# Infini-gram: Scaling Unbounded $n$-gram Language Models to a Trillion Tokens

**Jiacheng Liu**[♡] **Sewon Min**[♡]
**Luke Zettlemoyer**[♡] **Yejin Choi**[♡♠] **Hannaneh Hajishirzi**[♡♠]
[♡]Paul G. Allen School of Computer Science & Engineering, University of Washington
[♠]Allen Institute for Artificial Intelligence
liujc@cs.washington.edu

arXiv:2401.17377v1 [cs.CL] 30 Jan 2024

## Abstract

Are $n$-gram language models still relevant in this era of neural large language models (LLMs)? Our answer is *yes*, and we show their values in both text analysis and improving neural LLMs. Yet this necessitates modernizing $n$-gram models in two aspects. First, we train them at the same data scale as neural LLMs – **1.4 trillion tokens**. This is the largest $n$-gram model ever built. Second, existing $n$-gram models use small $n$ which hinders their performance; we instead allow $n$ to be arbitrarily large, by introducing a new **∞-gram LM** with backoff. Instead of pre-computing $n$-gram count tables (which would be very expensive), we develop an engine named **infini-gram** – powered by suffix arrays – that can compute ∞-gram (as well as $n$-gram with arbitrary $n$) probabilities with **millisecond-level latency**. The ∞-gram framework and infini-gram engine enable us to conduct many novel and interesting analyses of human-written and machine-generated text: we find that the ∞-gram LM has fairly high accuracy for next-token prediction (47%), and can complement neural LLMs to greatly reduce their language modeling perplexities. When analyzing machine-generated text, we also observe irregularities in the machine–∞-gram agreement level with respect to the suffix length, which indicates deficiencies in neural LLM pretraining and the positional embeddings of Transformers. We open-source our infini-gram engine in the hopes of enabling more study on how to best use verbatim information retrieved from large text corpora.

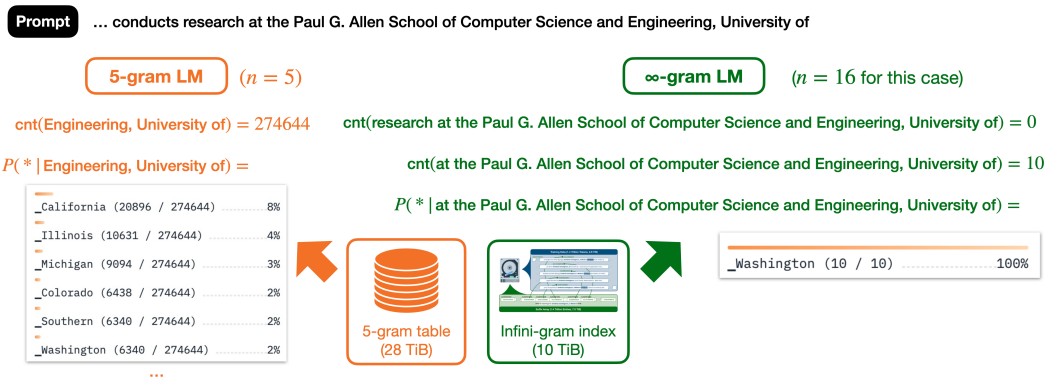



Figure 1: When trained on an 1.4-trillion-token corpus, a 5-gram LM is unable to correctly predict the next token of the given prompt, because it uses a short, fixed context length of 4. Our ∞-gram LM adaptively selects $n$ based on the longest suffix of the prompt that has a non-zero count in the corpus, and in this case, using a larger $n$ yields a correct prediction. The counting and distribution estimate in ∞-gram LM are powered by our infini-gram engine.

## 1 Introduction

When pretrained on trillion-token corpora, neural large language models (LLMs) achieve groundbreaking performance (Touvron et al., 2023a; Geng & Liu, 2023). However, we do not yet know how such data scale would benefit other language modeling approaches. In particular, how well do classical, $n$-gram language models (LMs) perform if estimated from such massive corpora? In other words, *are $n$-gram LMs still relevant in this era of neural LLMs?*

Our answer is *yes*. As we will show, $n$-gram LMs are useful for both text analysis and improving neural LLMs. Yet we need to first modernize the traditional $n$-gram LM in two aspects: the training data size, and the value of $n$. To achieve broader data coverage, we scale up the training data for $n$-gram LMs to **1.4 trillion tokens**, which rivals the size of the largest open-source text corpora (Together, 2023; Soldaini et al., 2023). To our knowledge, this is the largest $n$-gram LM ever built. Historically, $n$-gram indexes have been built only for small $n$'s (e.g., $n \leq 5$; Franz & Brants (2006)), because the size of naive $n$-gram count table grows almost exponentially with respect to $n$. We instead find there is significant value in increasing the value of $n$. As illustrated in Figure 1, $n$-gram LMs with small $n$ ($n = 5$) are often poorly predictive of the next token, because it discards the rich context in the long prompt; meanwhile, if we can use a larger $n$ (in this case $n = 16$), the prediction can become much more accurate. As such, we develop our $n$-gram LM with *unbounded* $n$, or in other words, an **$\infty$-gram LM**. We use a variant of *backoff* (Jurafsky & Martin, 2000), where we resort to smaller $n$ when longer $n$-grams have a zero count. Due to *sparsity* in the $\infty$-gram estimates, in some of the later experiments (e.g., §5), we will interpolate between the $\infty$-gram LM and neural LMs to yield a mixture LM upon which perplexity can be computed.

We develop a low-latency, resource-efficient engine to serve this massive $\infty$-gram LM. Instead of building an explicit $n$-gram count table, which is infeasible for arbitrarily large $n$ and such extreme data scale, we implement the $\infty$-gram LM with a *suffix array* – a data structure that supports fast $n$-gram counting at inference time, and is efficient in both storage space and compute. Our suffix array implementation takes 7 bytes of storage per token (3.5x overhead compared to the raw data), and on the training data with 1.4 trillion tokens, it can be built with a single 80-core CPU node in less than 3 days, using 10 TiB of disk storage. Average inference latency is less than **20 milliseconds for counting an $n$-gram and finding *all* positions of its occurrence** (regardless of how large $n$ is or how frequently the $n$-gram appears), and **200 milliseconds per token** for $n$-gram/$\infty$-gram probability estimation and decoding (more in §3.4 and Table 1). All indexes stay on-disk at inference time. We refer to this $\infty$-gram engine as **infini-gram**.

Analyses with $\infty$-gram (§4) offers new insights into human-written and machine-generated text. We found that $\infty$-gram has a fairly high accuracy (47%) when predicting the next token given a prefix of a human-written document, and this accuracy is higher on tokens where the effective $n$ is larger. In contrast, conventional $n$-grams (with small $n$) are insufficient in capturing a long enough context to predict the next token (29% accuracy). The correctness of $\infty$-gram predictions does *not* perfectly coincide with those predictions made by neural LMs, implying that $\infty$-gram can complement and improve neural LMs and reach better performance when combined. In fact, our experiments (§5) show that heuristically interpolating between the estimates made by $\infty$-gram and neural LMs can greatly reduce language modeling perplexity (by up to 73%) compared to the neural LMs alone, even when the neural LM is as large as 70B. When analyzing the level of agreement with $\infty$-gram, we find that nucleus sampling (Holtzman et al., 2019) from neural LMs produces machine-generated text with an agreement plot most similar to human-written text, among other decoding methods like greedy decoding and temperature sampling; for greedy decoding, we observe significant fluctuation in the agreement level with respect to the suffix length, which indicates deficiencies in neural LM pretraining and the positional embeddings of Transformers.

We open-source[1] the pre-built infini-gram indexes on some datasets, the code for performing $n$-gram/$\infty$-gram inferences, and the code for building indexes for new corpora. We also

[1]Code and pre-built indexes: https://github.com/liujch1998/infini-gram (coming soon)
Demo: https://huggingface.co/spaces/liujch1998/infini-gram

plan to host a public API service for $n$-gram/$\infty$-gram querying. We hope these resources can enable more insightful analysis and understanding of large text corpora, and open up new avenues for data-driven language modeling.

## 2 $\infty$-gram LM: Extending $n$-gram LMs with Unbounded $n$

**Background: $n$-gram LM.** The $n$-gram LM is a classical, statistical language model based on counting the occurrences of $n$-grams. In its most simple form, the probability of a token $w_i$ given a context $w_{i-(n-1):i-1}$ is estimated as

$$P_n(w_i|w_{i-(n-1):i-1}) = \frac{\text{cnt}(w_{i-(n-1):i-1}w_i \mid \mathcal{D})}{\text{cnt}(w_{i-(n-1):i-1} \mid \mathcal{D})} \quad (1)$$

where $\text{cnt}(\mathbf{w} \mid \mathcal{D})$ is the number of times the $n$-gram $\mathbf{w}$ appears in the training data $\mathcal{D}$ (i.e., a corpus), and $n$ is a pre-defined hyperparameter. (When $n = 1$, we define $w_{i-(n-1):i-1}$ as the empty string $\varepsilon$, whose count is equal to $|\mathcal{D}|$.) However, this naive version of $n$-gram LM may run into the sparsity issue: the numerator in Equation 1 may be zero, resulting in an infinite perplexity. One common technique to address this issue is **backoff** (Jurafsky & Martin, 2000): on an instance-wise basis, when the numerator is zero we decrease $n$ by one, and we can do this repeatedly until the numerator becomes positive. One caveat in the backoff strategy is that it does not yield a valid distribution for $P_n(*|w_{i-(n-1):i-1})$, because the **effective** $n$ depends on $w_i$. Therefore, further probability discounting is required to normalize the distribution (e.g., Katz backoff (Katz, 1987)).

Historically, $n$-gram LMs have been implemented by building an $n$-gram count table of the training data. This table stores all unique $n$-grams that appear in the training data, each associated with its count. Such $n$-gram count tables are huge and grow almost exponentially with respect to $n$. For example, we estimate that the 5-gram count table for a 1.4-trillion-token corpus would consume 28 TiB of disk space. As a result, previous $n$-gram LMs are limited to very small $n$, most commonly $n = 5$ (Franz & Brants, 2006; Aiden & Michel, 2011). As we illustrated in Figure 1 and will further quantify in §4, the problem with small $n$ is that it discards richer context, making such $n$-gram LMs poorly predictive of future tokens.

**$\infty$-gram LM.** The $\infty$-gram LM is a generalization of the $n$-gram LM, where conceptually we start backing off from $n = \infty$. We use a variant of backoff: we backoff only when the denominator in Equation 1 is zero. This means we stop backing off as soon as the denominator becomes positive, upon which the numerator might still be zero. This is fine because we will *not* be evaluating the perplexity of the $\infty$-gram LM itself. On an instance-wise basis, the effective $n$ is equal to one plus the length of the prompt's **longest suffix** that appears in the training data.

**For the rest of this paper, we will use "$\infty$-gram" to refer to the $\infty$-gram LM.** $\infty$-gram is formally defined as

$$P_\infty(w_i|w_{1:i-1}) = \frac{\text{cnt}(w_{i-(n-1):i-1}w_i \mid \mathcal{D})}{\text{cnt}(w_{i-(n-1):i-1} \mid \mathcal{D})}$$

where $w_{1:i-1}$ are all tokens preceding $w_i$ in the document, and

$$n = \max\{n' \in [1, i] \mid \text{cnt}(w_{i-(n'-1):i-1} \mid \mathcal{D}) > 0\}.$$

Unlike Katz backoff, $P_\infty(*|w_{1:i-1})$ is a valid distribution by construction and does not require discounting. This is because the effective $n$ is solely dependent on $w_{1:i-1}$ and does not depend on $w_i$, and $\sum_{w_i \in \mathcal{V}} \text{cnt}(w_{i-(n-1):i-1}w_i \mid \mathcal{D}) = \text{cnt}(w_{i-(n-1):i-1} \mid \mathcal{D})$.

