

# LSHBLOOM: MEMORY-EFFICIENT, EXTREME-SCALE DOCUMENT DEDUPLICATION

Arham Khan<sup>1</sup> Robert Underwood<sup>2</sup> Carlo Siebenschuh<sup>1</sup> Yadu Babuji<sup>1</sup> Aswathy Ajith<sup>1</sup> Kyle Hippe<sup>1</sup>  
Ozan Gokdemir<sup>1</sup> Alexander Brace<sup>1</sup> Kyle Chard<sup>1</sup> Ian Foster<sup>2</sup>

## ABSTRACT

Deduplication is a major focus for assembling and curating training datasets for large language models (LLM) – detecting and eliminating additional instances of the same content – in large collections of technical documents. Unrestrained, duplicates in the training dataset increase training costs and lead to undesirable properties such as memorization in trained models or cheating on evaluation. Contemporary approaches to document-level deduplication are often extremely expensive in both runtime and memory. We propose LSHBloom, an extension to MinhashLSH, which replaces the expensive LSHIndex with lightweight Bloom filters. LSHBloom demonstrates the same deduplication performance as MinhashLSH with only a marginal increase in false positives (as low as  $1e-5$  in our experiments); demonstrates competitive runtime (270% faster than MinhashLSH on peS2o); and, crucially, uses just 0.6% of the disk space required by MinhashLSH to deduplicate peS2o. We demonstrate that this space advantage scales with increased dataset size—at the extreme scale of several billion documents, LSHBloom promises a 250% speedup and a  $54\times$  space advantage over traditional MinHashLSH scaling deduplication of text datasets to many billions of documents.

## 1 INTRODUCTION

Scholarly publications and related literature represent a tremendous body of often carefully encoded human knowledge. Google Scholar is estimated to index some 400 million documents (Gusenbauer, 2019), suggesting that the total number of unique documents worldwide numbers at least in the hundreds of millions. And a growing number of digital repositories mean that researchers can today access millions of such documents with ease, and with effort, tens of millions or more.

Developments in large language models (LLMs) suggest that such corpora can be used to develop powerful agents able to assist with technical work in a variety of fields (Zhang et al., 2024). However, seizing this opportunity at the largest scale requires addressing challenging deduplication problems. Any large corpus of technical documents assembled from diverse locations will include multiple copies of many documents: not only identical copies but also copies that vary, subtly or otherwise, for reasons such as different publication formats, the use of different parsers, copyediting, and versioning. As is well known, duplication in training

datasets can lead to a host of undesirable behaviors, from excessive training times to predictive biases and training data memorization (Lee et al., 2021) in trained models. Text duplication between training and evaluation sets also hinders thorough, unbiased assessments of model performance, with much evidence suggesting that standard open-source benchmarks have leaked into LLM training sets (Chandran et al., 2024), illegitimately increasing perceived model performance by 8–11% (Dekoninck et al., 2024).

These considerations motivate the problem that we tackle in this paper, namely the **accurate and efficient deduplication of large numbers (10s to 100s of millions) of scholarly and technical documents**. We approach this problem from two perspectives. First, we demonstrate via careful evaluation that existing state-of-the-art approaches to the deduplication of technical documents suffer from poor deduplication performance (i.e., low precision and/or recall) and/or excessive computational costs. Due to the frequent loss of metadata and variation in documents, deduplication of technical documents requires approximate textual comparison methods such as Jaccard similarity, which may be accelerated via the approximating MinHashing method (Broder et al., 1998). The infeasibility of pairwise comparisons in large datasets motivated the development of the MinHashLSH method (Leskovec et al., 2014), which applies a locality-sensitive hashing (LSH) scheme to MinHash signatures to avoid comparing dissimilar documents. How-

<sup>1</sup> Department of Computer Science, University of Chicago; Chicago, IL, United States <sup>2</sup>Data Science & Learning Division, Argonne National Laboratory; Lemont, IL, United States. Correspondence to: Arham Khan <arham@uchicago.edu>.

ever, the associated MinHashLSH index can grow extremely large. For example, applying MinHashLSH to the peS2o dataset (Soldaini & Lo, 2023) of just 8M full-text and 31M abstracts takes more than 14 hours on a single 32-core node of the Polaris supercomputer and requires more than 300GB of disk space (see Section 5.3.1).

Second, we introduce a new method, LSHBloom, that improves significantly on the state of the art in terms of the computational costs required to achieve a desired level of deduplication performance. The key insight here is that the process that MinHashLSH employs to find matches between MinHash signatures can be approximated, with a modest and controllable loss of accuracy, by using Bloom filters. This use of Bloom filters enables LSHBloom to run almost three times faster than MinHashLSH by interfacing with inexpensive bit arrays rather than inserting and querying against the comparatively slow tree data structures that comprise the traditional LSHIndex. Most importantly, this use of Bloom filters means that LSHBloom requires just 0.6% of the disk space consumed by the traditional MinHashLSH index to deduplicate peS2o (Soldaini & Lo, 2023), a dataset of 39 million documents. We demonstrate that this space advantage remains as the number of documents increases and makes it feasible for LSHBloom to process several billion documents with space savings of  $54\times$  relative to MinHashLSH without sacrificing deduplication quality.

## 2 BACKGROUND

We first provide background information on the deduplication problem and describe the current state-of-the-art for large-scale text deduplication pipelines.

### 2.1 Defining the Deduplication Problem

Text deduplication algorithms seek to identify and remove duplicate instances of the same or similar content from a document, dataset, or text corpus. These methods can be applied at different resolutions (e.g., to documents, paragraphs, sentences, or even tokens) and scopes (e.g., across a corpus or within a single document). Deduplication methods may also differ in how they define duplication (e.g., exact matches, n-gram overlap, “fuzzy”). In evaluating a deduplication method for a particular application, we are concerned with its precision, recall, and computational performance. Low *precision*, i.e., excessive false positives, is harmful as it unnecessarily reduces the size of the training dataset and limits an LLM’s ability to learn the underlying distribution of the text data. Low *recall*, i.e., excessive false negatives, admits a great degree of duplicate content in our dataset, with negative effects on learning as well as potentially inducing degenerate behavior such as memorization. High *computational performance* is critical due to the extreme size of modern LLM datasets. For example, peS2o (Sol-

daini & Lo, 2023) contains roughly 39 million documents, Google Scholar is estimated to index some 400 million academic papers and other scholarly literature (Gusenbauer, 2019), DOLMA (Soldaini et al., 2024) contains roughly 5B documents, and RedPajama (Computer, 2023) has just over 100B documents—scales at which naive pairwise comparisons become infeasible. Text deduplication algorithms are needed that can efficiently process billions of documents while maintaining high precision and recall.

Here, we focus on methods for document-level deduplication. Suppose an operator  $S(d_i, d_j) : D \times D \rightarrow [0, 1]$  that takes as input two documents and outputs a similarity score between 0 and 1. We define our task as follows: Given a corpus of  $N$  documents,  $D_N = \{d_1, \dots, d_N\}$ , and a similarity threshold,  $T \in [0, 1]$ , identify every document  $d$  such that  $\exists d' \in D_N$  such that  $d' \neq d$  and  $S(d, d') \geq T$ .

