OLMOTRACE: Tracing Language Model Outputs Back to Trillions of Training Tokens

Jiacheng Liu[α][ω]   Taylor Blanton[α]   Yanai Elazar[α][ω]   Sewon Min[α][β]

YenSung Chen[α]   Arnavi Chheda-Kothary[α][ω]   Huy Tran[α]   Byron Bischoff[α]   Eric Marsh[α]
Michael Schmitz[α]   Cassidy Trier[α]   Aaron Sarnat[α]   Jenna James[α]   Jon Borchardt[α]
Bailey Kuehl[α]   Evie Cheng[α]   Karen Farley[α]   Sruthi Sreeram[α]   Taira Anderson[α]
David Albright[α]   Carissa Schoenick[α]   Luca Soldaini[α]   Dirk Groeneveld[α]   Rock Yuren Pang[ω]

Pang Wei Koh[α][ω]   Noah A. Smith[α][ω]   Sophie Lebrecht[α]   Yejin Choi[σ]
Hannaneh Hajishirzi[α][ω]   Ali Farhadi[α][ω]   Jesse Dodge[α]

[α]Allen Institute for AI   [ω]University of Washington   [β]UC Berkeley   [σ]Stanford University

Abstract

We present OLMOTRACE, the first system that traces the outputs of language models back to their full, multi-trillion-token training data in real time. OLMOTRACE finds and shows verbatim matches between segments of language model output and documents in the training text corpora. Powered by an extended version of infini-gram (Liu et al., 2024), our system returns tracing results within a few seconds. OLMOTRACE can help users understand the behavior of language models through the lens of their training data. We showcase how it can be used to explore fact checking, hallucination, and the creativity of language models. OLMOTRACE is publicly available and fully open-source.

1 Introduction

Tracing the outputs of language models (LMs) back to their training data is an important problem. As LMs gain adoption in higher-stakes scenarios, it is critical to understand *why* they generate certain responses. However, these modern LMs are trained on massive text corpora with trillions of tokens, which are often proprietary. Fully open LMs (e.g., OLMo; OLMo et al. 2024) enable access to the training data, but existing behavior tracing methods (Koh and Liang, 2017; Khalifa et al., 2024; Huang et al., 2024) have not been scaled to work with this multi-trillion-token setting due to their heavy computation needs.
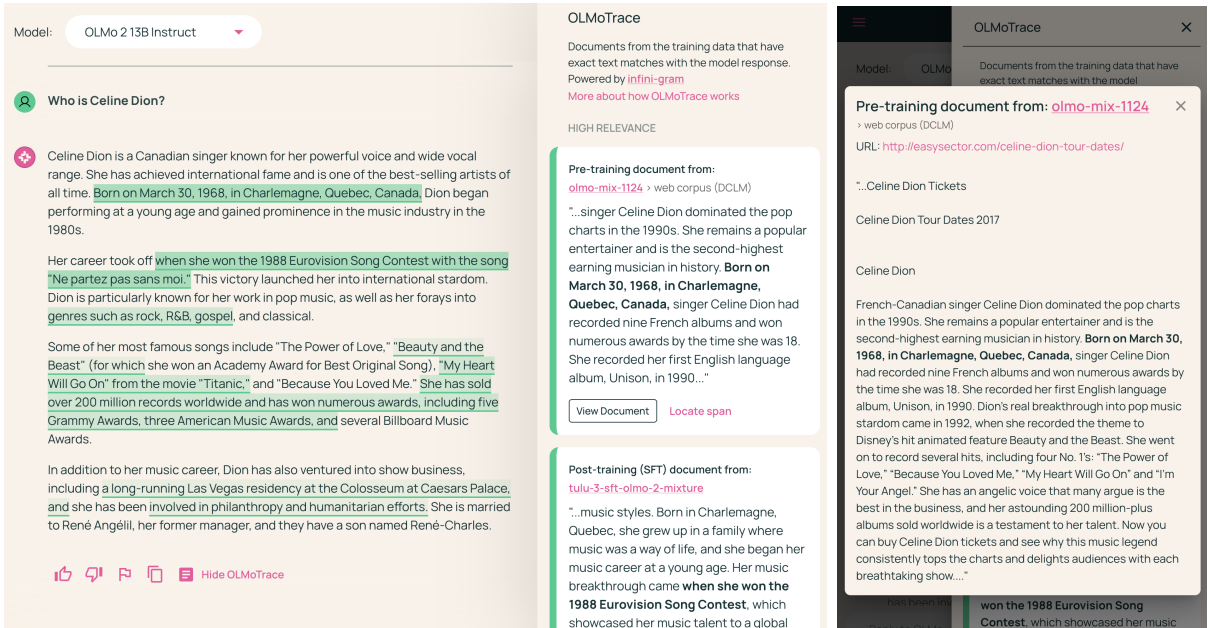
In this paper, we introduce OLMOTRACE, a

Figure 1: OLMOTRACE on Ai2 Playground. **Left:** On a response generated by OLMo, OLMOTRACE highlights text spans found verbatim in the model's training data and shows their source documents. Brighter highlights indicate spans from more relevant training documents, while darker highlights denote less relevant ones. **Right:** When user clicks the "View Document" button, the document is shown with extended context. Try OLMOTRACE at https://playground.allenai.org.

system that traces LM outputs *verbatim* back to its **full** training data and displays the tracing results to LM users in **real time**. Given an LM response to a user prompt, OLMoTRACE retrieves documents from the model's training data that contain exact matches with pieces of the LM response that are long, unique, and relevant to the whole response; see Figure 1 for an example.

