Phillipp Schoppmann, Lennart Vogelsang, Adrià Gascón, and Borja Balle

Secure and Scalable Document Similarity on Distributed Databases: Differential Privacy to the Rescue

Abstract: Privacy-preserving collaborative data analysis enables richer models than what each party can learn with their own data. Secure Multi-Party Computation (MPC) offers a robust cryptographic approach to this problem, and in fact several protocols have been proposed for various data analysis and machine learning tasks. In this work, we focus on secure similarity computation between text documents, and the application to k-nearest neighbors (k-NN) classification. Due to its non-parametric nature, k-NN presents scalability challenges in the MPC setting. Previous work addresses these by introducing non-standard assumptions about the abilities of an attacker, for example by relying on non-colluding servers. In this work, we tackle the scalability challenge from a different angle, and instead introduce a secure preprocessing phase that reveals differentially private (DP) statistics about the data. This allows us to exploit the inherent sparsity of text data and significantly speed up all subsequent classifications.

Keywords: text analysis, document similarity, multi-party computation, differential privacy

DOI 10.2478/popets-2020-0024
Received 2019-08-31; revised 2019-12-15; accepted 2019-12-16.

1 Introduction

Aggregating data held by different organizations has the potential to unlock novel data analysis applications. By increasing either the amount of training data or its dimensionality, more accurate models than the ones each organization could build using only its local dataset can be obtained. However, organizations often have conflicting interests in such scenarios: while collaboration would result in more accurate predictions, disclosing their proprietary datasets might be undesirable or infeasible from competitive and legal standpoints. Such constraints rule out solutions involving an external trusted party, thus presenting a scenario outside the scope of most classical approaches to privacy-preserving data analysis.

Multi-Party Computation (MPC) is an area of cryptography that addresses this challenge, while providing cryptographic guarantees under several natural threat models. Applied to distributed data analysis, MPC provides formal guarantees ensuring that no additional data about the inputs can be inferred from the protocol execution beyond what is already revealed by the outputs provided to each party. This has lead to several specialized MPC protocols for linear and logistic regression training [28, 47, 51], neural network training [47] and evaluation [11, 37, 42], matrix factorization [50], principal components analysis [6], as well as evaluation of decision trees and naive Bayes classifiers [14].

While MPC provides strong security guarantees, it comes at the cost of running times several orders of magnitude higher than conventional analysis. This motivates relaxed security notions for MPC, where the parties obtain some information beyond the desired result while still *provably quantifying and limiting* the incurred privacy leakage. For the latter, Differential Privacy (DP) [23] is a natural candidate which has led to recent works exploring variants of *MPC with differentially private leakage* [31, 34, 44]. Protocols under such relaxed definitions may reveal additional information, as long as that information leakage adheres to DP.

In this paper, we follow a similar approach and apply it to secure classification of text documents with k-NN on a standard TF-IDF feature representation. In k-NN, a query document is assigned a class by taking a majority vote among the k most similar documents in the training database. Despite its simplicity, k-NN enjoys remarkable theoretical properties [17] and provides competitive accuracies in a wide range of applications [26]. This power comes at the price of scalability: due to its non-parametric nature k-NN requires similarity computations against the whole dataset at prediction time. It is therefore crucial to reduce the time for each similarity computation as much as possible. Our

Phillipp Schoppmann, Lennart Vogelsang: Humboldt-Universität zu Berlin and Alexander von Humboldt Institute for Internet and Society, Berlin, Germany. Emails: schoppmann@informatik.hu-berlin.de, lennart.vogelsang@hiig.de.
Adrià Gascón: Work done while at the Alan Turing Institute, London, UK. Now at Google, London, UK. Email: adriagascon@gmail.com.
Borja Balle: Work done at Amazon Research, Cambridge, UK. Now at DeepMind, London, UK. Email: borja.balle@gmail.com.

main observation is that if we allow a one-time precomputation of differentially private statistics about the distributed dataset, we can significantly speed up classification time by using a novel sparse inner product protocol. We review our contributions and how they compose to a full k-NN protocol in the following subsection.

1.1 Our Contributions

The core of our paper consists of two novel contributions: (i) a secure two-party protocol for sparse inner products, and (ii) a mechanism for extracting differentially private IDF coefficients from text documents, and a corresponding two-party implementation. Both protocols are tailored to exploit inherent sparsity and word frequency properties commonly found in text data. These are composed to obtain a three-party protocol for distributed k-NN classification capable of withstanding arbitrary collusions. All the protocols presented in this paper are formally secure in the semi-honest model (cf. Appendix A).