Just how to define duplication is a subtle question. For example, one could stipulate simply that two documents must be byte-for-byte identical: equivalent to setting  $S$  to measure byte overlap and  $T = 1.0$  above. However, this definition may exclude many document pairs that we would like to consider duplicates due to great similarity in their token distributions or semantic content, as are encountered frequently in practical text ingestion pipelines. For example, the same document may be available from two separate venues (say, a pre-print on Arxiv and a camera-ready conference publication) that differ only slightly due to editorial decisions. When web-scraping text or obtaining articles from publishers, text data may be obtained in varied formats (e.g., HTML, PDF, XML), and various format-specific text parsing pipelines tend to induce small differences even when applied to the same underlying document. Even simply parsing two PDFs of the same document introduces complications—for instance, the choice of PDF parser may be influenced by the document’s origin, publisher, or language. PDF parsers that rely on optical character recognition (OCR) can also introduce artifacts that render exact matching woefully ineffective.

For these reasons, many deduplication techniques choose to take a “fuzzy” perspective on duplication, allowing for minor differences between items in duplicate pairs. One way to do this is to alter one’s choice of similarity metric to encapsulate a broader semantic notion of similarity. This is common in so-called semantic deduplication techniques such as SemDeDup (Abbas et al., 2023), which employ embeddings from pre-trained language models to identify semantically similar text data. Unfortunately, these techniques require training language models and performing inference on each data point before clustering similar items to identify duplicates, which is prohibitively expensive in the extreme-scale regime with billions of documents. It is more efficient to instead establish a similarity metric based

on the raw text content of each document.

## 2.2 Jaccard Similarity and MinHash Approximation

Popular techniques such as those used to build the DOLMA (Soldaini et al., 2024) and DataComp-LM (Li et al., 2024) datasets measure n-gram overlap between documents to identify duplicates. Alternatively, some techniques—such as MinHashLSH—choose a set-theoretic notion of document similarity.

A text document can be viewed as a bag of n-grams, and the similarity between two such sets,  $A$  and  $B$ , defined in terms of the Jaccard similarity:  $J(A, B) = \frac{|A \cap B|}{|A \cup B|}$ .

However, computing Jaccard similarity requires evaluating set membership for each item present in the union of both sets. For documents, this set size can be large, and computing the Jaccard similarity explicitly can be time-consuming. The MinHashing procedure (Broder et al., 1998) provides a means of efficiently approximating the Jaccard similarity of two documents. Rather than compare the elements of each set directly, this procedure randomly permutes the order of set elements to produce so-called MinHash signatures. The probability that the MinHash signatures for two sets are equal is exactly equal to the Jaccard similarity of the two sets (Broder et al., 1998; Leskovec et al., 2014). Therefore, we can repeatedly apply random permutations to the two sets’ elements and compare them for equality to obtain an estimate of the Jaccard similarity. Our estimate grows more accurate as we increase our sample size.

## 2.3 MinHashLSH for Scalable MinHash

The use of MinHash-estimated Jaccard similarity,  $J_H$ , as an approximation for Jaccard similarity,  $J$ , provides for more efficient similarity computations because the procedure employed computed to estimate the similarity of two documents has linear complexity in the number of random permutations applied, rather than quadratic complexity in the number of n-grams present in each document. However, the need to compare every pair of documents for similarity during deduplication still introduces a quadratic cost when comparing all documents in a corpus.

Fortunately, there exists a technique that allows us to compare only those pairs with a high likelihood of similarity, rather than all pairs. MinHashLSH (Leskovec et al., 2014) applies a locality-sensitive hashing scheme for MinHash signatures to avoid comparing dissimilar documents. Given a set of MinHash signatures, one can construct a MinHash signature matrix in which each column contains all MinHash values for a single document. Then, given a desired Jaccard similarity threshold,  $T$ , one can determine an optimal band size,  $b$ , with which to group the rows of the MinHash signature matrix. This results in  $b$  groupings of

rows (bands) of size  $r$ . With high probability, if two bands of the MinHash signature matrix are exactly identical for two documents (columns), then those two documents have Jaccard similarity greater than or equal to  $T$ . This enables us to produce a series of candidate pairs of documents that we can take as duplicate pairs or further evaluate for similarity. This method avoids pairwise comparison between every pair of documents in the corpus but introduces associated false positive and false negative probabilities due to the use of locality-sensitive hashing.

One can implement a prefix-tree or suffix array to search efficiently for candidate duplicate documents when evaluating a new document for duplication. However, this requires storing a representation of each document, and while MinHash representations are small compared to raw documents, the MinHashLSH index becomes large for large datasets: e.g., 23 TB for 5B documents. Additionally, inserting and querying these tree-structured indices constitutes a significant portion (>80%) of algorithm runtime, which can become prohibitive at extreme scales (see Figure 1). Here we remedy this space and runtime issue by employing a novel combination of MinHashLSH and Bloom filters.

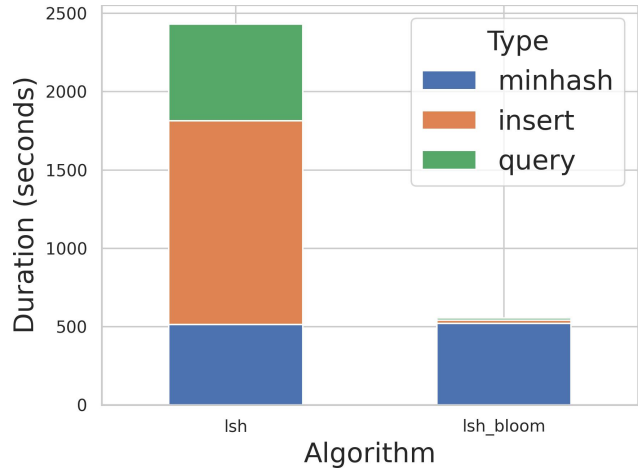


Figure 1. Breakdown of wall clock time on 5% of peS2o for the conventional MinHashLSH algorithm and our LSHBloom method

## 2.4 Bloom Filter

The Bloom filter (Bloom, 1970) is a space-efficient probabilistic data structure used to test whether an element is a member of a set. It can determine with certainty if an element is not in the set, but if it indicates the element is in the set, there is a small probability of error. That is, the Bloom filter may admit false positives but not false negatives.

A Bloom filter consists of a bit array of size  $m$ , with all bits initially set to 0.  $k$  independent hash functions are defined, each of which hashes an element to an integer in the range 1 to  $m$ . To insert an element into the Bloom filter, it is passed

through the  $k$  hash functions, and the resulting  $k$  positions in the bit array are set to 1. We can then check if an element is in the set by hashing using the same  $k$  hash functions. If all  $k$  of the corresponding positions in the bit array are 1, the filter returns a positive. If any of those  $k$  bits is 0, the element is not contained in the set.

Bloom filters have been used extensively in distributed computing for a variety of purposes (Broder & Mitzenmacher, 2004; Chervenak et al., 2008; Tarkoma et al., 2011).

### 3 RELATED WORK

Having provided background on efficient document similarity measures, we briefly review related work on deduplication methods employed by groups developing LLMs. LLM developers use various methods, many ad hoc, to deduplicate the large quantities of text content used for LLM training regardless of whether the text comes from a traditional source or synthetic data. Many leading LLMs do not discuss the deduplication and curation methods they use, such as Llama-2 (Touvron et al., 2023b), Claude, Gemini (Gemini Team, 2024a;b), and GPT versions 3 and later (OpenAI, 2024). Others, such as Palm2 (Anil et al., 2023) and Gemma (Gemma Team, 2024; Gemini Team, 2024b), provide only limited details. Others, including several leading models, provide extensive details about how data for their model was deduplicated including Olmo’s Dolma dataset (Soldaini et al., 2024) as well as its successor DataComp-LM (Li et al., 2024), Llama 1 (Touvron et al., 2023a), Llama 3 (Dubey et al., 2024), and OPT (Zhang et al., 2022). For example, Llama 3 (Dubey et al., 2024) was trained on a dataset that was deduplicated by using MinHash over the entire dataset, line-level deduplication using CCNet for lines replicated over six times in each bucket of text based on coarse categorization (e.g., math reasoning, geometry/trigonometry), and URL deduplication for web data. Phi uses an approach described by Gunasekar et al. (2023) and elaborated on in their Phi 2.0 report (Jawaheripi, 2023) that starts from deduplicated code and text, but they do not discuss deduplicating after generation.