The key idea that makes OLMoTRACE fast is that exact matches can be quickly located in a large text corpus if we pre-sort all of its suffixes lexico-graphically. We use infini-gram (Liu et al., 2024) to index the training data and develop a novel parallel algorithm to speed up the computation of matching spans (§3). In our production system, OLMoTRACE completes tracing for each LM response (avg. ∼450 tokens) within 4.5 seconds on average.

The purpose of OLMoTRACE is to give users a tool to explore where LMs *may* have learned to generate certain word sequences, focusing on verbatim matching as the most direct connection between LM outputs and the training data. OLMoTRACE offers an interactive experience, so that users can explore which training documents contain a specific span in the LM response, or inspect a particular document and locate its matching spans in the LM response. We present three case studies for ways to use OLMoTRACE (§5): (1) fact checking, (2) tracing the LM-generated "creative" expressions, and (3) tracing math capabilities. We invite the community to explore more use cases to better understand the relationship between data and models.

OLMoTRACE is available in the Ai2 Playground[1] and supports the three flagship OLMo models (OLMo et al., 2024; Muennighoff et al., 2024) including OLMo-2-32B-Instruct.[2] For each model, it matches against its full training data, including pre-training, mid-training, and post-training. OLMoTRACE can be applied to any LM as long as the service provider has access to its full training data. The core part of the system is open-sourced under the Apache 2.0 license.[3]

2 System Description

Features of OLMoTRACE. Figure 1 shows OLMoTRACE applied to an LM response. When OLMoTRACE is enabled in the Ai2 Playground, it highlights the matching spans in the response, and

Stage	Dataset	# Docs	# Tokens
pre-training	allenai/olmo-mix-1124	3081 M	4575 B
mid-training	allenai/dolmino-mix-1124	81 M	34 B
post-training	SFT & DPO & RLVR	1.7 M	1.6 B
Total		**3164 M**	**4611 B**

Table 1: The full training data of OLMo-2-32B-Instruct, which OLMoTRACE matches against. For mid-training data, we excluded sources that already appeared in the pre-training data, from both the statistics and the index.

shows all training documents matching at least one of these spans in a document panel. OLMoTRACE supports inspecting the documents that match with any particular highlighted span (App. Figure 5, left), and locating the spans enclosed by any particular document (App. Figure 5, right). In the document panel, each document is shown with a snippet of 80 tokens surrounded the matched span; OLMoTRACE allows users to further inspect the document with an extended context (500 tokens).

The training data. The three supported OLMo models are trained on the same pre-training and mid-training data, and slightly different post-training data. OLMoTRACE matches against the entirety of an LM's training data. Table 1 shows links and statistics of the training data of OLMo-2-32B-Instruct, which totals 3.2 billion documents and 4.6 trillion (Llama-2) tokens. The other two OLMo models have similar training data size.

3 The Inference Pipeline

OLMoTRACE takes as input an LM response to a user prompt, and outputs (1) a set of text spans in the LM response, each marked by its starting and ending position, and (2) a list of documents from the training data of this LM, each containing one or more of the aforementioned text spans. The OLMoTRACE inference pipeline consists of the following five steps (illustrated in Figure 2):

Step 1: Find maximal matching spans. We find all maximal spans in the LM output that appear verbatim in the training data. Specifically, we first tokenize the LM output with the Llama-2 tokenizer, and find all spans of the token ID list that satisfy the following criteria:

1. **Existence:** The span appears verbatim at least once in the training data;

