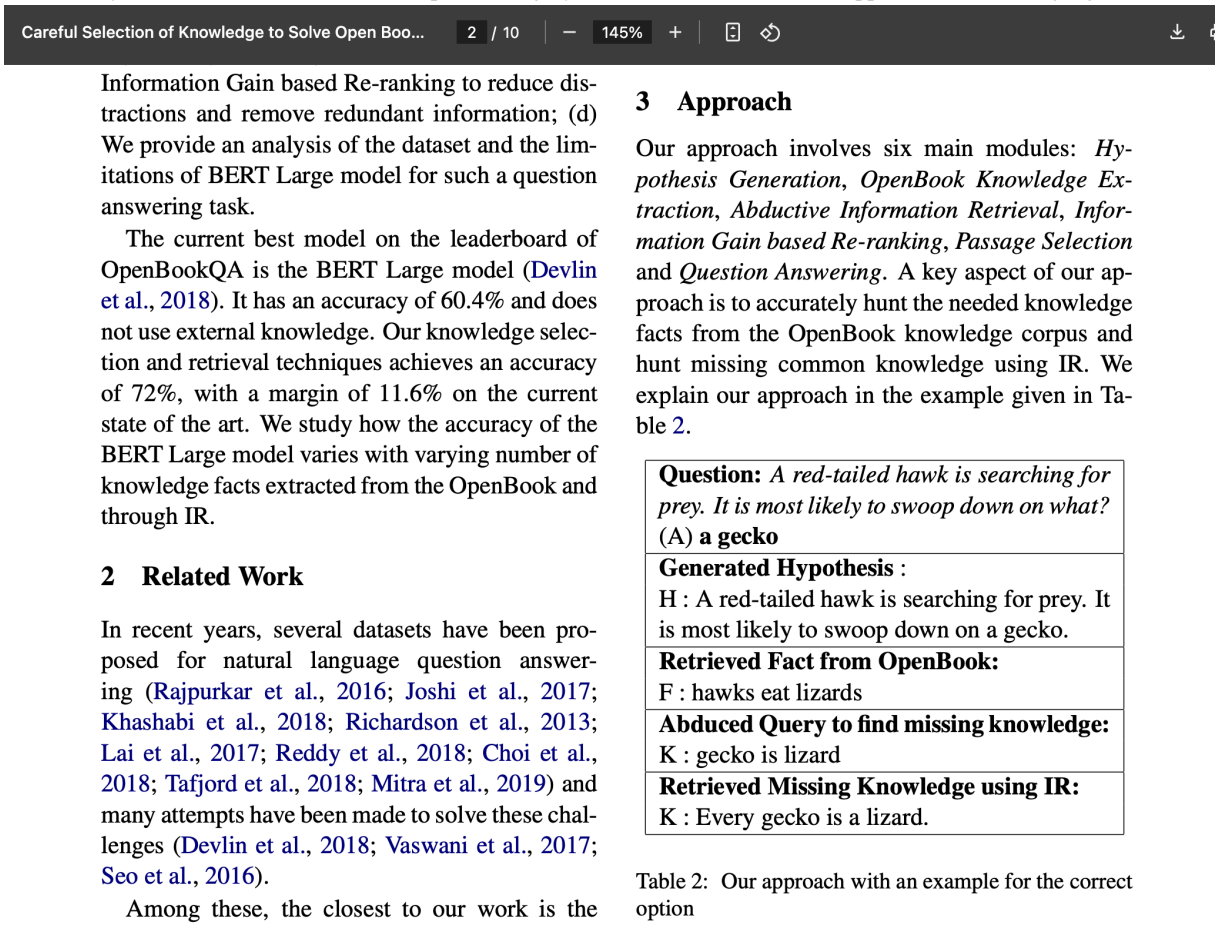


(b) The original webpage responsible for the contamination.

Figure 11: BigBenchHard dirty entry example in CC-2025-05. The contamination source is a HuggingFace commit history that list it as few-shot example.