The most common methods used across these model sources include metadata-based approaches, MinHashLSH, CCNet, and n-gram-based methods. Metadata-based approaches (e.g., exact URL matching, DOIs) are by far the simplest approach; however, they are applicable only in cases in which a reliable source of metadata exists, such as those provided by a single publisher. Unfortunately, for our target datasets this metadata is often unavailable or is otherwise untrustworthy. This can occur, for example, when: 1) a document appears in two or more collections with different metadata schemes (e.g., ArXiv vs. ACM, different GitHub repositories); 2) sections/paragraphs of text are copied, as in extended journal versions of computer science papers; or

3) metadata is not curated (e.g., an ArXiv entry may not be updated to reflect a subsequent publication).

**CCNet** (Wenzek et al., 2020) (CCNET) is an automated preprocessing pipeline that aims to extract high-quality monolingual language datasets from the Common Crawl corpus of Web content. CCNet performs dataset sharding, deduplication, language identification, and quality filtering. CCNet’s deduplication technique begins by preprocessing text so that all characters are lowercase and eliminates special Unicode characters. CCNet then splits documents by newline characters, computes SHA1 hashes for each resulting unit of text, and compares SHA1 hashes between units of text to identify duplicates. CCNet can optionally compare to a single document, a subset of documents, or all documents.

**Dolma** (Soldaini et al., 2024) deduplicates paragraphs by exact matching using a Bloom filter (DOLMA). An alternative procedure (DOLMA-NGRAM) splits paragraphs into n-grams and checks whether n-grams are duplicated by querying against a Bloom filter. If the proportion of duplicated n-grams in a paragraph exceeds a threshold,  $T \in [0, 1]$ , then that paragraph is marked as a duplicate. By default,  $T = 1.0$  in their configuration.

**DataComp-LM** (Li et al., 2024) (DCLM) deduplicates items at the document and paragraph levels simultaneously by using a Big Friendly Filter, an extension of the Bloom filter. They split each document into paragraphs on newline characters and tokenize paragraphs with the UniSeg tokenizer (Ye et al., 2023). For each paragraph, they query each n-gram against their Bloom filter and remove duplicated n-grams. If the proportion of duplicated n-grams in a paragraph exceeds a user-defined threshold,  $T \in [0, 1]$ , the entire paragraph is removed from the document. If the proportion of duplicated n-grams in the entire document exceeds the user-defined threshold, then the document is removed from the dataset. They employ the same threshold parameters for paragraph-level and document-level deduplication.

## 4 THE LSHBLOOM METHOD

### 4.1 Overview

We propose a memory-efficient alternative to MinHashLSH for document deduplication. In particular, rather than employ the traditional prefix-tree index used to find exact matches between the MinHash signatures in each band, we develop a novel Bloom filter index structure for MinHashLSH. We instantiate a series of Bloom filters, one corresponding to each band of the MinHash signature matrix. By inserting several rows of MinHash signatures into a fixed number of Bloom filters of fixed size, we achieve a large reduction in space usage by the index and a significant speedup in searching for duplicates. The size of each Bloom filter and the number of hash functions utilized are



determined according to the optimal filter size as computed by the method of [Bender et al. \(2022\)](#).

## 4.2 Insertion

Given a Jaccard similarity threshold  $T \in [0, 1]$ , MinHashLSH traditionally groups signatures into a series of  $b$  bands of size  $r$  such that an exact match in one band for two columns of the MinHash signature matrix denotes a duplicate document. That is, with high probability, the two documents with a matching band have Jaccard similarity greater than or equal to the threshold. To avoid storing the signatures in full, as is required by MinHashLSH, we assign each band of the minhash signature matrix its own Bloom filter. Thus we have in total  $b$  Bloom filters, and a collision in any one indicates a duplicate document.

When we wish to insert a document into the index, we first compute the MinHash signature for a document, which is a vector of integers (each produced by a universal hash function). Then, we group the rows into  $b$  bands of size  $r$ , as dictated by the MinHashLSH algorithm. It is desirable to hash this list of integers into a single value that we can readily insert into a Bloom filter in a manner that minimizes collisions. To do so, we take advantage of the fact that each MinHash signature value is computed using a universal hash function (in our case, SHA1 ([3rd & Jones, 2001](#))), with the property that the probability of a hash collision is  $1/N$  where  $N$  is the range of possible hash values. Then, we can hash a vector  $\bar{x}$  of  $r$  integers as follows ([Carter & Wegman, 1977](#)):

$$h(\bar{x}) = \left( \sum_{i=1}^r h_i(x_i) \right) \bmod N$$

This approach enables us to map each band to a single integer value with known collision probability. To insert a document into the index, we simply insert each of the  $b$  final integer values into each of the  $b$  Bloom filters corresponding to the  $b$  bands in the signature matrix.

## 4.3 Querying

Querying a document against the index is a similar process to insertion. We compute the document’s MinHash signature, group the rows into bands, and then hash each band as described in [Section 4.2](#). If there is a collision for any band of signatures, we mark that document as a duplicate. This approach mimics the original MinHashLSH procedure and gives the same probabilistic guarantees with respect to the Jaccard similarity of candidate pairs, with a slight increase in false positive probability due to our use of Bloom filters. However, the false positive probability is configurable and can be set to be arbitrarily low (in our experiments, we set the effective false positive rate as low as  $1e-5$ ).

## 4.4 Error Rate

In traditional MinHashLSH, a collision in any band causes a document to be marked as a candidate duplicate. This is also the case in LSHBloom. However, we make use of Bloom filters, which incur additional false positive overhead. Here, we analyze the error rate of LSHBloom. A false positive is a non-duplicate document that is incorrectly identified as a duplicate by LSHBloom, and a false negative is a duplicate document that is not marked as a duplicate by LSHBloom.

Given a Minhash signature matrix with  $b$  bands with  $r$  rows each corresponding to  $b$  Bloom filters with their individual false positive rates set to  $p$ , the net false positive overhead of using Bloom filters will be  $p_{\text{effective}} = 1 - (1 - p)^b$ . This is because a single collision in any band constitutes a duplicate, and false positive rates are independent for each Bloom filter. Therefore, the probability that no Bloom filter reports a false positive is  $(1 - p)^b$ , and the overall false positive rate overhead introduced by using Bloom filters is the complement of this value.

The relationship between the false positive rate of the Bloom filter and the overall error rate of LSHBloom can be understood as follows. [Leskovec et al. \(2014\)](#) determine that the false positive and false negative rate of MinHashLSH can be given by the following integrals, where  $T$  is the Jaccard similarity threshold,  $b$  is the number of bands in the MinHash signature matrix, and  $r$  is the number of rows in each band:

$$FP = \int_0^T 1 - (1 - t^r)^b dt \quad (1)$$

$$FN = \int_T^1 1 - (1 - (1 - t^r)^b) dt \quad (2)$$

[Zhu et al. \(2016\)](#) demonstrate how to tune the values of  $b$  and  $r$  to minimize these errors in MinhashLSH. We follow this approach in our work.