2. **Self-contained:** The span does not contain a period token (.) or newline token (\n) unless it appears at the end of the span; and the span does not begin or end with incomplete words;
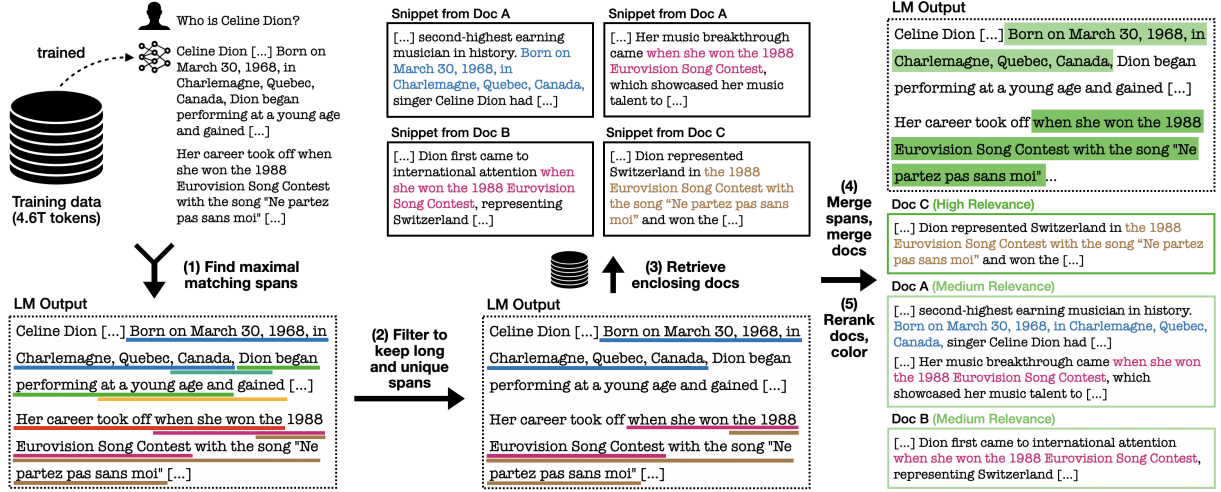
Figure 2: The OLMoTRACE inference pipeline, as described in §3. For better illustration, we slightly adjusted the highlighted spans and document relevance from the actual example.

3. **Maximality:** The span is not a subspan of another span that meets the above two criteria.

This is the most compute-heavy step, since naively we need to enumerate all $O(L^2)$ spans of the LM output (where L is the length of the LM output in tokens, and typically $L \in [10^2, 10^3]$) and scan the entire training data (with N tokens where $N > 10^{12}$). We propose a fast algorithm to compute these maximal matching spans (§3.1), which reduces the time complexity to $O(L \log N)$, and latency to $O(\log N)$ when fully parallelized. After this step, we have a set of relatively long spans that appear in the training data.

Step 2: Filter to keep long and unique spans. To declutter the UI and only show spans that are more likely "interesting", we filter spans to keep ones with the smallest *span unigram probability*, a metric that captures both length and uniqueness. The span unigram probability is defined as the product of unigram probabilities of all tokens in the span, where the token unigram probability derived from statistics of the LM's entire training data. (We pre-compute and cache the token unigram probability for the entire vocabulary.) A lower span unigram probability usually means the span is relatively long and contains non-common tokens. We keep K spans with the smallest unigram probability, where $K = \lceil 0.05 \times L \rceil$.

During development, we tried keeping the longest spans instead of those with smallest span unigram probability. However, we found that ranking with the span length metric leads to worse relevance level on documents retrieved from the filtered spans (see App. §C and Table 3), and thus

we favored the span unigram probability metric. We chose unigram over bigram or trigram because pre-caching them is non-trivial.

Step 3: Retrieve enclosing documents. For each kept span, we retrieve up to 10 document snippets from the training data that enclose this span. Due to the maximality criterion in step 1, most spans appear no more than 10 times. If a span exceeds this limit, we randomly sample 10 to keep retrieval time manageable and avoid UI overload.

Step 4: Merge spans, merge documents. To further declutter the UI, we merge (i.e., take the union of) overlapping spans into a single span to be highlighted in the LM output. Also, if two snippets are retrieved from the same document, we merge them into a single document to be displayed in the document panel.

Step 5: Rerank and color documents by relevance. To prioritize showing the most relevant documents, in the document panel we rank all documents by a BM25 score in descending order. The per-document BM25 score is computed by treating the collection of retrieved documents as a "corpus", and the concatenation of user prompt and LM response as the "query".[4] We use this BM25 score because it has fairly high agreement with human judgment on topical relevance (§4.2), and can be quickly computed using CPUs. Subsequently, we bucket the relevance scores into three levels – high relevance, medium relevance, and low relevance – and display a colored sidebar on each document to represent its relevance level. High relevance are

---

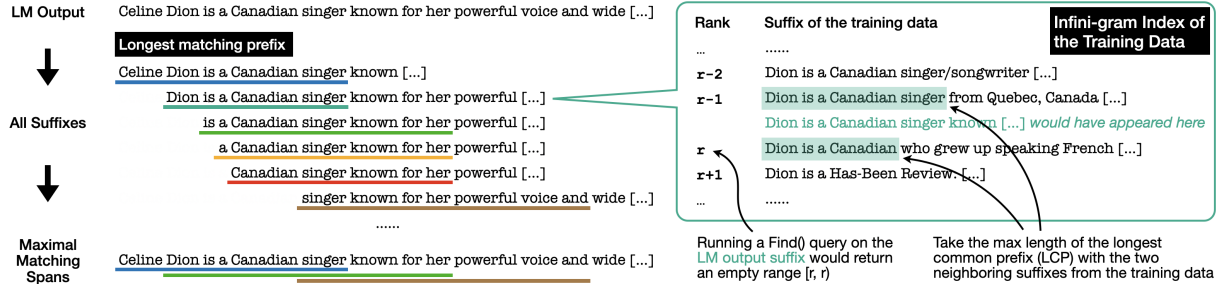[4]We use the implementation in https://github.com/dorianbrown/rank_bm25