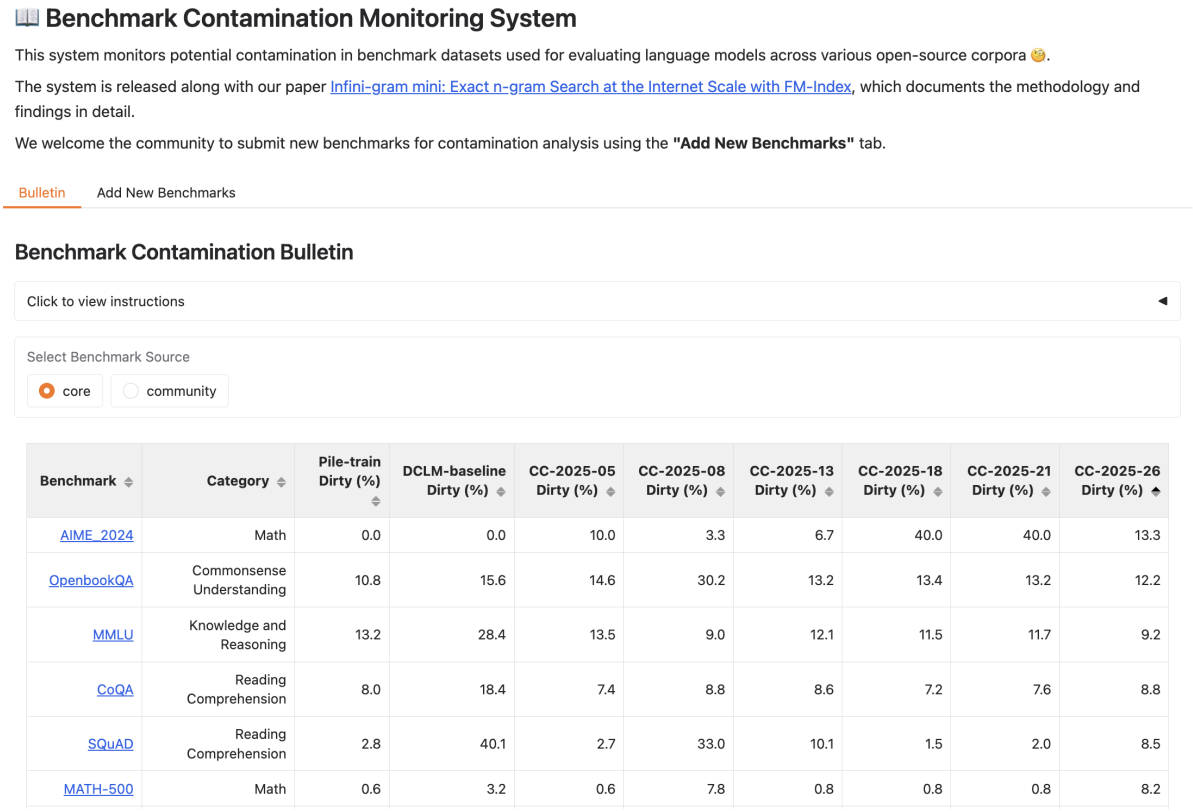


(a) **Left bottom:** An entry in the OpenbookQA benchmark. **Right:** A document contaminating this entry, retrieved from Pile-train by INFINI-GRAM MINI. This example is Category 2 where the correct answer appears in natural language.



(b) The original webpage responsible for the contamination.

Figure 12: OpenbookQA dirty entry example in Pile-train. The contamination source is a paper citing an example from OpenbookQA benchmark.



Benchmark	Category	Pile-train Dirty (%)	DCLM-baseline Dirty (%)	CC-2025-05 Dirty (%)	CC-2025-08 Dirty (%)	CC-2025-13 Dirty (%)	CC-2025-18 Dirty (%)	CC-2025-21 Dirty (%)	CC-2025-26 Dirty (%)
AIME_2024	Math	0.0	0.0	10.0	3.3	6.7	40.0	40.0	13.3
OpenbookQA	Commonsense Understanding	10.8	15.6	14.6	30.2	13.2	13.4	13.2	12.2
MMLU	Knowledge and Reasoning	13.2	28.4	13.5	9.0	12.1	11.5	11.7	9.2
CoQA	Reading Comprehension	8.0	18.4	7.4	8.8	8.6	7.2	7.6	8.8
SQuAD	Reading Comprehension	2.8	40.1	2.7	33.0	10.1	1.5	2.0	8.5
MATH-500	Math	0.6	3.2	0.6	7.8	0.8	0.8	0.8	8.2