A Bloom filter will not yield false negatives, so the overall false negative rate is unaffected in LSHBloom. If MinHashLSH yields a false positive (by “mistakenly” yielding two identical bands), we get a false positive result. If MinHashLSH does not yield a false positive, the bloom filter index may still yield a false positive result with probability  $p_{\text{effective}}$ . Therefore, the overall false positive rate for LSHBloom is given by:

$$FP_{\text{total}} = FP + (1 - FP) * p_{\text{effective}}$$

Note that  $p_{\text{effective}}$  can be set arbitrarily small in practice so that this overhead is negligible.

## 4.5 Advantages

We seek to measure duplication based on the Jaccard similarity between set-theoretic representations of documents as bags of  $n$ -grams. Since this representation stems from the  $k$ -shingling of documents and many practical scenarios may admit an extremely large universe of such  $k$ -shingles (e.g., if we allow our set of documents to include a variety of encodings, latex, tabature, figures, etc.), practitioners may want to reflect this additional complexity by increasing the range of MinHash hash values or increasing the accuracy of MinHash estimates of the Jaccard similarity. This can be done by increasing the number of permutations applied to prevent as much information loss as possible via compression of the text representation. A scalable deduplication algorithm should be amenable to such considerations. LSHBloom allows for the space-efficient representation of extremely large deduplication indices. This is due to the Bloom filter’s ability to determine the set membership properties of objects of arbitrary size with the same number of bits. If we expect  $n$  documents and desire a false positive probability of  $p$ , then each Bloom filter requires exactly  $m = -n * \log(p) / (\log(2))^2$  bits (Bender et al., 2022).

As an example, say we use a Jaccard similarity threshold  $T$  of 0.8, and 128 random permutations for MinHashing. For these parameter settings, MinHashLSH creates nine bands for the MinHash signature matrix, meaning that we require nine Bloom filters for our index. For these settings, setting a conservative  $p_{\text{effective}} = 1e-10$  for a dataset of 10B documents would only require 590 GB of space. Meanwhile, traditional MinHashLSH may require 46 TB to store the equivalent LSHIndex—exemplifying the scalability of our technique.

While the use of Bloom filters to satisfy our memory and runtime constraints introduces additional false positive errors, it also provides another level of configuration. This is another advantage of our technique: we can observe massive gains in space while only introducing a single additional hyperparameter, meaning that our hyperparameter tuning burden is comparable to that of MinHashLSH.

The additional false positive error overhead scales with the number of documents we intend to insert and scales inversely with the size of each filter. Therefore, there is a trade-off between memory usage and correctness that is mediated by the size of each filter. However, even for very large datasets of, say, 100M documents, this does not prove to be an issue as we can set an effective false positive probability overhead as low as  $1e-15$  and only incur a space cost of 8.5 GB for our index compared to 460 GB for the equivalent LSHIndex.

## 5 EVALUATION

Broadly our evaluation considers two concerns important for large-scale deduplication: correctness and performance. As expressed above, both the correctness and performance of these methods are critical for their application to extremely large datasets. For correctness, we compare LSHBloom against state-of-the-art deduplication methods when applied to a dataset of 25K PDFs containing labeled duplicates. Without correctness, the system may either under-identify duplicates, resulting in decreased downstream inference quality; or over-identify duplicates, unnecessarily decreasing the volume of the training dataset. For performance, we consider the storage utilization and runtime of each method on a larger collection of PDFs, peS2o (Soldaini & Lo, 2023). Without performance, it may not be possible to process large datasets on available hardware resources.

We conduct a comprehensive analysis of various deduplication methods. Specifically, as each method includes a set of parameters whose values affect both their correctness and resource usage, we evaluate each using a range of parameter settings. Understanding these relationships is crucial to making the best use of available resources.

In the following subsections, we describe the details of our evaluation beginning with the evaluation methodology, then proceeding to correctness, performance, and finally the tradeoffs between performance and correctness.

### 5.1 Experimental Methodology

In this subsection, we describe our experimental methodology including testbed hardware, state-of-the-art methods for comparison, our evaluation metrics, and datasets.

#### 5.1.1 Testbed Hardware

We ran our experiments on the Polaris supercomputer, a 560-node HPE Apollo 6500 Gen 10+-based system. Each node is equipped with a 32-core AMD Zen 3 2.8 GHz processor, with 512 GB of DDR4 RAM, and 3.2 TB of local SSD: representative of nodes found in large-scale high-performance computing facilities used for deduplication tasks.

#### 5.1.2 Baselines

We compare LSHBloom with state-of-the-art deduplication methods widely used in large-scale LLM training: MinHashLSH, Dolma, Dolma-ngram, CCNet, and DataComp-LM. We normalized all implementations to Python-only implementations to avoid performance differences due to implementation language. This also enabled us to use the same Bloom filter implementation for each technique. Below, we describe any changes or configurations to the methods.

**MinhashLSH** (Broder, 1997) (LSH): We configure the tra-

ditional MinhashLSH algorithm as described in Section 2.3.

**DOLMA:** We apply both standard DOLMA and DOLMA-NGRAM methods to document-level deduplication. Specifically, for DOLMA we determine whether a document is duplicated by identifying duplicated paragraphs and comparing the percentage of document text duplicated to the overlap threshold,  $T$ . For DOLMA-NGRAM, we split each document into a set of n-grams with a given size, determine how many n-grams are duplicated according to the Bloom filter, and compare the percentage of n-grams that are duplicated within that document to the overlap threshold,  $T$ . Our implementation aims to faithfully extend Dolma’s two primary deduplication methods to document-level deduplication. **CCNet** originally performs deduplication at the paragraph level; however, here, we apply it at the document level by measuring the proportion of duplicated paragraphs in any given document against some tolerance threshold,  $T \in [0, 1]$ . For a fair comparison, we compare documents to all other documents—just as MinhashLSH does. When evaluating memory usage, we apply all CCNet optimizations, such as the use of memory-efficient hash sets. Additionally, we augment CCNet with multiprocessing where appropriate (e.g., when computing hashes over each document) for a more appropriate comparison during our scaling tests. **DataComp-LM:** For our evaluation, we compare only with their document-level deduplication procedure.

When considering the correctness of techniques that operate on n-grams and use Bloom filters, we must give an expected n-gram count in the corpus; choosing this count accurately is crucial to make a fair comparison. Because counting the number of n-grams of a given size in a corpus is far more computationally expensive than counting the number of documents (as is required for our technique), we compute an estimate of the n-gram count in a corpus. To do so we randomly sample  $N = 1000$  documents from the corpus, compute the mean n-gram count from our sample, and multiply the mean n-gram count by the number of total documents in that corpus to obtain an estimate of the number of n-grams in the entire corpus. We parameterize Bloom filters for n-gram techniques using this estimate.

### 5.1.3 Metrics

**Correctness:** We require methods that can distinguish meaningfully between duplicate and non-duplicate documents. We want these methods to yield a few false positives (to retrain the maximal number of training examples), few false negatives to reduce the prevalence of duplication in our dataset, and to provide a reasonable combination of accuracy and performance. Particularly for large-scale deep learning, retaining the maximal number of training examples is highly desirable—therefore, we want few false positives. Similarly, allowing a large number of duplicates into the training corpus is unacceptable as this affects the model’s

training efficiency and may increase the likelihood that our model develops degenerate behaviors such as memorization. To account for these considerations, we evaluate each deduplication technique based on its ability to mitigate both false positives and false negatives using the F1 score:

$$F1 = \frac{TP}{TP + \frac{1}{2}(FP + FN)}$$

**Performance:** To assess the resource costs of each technique we measure wall clock time and disk space usage. We measure wall clock time using `/user/bin/time` and assess each method’s disk space requirements using Python’s standard `os.path.getsize` method.