Figure 3: Computation of the maximal matching spans (§3.1). For each suffix of the LM output, OLMoTRACE computes its longest matching prefix (color-underlined) with a single FIND query on the infini-gram index of the LM training data. All suffixes of the LM output are processed in parallel. Finally, non-maximal spans are suppressed.

highlighted with the most saturated color, and low relevance with the least saturated color. We also apply this differential coloring on span highlights: a span's relevance level is computed as the maximum relevance level among documents enclosing the span. As a result, users are more likely to find highly relevant documents for spans highlighted with the most saturated color.

3.1 Fast Span Computation

Efficiently identifying all maximal matching spans across multi-trillion-token corpora is a non-trivial challenge. To tackle this, we index the training corpora with infini-gram (Liu et al., 2024) and develop a new parallel algorithm for fast span computation.

Infini-gram. Infini-gram is a text search engine. It supports efficiently counting text queries and retrieving matching documents in massive text corpora with trillions of tokens. To make operations fast, infini-gram indexes text corpora with the suffix array (SA) data structure, and at inference time keeps the huge index files on low-latency SSD disks to avoid loading them into RAM. For OL-MOTRACE, we build an infini-gram index on the tokenized version of the LMs' training data (using the Llama-2 tokenizer). On top of this index, in this work we devise a novel parallel algorithm to compute maximal matching spans with low compute latency (Figure 3); we discuss this algorithm and its implementation below.

Problem analysis. The problem of finding all maximal matching spans can be broken down into two steps: (1) finding the longest matching prefix of each suffix of the LM output; and (2) suppressing the non-maximal spans. This is because starting from each position, there can be at most one span that is a maximal matching span (if there are two, then one is a subspan of the other and thus is not maximal). Now the first step consists of multiple

independent tasks that can be parallelized, and as we will show below, each task can be done with one FIND query. FIND is a core query operation in infini-gram; it returns the segment of SA that corresponds to all occurring positions of a search term in the text corpus. Since in infini-gram, the processing speed of FIND queries is bounded by disk I/O latency and there is a lot of unused throughput, parallelizing these queries can reduce the overall compute latency. In fact, with parallelization, the overall processing speed is bottlenecked by the disk I/O throughput, and thus in our production system we put the index files on high-IOPS SSD disks.

Finding the longest matching prefix of a suffix. With FIND queries, the length of the outputted segment is the count of the search term in the text corpus. Naively, we can run FIND queries on incrementally long prefixes of the LM output's suffix until the count becomes zero (which takes $O(L)$ queries), or we can do binary-lifting + binary-search to reduce to $O(\log L)$ queries. However, we show below that we can do this with one single FIND query ($O(1)$).

We use the fact that when the search term does not exist in the text corpus, FIND would return a 0-length segment (delimited by a left-inclusive starting position and a right-exclusive ending position that are identical), where the previous (or next) SA element corresponds to the suffix in the text corpus that lexicographically precedes (or succeeds) the search term (see Figure 3). Consequently, the suffix in the text corpus that shares the longest common prefix (LCP) with the search term must come from one of these two neighboring suffixes, and inspecting these two suffixes would tell us the length of the longest matching prefix for this search term. Therefore, we can simply run FIND once with the entire LM output's suffix to find out its longest matching prefix.

In reality, we shard the infini-gram index because each shard is limited to 500B tokens. In case there are multiple shards, we run FIND on each one in parallel, and take the maximum of LCP length from all shards.

Note that to retrieve documents containing the longest matching prefix, we need to run a second FIND query to locate all its occurrences in the SA. In practice, we run this query immediately after the first one to leverage temporal locality in the disk cache.

Suppressing non-maximal spans. We gather the longest matching prefix of all suffixes into a list of spans. These spans begin at monotonically increasing positions, but end at monotonically non-decreasing positions that may still be identical, and thus there may still be non-maximal spans (see Figure 3). To remove the non-maximal spans, we make a pass on the spans in increasing order of the beginning position, and only keep spans with an ending position larger than that of the previously encountered spans.

4 Benchmarking and Analysis

4.1 Benchmarking Inference Latency

We host the inference pipeline on a CPU-only node in the Google Cloud Platform. The node has 64 vCPUs and 256GB RAM, and we store the index files on 40TB of SSD disks. See App. §B for a detailed description of our production system.

We empirically benchmark the latency of the most compute-intensive part of our inference pipeline: steps 1–3, which include computing maximal matching spans and retrieving document snippets. We collect 98 conversations from internal usage of OLMo models in the Ai2 Playground, and send them to OLMoTRACE. On average, each LM response has 458 tokens, and the OLMoTRACE inference latency per query is **4.46 seconds**. This is in line with our disk I/O analysis in App. §B. The low inference latency allows us to present OLMoTRACE results to users in real time and offer a smooth user experience.