Further, we define the **sparsity** of this $\infty$-gram estimate: an estimate is sparse iff $P(w_i|w_{i-(n-1):i-1}) = 1$ for one of the $w_i \in \mathcal{V}$, and is zero for all other tokens in the vocabulary. Intuitively, this means there is only one possible next token given this context, according to the training data. As we will show in §4, sparse estimates are more predictive of the ground truth tokens than non-sparse ones.

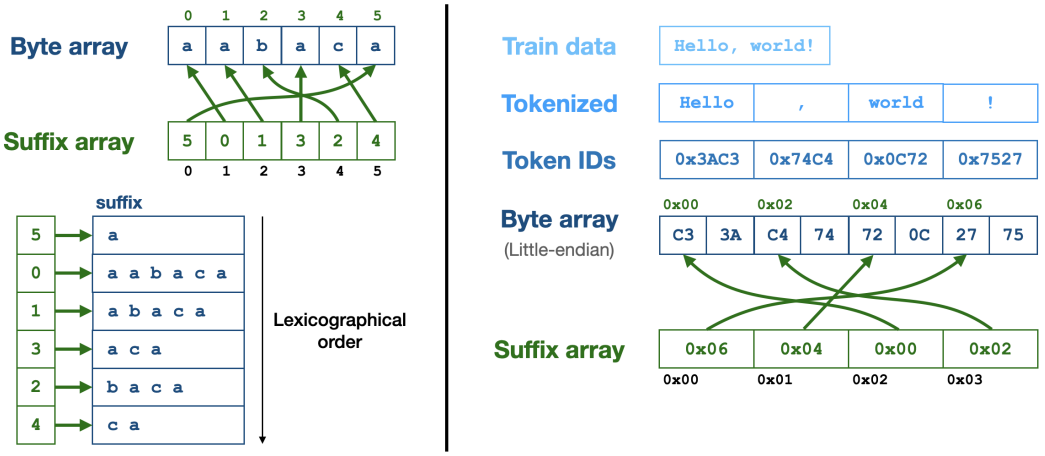



Figure 2: **Left:** the suffix array for a toy string. **Right:** illustration of the suffix array in our infini-gram implementation, with $N = 4$ tokens in the training data.

**Interpolating with neural LMs.** Sparsity in the ∞-gram estimates causes issue in its evaluation: a zero probability assigned to the ground truth token would give an infinite perplexity. We do *not* attempt to compute the perplexity of the ∞-gram in isolation. Instead, we interpolate it with neural LMs and show perplexity improvement compared to the neural LMs alone (§5). The combined model is formally

$$P(y \mid x) = \lambda P_\infty(y \mid x) + (1 - \lambda) P_{\text{neural}}(y \mid x),$$

where $\lambda \in [0, 1]$ is a hyperparameter.

## 3 Infini-gram: A Performant Engine for $n$-gram/∞-gram Queries

To maximize their usefulness, we would like to train ∞-gram on the modern, trillion-token text corpora. However, it is practically infeasible to build the $n$-gram count table with unbounded $n$ for such massive training data, since such a table would contain $O(|\mathcal{D}|^2)$ entries. In this section, we describe our infini-gram engine that processes $n$-gram/∞-gram queries efficiently. Infini-gram is powered by a data structure called **suffix array** (§3.1). We will show how to build this suffix array index (§3.2) and how to perform $n$-gram/∞-gram inferences with it. In §3.4 we list the six types of queries supported by infini-gram and benchmark their latency.

### 3.1 Suffix Array

The essence of $n$-gram and ∞-gram LMs is counting a given $n$-gram in the training data. For this reason, we leverage the suffix array data structure, which is originally designed for efficiently counting the number of times a given "needle" string (length $L$) appears as substring of a huge "haystack" string (length $N$). When the suffix array is built for the haystack string, counting any given needle string would have time complexity $O(L + \log N)$.

A suffix array represents the lexicographical ordering of all suffixes of an array (or a string, which is an array of characters). For an array of length $N$, the suffix array contains $N$ unique integers, where the $i$-th element is the starting position of the suffix ranked $i$-th among all suffixes. Figure 2 (left) shows the suffix array for an example string, `aabaca`.

We build the suffix array on the byte array of the tokenized training data (Figure 2, right). Documents are prefixed by a 4-byte document ID and separated by the `\xff\xff` token. In the byte array, each consecutive two bytes represent a token ID (assuming that $|\mathcal{V}| < 2^{16} = 65536$). Given that the training data has $N$ tokens, the size of the byte array is $2N$ bytes. The suffix array contains $N$ elements, each pointing to a token in the byte array by storing its

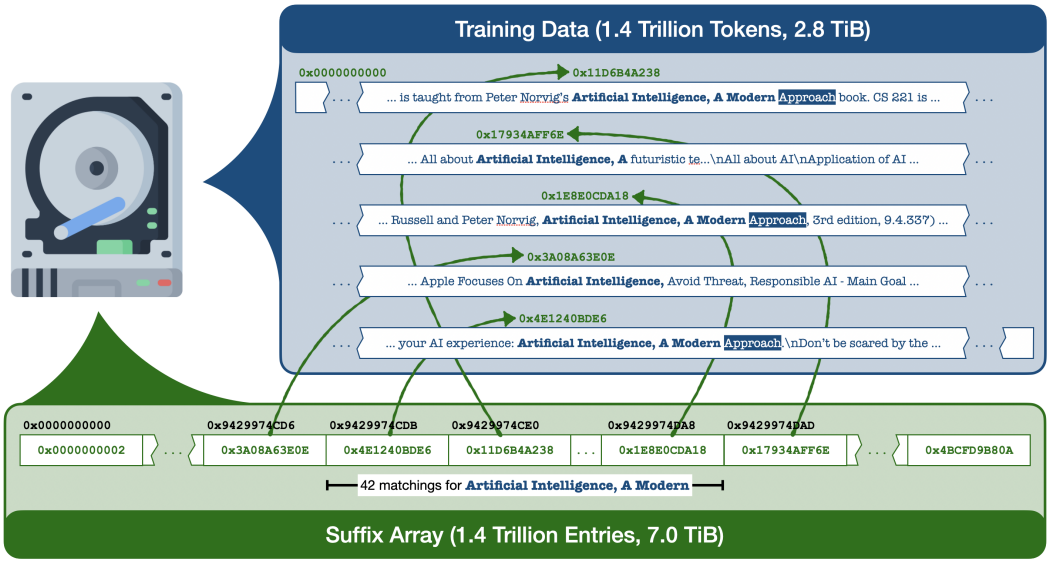



Figure 3: $n$-gram/$\infty$-gram queries on a training data are supported by an associated *suffix array*. Both the training data and the suffix array are stored on-disk as regular files. Contents on the white strips are file data, and addresses above the strips are byte offsets. Querying for a particular $n$-gram returns a consecutive segment of the suffix array, where each element is a pointer into the training data where the $n$-gram appears. E.g., in the trillion-token training data, `Artificial Intelligence, A Modern` appears 42 times, and in all cases the following token is `Approach`.

byte offset. All elements in the suffix array are even numbers (so that they point to valid tokens), and all token positions in the byte array appears exactly once in the suffix array. Each pointer can be stored with $\lceil \log_2(2N)/8 \rceil$ bytes. For corpora with 2B to 500B tokens (which is the range we deal with, after sharding (§3.2)), this is 5 bytes per pointer, and thus the size of the suffix array is $5N$ bytes. The combined size of tokenized data and suffix array (i.e., the **infini-gram index**) is thus $7N$ bytes.

### 3.2 Building the suffix array

Suffix arrays can be built in linear time with respect to the length of the byte array (Kärkkäinen et al., 2006). We adapted from the suffix array implementation in Lee et al. (2022) and further optimized it for efficiency. We then built the suffix arrays for the 360B-token Pile (Gao et al., 2020) and the 1.4T-token RedPajama (Together, 2023). It took us ∼56 hours to build the suffix array for RedPajama on a single node with 80 CPUs and 512G RAM.

**Sharding.** Building the suffix array requires heavy random access to the byte array, and thus the entire byte array must be kept in RAM so that the building time is reasonable. However, the byte array may be too large to fit into RAM. We shard the byte array into multiple shards, and build a suffix array for each shard. Sharding would induce additional inference latency, which we discuss and mitigate below (§3.3).

### 3.3 Inference with the suffix array

**$n$-gram counting.** Computing the $n$-gram probability involves counting the number of occurrences of a token string, i.e., $\text{cnt}(x_1...x_n)$. Since the suffix array represents a lexicographical ordering of all suffixes of the training data, the occurrence positions of strings starting with $x_1...x_n$ lies in a single, consecutive segment in the suffix array. Thus we only need to find the first and last occurrence positions, and the count would be the difference between these positions. Both the first and last occurrence positions can be found with

binary search, with time complexity $O(n \cdot \log N)$ and $O(\log N)$ random array accesses. The two binary searches can be parallelized, reducing the latency by roughly 2x. The impact of query length $n$ is negligible, because computers usually fetch memory in pages of (typically) 4096 bytes, and string comparison is much faster than page fetching. Therefore, when we analyze time complexity below, we refer to the number of random array accesses.

**Finding occurrence positions and documents.** $n$-gram counting with suffix arrays has a by-product: we also get to know all positions where the $n$-gram appears in the training data, for free. This position information is implicitly contained in the suffix array segment we obtained during counting, and to retrieve the original documents where the $n$-gram appears, all we need to do is to follow each pointer within this segment back into the training data, and expand to both directions until hitting the document separator.

**Impact of sharding.** When the suffix arrays are built on sharded byte arrays, we can simply perform counting on each individual shard and accumulate the counts across all shards. The latency is proportional to the number of shards: time complexity would become $O(S \cdot \log N)$. The processing of different shards can be parallelized, reducing the time complexity back to $O(\log N)$.

**Speeding up $n$-gram computation by re-using previous search results.** On the suffix array, the segment for $x_1...x_n$ must be a sub-segment of that for $x_1...x_{n-1}$. Therefore, when computing the $n$-gram probability $P_n(x_n \mid x_1...x_{n-1})$, we can first count $x_1...x_{n-1}$, and then when counting $x_1...x_n$, we only need to search for the first and last occurrence positions within the segment of $x_1...x_n$, which reduces the latency by at most 2x.

**On-disk search.** The byte array and suffix array may be too large to fit into RAM, so in practice, we keep them on disk and read them as memory-mapped files. However, this creates a significant latency as the binary search requires random access to the byte array and suffix array. To mitigate this, we implemented a memory **pre-fetching** method that informs the system of the array offsets we will likely be reading in the near future. Pre-fetching reduces average latency by roughly 5x.

**Speeding up $\infty$-gram computation.** To compute the $\infty$-gram probability, we need to count the occurrence of each suffix $x_{l-n+1}...x_l$ up to the maximum $n$ so that the suffix still meets the sufficient appearance requirement (we denote this maximum $n$ as $L$). This means $O(L)$ counting operations, and the time complexity for each $\infty$-gram computation is $O(L \cdot \log N)$. However, a simple binary-lifting + binary-search algorithm for searching $L$ can reduce the number of counting operations to $O(\log L)$, and thus the time complexity for each $\infty$-gram computation becomes $O(\log L \cdot \log N)$.

**Speeding up dense $\infty$-gram computation.** During evaluation, we need to compute the $\infty$-gram probability of each token in the test document. We can save computation by observing that the effective $n$ for one token is at most one token longer than that for the previous token. This brings the amortized time complexity for evaluating each token down to $O(\log N)$.