### 5.1.4 Constructing the correctness evaluation dataset

To permit the rigorous evaluation of de-duplication methods, we created an evaluation dataset By acquiring 24,000 scientific articles (4156 scraped from HTML and 20,844 parsed from PDF) from a variety of sources, spanning a diverse set of disciplines (mathematics, physics, chemistry, biology, economics, engineering, and medicine) as identified through keywords or classification algorithms based on metadata. We then duplicated 1000 of those documents for a total of 25,000. We then translated each PDF into a sequence of tokens by applying a text extraction or OCR tool—one of PyMupDF (Smith, 2024), Nougat (Blecher et al., 2023), Marker (Paruchuri, 2024), Grobid (Lopez, 2009), or Tesseract (Smith, 2007), each used with roughly the same frequency—for a total of 24,956 HTML or parsed PDF documents (as 44 failed to parse). Finally, we transformed 419 of these documents by removing a randomly selected subset of its tokens

### 5.1.5 Choice of values for Tuning Parameters

We evaluate each technique over a range of evenly spaced parameter settings using a grid search. Specifically, for any threshold parameters such as the Jaccard similarity threshold for MinHashLSH and the text overlap threshold for DataComp-LM, we evaluate values between 0.2 and 1.0 in steps of 0.2. We vary the number of permutations for MinHashLSH from 32 to 256 by powers of two. For n-gram techniques, we explore n-gram sizes 1, 2, 5, 7, 13, and 26. For MinHashLSH and LSHBloom, we additionally gathered results for  $T = 0.5$  and 48 random permutations as the surrounding parameter values demonstrated sharp jumps in the F1 score, and we wanted to explore this change in the score at a finer granularity. All Bloom filters (for DOLMA/DOLMA-ngram, DataComp-LM, LSHBloom) are assigned a false positive rate  $p$  of  $1e-5$ . For LSHBloom, this equates to  $p_{\text{effective}}$  between  $1e-5$  and  $8e-4$ , depending on the value of  $T$  and the number of MinHash permutations. The value of  $p_{\text{effective}}$  at various parameter settings for this

benchmark is illustrated in Figure 2.

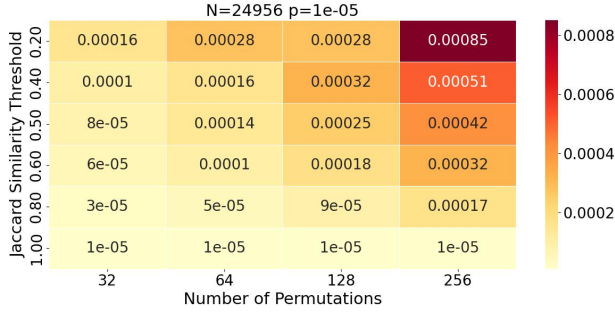


Figure 2. Value of  $p_{\text{effective}}$  (the effective false positive rate overhead) in our correctness benchmark at various parameter settings given  $N=24,956$  documents and  $p=1e-5$

It is possible that one could properly tune these deduplication methods further using incremental changes in their parameter settings. However, the expensive runtime of n-gram techniques on large corpora coupled with the need to comb over parameter settings in small increments points to the inherent difficulty and cost associated with properly tuning these methods for maximal deduplication performance. Meanwhile, MinHashLSH-based techniques readily admit parameter tuning with relatively low runtime and memory costs. Furthermore, MinHashLSH and LSHBloom both benefit from established theoretical analysis that effectively bounds their false positive and false negative rates given the values for the Jaccard similarity threshold and number of permutations. MinHashLSH-based techniques offer the greatest opportunity to tune parameters for the best deduplication performance, particularly when faced with a constrained parameter tuning budget. This is particularly true for LSHBloom, which is significantly cheaper to run than MinHashLSH in terms of wall clock time and disk usage—particularly as we increase the size of our dataset to several million or several billion documents (see Section 5.3.1).

## 5.2 Deduplication Correctness

### 5.2.1 LSH Methods

The parametrization of methods such as MinHashLSH and LSHBloom has dramatic effects for their accuracy so tuning the parameters is a critical aspect of deploying them for large-scale deduplication efforts. Figures 3a and 3b show the F1 score for MinHashLSH and LSHBloom, respectively, with different settings for the Jaccard similarity threshold,  $T$ , and the number of permutation functions for MinHashing on the 25K dataset. As the number of permutation functions increases, the MinHash estimate of the Jaccard similarity becomes more accurate. However, there is a clear saturation point at 128 permutation functions, after which we observe only marginal performance increases. This is particularly

relevant for LSHBloom, for which increasing the number of permutation functions blindly can lead to slight degradations in performance—due to the introduction of more bands in the signature matrix by the LSH function, requiring more Bloom filters, and therefore leading to an increase in the false positive rate. This can be avoided, however, by jointly tuning the false positive rate and the number of permutation functions. The threshold parameter affects the F1 score the most, as it defines the sensitivity of the deduplication algorithm to overlap in text content. For our particular benchmark, a threshold of 0.6 yields the best performance, as more stringent thresholds lead to a diminishing F1 score.

### 5.2.2 Paragraph Level and N-Gram Methods

Figure 5 shows the impact of adjusting the overlap threshold parameter for paragraph-level methods, DOLMA and CCNet, on the F1 score. Distinguishing documents at the granularity of individual paragraphs is highly error-prone and results in poor performance. N-gram methods such as DOLMA-ngram and CCNet do comparably better in terms of F1 score but still lag just behind the scores of the MinHashLSH-based techniques. Figures 4a and 4 show the results of varying both the n-gram size and overlap threshold parameters for each n-gram method. While n-gram methods enable a finer-grained comparison between individual documents and the rest of the corpus, these methods evaluate overlap by determining the portion of n-grams in a document that is exactly duplicated in the corpus. This is similar to the notion of evaluating the set overlap between documents but can be sensitive to the relative frequency of n-grams within a document, resulting in increased error. Moreover, both paragraph and n-gram methods demonstrate a lack of sensitivity to changes in their parameter settings in this benchmark, indicating bias in their predictions. Specifically, we find that paragraph-level methods tend to be biased toward marking documents as duplicates. In contrast, n-gram methods yield many false negatives at high thresholds and yield many false positives at low thresholds.

## 5.3 Deduplication Resource Usage

As LLM training datasets grow ever larger, it is important that text deduplication strategies can scale effectively to larger quantities of documents. Here we work with the peS2o dataset (Soldaini & Lo, 2023) of roughly 8M full-text and 31M abstracts. We evaluate each deduplication method on subsets of peS2o of various sizes and compare them in terms of wall clock time and disk usage (e.g., for storing an index or Bloom filter). We do not evaluate the n-gram techniques for scaling capacity on peS2o due to their prohibitive cost. DataComp-LM, for example, takes one hour just to deduplicate 25,000 examples in our previous benchmark in Section 5.1.4. Even processing just 10% of peS2o using this technique would require 160 hours by



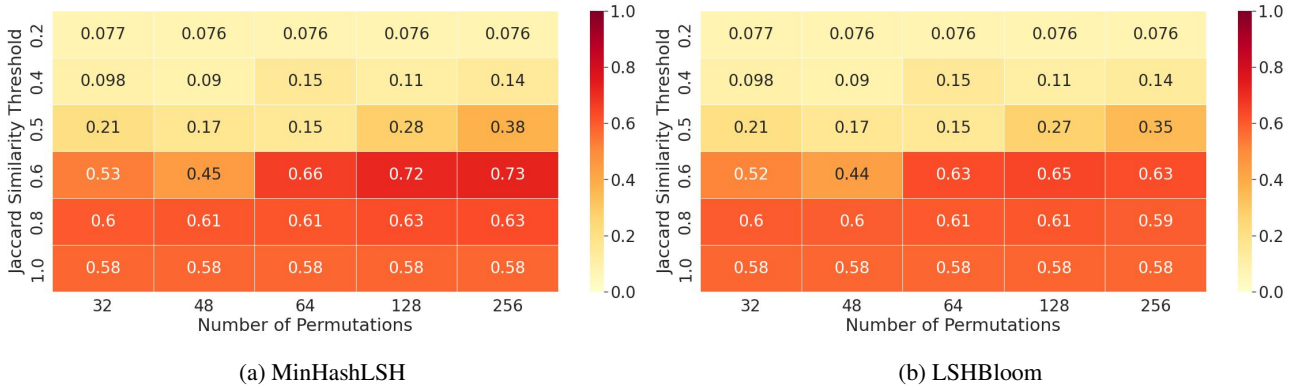


Figure 3. F1 score for LSH techniques as a function of the number of permutations (x-axis) and Jaccard similarity threshold (y axis).