4.2 Evaluating Document Relevance

To improve the user experience, OLMoTRACE reranks the retrieved documents by relevance to the LM output. We conducted a study to evaluate the relevance level of the top displayed documents. We first composed a rubric for scoring document relevance on a 0–3 Likert scale (App. Table 2, left),

and asked a human expert to annotate the top-5 displayed documents for each of the 98 threads used in §4.1 according to this rubric. This human evaluation round was done with OLMoTRACE results under a different hyperparameter setting than our final setting, and we later improved the setting under the guidance of LLM-as-a-Judge evaluation. The first document displayed in each thread received an average relevance score of 1.90, and the top-5 documents scored an average of 1.43 (App. Table 3). We then switched to LLM-as-a-Judge evaluation (Zheng et al., 2023) with gpt-4o, and found that it mostly agrees with human evaluation (with a Spearman correlation coefficient of 0.73). LLM-as-a-Judge assigned slightly lower scores overall, with average scores of 1.73 and 1.28 on first and top-5 documents, respectively. We then used LLM-as-a-Judge to guide the tuning of several hyperparameters in OLMoTRACE, and our final setting achieved average LLM-as-a-Judge scores of 1.82 on first documents and 1.50 on top-5 documents. See App. §C for additional details on relevance evaluation and hyperparameters tuning.

5 Case Studies

We envision that researchers and the general public can use OLMoTRACE in many ways to understand the behavior of LMs. Below we discuss three example use cases, and we invite the community to explore more use cases.

Fact checking. If the LM states a fact, users may be able to fact-check the statement against its training data. In Figure 4(a), OLMo outputs "*The space needle was built for the 1962 World Fair,*". OLMoTRACE indicates through the highlight that this span of tokens appears verbatim in the training data and shows the corresponding documents (the screenshot captured one of the 10). For most documents from the pretraining data (like this one), users can click on the "View Document" button and find the URL to the original webpage where this document was crawled. Inspecting the document context and source helps users make a more informed judgment on the factuality of the statement, as words can be misleading out of context, and some web sources may be unreliable.

Tracing "creative" expressions. While LMs can be creative in piecing expressions together in new ways, seemingly novel expressions may not be truly new, as LMs may have learned them during training. In such cases, OLMoTRACE reveals the po-
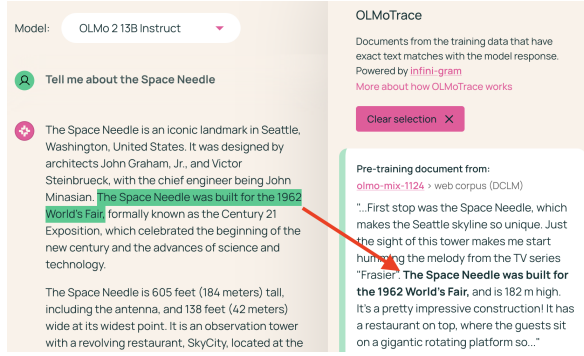
(a) **Fact checking:** Inspecting the document (and its source URL) helps verify the factual claim made in the span.
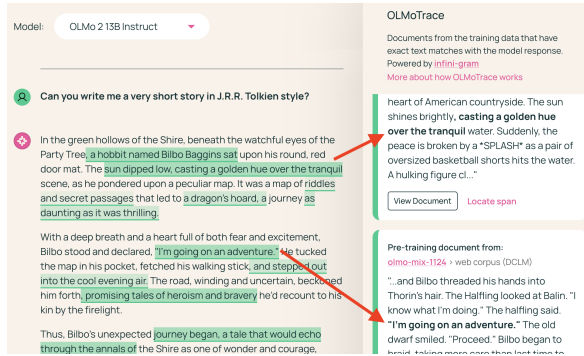
(b) **Tracing "creative" expressions:** Matching spans reveal potential source of LM-generated "creative" expressions.
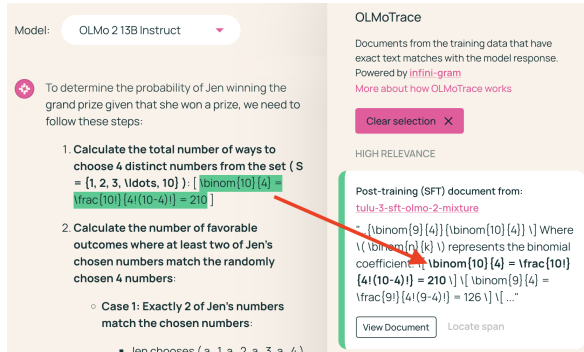
(c) **Tracing math capabilities:** Arithmetics carried out by LMs can be traced verbatim to their training data.