### 3.4 Supported query types and latency benchmarking

Infini-gram supports the following types of $n$-gram/$\infty$-gram queries:

1. Counting an $n$-gram (COUNT);
2. Computing a token probability from $n$-gram LM (with given $n$, no backoff) (NGRAMPROB);
3. Computing the full next-token distribution from $n$-gram LM (NGRAMDIST);
4. Computing a token probability from $\infty$-gram LM (INFINIGRAMPROB);
5. Computing the full next-token distribution from $\infty$-gram LM (INFINIGRAMDIST);
6. Returning a random document containing an $n$-gram, or a CNF logical expression of $n$-gram terms, connected with AND's and/or OR's (e.g., `(natural language processing OR artificial intelligence) AND (deep learning OR machine learning)`) (GETDOCUMENT).

| **Reference Data** ($\rightarrow$)<br>**Query Type** ($\downarrow$) | **Pile-train**<br>$N = 0.36$T<br>$S = 2$ | **RPJ**<br>$N = 1.4$T<br>$S = 8$ | **Time Complexity**<br>(measured by number<br>of random disk accesses) |
|---|---|---|---|
| 1. Counting an $n$-gram | | | $O(\log N)$ |
| ... ($n = 1$) | 7 ms | 9 ms | |
| ... ($n = 2$) | 13 ms | 20 ms | |
| ... ($n = 5$) | 14 ms | 19 ms | |
| ... ($n = 10$) | 13 ms | 18 ms | |
| ... ($n = 100$) | 13 ms | 19 ms | |
| ... ($n = 1000$) | 14 ms | 19 ms | |
| 2. Computing a token probability from $n$-gram LM ($n = 5$) | 19 ms | 30 ms | $O(\log N)$ |
| 3. Computing full next-token distribution from $n$-gram LM (n = 5) | 31 ms | 39 ms | $O(V \cdot \log N)$ |
| 4. Computing a token probability from $\infty$-gram LM | 90 ms | 135 ms | $O(\log L \cdot \log N)$ |
| ... on consecutive tokens | 12 ms | 20 ms | $O(\log N)$ |
| 5. Computing full next-token distribution from $\infty$-gram LM | 88 ms | 180 ms | $O((\log L + V) \cdot \log N)$ |

Table 1: Inference-time latency of infini-gram on different types of queries. Average latency per query is reported. Benchmarked with inference engine written in C++ (with parallelized shard processing) and running on a single, 80-core CPU node. Notations for time complexity: $N =$ number of tokens in the reference data; $S =$ number of shards for the suffix array; $L =$ number of tokens in the query document; $V =$ vocabulary size.

See Algorithms 1 and 2 (Appendix) for the algorithms that implement these query types. Our online demo fully supports the above query types.

We benchmark the latency of infini-gram on different types of $n$-gram and $\infty$-gram queries, and show results in Table 1. During inference, the training data and the suffix array are stored on an SSD. For each type of query, the benchmarking is conducted on 1,000 tokens randomly and independently sampled from Pile's validation data (except for the task "computing a token probability from $\infty$-gram LM on consecutive tokens", where we sampled 10 documents and processed 1000 consecutive tokens in each document).

All types of queries demonstrate sub-second latency on the trillion-token training data. Computing a token probability from the $\infty$-gram with RedPajama takes merely 135 milliseconds. Furthermore, our implementation supports counting the occurrence of an $n$-gram with arbitrarily large $n$, with roughly constant latency at 20 milliseconds (we experimentally validated up to $n = 1000$). Decoding requires computing the full next-token distribution and is thus slightly slower: 39 milliseconds per token with $n$-gram LMs and 180 milliseconds per token with $\infty$-gram.

## 4 Analyzing Human-written and Machine-generated Text using $\infty$-gram

In this section, we present some analyses of human-written and machine-generated text from the perspective of the $\infty$-gram, mostly focusing on the token-wise agreement between $\infty$-gram and the actual text. In summary, we found that:

1. $\infty$-gram has a fairly high accuracy (47%) when predicting the next token given a prefix of a human-written document, and this accuracy is higher when a longer suffix of the prompt can be used (i.e., when the effective $n$ is larger);
2. Conventional $n$-grams ($n \leq 5$) are insufficient in capturing a long enough context to determine the next token, while our $\infty$-gram method is highly predictive of human-written and machine-generated text;
3. $\infty$-gram has significant potential to complement and improve neural LMs when predicting human-written text (which we further investigate in §5);
4. When plotting the agreement level with respect to the suffix length, text generated by neural LMs with nucleus sampling is most similar to human-written text, among other decoding methods like greedy decoding and temperature sampling. For greedy decoding, the agreement plot suffers from significant fluctuation, which may be rooted in deficiencies in neural LM pretraining and the positional embeddings of Transformers.

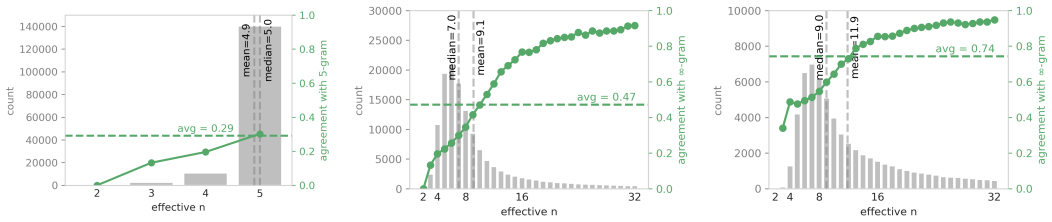



Figure 4: Token-wise agreement between human-generated text and $n$-gram/$\infty$-gram. **Left:** $n$-gram; **Middle:** $\infty$-gram; **Right:** $\infty$-gram, on tokens with sparse estimates only.

**$\infty$-gram training data.** For analyses in this section, we use Pile's training set (Gao et al., 2020) as training data in the $\infty$-gram, which consists of 360 billion tokens (based on the LLaMA tokenizer) after de-contamination.

**Decontamination.** It is important that the training data is decontaminated against the evaluation data in order to avoid test leakage. We run decontamination on Pile's training data against the validation and test data of the Pile (which we will use for evaluation below and also in §5), using the method from Groeneveld (2023) that filters out a document if there is too much $n$-gram overlap with the evaluation data. See §A.1 for more details.

Decontamination is non-trivial, and its definition could vary, e.g., when there is an identical sentence, is it contamination, or is it a quote that naturally occurs in real test-time scenarios? We follow standard decontamination best practices, and conduct detailed analysis in the later sections.

### 4.1 Human-written Text

**Setup.** We use Pile's validation data as samples of human-written text. For this analysis, we sampled 50 documents from each domain of Pile's validation set, and truncated each document to up to 1024 tokens (so the total number of tokens per domain is about 50k). We aggregate results from all domains.

We measure the token-wise *agreement* between $\infty$-gram's estimate and the actual human-written text. Since computing the full next-token distribution (or the argmax of it) in $\infty$-gram is relatively slow, we compute the $\infty$-gram probability of the actual next-token, and deem it as accurate if this probability is higher than 0.5.[2] We further categorize all tokens by their effective $n$, i.e., one plus the length of their prompt's longest suffix that has a non-zero count in the training data. For each category, we visualize the number of such tokens (in gray bars), as well as the agreement level (in green dots).

The visualization is shown in Figure 4 (middle plot). Overall, $\infty$-gram agrees with the human-written text on 47% of the tokens. We see that $\infty$-gram becomes more accurate with the increase of the effective $n$: when the effective $n \geq 16$, agreement is higher than 75%. Further analysis (Appendix Figure 14) shows that the count of this longest suffix in the training data does not affect agreement significantly.

In the left plot of Figure 4, we show the same analysis for a 5-gram LM using the same training data, and it has much lower agreement than the $\infty$-gram. 5-grams, which has been used extensively in previous literature (Franz & Brants, 2006; Aiden & Michel, 2011), does not capture a long enough context to correctly predict the next token: over 90% tokens in the evaluation data has an effective $n$ of at least 5, and the $\infty$-gram analysis shows that the median of effective $n$ is 7 tokens (and mean is 9.1 tokens).

In the right plot of Figure 4, we show the same analysis for only tokens with a sparse $\infty$-gram estimate, which covers more than 50% of all tokens. The overall agreement is even higher (75%), and when the effective $n \geq 14$, agreement is higher than 80%. This means

[2]This is a lower-bound of argmax accuracy, though the gap is small.

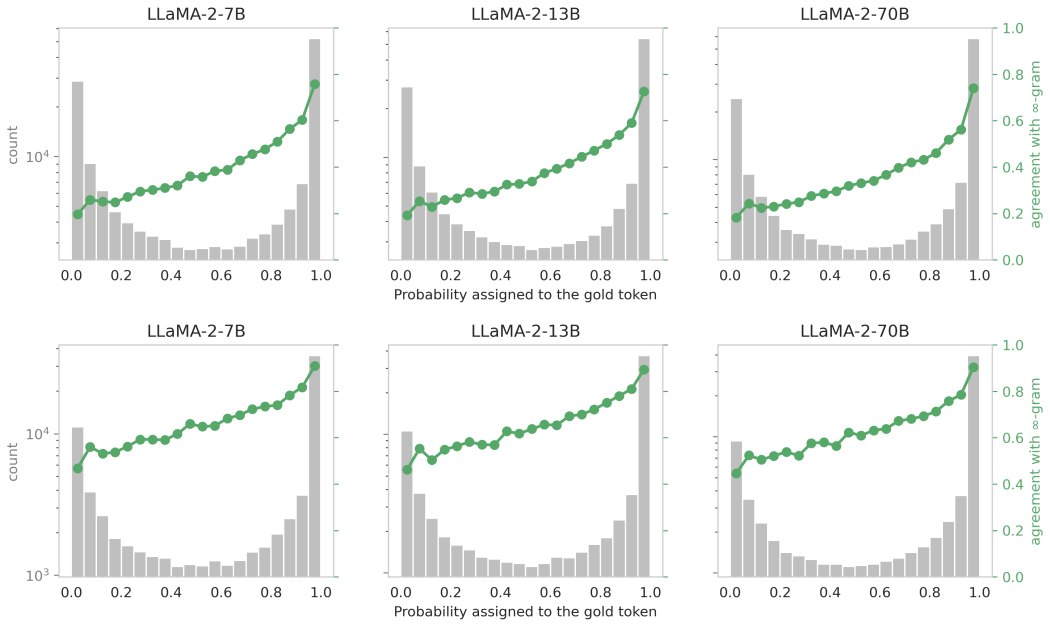



Figure 5: Distribution of probabilities assigned by neural LMs to human-written text tokens, and ∞-gram's agreement with these tokens. **Upper:** on all tokens; **Lower:** on tokens with sparse ∞-gram estimates. **Takeaway:** ∞-gram and neural LMs are predictive of actual human text on different tokens, and thus ∞-gram estimates – especially sparse ∞-gram estimates – can be used to complement neural LMs.

when the next token is unique according to the training data, that unique token is very likely to be the actual token in human-written text.

**∞-gram can shine where neural LMs fail.** In Figure 5, we plot the distribution of probabilities assigned by the LLaMA-2 models (Touvron et al., 2023b) to the actual tokens in human-written text, and the human–∞-gram agreement for tokens in each probability range. (The higher the assigned probability, the higher agreement LLaMA-2 has with the actual tokens.) We observe a positive, yet imperfect, correlation between neural LMs and ∞-gram regarding their agreement with the actual text. In particular, when the neural LM performance is very bad (left side of the histogram), ∞-gram still gives a non-trivial agreement of above 20%; if only considering tokens with sparse ∞-gram estimates, this is as high as 50%. This implies a huge potential of complementing and improving the performance of neural LMs with ∞-gram when predicting human-written text, which we further investigate in §5.