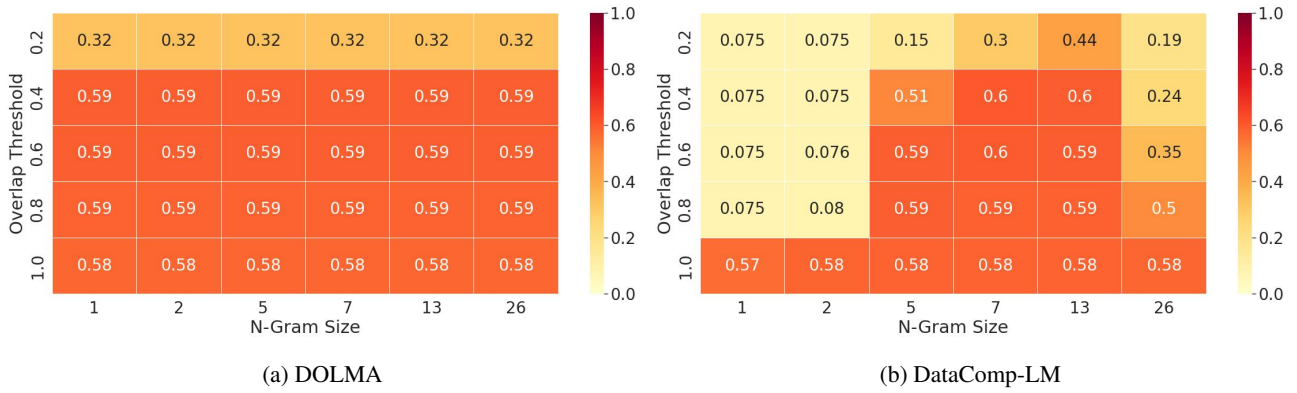


Figure 4. F1 Score for N-Gram techniques as a function of N-Gram size (x axis) and overlap threshold (y axis).

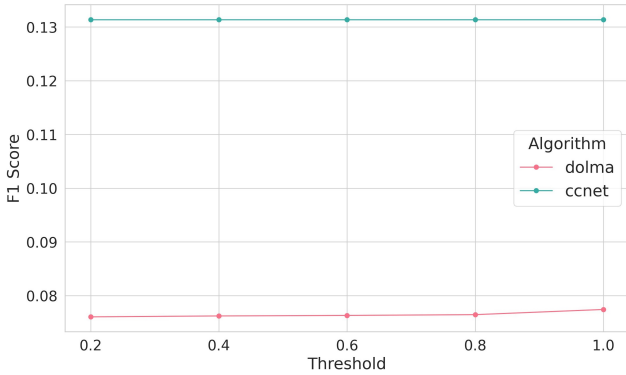


Figure 5. F1 score vs. threshold for paragraph-level techniques.

extrapolation. We discuss the results for other methods below.

### 5.3.1 Scaling of Resource Usage

In this section, we study how resource usage grows with the number of documents. In these experiments, we set any threshold parameters to 0.8, the number of random permu-

tations for our LSH techniques to 128, and  $p = 1e-5$  for all Bloom filters. Ideally, we would have informed these parameter settings based on the optimal settings from [Section 5.1.4](#); however, scaling experiments on peS2o regularly took over a day to complete, limiting our ability to tune parameters and necessitating heuristic choices for parameter settings as we evaluated deduplication quality in parallel.

[Figure 8](#) shows the wall clock time to deduplicate various subsets of peS2o with each method. Notably, MinHashLSH incurs the largest runtime while paragraph-level techniques such as DOLMA and CCNet are relatively fast. However, DOLMA and CCNet achieve this speed with significantly lower deduplication performance: see [Section 5.1.4](#). Meanwhile, LSHBloom, which has comparable deduplication performance to MinHashLSH, is much faster. To deduplicate all of peS2o, LSHBloom takes just five hours, whereas MinHashLSH takes 14 hours—almost a  $3\times$  speedup. [Figure 1](#) shows the breakdown of this time for MinHashLSH and LSHBloom; we can see that, as expected, the time to compute MinHashes is the same; however, the insertion and query latencies are significantly reduced in the Bloom filter-based index compared to the traditional LSHIndex.

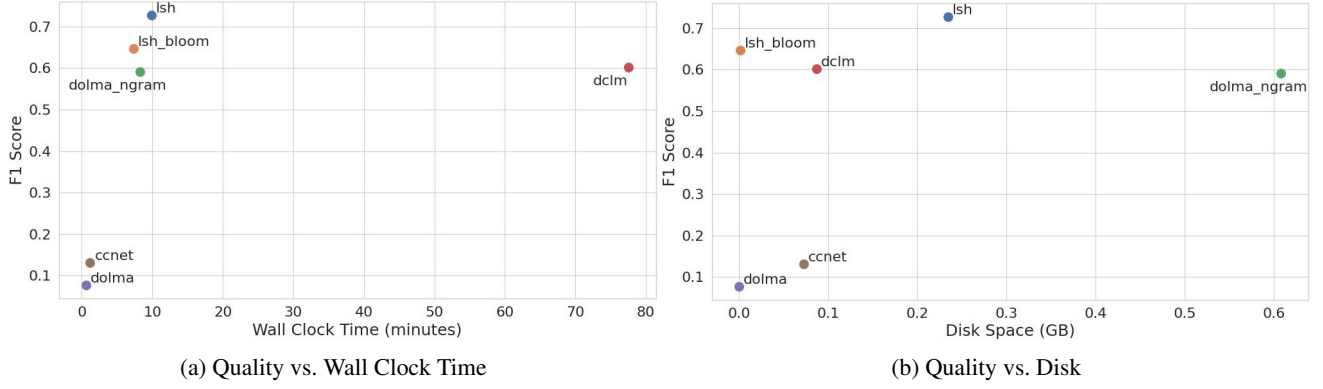


Figure 6. Pareto plots for F1 score vs. resource usage.

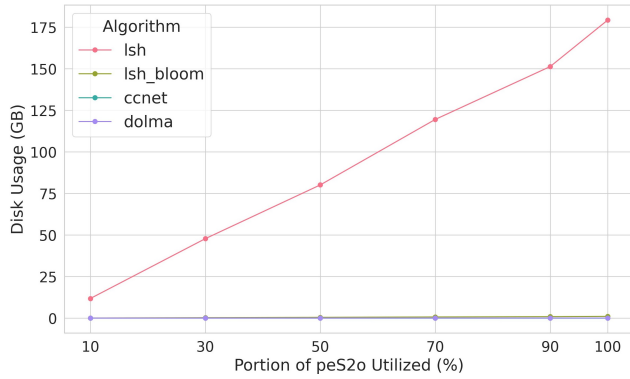


Figure 7. Disk size of index for different methods, as measured for peS2o subsets of increasing size.