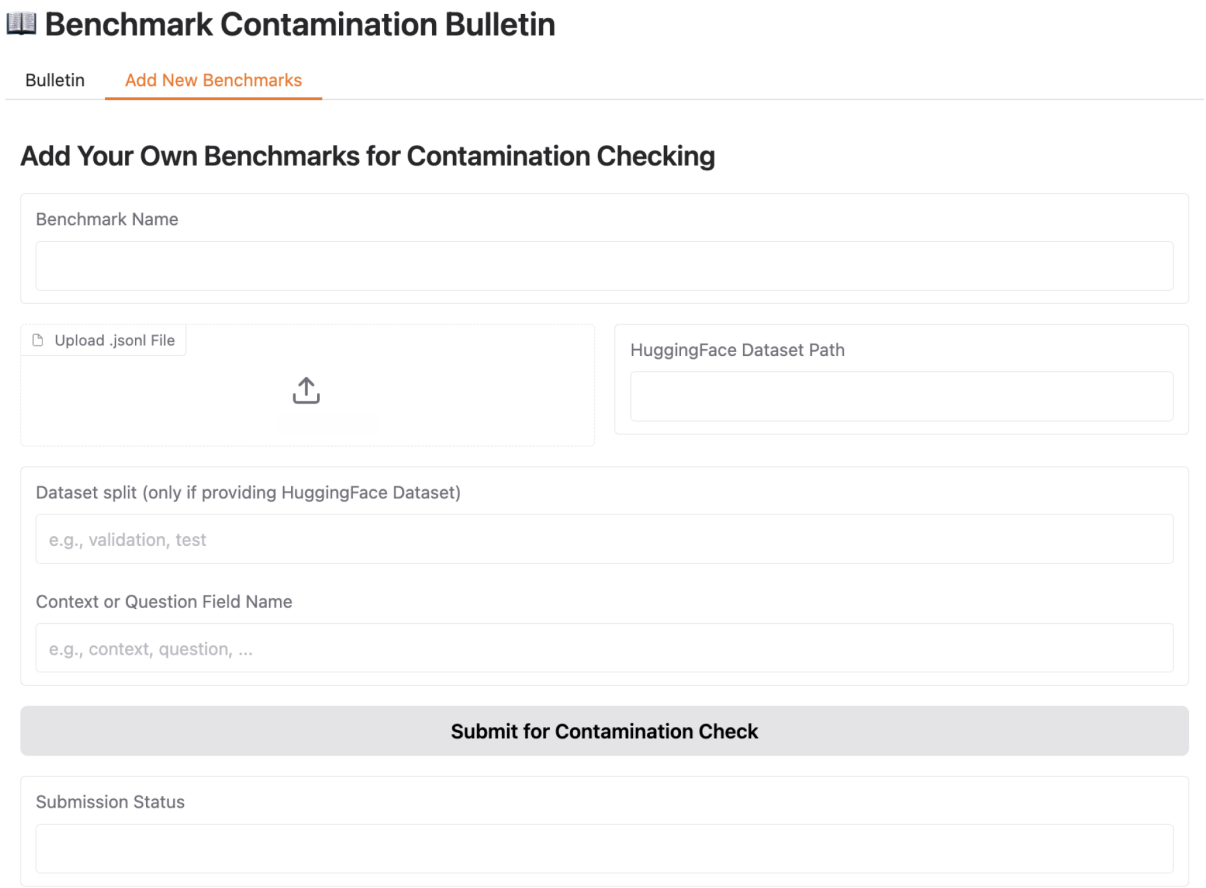


Figure 13: **Upper:** A screenshot of our online Benchmark Contamination Bulletin. **Lower:** We invite the community to add new benchmarks or upload new ones for contamination analysis, which will be added to the bulletin.