### 4.2 Machine-generated Text

**Setup.** Similar to the analysis with human-written text, we sampled 50 documents from each domain of the Pile. We use the first 50 tokens of each document to prompt neural LMs to generate a continuation. Generation continues up to the original length of the document, or when an [EOS] token is generated. We experiment with three decoding methods: greedy decoding, temperature sampling, and nucleus sampling (Holtzman et al., 2019). The neural LMs are GPT-Neo (1.3b, 2.7b), GPT-J (6b), and LLaMA-2 (7b, 13b, 70b).

**Impact of decoding method.** The top row of Figure 6 shows the three decoding method on the same neural LM – LLaMA-2-70b. In general, increasing stochasticity shifts the effective $n$ to the smaller side, and also decreases the agreement level. Nucleus sampling (with $p = 0.8$) has the most similar distribution of effective $n$ to human-written text (Figure 4, middle plot), which is probably why nucleus sampling is usually preferred in text generation. In comparison, greedy decoding has even higher effective $n$ than human-written text, which

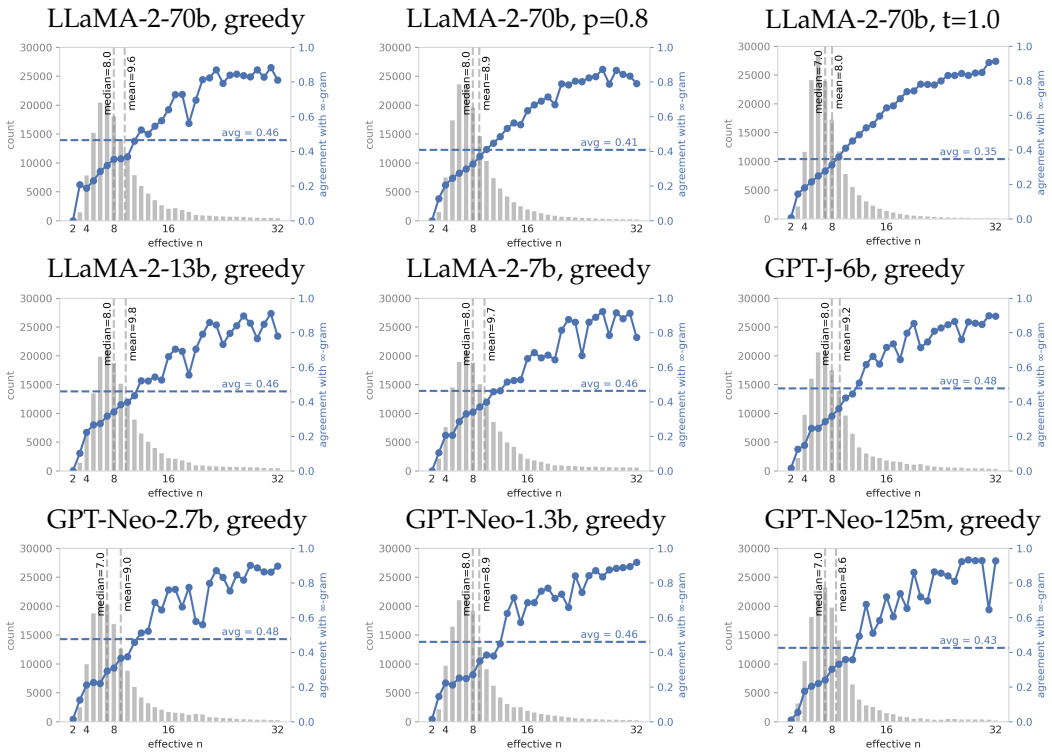



Figure 6: Token-wise agreement between machine-generated text and ∞-gram. All tokens are considered.

implies that greedy decoding could lead to over-memorization of training data as well as lack of diversity.

One very curious phenomenon is that, as effective $n$ increases, the agreement level fluctuates greatly in greedy decoding (but not nucleus or temperature sampling, where agreement level almost increases monotonically). Such fluctuation is even more rapid for smaller models (LLaMA-2-13b/7b, GPT-J-6b, GPT-Neo-2.7b/1.3b/125m), and for LLaMA-2-7b the fluctuation is even periodic (rapidly dropping at effective $n = 20, 24, 28, 32$; this is statistically significant, a two-proportion z-test gives a p-value of $< 10^{-99}$). We suspect that this may be caused by the application of positional embeddings when pretraining these Transformer-based models, and we welcome further investigation from the community.

**Impact of model size.** The last two rows of Figure 6 shows the same analysis for different sizes of neural LM under greedy decoding. In general, increasing model size slightly shifts the effective $n$ to the larger side, and also increases the agreement level. This indicates that larger models memorizes more from the training data, and are also more inclined to copy verbatim. The agreement level of GPT-Neo/J models is higher than LLaMA-2 models, probably because GPT-Neo/J are trained on the same data as our ∞-gram training data (i.e., Pile's training set). Overall, text generated by these neural LMs has similar agreement level with ∞-gram as human text.

## 5 Improving Neural LMs with the ∞-gram

The results in §4 motivate us to combine (interpolate; §2) neural LMs and ∞-gram to yield better language models. This section shows strong experimental results of the combined model. In §4 we found that the ∞-gram estimate has higher agreement with human-written text when it is *sparse*. Therefore, we use two separate interpolation hyperparameters: $\lambda_1$ for

sparse and $\lambda_2$ for non-sparse $\infty$-gram estimates. These hyperparameters are tuned on the validation set to minimize the perplexity of the combined model.

### 5.1 Experimental Setup

**Evaluation.** We compute perplexity on the Pile validation and test data (Gao et al., 2020). We split each document in the data into batches with a maximum sequence length of 1,024 and a sliding window of 512, a setup that is standard in prior language modeling literature (Baevski & Auli, 2019; Khandelwal et al., 2020).

**Metric.** We measure the perplexity of each model on the evaluation data, as well as the relative improvement of perplexity between models. The relative improvement of model $M$ against model $M_o$ is computed as

$$\Delta = (1 - \frac{\text{PPL}(M) - 1}{\text{PPL}(M_o) - 1}) \times 100\% \tag{2}$$

which is the percentage of perplexity gap closed towards perfect language modeling (i.e., PPL = 1).

**Reference data.** To reduce confusion, in this section, we will use "reference data" to refer to the training data of the $\infty$-gram. In addition to Pile's training set (Gao et al., 2020), which we used in the previous analyses (§4), we also consider RedPajama (Together, 2023) as reference data. The Pile and the Redpajama have 360 billion and 1.4 trillion tokens, respectively, summing up to 1.8 trillion tokens (based on the LLaMA tokenizer). We later perform ablations on varying sizes and domains of the reference data.

**Neural LMs.** We use a range of large, competitive neural LMs, both as a baseline and as a model to interpolate with the $\infty$-gram. In total, 14 models are considered.

- **GPT-2** (Radford et al., 2019), one of the earliest autoregressive language models whose sizes range from 117M, 345M, and 774M to 1.6B. Their training data is a diverse set of web text, although is not public.
- **GPT-Neo** (Gao et al., 2020) and **GPT-J** (Wang & Komatsuzaki, 2021), language models trained on the Pile whose sizes vary from 125M, 1.3B, and 2.7B to 6.7B.
- **LLaMA-2** (Touvron et al., 2023b), a subsequent version of LLaMA (Touvron et al., 2023a) trained on two trillion tokens and has sizes of 7B, 13B, and 70B. LLaMA-2 is one of the most competitive language models whose weights are available at the time of writing the paper. The training data of LLaMA-2 is unknown, although the precedent version is trained on a large corpus of Common Crawls, Wikipedia and code, which is replicated by RedPajama (Together, 2023).
- **SILO** (Min et al., 2023a), 1.3B language models trained on permissively licensed data only. The original paper showed that training on permissively licensed data leads to the challenge of *extreme domain generalization* because the training data is skewed to highly specific domains like code and government text. We use three different variants, PD, PDSW and PDSWBY, which are trained on different levels of permissivity, leading to varying levels of the domain generalization challenge.

The perplexity of GPT-2, GPT-Neo and GPT-J are comparable to each other, but perplexity of LLaMA-2 and SILO are not comparable to them nor to each other, because they are based on different tokenizers. We built three versions of the infini-gram index, one for each type of tokenizer.

### 5.2 Results

Experimental results are shown in Table 2 and Table 3. Interpolating with $\infty$-gram greatly and consistently improves the perplexity of neural LMs. The amount of improvement trends smaller as the neural LM size grows, within the same model series, while the largest models can still benefit a lot from our method (e.g., the Pile alone improves LLaMA-2 (70B) by 12%).

| Neural LM | (Size) | Ref Data | Validation | | | Test | | |
|---|---|---|---|---|---|---|---|---|
| | | | Neural | + ∞-gram | | Neural | + ∞-gram | |
| GPT-2 | 117M | Pile | 22.82 | **13.71** | (42%) | 22.86 | **13.58** | (42%) |
| GPT-2 | 345M | Pile | 16.45 | **11.22** | (34%) | 16.69 | **11.18** | (35%) |
| GPT-2 | 774M | Pile | 15.35 | **10.39** | (35%) | 15.40 | **10.33** | (35%) |
| GPT-2 | 1.6B | Pile | 14.42 | **9.93** | (33%) | 14.61 | **9.93** | (34%) |
| GPT-Neo | 125M | Pile | 13.50 | **10.76** | (22%) | 14.08 | **10.79** | (25%) |
| GPT-Neo | 1.3B | Pile | 8.29 | **7.31** | (13%) | 8.61 | **7.36** | (16%) |
| GPT-Neo | 2.7B | Pile | 7.46 | **6.69** | (12%) | 7.77 | **6.76** | (15%) |
| GPT-J | 6.7B | Pile | 6.25 | **5.75** | (10%) | 6.51 | **5.85** | (12%) |
| LLaMA-2 | 7B | Pile | 5.69 | **5.05** | (14%) | 5.83 | **5.06** | (16%) |
| LLaMA-2 | 13B | Pile | 5.30 | **4.75** | (13%) | 5.43 | **4.76** | (15%) |
| LLaMA-2 | 70B | Pile | 4.59 | **4.21** | (11%) | 4.65 | **4.20** | (12%) |
| LLaMA-2 | 7B | Pile + RPJ | 5.69 | **4.66** | (22%) | 5.83 | **4.66** | (24%) |
| LLaMA-2 | 13B | Pile + RPJ | 5.30 | **4.41** | (21%) | 5.43 | **4.42** | (23%) |
| LLaMA-2 | 70B | Pile + RPJ | 4.59 | **3.96** | (18%) | 4.65 | **3.95** | (19%) |

Table 2: Perplexity (the lower the better) on the validation and the test datasets of the Pile. The numbers in parentheses are percentage of improvement as computed by Equation 2. The first eight rows share the tokenizer with each other, and the last six rows share the tokenizer with each other. Pile is Pile-train (360 billion tokens); RPJ is RedPajama (1.4 trillion tokens).