Figure 4: Example use cases of OLMoTRACE. In (a) and (c), one span has been selected to inspect its enclosing documents; the selected span is colored with solid green while other span highlights are hidden.

tential source of LM-generated expressions. In Figure 4(c), OLMo outputs a story in the Tolkien style, and OLMoTRACE highlights verbatim matches with the training data, e.g., "*I'm going on an adventure*" matches with the shown document, which is a fan fiction about the Hobbits.

Tracing math capabilities. OLMoTRACE helps understanding how LMs learned to carry out arithmetic operations and solve math problems. In Figure 4(d), OLMo correctly answers Problem 4 from the AIME 2024 I exam (a combinatorics problem).

OLMoTRACE shows that the calculation step, "$\binom{10}{4} = \frac{10!}{4!(10\text{-}4)!} = 210$" appears verbatim in the post-training dataset.

6 Related Work

Comparison with RAG. Retrieval-augmented generation (RAG) systems retrieve relevant documents from a database and condition the LM generation on the retrieved documents. Examples of them include Bing Chat, Google AI Overview, and Perplexity AI. Despite looking similar, OLMoTRACE is fundamentally different from RAG: OLMoTRACE retrieves documents *post-hoc* and does not intervene with the LM generation. The purpose of retrieval in OLMoTRACE is to show the connection between an LM's output and its training data, not to improve the generation itself.

Comparison with search engines. Traditional search engines (e.g., Google) retrieve documents from their web index and may drop results for efficiency or profitability. OLMoTRACE faithfully retrieves all matches in an LM's training data.

Tracing LM generation into training data. One classical approach to trace LM generation is using influence functions (Koh and Liang, 2017; Han et al., 2020; Han and Tsvetkov, 2022), which leverage gradient information to find influential training examples for a given test example. While effective on a small scale, influence functions are intractable for trillion-token training data due to their high computational cost. Our work takes a different approach: we directly retrieve similar training examples by lexical overlap, with the heuristic that such training examples are likely to be influential for the given output.

Other types of tracing. Khalifa et al. (2024) train LMs to cite documents from the pretraining data, which is an intervention on the training process of LMs. Some work traces LM behavior into sources other than the training data. Huang et al. (2024) extend RAG to have LMs cite retrieved documents provided in-context, whereas Chuang et al. (2025) train LMs to cite content from the long context provided to the LM at inference time. Gao et al. (2022) retrieve supporting evidence for LM generations from Google Search; the Gemini App has a "double-check response" feature that highlights parts of the LM response and shows similar results from Google Search, which is updated in real time and thus not identical to Gemini's training data, making it less useful for scientific exploration.

Limitations

OLMOTRACE finds lexical, verbatim matches between an LM's output and its training data. The retrieved documents should not be interpreted as having a causal effect on the LM output, or as supporting evidence or citations for the LM output. In addition, OLMOTRACE could make potentially problematic contents in the LM training data more easily exposed, including copyright, PII (personal identifiable information), and toxicity.

Acknowledgements

Author Contributions

Core contributors: Jiacheng Liu, Taylor Blanton

Research: Yanai Elazar, Sewon Min, Luca Soldaini, Dirk Groeneveld, Rock Yuren Pang

Engineering: YenSung Chen, Arnavi Chheda-Kothary, Huy Tran, Byron Bischoff, Eric Marsh

Design: Cassidy Trier, Aaron Sarnat, Jenna James, Jon Borchardt

UX: Bailey Kuehl, Evie Cheng

PM: Karen Farley, Sruthi Sreeram, Taira Anderson

Legal: Will Smith, Crystal Nam

Comms: David Albright, Carissa Schoenick

Advising: Jesse Dodge, Ali Farhadi, Hannaneh Hajishirzi, Yejin Choi, Sophie Lebrecht, Noah A. Smith, Pang Wei Koh

References

Yung-Sung Chuang, Benjamin Cohen-Wang, Shannon Zejiang Shen, Zhaofeng Wu, Hu Xu, Xi Victoria Lin, James Glass, Shang-Wen Li, and Wen tau Yih. 2025. Selfcite: Self-supervised alignment for context attribution in large language models.

Luyu Gao, Zhuyun Dai, Panupong Pasupat, Anthony Chen, Arun Tejasvi Chaganty, Yicheng Fan, Vincent Zhao, N. Lao, Hongrae Lee, Da-Cheng Juan, and Kelvin Guu. 2022. Rarr: Researching and revising what language models say, using language models. In *Annual Meeting of the Association for Computational Linguistics*.

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.

Xiaochuang Han, Byron C. Wallace, and Yulia Tsvetkov. 2020. Explaining black box predictions and unveiling data artifacts through influence functions. *ArXiv*, abs/2005.06676.

Chengyu Huang, Zeqiu Wu, Yushi Hu, and Wenya Wang. 2024. Training language models to generate text with citations via fine-grained rewards. In *Annual Meeting of the Association for Computational Linguistics*.

Muhammad Khalifa, David Wadden, Emma Strubell, Honglak Lee, Lu Wang, Iz Beltagy, and Hao Peng. 2024. Source-aware training enables knowledge attribution in language models. *ArXiv*, abs/2404.01019.