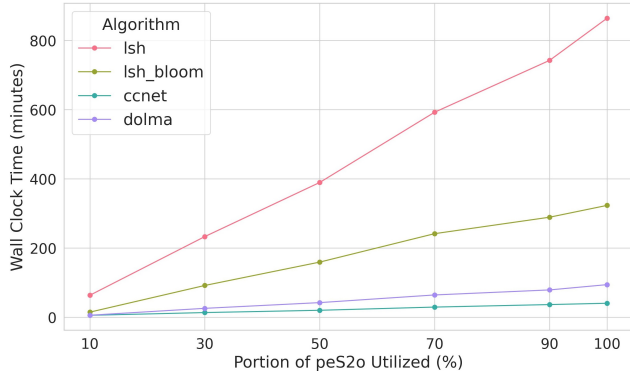


Figure 8. Wall clock times for different methods, as measured for peS2o subsets of increasing size.

Figure 7 shows the disk usage for each method. We again see a similar trend. The paragraph-level techniques maintain only a single Bloom filter which leads to extremely low disk usage. Meanwhile, MinHashLSH’s index grows steadily in size as the number of documents increases. LSHBloom’s index also grows linearly in the number of documents but at a reduced rate. When processing all of peS2o, LSHBloom

requires just 1 GB vs. the 180 GB taken by MinHashLSH—a  $180\times$  reduction.

### 5.3.2 Estimated Resource Usage at Scale

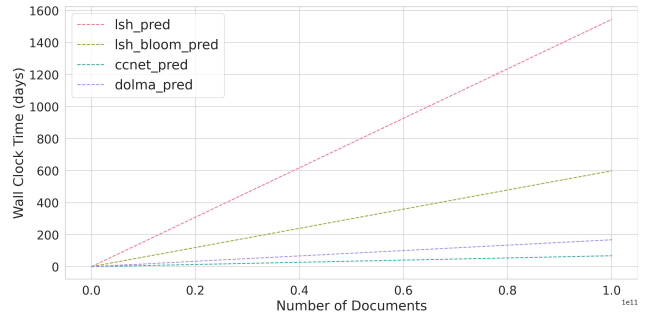


Figure 9. Predicted wall clock times vs. number of documents (in Billions) for different methods

LSHBloom shows clear performance benefits over MinHashLSH; however, our data also indicates that these gains grow linearly in the number of documents processed. Given the strong linear trend in runtime for each method, we attempt to extrapolate their resource usage by modeling runtime with respect to the number of documents by fitting a set of linear regressors. We choose to extrapolate our results forward to predict each method’s resource utilization if processing roughly 5B documents (the approximate size of the DOLMA corpus). Figure 9 shows the extrapolated runtime for each technique against corpus size in the number of documents. MinHashLSH would require 77.2 days to process DOLMA. Meanwhile, LSHBloom could achieve the same in only 30 days and of processing time—a speedup of over 250%. Disk usage is also a crucial limiting factor, if index sizes grow beyond that which is feasible to store, read, and write to then deduplication efforts would be infeasible at an extreme scale. Table 1 Shows the linearly extrapolated index size for MinHashLSH and the computed index size for LSHBloom at various effective false posi-

tive rate settings,  $p_{\text{effective}}$ , for its Bloom filter index using the method described in Section 4.5. Assuming a corpus of 5B documents, MinHashLSH requires just over 23 TB. By contrast, even an extremely conservative effective false positive rate of  $1e-15$  for LSHBloom yields an index of only 425 GB. At the scale of an extremely large corpus like DOLMA, LSHBloom offers just over  $54\times$  space savings over the traditional MinHashLSH algorithm. For a corpus of 100B documents like RedPajama, MinHashLSH would require 460 TB of disk space, whereas LSHBloom with  $p_{\text{effective}} = 1e-15$  would only require 8.5 TB—a  $54\times$  space advantage (see Table 2). Here, we observe a similar  $2.5\times$  speedup for LSHBloom compared to MinHashLSH.

Table 1. Predicted index sizes for 5B documents, for LSH and LSHBloom with different false positive overheads.

Technique	Bloom filter false positive overhead	Index disk usage (GB)
MinHashLSH	—	23,008.04
LSHBloom	$1e-5$	160.51
LSHBloom	$1e-10$	295.30
LSHBloom	$1e-15$	425.35

Table 2. Predicted index sizes for 100B documents, for LSH and LSHBloom with different false positive overheads.

Technique	Bloom filter false positive overhead	Index disk usage (TB)
MinHashLSH	—	460.16
LSHBloom	$1e-5$	3.21
LSHBloom	$1e-10$	5.91
LSHBloom	$1e-15$	8.51

#### 5.4 Performance vs. Quality Tradeoffs at Scale

Here we investigate tradeoffs between the costs of various deduplication techniques (in terms of runtime and disk usage) and their deduplication performance. Figures 6a and 6b plot the maximum F1 score for each deduplication technique against runtime and disk usage, respectively. MinHashLSH and LSHBloom dominate the alternative techniques, achieving the highest F1 scores while also exhibiting competitive runtime. Additionally, LSHBloom displays a comparable F1 score to traditional MinHashLSH while using just a fraction of the disk space for its index. As shown in Section 5.3.1, this is a crucial distinction, as rapidly increasing disk usage renders deduplication efforts infeasible at the scale of many millions or billions of documents.

## 6 CONCLUSIONS

We have introduced a new text deduplication technique, LSHBloom, that offers significant runtime and space savings

over the current state-of-the-art technique, MinHashLSH. We also conducted a thorough study of deduplication quality between our technique and other state-of-the-art deduplication algorithms commonly used in LLM training pipelines, using a custom benchmark of 25,000 scientific articles. We find that even with extensive parameter tuning for conventional MinHashLSH, LSHBloom is extremely competitive in terms of its deduplication quality, with an F1 score within 10% of the best achieved by MinHashLSH, while using far less storage and runtime. Using the peS2o dataset, we find that LSHBloom can feasibly scale to deduplicating billions of documents due to its low space requirements (a  $180\times$  space reduction from MinHashLSH) and relatively low wall clock time ( $3\times$  speedup over MinHashLSH). In contrast, alternatives demonstrate either poor deduplication quality, excessive runtime, or prohibitive space requirements for extreme-scale datasets of this size. In the future, we aim to further reduce the wall clock time of LSHBloom by employing parallelization over subsets of text datasets when inserting them into our index—an approach that we expect to enable significant acceleration.

## REFERENCES

- 3rd, D. E. E. and Jones, P. US Secure Hash Algorithm 1 (SHA1). RFC 3174, September 2001. URL <https://www.rfc-editor.org/info/rfc3174>.
- Abbas, A., Tirumala, K., Simig, D., Ganguli, S., and Morcos, A. S. Semdedup: Data-efficient learning at web-scale through semantic deduplication, 2023. Preprint arXiv:2303.09540.
- Anil, R., Dai, A. M., Firat, O., Johnson, M., Lepikhin, D., Passos, A., Shakeri, S., Taropa, E., Bailey, P., Chen, Z., et al. Palm 2 technical report, 2023. Preprint arXiv:2305.10403.
- Bender, M. A., Farach-Colton, M., Kuszmaul, J., Kuszmaul, W., and Liu, M. On the optimal time/space tradeoff for hash tables. In *54th Annual ACM SIGACT Symposium on Theory of Computing*, pp. 1284–1297, 2022.
- Blecher, L., Cucurull, G., Scialom, T., and Stojnic, R. Nougat: Neural optical understanding for academic documents, 2023. Preprint arXiv:2308.13418.
- Bloom, B. H. Space/time trade-offs in hash coding with allowable errors. *Communications of the ACM*, 13(7): 422–426, 1970.
- Broder, A. and Mitzenmacher, M. Network applications of Bloom filters: A survey. *Internet Mathematics*, 1(4): 485–509, 2004.