| Neural LM | Validation | | | | Test | | | |
|---|---|---|---|---|---|---|---|---|
| | Neural | + ∞-gram | + $k$NN-LM† | + RIC-LM† | Neural | + ∞-gram | + $k$NN-LM† | + RIC-LM† |
| ***Eval data: Wikipedia*** | | | | | | | | |
| Silo PD | 26.60 | **15.30** (43%) | 20.62 | 27.91 | 28.42 | **14.44** (51%) | – | – |
| Silo PDSW | 18.93 | **12.36** (36%) | 14.10 | 18.90 | 20.02 | **11.84** (43%) | 14.5 | 19.4 |
| Silo PDSWBY | 10.66 | **8.77** (19%) | 10.14 | 10.87 | 10.76 | **8.41** (24%) | – | – |
| Pythia | 9.00 | – | 8.50 | 8.84 | 9.1 | – | – | – |
| ***Eval data: Enron Emails*** | | | | | | | | |
| Silo PD | 19.56 | **6.31** (70%) | 8.56 | 15.45 | 15.71 | **4.85** (73%) | – | – |
| Silo PDSW | 14.66 | **5.58** (65%) | 6.70 | 10.80 | 11.23 | **4.35** (66%) | 5.9 | 9.9 |
| Silo PDSWBY | 14.67 | **5.61** (65%) | 7.24 | 10.91 | 11.52 | **4.44** (66%) | – | – |
| Pythia | 7.577 | – | 4.99 | 6.16 | 6.9 | – | – | – |
| ***Eval data: NIH ExPorters*** | | | | | | | | |
| Silo PD | 27.46 | **16.26** (41%) | 19.27 | 25.51 | 27.94 | **16.00** (44%) | – | – |
| Silo PDSW | 19.35 | **12.70** (35%) | 14.95 | 18.35 | 19.12 | **12.39** (37%) | 15.0 | 18.5 |
| Silo PDSWBY | 15.01 | **10.62** (30%) | 12.33 | 14.29 | 14.81 | **10.33** (32%) | – | – |
| Pythia | 11.20 | – | 11.20 | 10.83 | 11.1 | – | – | – |

Table 3: Perplexity (the lower the better) on the validation and the test datasets of the Wikipedia, Enron Emails, and NIH ExPorters of the Pile. All neural models are 1.3B models, and the reference data is always the Pile. ■ indicates in-domain; ■ indicates out-of-domain; ■ indicates out-of-domain but has relevant data in-domain, all with respect to the training data of the neural LM. †: Results retrived from Min et al. (2023a), which use much smaller reference data: 45-million to 1.2-billion tokens, compared to our 360-billion tokens.

However, this trend does not hold across different series of LMs. For example, ∞-gram can improve GPT-2 (1.6B) by 34%, but only improves a smaller model, GPT-Neo (1.3B), by 16%. This may be attributed to the fact that the GPT-Neo/J models are trained on the Pile, while the GPT-2 models are not. ∞-gram works best when the reference data distribution differs from, or complements, the pretraining data distribution, which emphasizes the importance of data diversity. Meanwhile, the fact that ∞-gram also improves neural LMs already pretrained on its reference data shows that the advantage of using ∞-gram is consistent.

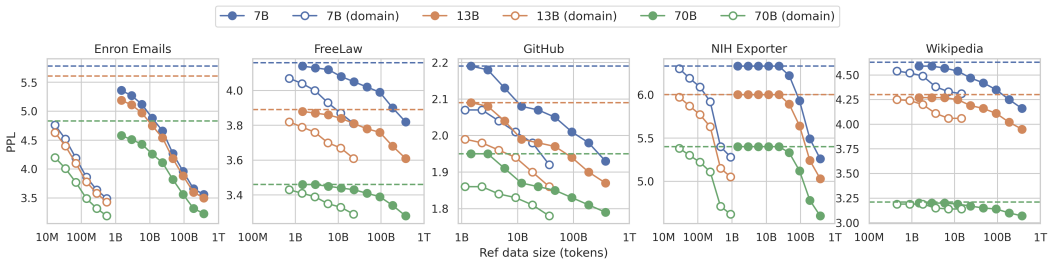



Figure 7: Impact of scaling the datastore of the ∞-gram, all using the LLaMA-2 models (7B, 13B, and 70B) as neural LMs, and the Pile as the reference data. - - -: neural LM only (baseline). ●: ∞-gram uses the full Pile; ○: ∞-gram uses only the in-domain portion of the Pile. **Gains increase consistently as the datastore scales.**

On the choice of ∞-gram reference data, the combination of the Pile and RedPajama yields larger improvements on the LLaMA-2 models than the Pile alone. The interpolation of LLaMA-2 (13B) and ∞-gram with Pile + RPJ outperforms LLaMA-2 (70B), and interpolating with ∞-gram pushes the perplexity of LLaMA-2 (70B) below 4.0.

When the neural LM is SILO (which is trained on permissive-licensed data only and thus has less training data), adding the ∞-gram component is more helpful when SILO is trained on more restrictive data (i.e., PD > PDSW > PDSWBY). The usage of ∞-gram can be precisely traced back to the contributing document(s) in the reference data, which is in-line with the philosophy of SILO: to allow crediting the source data when using them for language modeling. When compared to the existing retrieval-augmentation methods used by SILO, i.e., *k*NN-LM and RIC-LM, ∞-gram yields better improvement in perplexity. Therefore, ∞-gram can serve as a better alternative as the retrieval-augmentation method for SILO.

### 5.3 Ablations

**Effect of the size of the reference data.** Figure 7 reports the performance of the combined model wrt the size of the reference data. To create progressively smaller reference data, we repeatedly subsampled the full reference data by 2x (up to 256x, resulting in 9 sizes). We see the improvement brought by ∞-gram widens as reference data size grows, and the relationship is roughly log-linear (except for the `NIH ExPorter` domain, where ∞-gram doesn't help when the reference data is too small).

**Effect of the domain of the reference data.** Figure 7 compares the performance of the combined model where the ∞-gram uses either the full reference data or the in-domain reference data. Using only the in-domain reference data is roughly as powerful as using the full reference data, which implies that almost all improvement we have witnessed is thanks to in-domain data (where contaminations have been eliminated). This means it would not hurt to use the full reference data, especially when the test domain is unknown or an in-domain reference data is unavailable; however, having in-domain reference data is most helpful.

### 5.4 Evaluating on Time-Shifted Data

To further show the effectiveness of ∞-gram and eliminate doubts that our performance gains might be due to insufficient decontamination, we evaluate on **time-shifted data**: documents that were created after the cutoff time of the ∞-gram reference data. We use new Wikipedia articles created during April and August, 2023, which is after the cutoff time of both Pile and RedPajama.

Table 4 reports the perplexity of neural LM as well as the combined model. On documents in four out of the five months, interpolating with ∞-gram improves the perplexity of the neural LM. We find that this improvement can be further boosted by applying a Random Forest to decide an instance-wise interpolation hyperparameter, where the features of the

| Eval Data (Wikipedia) | simple interpolation | | w/ Random Forest | |
|---|---|---|---|---|
| | Neural | + ∞-gram | Neural | + ∞-gram |
| April 2023 | 5.64 | **5.48** (3%) | 5.86 | **4.89** (20%) |
| May 2023 | 5.43 | **5.27** (4%) | 6.01 | **5.70** ( 6%) |
| June 2023 | 5.49 | **5.21** (6%) | 5.69 | **4.87** (17%) |
| July 2023 | 4.93 | 4.93 (0%) | 4.91 | **4.78** ( 3%) |
| August 2023 | 4.64 | **4.46** (5%) | 4.81 | **4.50** ( 8%) |

Table 4: Evaluation on time-shifted data. The evaluation data is taken from newly-added Wikipedia articles since April 2023, which is after the creation of both the Pile and RedPajama. The neural model is LLaMA-2 (13B), and the ∞-gram reference data is Pile + RPJ.

Random Forest are the suffix lengths (1 up to the effective $n$) as well as the frequency of each suffix in the reference data. When Random Forest is applied, the perplexity improvement ranges from 3% – 20%.

### 5.5 A note on text generation

While ∞-gram can be interpolated with neural LMs and greatly improve their perplexity, our preliminary experiments show that such method might not be helpful, and even harmful, to open-ended text generation tasks. During generation, ∞-gram can make odd mistakes (e.g., retrieving very irrelevant tokens) which makes the model to completely fail. Thus this combined model is *not* ready to replace neural LMs. Additional investigation is required to make ∞-gram best contribute to text generation (e.g., adaptively routing between ∞-gram and neural LMs).

## 6 Discussion and Broader Implications

In §4 and §5, we showcased some very preliminary use cases of the infini-gram engine. However, we believe that infini-gram can enable much broader investigations and applications, including but not limited to:

**Understanding text corpora.** Text corpora used for pretraining language models have become prohibitively large, and we have relatively limited understanding of their contents (Elazar et al., 2023). Infini-gram can be a useful tool to quickly find out what *is* in the corpus and what is *not*, using $n$-gram lookup (the COUNT query).

**Data curation.** Data engineers often want to remove problematic content in corpora scraped from the Internet, such as toxicity, hate speech, and personal identifiable information (PII). Using infini-gram's GETDOCUMENT query (which can be easily modified to return all documents), one can retrieve all documents containing an $n$-gram term (or a CNF expression with multiple $n$-gram terms) and remove them from the corpus. Removal can even be done iteratively: infini-gram indexes are additive/subtractive, so we can obtain an index of the corpus after round one removal, by indexing the removed set and take the difference of the original index and the removal index.

**Attribution.** When using neural LMs to make predictions, people might want to know which training data most influenced the model's decision. Using $n$-gram lookup with key phrases, we can trace back to related documents in the training data of neural LMs.

**Detecting data contamination, memorization, and plagiarism.** Test set contamination has become a major issue for language model evaluation. $n$-gram lookup enables us to check if evaluation queries have sneaked into the training data of neural LMs. It also opens up possibility to detect memorization in machine-generated text, or plagiarism in human-written text.

**Mitigating copyright infringement.** Recently, generative AIs are facing numerous lawsuits for generating arguably copyrighted materials. Infini-gram may be able to mitigate

| Method | # tokens (↑) | # entries (↑) | Storage usage (↓) | max $n$ |
|---|---|---|---|---|
| ***Vector-based index*** | | | | |
| RETRO (Borgeaud et al., 2022) | 1.8 T | $2.8 \times 10^{10}$ | 432 TB (16$k$ bytes / entry) | – |
| Atlas (Izacard et al., 2022) | 27 B | $4 \times 10^{8}$ | 200 GB (8 bytes / entry) | – |
| $k$NN-LM (Khandelwal et al., 2020) | 3 B | $3 \times 10^{9}$ | 200 GB (64 bytes / entry) | – |
| NPM (Min et al., 2023b) | 1 B | $1 \times 10^{9}$ | 1.4 TB ($\sim$ 2$k$ bytes / entry) | – |
| ***n-gram-based index*** | | | | |
| Google's (Franz & Brants, 2006) | 1 T | $3.8 \times 10^{9}$ | 24 GB | 5 |
| Google Books Ngram (Aiden & Michel, 2011) | 500 B | unreported | unreported | 5 |
| Stehouwer & van Zaanen (2010) | 90 M | unreported | unreported | $\infty$ |
| Kennington et al. (2012) | 3 M | $5 \times 10^{12}$ | 330 MiB (110 bytes / token) | $\infty$ |
| Shareghi et al. (2015) | 9 B | $8 \times 10^{18}$ | 63 GiB (7 bytes / token) | $\infty$ |
| infini-gram (ours) | 1.8 T | $1.6 \times 10^{24}$ | 12 TiB (7 bytes / token) | $\infty$ |

Table 5: Comparison with other nonparametric language modeling methods. **# tokens**: number of tokens in the inference-time reference data. **# entries**: number of representations (counts) in the index. **max** $n$: maximum number of context tokens considered. For infini-gram, we consider the combination of Pile-train and RedPajama as reference data.

copyright infringement, by diverting neural LMs to alternative (yet still plausible) generation paths when they are about to generate long $n$-grams that appear in the training data, especially if they mostly appear in documents from copyrighted sources.

**Reducing hallucination in factual knowledge.** Parametric-only models are prone to generating non-factual statements, which is widely known as the hallucination problem. Infini-gram can potentially be used to mitigate hallucination by reading verbatim from the training data. We have found evidence that the $\infty$-gram can greatly outperform LLaMA-2-70B on factual probing benchmarks such as LAMA (Petroni et al., 2019).