Pang Wei Koh and Percy Liang. 2017. Understanding black-box predictions via influence functions. In *International Conference on Machine Learning*.

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

Niklas Muennighoff, Luca Soldaini, Dirk Groeneveld, Kyle Lo, Jacob Daniel Morrison, Sewon Min, Weijia Shi, Pete Walsh, Oyvind Tafjord, Nathan Lambert, Yuling Gu, Shane Arora, Akshita Bhagia, Dustin Schwenk, David Wadden, Alexander Wettig, Binyuan Hui, Tim Dettmers, Douwe Kiela, and 5 others. 2024. Olmoe: Open mixture-of-experts language models. *ArXiv*, abs/2409.02060.

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. 2024. 2 olmo 2 furious. *ArXiv*, abs/2501.00656.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Haotong Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *ArXiv*, abs/2306.05685.

A More Screenshots of OLMoTRACE

Figure 5 is an extension of Figure 1 and shows screenshots of OLMoTRACE when user interacts with the UI.

B Production System Setup

We host the production system of OLMoTRACE on Google Cloud Platform. We store the infini-gram index files on pd-balanced SSD disks with up to 80,000 read IOPS per VM. To achieve the maximum IOPS, we mount the disks to an N2 VM with 64 vCPUs. We keep the index files on disk for inference to avoid needing an unrealistic amount of RAM, and allocate 256GB RAM for the VM to fit the fully-materialized page tables of the mmap'ed index files (0.2% the full file size). To enhance system availability and throughput, we keep 2 VM replicas and multi-mount the same disks to both VMs, and we keep OLMoTRACE processing in separate workers.

In the infini-gram engine, we turn off prefetching (setting all prefetch depth to 0) because it would slow down the overall inference. (Prefetching reduces the latency of single query at the cost of performing more disk read ops speculatively, which is not beneficial when disk I/O throughput is the bottleneck.) We also implemented a batched version of GETDOCBYPTR query to retrieve multiple training documents in parallel and reduce latency.

Disk I/O analysis. Here we compute the number of random disk reads needed in the span computation step. For each beginning token position in the LM output, we need to find its longest matching prefix which means 2 FIND queries; effectively this only counts as 1 FIND query because most disk reads are shared and cached. Each Find takes 2 binary searches over the SA, but our implementation combines them into 1 binary search. Each binary search takes $\log N \approx 40$ steps, where each step takes 2 disk reads – one on the SA and one on the text corpus. In practice we partition the training data into 12 shards, so multiply the number of disk reads by 12. This means for each token in the LM output, we need $40 \times 2 \times 12 = 960$ disk reads. Given that our disks have 80,000 IOPS, OLMoTRACE can processes, for example, a 100-token LM output within 1.2 seconds.

C More Details on Document Relevance Evaluation

For human evaluation, we used the rubric in Table 2 (left). For LLM-as-a-Judge evaluation, we used the prompt in Table 2 (right), which closely follows the rubric, and gpt-4o-2024-08-06 as the judge model.

Table 3 shows the evaluation results. We report 4 metrics: average score among the first and top-5 displayed documents, and the percentage of relevant documents among the first and top-5 displayed documents. We report different settings in reversed chronological order. For the last row, we used an early hyperparameter setting of OLMoTRACE with human evaluation, and for the second-last row we used the same hyperparameter setting but switched to LLM-as-a-Judge. The early hyperparameter setting differs from our final setting in that:
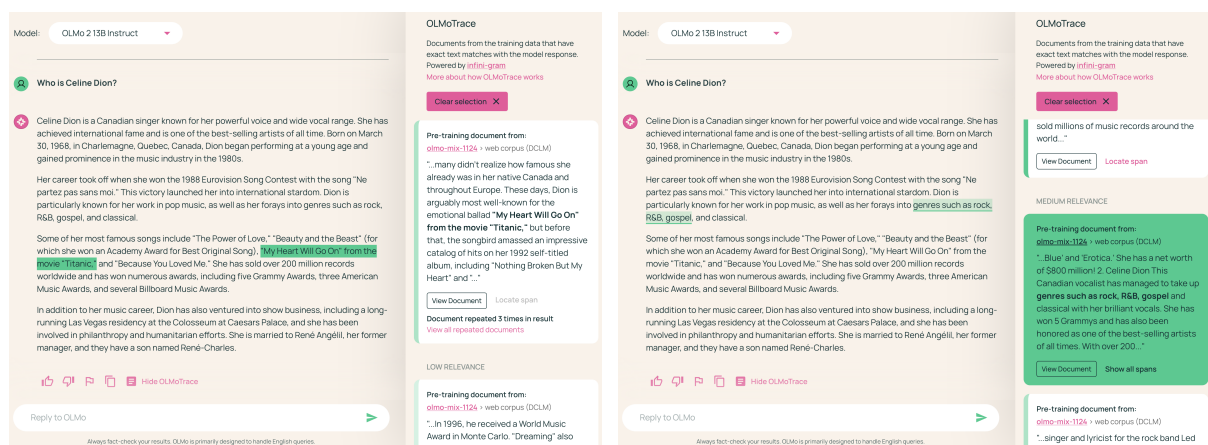
Figure 5: Screenshots on interacting with OLMoTRACE on Ai2 Playground. **Left:** When user clicks on a highlighted span, the document panel is filtered to only present documents enclosing the selected span. **Right:** When user clicks the "Locate Span" button on a document, the span highlights will narrow down to those enclosed in the selected document. Clicking on the same place again or the "Clear Selection" button will lead back to showing all spans and documents (Figure 1, left).