- Broder, A. Z. On the resemblance and containment of documents. In *Compression and Complexity of Sequences*, pp. 21–29. IEEE, 1997.
- Broder, A. Z., Charikar, M., Frieze, A. M., and Mitzenmacher, M. Min-wise independent permutations. In *30th ACM Symposium on Theory of Computing*, pp. 327–336, 1998.
- Carter, J. L. and Wegman, M. N. Universal classes of hash functions. In *9th ACM Symposium on Theory of Computing*, pp. 106–112, 1977.
- Chandran, N., Sitaram, S., Gupta, D., Sharma, R., Mittal, K., and Swaminathan, M. Private benchmarking to prevent contamination and improve comparative evaluation of llms, 2024. Preprint arXiv:2403.00393.
- Chervenak, A. L., Schuler, R., Ripeanu, M., Amer, M. A., Bharathi, S., Foster, I., Iamnitchi, A., and Kesselman, C. The Globus replica location service: Design and experience. *IEEE Transactions on Parallel and Distributed Systems*, 20(9):1260–1272, 2008.
- Computer, T. Redpajama: an open dataset for training large language models, 2023. URL <https://github.com/togethercomputer/RedPajama-Data>.
- Dekoninck, J., Muller, M. N., Baader, M., Fischer, M., and Vechev, M. T. Evading data contamination detection for language models is (too) easy, 2024. Preprint arXiv:2402.02823.
- Dubey, A. et al. The Llama 3 herd of models, 2024. Preprint arXiv:2407.21783.
- Gemini Team. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context, 2024a. Preprint arXiv:2403.05530.
- Gemini Team. Gemini: A family of highly capable multimodal models, 2024b. Preprint arXiv:2312.11805.
- Gemma Team. Gemma: Open models based on Gemini research and technology, 2024. Preprint arXiv:2403.08295.
- Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Del Giorno, A., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., Salim, A., Shah, S., Behl, H. S., Wang, X., Bubeck, S., Eldan, R., Kalai, A. T., Lee, Y. T., and Li, Y. Textbooks are all you need, 2023. Preprint arXiv:2306.11644.
- Gusenbauer, M. Google Scholar to overshadow them all? Comparing the sizes of 12 academic search engines and bibliographic databases. *Scientometrics*, 118(1):177–214, 2019.
- Javaheripi, M. The surprising power of small language models, 2023. URL [https://neurips.cc/media/neurips-2023/Slides/83968\\_5GxuY2z.pdf](https://neurips.cc/media/neurips-2023/Slides/83968_5GxuY2z.pdf).
- Lee, K., Ippolito, D., Nystrom, A., Zhang, C., Eck, D., Callison-Burch, C., and Carlini, N. Deduplicating training data makes language models better, 2021. Preprint arXiv:2107.06499.
- Leskovec, J., Rajaraman, A., and Ullman, J. D. *Mining of Massive Datasets, 3rd Edition*, chapter 3: Finding similar items, pp. 73–130. Cambridge University Press, 2014. <http://www.mmids.org>.
- Li, J., Fang, A., Smyrnis, G., Ivgi, M., Jordan, M., Gadre, S., Bansal, H., Guha, E., Keh, S., Arora, K., and Saurabh Garg, X. R., Muennighoff, N., Heckel, R., Mercat, J., Chen, M., Gururangan, S., Wortsman, M., Albalak, A., Bitton, Y., Nezhurina, M., Abbas, A., Hsieh, C.-Y., Ghosh, D., Gardner, J., Kilian, M., Zhang, H., Shao, R., Pratt, S., Sanyal, S., Ilharco, G., Daras, G., Marathe, K., Gokaslan, A., Zhang, J., Chandu, K., Nguyen, T., Vasiljevic, I., Kakade, S., Song, S., Sanghavi, S., Faghri, F., Oh, S., Zettlemoyer, L., Lo, K., El-Nouby, A., Pouransari, H., Toshev, A., Wang, S., Groeneveld, D., Soldaini, L., Koh, P. W., Jitsev, J., Kollar, T., Dimakis, A. G., Carmon, Y., Dave, A., Schmidt, L., and Shankar, V. DataComp-LM: In search of the next generation of training sets for language models, 2024. Preprint arXiv:2406.11794.
- Lopez, P. GROBID: Combining automatic bibliographic data recognition and term extraction for scholarship publications. In *13th European Conference on Research and Advanced Technology for Digital Libraries*, pp. 473–474. Springer, 2009.
- OpenAI. GPT-4 technical report, 2024. Preprint arXiv:2303.08774.
- Paruchuri, V. Marker parser, 2024. <https://github.com/VikParuchuri/marker>.
- Smith, J. PyMuPDF library, 2024. <https://pypi.org/project/PyMuPDF/>.
- Smith, R. An overview of the Tesseract OCR engine. In *9th International Conference on Document Analysis and Recognition*, volume 2, pp. 629–633. IEEE, 2007.
- Soldaini, L. and Lo, K. peS2o (pretraining efficiently on S2ORC) dataset, 2023. <https://github.com/allenai/peS2o>.
- Soldaini, L. et al. Dolma: An open corpus of three trillion tokens for language model pretraining research, 2024. Preprint arXiv:2402.00159.



- Tarkoma, S., Rothenberg, C. E., and Lagerspetz, E. Theory and practice of Bloom filters for distributed systems. *IEEE Communications Surveys & Tutorials*, 14(1):131–155, 2011.
- Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. LLaMA: Open and efficient foundation language models, 2023a. Preprint arXiv:2302.13971.
- Touvron, H. et al. Llama 2: Open foundation and fine-tuned chat models, 2023b. Preprint arXiv:2307.09288.
- Wenzek, G., Lachaux, M.-A., Conneau, A., Chaudhary, V., Guzmán, F., Joulin, A., and Grave, É. CCNet: Extracting high quality monolingual datasets from web crawl data. In *12th Language Resources and Evaluation Conference*, pp. 4003–4012, 2020.
- Ye, Y., Xie, Y., Zhang, J., Chen, Z., and Xia, Y. UniSeg: A prompt-driven universal segmentation model as well as a strong representation learner. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pp. 508–518. Springer, 2023.
- Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., Dewan, C., Diab, M., Li, X., Lin, X. V., Mihaylov, T., Ott, M., Shleifer, S., Shuster, K., Simig, D., Koura, P. S., Sridhar, A., Wang, T., and Zettlemoyer, L. OPT: Open pre-trained transformer language models, 2022. Preprint arXiv:2205.01068.
- Zhang, Y., Chen, X., Jin, B., Wang, S., Ji, S., Wang, W., and Han, J. A comprehensive survey of scientific large language models and their applications in scientific discovery, 2024. Preprint arXiv:2406.10833.
- Zhu, E., Nargesian, F., Pu, K. Q., and Miller, R. J. Lsh ensemble: internet-scale domain search. *Proc. VLDB Endow.*, 9(12):1185–1196, August 2016. ISSN 2150-8097. doi: 10.14778/2994509.2994534. URL <https://doi.org/10.14778/2994509.2994534>.