**Non-parametric speculative decoding.** Speculative decoding (Chen et al., 2023) speeds up text generation by employing a fast and a slow decoder, where the fast decoder is a smaller model that does the autoregressive token generation, and the slow decoder checks the fast decoder's proposals by parallelizing the forward passes of multiple tokens. Given the low latency of infini-gram, we can potentially use $\infty$-gram as the fast decoder, similar to He et al. (2023).

**Offloading rote memorization from neural models.** Fully-parametric language models needs to internalize a huge amount of factual knowledge into their parameters. We can potentially offload such rota memorization from neural models into non-parametric modules, for example, by training neural LMs to fit the residual of $\infty$-gram from the ground truth (Li et al., 2022).

We welcome the community to collaboratively build toward the aforementioned directions, by leveraging the open-sourced tools provided in infini-gram.

## 7 Related Work

***n*-gram language models.** $n$-gram has been one of the most classical language modeling methods since the inception of natural language processing (Jurafsky & Martin, 2000). People have been pushing the limits of $n$-gram LMs by scaling up its training data, and to date, the largest $n$-gram table (Franz & Brants, 2006) indexes 1 trillion tokens for 5-grams that appears at least 40 times.

While $n$-gram LMs are currently largely surpassed by neural LMs, there has been recent work that revisit $n$-grams and $n$-gram LMs. Khandelwal et al. (2020) interpolates neural LMs with the $n$-gram model but finds it not improving performance. In contrast, Li et al. (2022) finds that the $n$-gram model is as competitive as a small neural LM, and training a neural model to be complementary to the $n$-gram model and using both at inference time outperforms the neural-only LM. However, both use limited reference data (101M tokens)

and compare with small neural LMs (117–250M parameters). Some prior work has found value in scaling up the $n$-gram training data (Allamanis & Sutton, 2013).

Our $\infty$-gram LM does not use a fixed value of $n$. We significantly scale up the training data to more than a trillion tokens and interpolate it with the neural model, significantly outperforming the state-of-the-art neural models that consist of up to 70B parameters.

**Unbounded $n$-grams, suffix arrays, suffix trees.** Previous work has explored using suffix-based data structures to enable $n$-gram queries with unbounded $n$, with limited scale of the training data. Stehouwer & van Zaanen (2010) proposes to use suffix arrays for $\infty$-gram, and yet their formulation does *not* yield proper probability distributions and, consequently, a language model. Kennington et al. (2012) proposes to use suffix trees for the same purpose, and yet the storage overhead of suffix trees is very high such that it hinders scaling, which may be mitigated with highly intricate compression techniques (Shareghi et al., 2015). Among the three aforementioned papers, only the third evaluates on the general language modeling task, and the perplexity numbers are too high to be practically useful. We compare the scale of our $\infty$-gram index with these papers in Table 5; our training data is 200x larger than the largest one used in previous work.

**Other data structures for text indexing.** Beside suffix arrays and suffix trees, other data structures have been used to index text corpora to satisfy different trade-offs. The ROOTS Search Tool (Piktus et al., 2023) builds a BM25 index on the ROOTS corpus, and supports document searching via both exact match and fuzzy match of $n$-grams. Data Portraits (Marone & Durme, 2023) proposes a lightweight index based on Bloom Filter, and is tailored for probabilistic membership inference (exact match of $n$-grams of 50 characters, where $n \approx 8$) against the Pile and Stacks. ElasticSearch is a proprietary search engine based on the Lucene index, and it has been used by Dodge et al. (2021) to search documents in C4, and also by Elazar et al. (2023) to count $n$-grams and list most frequent $n$-grams in various corpora up to 480B tokens.

**Nonparametric language models.** A nonparametric LM refers to the LM whose complexity is not bounded as a priori, because the complexity can grow or update according to the data given at inference time. Prior work is broadly divided into two categories: a *token* retrieval approach that represents each token as one vector and uses a nonparametric prediction function (Khandelwal et al., 2020; Zhong et al., 2022; Lan et al., 2023; Min et al., 2023b;a; Shi et al., 2023), and a *chunk* retrieval approach that represents each chunk of text as a vector and incorporates nearest chunks to the neural language model (Guu et al., 2020; Izacard et al., 2022; Borgeaud et al., 2022). Scaling the reference data in nonparametric LMs is very expensive as it requires storing a vector for every unit (either token or chunk). To the best of our knowledge, prior work with the largest reference data is RETRO (Borgeaud et al., 2022), which uses the 7B-parameter LM and the reference data consisting of 1.8 trillion tokens. It stores and searches over 28 billion vectors, estimated to consume 432TB of disk space.[3] (Detailed comparisons in Table 5.)

Our $\infty$-gram LM is one instance of nonparametric LMs, and its simplicity makes it possible to significantly scale the reference data with modest resources (§3.1). To the best of our knowledge, our $\infty$-gram LM is the largest in both the size of the reference data size (1.8 trillion tokens when counting the union of Pile-train and RedPajama) and the size of the base neural LM (70B).

## 8 Conclusion and Future Work

In this paper, we modernized the classical $n$-gram language model by scaling it up to a trillion tokens and extending to unbounded $n$. We presented the infini-gram engine that performs efficient training and inference under this extreme setup. We also proposed the $\infty$-gram language model, powered by the infini-gram engine, and showed that it can offer

[3]This is in part because RETRO does not use any approximation in $k$NN search. Even if RETRO used approximate search as Khandelwal et al. (2020) did, it would still use 10TB. Moreover, there is no open-sourced software that easily supports fast $k$NN search over tens of billions of vectors.

novel insights into human-written and machine-generated text and can improve existing neural language models. We look forward to seeing more insightful analyses and creative usages powered by infini-gram.

## Acknowledgments

We would like to thank Zexuan Zhong, Mike Lewis, Yanai Elazar, Will Merrill, Tim Dettmers, Ximing Lu, Alisa Liu, Weijia Shi, Xiaochuang Han, members of the H2lab, and Ziqi Ma for their invaluable feedback. This work was funded in part by the DARPA MCS program through NIWC Pacific (N66001-19-2-4031), NSF IIS-2044660, NSF DMS-2134012, and ONR N00014-18-1-2826.

## References

Erez Lieberman Aiden and Jean-Baptiste Michel. Quantitative analysis of culture using millions of digitized books. *Science*, 331:176 – 182, 2011. URL https://api.semanticscholar.org/CorpusID:40104730.

Miltiadis Allamanis and Charles Sutton. Mining source code repositories at massive scale using language modeling. *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 207–216, 2013. URL https://api.semanticscholar.org/CorpusID:1857729.

Alexei Baevski and Michael Auli. Adaptive input representations for neural language modeling. In *Proceedings of the International Conference on Learning Representations*, 2019.

Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. Improving language models by retrieving from trillions of tokens. In *Proceedings of the International Conference of Machine Learning*, 2022.

Charlie Chen, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, L. Sifre, and John M. Jumper. Accelerating large language model decoding with speculative sampling. *ArXiv*, abs/2302.01318, 2023. URL https://api.semanticscholar.org/CorpusID:256503945.

Jesse Dodge, Ana Marasovic, Gabriel Ilharco, Dirk Groeneveld, Margaret Mitchell, and Matt Gardner. Documenting large webtext corpora: A case study on the colossal clean crawled corpus. In *Conference on Empirical Methods in Natural Language Processing*, 2021. URL https://api.semanticscholar.org/CorpusID:237568724.

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. What's in my big data? *ArXiv*, abs/2310.20707, 2023. URL https://api.semanticscholar.org/CorpusID:264803575.

Alex Franz and Thorsten Brants. All our n-gram are belong to you. *Google Machine Translation Team*, 20, 2006. URL https://blog.research.google/2006/08/all-our-n-gram-are-belong-to-you.html.

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

Xinyang Geng and Hao Liu. Openllama: An open reproduction of llama, May 2023. URL https://github.com/openlm-research/open_llama.

Dirk Groeneveld. The big friendly filter. https://github.com/allenai/bff, 2023.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *Proceedings of the International Conference of Machine Learning*, 2020.

Zhenyu He, Zexuan Zhong, Tianle Cai, Jason D Lee, and Di He. Rest: Retrieval-based speculative decoding. 2023. URL https://api.semanticscholar.org/CorpusID:265157884.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. *ArXiv*, abs/1904.09751, 2019. URL https://api.semanticscholar.org/CorpusID:127986954.

Gautier Izacard, Patrick Lewis, Maria Lomeli, Lucas Hosseini, Fabio Petroni, Timo Schick, Jane Dwivedi-Yu, Armand Joulin, Sebastian Riedel, and Edouard Grave. Few-shot learning with retrieval augmented language models. *arXiv preprint arXiv:2208.03299*, 2022.

Dan Jurafsky and James H. Martin. Speech and language processing - an introduction to natural language processing, computational linguistics, and speech recognition. In *Prentice Hall series in artificial intelligence*, 2000. URL https://api.semanticscholar.org/CorpusID:60691216.

Juha Kärkkäinen, Peter Sanders, and Stefan Burkhardt. Linear work suffix array construction. *J. ACM*, 53:918–936, 2006. URL https://api.semanticscholar.org/CorpusID:12825385.

Slava M. Katz. Estimation of probabilities from sparse data for the language model component of a speech recognizer. *IEEE Trans. Acoust. Speech Signal Process.*, 35:400–401, 1987. URL https://api.semanticscholar.org/CorpusID:6555412.

Casey Redd Kennington, Martin Kay, and Annemarie Friedrich. Suffix trees as language models. In *International Conference on Language Resources and Evaluation*, 2012. URL https://api.semanticscholar.org/CorpusID:12071964.

Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In *Proceedings of the International Conference on Learning Representations*, 2020.

Tian Lan, Deng Cai, Yan Wang, Heyan Huang, and Xian-Ling Mao. Copy is all you need. In *Proceedings of the International Conference on Learning Representations*, 2023.

Katherine Lee, Daphne Ippolito, Andrew Nystrom, Chiyuan Zhang, Douglas Eck, Chris Callison-Burch, and Nicholas Carlini. Deduplicating training data makes language models better. In *Proceedings of the Association for Computational Linguistics*, 2022.

Huayang Li, Deng Cai, Jin Xu, and Taro Watanabe. Residual learning of neural text generation with n-gram language model. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, 2022. URL https://aclanthology.org/2022.findings-emnlp.109.

Marc Marone and Benjamin Van Durme. Data portraits: Recording foundation model training data. *ArXiv*, abs/2303.03919, 2023. URL https://api.semanticscholar.org/CorpusID:257378087.

Sewon Min, Suchin Gururangan, Eric Wallace, Hannaneh Hajishirzi, Noah Smith, and Luke Zettlemoyer. SILO language models: Isolating legal risk in a nonparametric datastore. *arXiv preprint arXiv:2308.04430*, 2023a. URL https://arxiv.org/abs/2308.04430.

Sewon Min, Weijia Shi, Mike Lewis, Xilun Chen, Wen-tau Yih, Hannaneh Hajishirzi, and Luke Zettlemoyer. Nonparametric masked language modeling. In *Findings of ACL*, 2023b.

Fabio Petroni, Tim Rocktäschel, Patrick Lewis, Anton Bakhtin, Yuxiang Wu, Alexander H. Miller, and Sebastian Riedel. Language models as knowledge bases? *ArXiv*, abs/1909.01066, 2019. URL https://api.semanticscholar.org/CorpusID:202539551.