Score	Description
0	The snippet or context of the snippet is about a different topic than the query and model response (though possibly semantically similar): For example, for the query breast cancer symptoms, give a 0 to: A snippet about heart attack symptoms – wrong topic A snippet about brain cancer symptoms – may not necessarily apply to breast cancer symptoms
1	The snippet or context of the snippet is about a broader topic than the query and model response, or is potentially relevant but there's not enough information: For example, for the query breast cancer symptoms, give a 1 to: A snippet about cancer in general – missing key specifics of symptoms
2	The snippet or context of the snippet is on the right topic of the query and model response, but is in a slightly different context or is too specific to fit the exact query: For example, for the query breast cancer symptoms, give a 2 to: A snippet referring a breast cancer treatment side effect
3	The snippet or context of the snippet is about a subject that is a direct match, in topic and scope, of the most likely user intent for the query and model response: For example, for the query breast cancer symptoms, give a 3 to: A snippet discussing a symptom specific to breast cancer

LLM-as-a-Judge Prompt

You will be given a user prompt, a model's response to the prompt, and a retrieved document. Please rate how relevant the document is to the prompt and model response. Rate on a scale of 0 (not relevant) to 3 (very relevant). Respond with a single number, and do not include any other text in your response.

Rubric for rating:
0: The document is about a different topic than the prompt and model response.
1. The document is about a broader topic than the prompt and model response, or is potentially relevant but there's not enough information.
2. The document is on the right topic of the prompt and model response, but is in a slightly different context or is too specific.
3. The document is about a subject that is a direct match, in topic and scope, of the most likely user intent for the prompt and model response.

Prompt: {prompt}
Model response: {response}
Retrieved document: {document}

Table 2: **Left:** Rubrics for document relevance evaluation. **Right:** Prompt for automatically evaluating document relevance with LLM-as-a-Judge.

Setting	Avg score (1st doc)	Avg score (top-5 docs)	% relevant (1st doc)	% relevant (top-5 docs)
our final setting	**1.82**	**1.50**	63.3%	**55.1%**
+ BM25 doc reranking only considers LM response (no user prompt)	1.78	1.49	62.2%	54.5%
+ shorten doc context length to 100 tokens	1.74	1.44	**64.3%**	52.9%
+ span ranking w/ length	1.56	1.37	57.1%	49.4%
+ drop spans w/ frequency >10	1.73	1.28	62.2%	47.0%
+ switch to human annotator	1.90	1.43	63.0%	46.2%

Table 3: Evaluating the relevance level of top documents displayed by OLMoTRACE. Avg score is on a likert scale of 0-3, where 0 is "unrelated" and 3 is "highly relevant". For % relevant, we consider a document as relevant if it gets a score of 2 or 3. We use LLM-as-a-Judge with gpt-4o-2024-08-06, except in the last row where we collect annotation from a human expert.

1. Before step 2, it dropped maximal matching spans that appear more than 10 times in the training data (i.e., frequency >10);
2. In step 2, it ranked the spans by descending length instead of ascending span unigram probability;
3. When reranking documents in step 5, the BM25 scorer only considered a context length of 100 tokens around the span instead of 500;
4. The BM25 scorer only considered the LM response and did not consider the user prompt.

We tuned LLM-as-a-Judge so that it has high agreement and roughly matched statistics with the human evaluation, and our selection of model (gpt-4o-2024-08-06) and prompt (Table 2, right) was the best combination we reached.

We then incrementally adjusted the hyperparameter settings in OLMoTRACE and measured the document relevance with LLM-as-a-Judge. The first change we made is to no longer drop maximal matching spans that appear more than 10 times in the training data. The dropping was due to a limitation in the early version of our system, and we thought this would lead to incomplete results (many documents are duplicated more than 10 times in the pre-training data) and decided to not drop any maximal matching spans according to frequency. Not dropping spans decreased the metrics on the first displayed documents, but increased the metrics on the top-5. Subsequently, we incrementally flipped item 2, 3, and 4 in the above change list, and with every change applied, the overall document relevance metrics improved (with a small exception on % relevant among first displayed documents). Our final setting achieved an average relevance score of 1.82 among the first displayed documents, and 1.50 among the top-5 documents, according to LLM-as-a-Judge.