Aleksandra Piktus, Christopher Akiki, Paulo Villegas, Hugo Laurenccon, Gérard Dupont, Alexandra Sasha Luccioni, Yacine Jernite, and Anna Rogers. The roots search tool: Data transparency for llms. In *Annual Meeting of the Association for Computational Linguistics*, 2023. URL https://api.semanticscholar.org/CorpusID:257219882.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

Ehsan Shareghi, Matthias Petri, Gholamreza Haffari, and Trevor Cohn. Compact, efficient and unlimited capacity: Language modeling with compressed suffix trees. In *Conference on Empirical Methods in Natural Language Processing*, 2015. URL https://api.semanticscholar.org/CorpusID:225428.

Weijia Shi, Sewon Min, Michihiro Yasunaga, Minjoon Seo, Rich James, Mike Lewis, Luke Zettlemoyer, and Wen-tau Yih. REPLUG: Retrieval-augmented black-box language models. *arXiv preprint arXiv:2301.12652*, 2023.

Luca Soldaini, Rodney Kinney, Akshita Bhagia, Dustin Schwenk, David Atkinson, Russell Authur, Khyathi Chandu, Jennifer Dumas, Li Lucy, Xinxi Lyu, Ian Magnusson, Aakanksha Naik, Crystal Nam, Matthew E. Peters, Abhilasha Ravichander, Zejiang Shen, Emma Strubell, Nishant Subramani, Oyvind Tafjord, Evan Pete Walsh, Hannaneh Hajishirzi, Noah A. Smith, Luke Zettlemoyer, Iz Beltagy, Dirk Groeneveld, Jesse Dodge, and Kyle Lo. Dolma: An Open Corpus of 3 Trillion Tokens for Language Model Pretraining Research. Technical report, Allen Institute for AI, 2023. Released under ImpACT License as Medium Risk artifact, https://github.com/allenai/dolma.

Herman Stehouwer and Menno van Zaanen. Using suffix arrays as language models: Scaling the n-gram. 2010. URL https://api.semanticscholar.org/CorpusID:18379946.

Together. RedPajama: An open source recipe to reproduce LLaMA training dataset, 2023. URL https://github.com/togethercomputer/RedPajama-Data.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023a.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023b.

Ben Wang and Aran Komatsuzaki. GPT-J-6B: A 6 Billion Parameter Autoregressive Language Model. https://github.com/kingoflolz/mesh-transformer-jax, May 2021.

Zexuan Zhong, Tao Lei, and Danqi Chen. Training language models with memory augmentation. In *Proceedings of Empirical Methods in Natural Language Processing*, 2022.

# A Additional Details on Method

## A.1 De-contamination of Reference Data

To properly evaluate the effectiveness of ∞-gram LM on Pile's evaluation sets, we performed data de-contamination on the Pile's training set and RedPajama before using them as reference data for the ∞-gram LM. We run the Big Friendly Filter (BFF)[4] (Groeneveld, 2023) on Pile's training set and RedPajama, filtering out documents with too much $n$-gram overlap with Pile's evaluation sets. Table 6 reports the statistics of de-contamination.

When using BFF, we always remove whole documents, instead of by paragraphs. Following the default settings, we consider $n$-grams where $n = 13$, and discard the document if at least 80% of its $n$-grams are present in the evaluation set. For Pile's training set, we lowercase all documents to capture more potential contaminations.

RedPajama

| Subset | Total docs | Filtered docs | Ratio filtered |
|---|---|---|---|
| arxiv | 1558306 | 213 | 0.01% |
| book | 205744 | 711 | 0.3% |
| c4 | 364868892 | 53195 | 0.01% |
| common_crawl | 476276019 | 0 | 0% |
| github | 28793312 | 614259 | 2% |
| stackexchange | 29825086 | 40086 | 0.01% |
| wikipedia | 29834171 | 21973 | 0.07% |
| **Total** | **931361530** | **730437** | **0.08%** |

Pile (train)

| Subset | Total docs | Filtered docs | Ratio filtered |
|---|---|---|---|
| Arxiv | 2377741 | 1089 | |
| BookCorpus2 | 25355 | 6 | |
| Books3 | 277655 | 99 | |
| DM Mathematics | 1918535 | 0 | 0% |
| Enron Emails | 926132 | 18236 | 2% |
| EuroParl | 131723 | 21 | |
| FreeLaw | 5069088 | 11821 | 0.2% |
| Github | 18044218 | 961726 | 5.3% |
| Gutenberg (PG-19) | 66981 | 70 | 0.1% |
| HackerNews | 1571968 | 14 | |
| NIH ExPorter | 1777926 | 3739 | 0.2% |
| OpenSubtitles | 632485 | 5754 | 0.9% |
| OpenWebText2 | 32333654 | 136914 | 0.4% |
| PhilPapers | 63875 | 2324 | 0.4% |
| Pile-CC | 52441354 | 19928 | |
| PubMed Abstracts | 29329202 | 2312 | |
| PubMed Central | 5679903 | 4230 | 0.1% |
| StackExchange | 29529008 | 2072 | |
| USPTO Backgrounds | 11123325 | 80088 | 0.7% |
| Ubuntu IRC | 20067 | 10 | |
| Wikipedia (en) | 16939503 | 45052 | 0.3% |
| YoutubeSubtitles | 328030 | 871 | 0.3% |
| **Total** | **210607728** | **1296376** | **0.6%** |

Table 6: Statistics of de-contamination in RedPajama (left) and Pile's training set (right).

## A.2 Inference Algorithms

Algorithms 1 and 2 shows the implementation of $n$-gram/∞-gram queries listed in §3.4.

## A.3 Example Queries

Figures 8 to 13 show one example for each of the six query types supported by infini-gram. We used Pile-train and the GPT-2 tokenizer throughout these examples. Screenshots are taken from our demo.

# B Additional Results

Figure 14

[4] https://github.com/allenai/bff

**Algorithm 1** $n$-gram/$\infty$-gram queries

**Data** reference data byte array $data$ with $2N$ bytes ($N$ tokens), suffix array $sa$ with $P \cdot N$ bytes ($N$ elements, $P$ is the pointer size), both memory-mapped

**procedure** PREFETCH($B, l, r, depth = 0$)
  $m \leftarrow \lfloor (l + r)/2 \rfloor$
  **if** $m == -1$ **then return**
  **if** $depth == 1$ **then**
    $p \leftarrow$ READPTR($sa, m, P$)
    Read disk page containing bytes $data[p : p + B]$
  **else if** $depth == 3$ **then**
    Read disk page containing bytes $sa[P \cdot m : P \cdot (m + 1)]$
    **return**
  PREFETCH($B, l, m, depth + 1$)
  PREFETCH($B, m, r, depth + 1$)
**procedure** FIND($promptIds, hint = NULL$)
  **if** $len(promptIds) == 0$ **then**
    **return** $(0, N)$
  $promptBuf \leftarrow$ ENCODE($promptIds$) ▷ little-endian byte array
  $B \leftarrow promptBuf.numbytes$
  **if** $hint == NULL$ **then**
    $l \leftarrow -1, r \leftarrow N$
  **else**
    $l \leftarrow hint[0] - 1, r \leftarrow hint[1]$
  **while** $r - l > 1$ **do**
    PREFETCH($B, l, r$)
    $m \leftarrow \lfloor (l + r)/2 \rfloor$
    $p \leftarrow$ READPTR($sa, m, P$)
    **if** $ds[p : p + B] < promptBuf$ **then**
      $l \leftarrow m$
    **else**
      $r \leftarrow m$
  $start \leftarrow r$
  **if** $hint == NULL$ **then**
    $l \leftarrow -1, r \leftarrow N$
  **else**
    $l \leftarrow hint[0] - 1, r \leftarrow hint[1]$
  **while** $r - l > 1$ **do**
    PREFETCH($B, l, r$)
    $m \leftarrow \lfloor (l + r)/2 \rfloor$
    $p \leftarrow$ READPTR($sa, m, P$)
    **if** $ds[p : p + B] \leq promptBuf$ **then**
      $l \leftarrow m$
    **else**
      $r \leftarrow m$
  $end \leftarrow r$
  **return** $(start, end)$
**procedure** COUNT($promptIds$)
  $(start, end) \leftarrow$ FIND($promptIds$)
  $count \leftarrow end - start$
  **return** $count$
**procedure** NGRAMPROB($promptIds, tokenId, hint = NULL$)
  **if** $hint == NULL$ **then**
    $hint \leftarrow$ FIND($promptIds$)
  $(start, end) \leftarrow hint$
  $count_{prompt} \leftarrow end - start$
  **if** $count_{prompt} == 0$ **then**
    **return** NaN
  $(start, end) \leftarrow$ FIND($promptId + [tokenId], hint = (start, end)$)
  $count_{full} \leftarrow end - start$
  $prob = count_{full}/count_{prompt}$
  **return** $prob$

**Algorithm 2** $n$-gram/$\infty$-gram queries (continued)

**procedure** NGRAMDIST($promptIds, hint = NULL, startTokenId = NULL, endTokenId = NULL$)
  **if** $hint == NULL$ **then**
    $hint \leftarrow$ FIND($promptIds$)
  $promptBuf \leftarrow$ ENCODE($promptIds$)
  $B \leftarrow promptBuf.numbytes$ ▷ this can be pre-computed
  $freqByTokenId \leftarrow \{\}$ ▷ a map from token id to frequency
  $(start, end) \leftarrow hint$
  **if** $end - start \leq 4$ **then** ▷ the threshold for this base case is heuristically determined
    **for** $rank = start...end - 1$ **do**
      $p \leftarrow$ READPTR($sa, rank, P$)
      **if** $p + B < 2N$ **then**
        $tokenId \leftarrow$ READTOKENID($ds, p + B$)
        $freqByTokenId[tokenId] \leftarrow freqByTokenId[tokenId] + 1$
  **else**
    **if** $startTokenId == NULL$ **then**
      $p \leftarrow$ READPTR($sa, start, P$)
      $startTokenId \leftarrow$ READTOKENID($ds, p + B$)
    **if** $endTokenId == NULL$ **then**
      $p \leftarrow$ READPTR($sa, end - 1, P$)
      $endTokenId \leftarrow$ READTOKENID($ds, p + B$)
    **if** $startTokenId == endTokenId$ **then** ▷ if start and end has the same next-token, then this entire segment must have the same next-token
      $freqByTokenId[startTokenId] \leftarrow end - start$
    **else** ▷ divide and conquer
      $rank \leftarrow \lfloor (start - end)/2 \rfloor$
      $freqByTokenIdLeft \leftarrow$ NGRAMDIST($promptIds, hint = (start, rank), startTokenId = startTokenId$)
      $freqByTokenIdRight \leftarrow$ NGRAMDIST($promptIds, hint = (rank, end), endTokenId = endTokenId$)
      $freqByTokenId \leftarrow$ MERGEDICTSBYSUM($freqByTokenIdLeft, freqByTokenIdRight$)
  $probByTokenId \leftarrow$ NORMALIZE($freqByTokenId$)
  **return** $probByTokenId$

**procedure** INFINIGRAMPROB($promptIds, tokenId$)
  $L \leftarrow promptIds.length$
  $prob \leftarrow$ NaN ▷ this is a placeholder, because $\infty$-gram can always fallback to $n = 1$ and give a valid probability
  **for** $cutoff = L...0$ **do** ▷ this can be speed up by binary-lifting + binary-search
    $(start, end) \leftarrow$ FIND($promptIds[cutoff:]$)
    **if** $end - start == 0$ **then**
      **return** $prob$
    $prob \leftarrow$ NGRAMPROB($promptIds[cutoff:], tokenId, hint = (start, end)$)
  **return** $prob$

**procedure** INFINIGRAMDIST($promptIds$)
  $L \leftarrow promptIds.length$
  $bestCutoff \leftarrow L$
  **for** $cutoff = L...0$ **do**
    $(start, end) \leftarrow$ FIND($promptIds[cutoff:]$)
    **if** $end - start == 0$ **then**
      **break**
    $bestCutoff \leftarrow cutoff$
  **return** NGRAMDIST($promptIds[bestCutoff:]$)

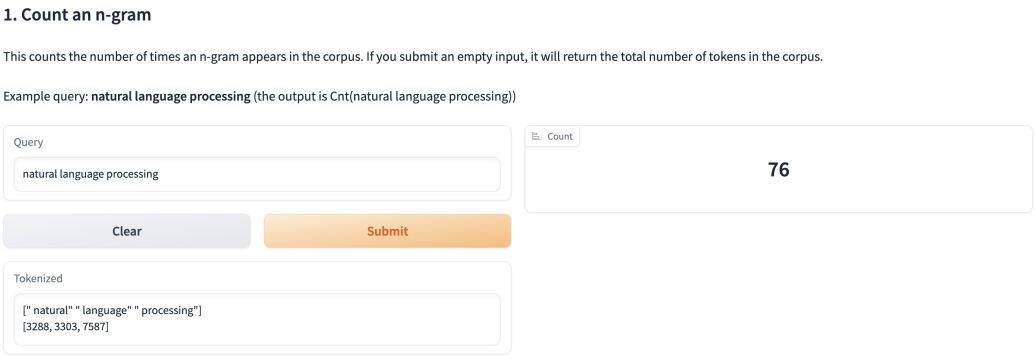



Figure 8: Example for query type 1: Counting an $n$-gram (COUNT).

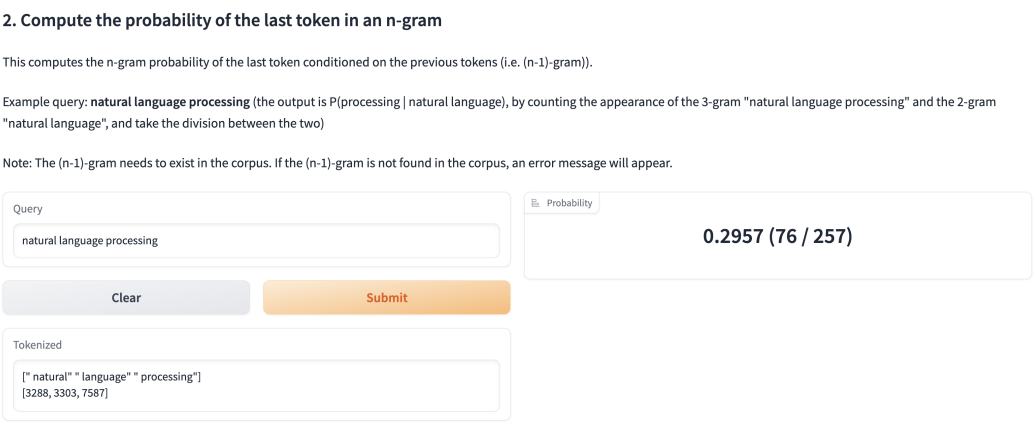



Figure 9: Example for query type 2: Computing a token probability from $n$-gram LM (with given $n$, no backoff) (NGRAMPROB).

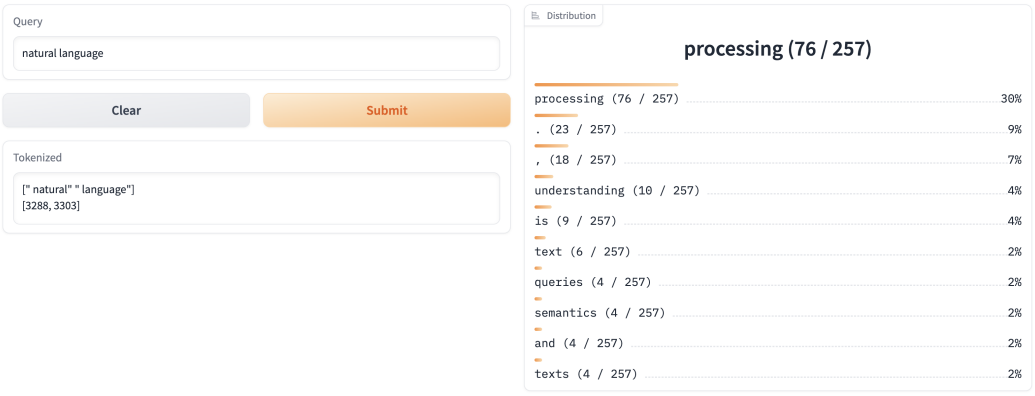



Figure 10: Example for query type 3: Computing the full next-token distribution from $n$-gram LM (NGRAMDIST). Due to space limits, only top-10 tokens are shown.

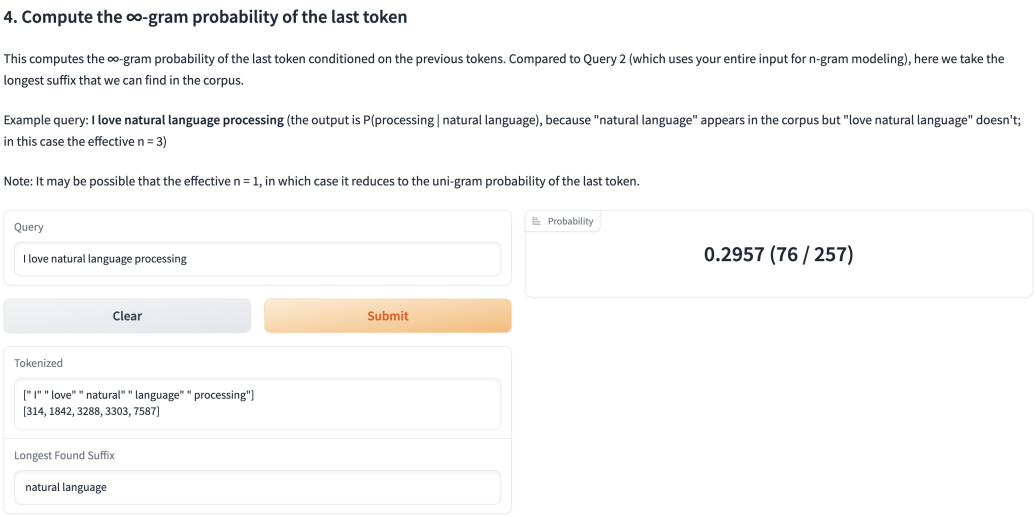



Figure 11: Example for query type 4: Computing a token probability from ∞-gram LM (INFINIGRAMPROB).

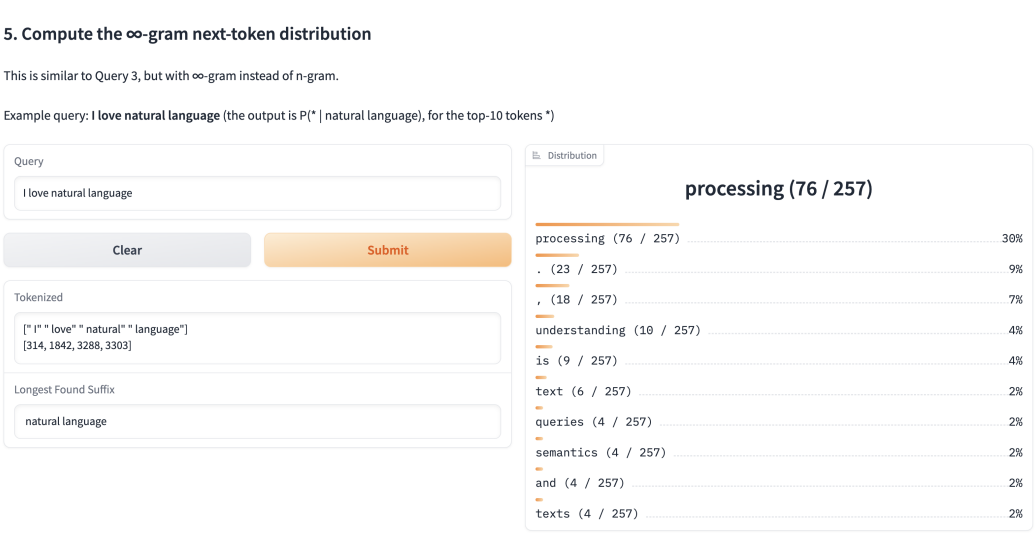



Figure 12: Example for query type 5: Computing the full next-token distribution from ∞-gram LM (INFINIGRAMDIST). Due to space limits, only top-10 tokens are shown.

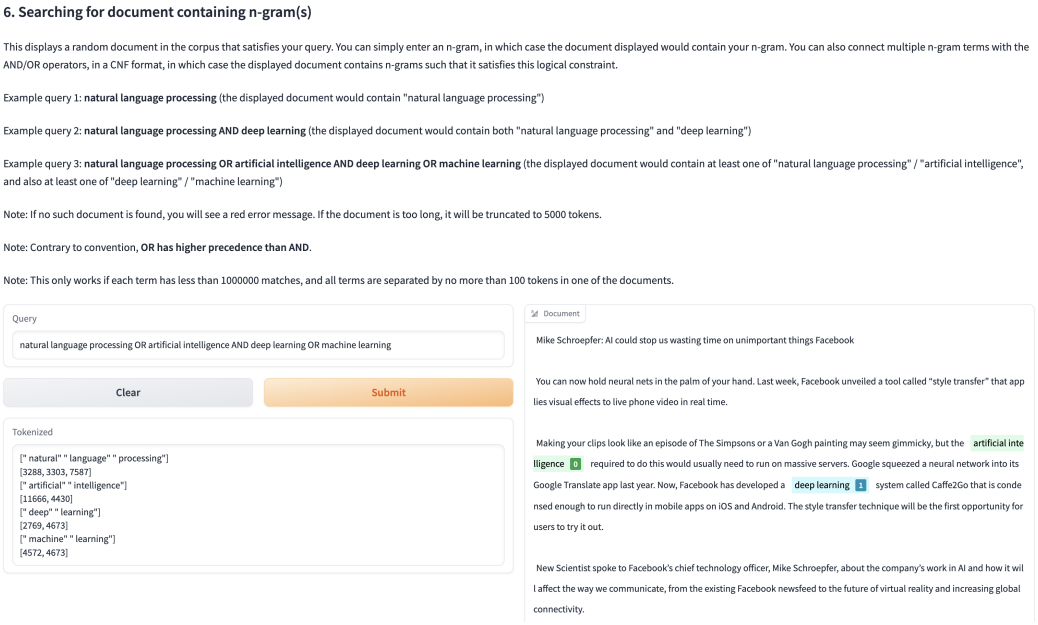



Figure 13: Example for query type 6: Returning one document containing an $n$-gram, or a CNF logical expression of $n$-gram terms, connected with AND's and/or OR's.

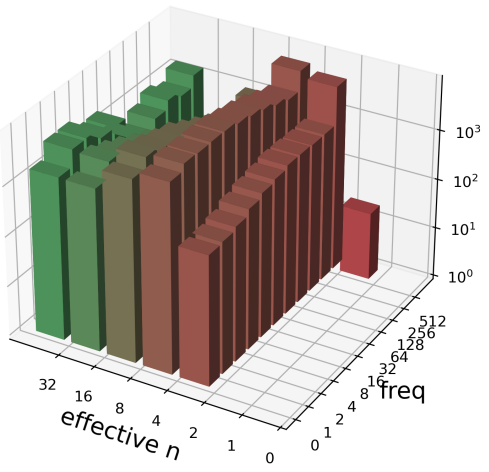



Figure 14: Token-wise agreement between human-generated text and ∞-gram, broken down by "effective $n$" and frequency of the corresponding longest suffix in the reference data. The height of each bar represents token count, and the color represents agreement (red is 0.0, green is 1.0).