Secure Document Similarity from Sparse Inner Products (Section 3). Our first contribution is a novel protocol for secure sparse inner product. It allows two parties holding private sparse vectors to compute additive shares of their inner product, while revealing nothing except an upper bound on the number of non-zero entries. As described in Section 3, this can be used to compute similarities between sparse representations of text documents. We also propose a batched version of our protocol to improve the scalability of computing many inner products in parallel. In Section 6.1.1, we experimentally evaluate the running time of our protocol for a wide range of parameters, and show that it outperforms its state-of-the-art dense counterparts by at least one order of magnitude.

Differentially Private IDF Coefficients (Section 4). Secondly, we develop a mechanism for extracting Inverse Document Frequency (IDF) coefficients from a distributed database of text documents, while guaranteeing Differential Privacy (DP). We show why a standard approach based on adding Laplace noise to each coefficient fails at this task, and formally prove privacy and accuracy guarantees for our custom mechanism. While our proposal is already of interest in the centralized setting, we further show how to instantiate it for generic circuit-based MPC, thus achieving Multi-Party Computational Differential Privacy (MPC-DP) [11]. To that end, we rely on a method for oblivious sampling without replacement that has, to the best of our knowledge, not been reported in the academic literature before. Our experiments (Section 6.2.4) showcase the advantage of having access to privatized data-dependent IDFs over a vanilla data-independent Term Frequency (TF) representation, and demonstrate that the noise introduced by our DP protocol incurs only a small accuracy loss. Finally, we implement our protocol for the special case of two parties holding a databases of secret documents and show that it scales to real-world vocabulary sizes (Section 6.1.2).

Application to Secure k-NN (Section 5). While both of the above are of independent interest, we show how in combination they allow us to implement an efficient k-Nearest Neighbors protocol in a three-party setting with two servers and one client. Here, the two servers each hold a collection of labeled documents, and the client would like to classify a document against the union of the servers' datasets. Our protocol achieves this by combining the two sub-protocols for IDF precomputation and secure inner products with a generic MPC phase for top-k selection. Apart from the final classification, our full protocol releases differentially private statistics about the dataset (i.e., IDF coefficients) as a one-time precomputation. We formalize this as *differentially private leakage* similar to previous work [31, 44] and prove security of our protocol in that model. We emphasize that these differentially private IDF coefficients need to be computed only once, and can then be reused in any subsequent classifications.

Unlike previous work on distributed k-NN (cf. Section 7), our protocol withstands arbitrary collusions among the three parties, and therefore also captures settings where there is only one server, or where the client is one of the two servers. We also stress that we are the first to even consider the feature extraction phase as part of a distributed k-NN protocol. Previous work starts directly with feature vectors as inputs and therefore implicitly assumes feature extraction can be done locally by the parties, which is not the case for TF-IDF. Thus, the fact that our protocol releases differentially private IDF values in fact strengthens the privacy guarantees compared to previous work.

We implement our k-NN protocol and show that it scales to real-world dataset sizes. For example, the time needed for a classification of a query document against a database of 28K documents is less than 40 minutes.

2 Background

2.1 Multi-Party Computation

The protocols we present in Sections 3 and 4 are in the two-party setting of secure computation. We use the standard cryptographic notion of security against semi-honest adversaries [30, 40], which we repeat in Appendix A. Intuitively, this definition guarantees that the information obtained by any adversary is the same information that is revealed in an idealized setting where a trusted party performs the whole computation and returns the result to the parties.

For our full k-NN protocol (Section 5), we need to account for the fact that we rely on the precomputation of differentially private IDF coefficients that are given to the client in addition to the classification result. Unlike previous work on MPC with DP leakage [31, 34, 44], this is a one-time preprocessing step that is used as input to subsequent classification queries. We account for this in our definition of security with differentially private leakage in Section 5.1.

2.2 TF-IDF Features

Throughout this paper, we work on *text document data*. Before being able to process this data it is necessary to construct a vectorial representation for each document. Here, we rely on the *term frequency–inverse document frequency* (TF-IDF) feature representation, which is one of the most common encodings for text data. For example, 83% of text-based document search and recommendation systems in digital libraries use TF-IDF [10].

The TF-IDF feature representation of a text document is defined with respect to a fixed vocabulary and a database of documents. Let $\mathcal{V}$ be a fixed vocabulary – e.g., $\mathcal{V}$ might be all the words in a given dictionary – and consider a database Z of documents over the common vocabulary $\mathcal{V}$. Given an arbitrary document x with words in $\mathcal{V}$, its TF-IDF representation is a $|\mathcal{V}|$-dimensional vector $\psi(x) \in \mathbb{R}^{\mathcal{V}}$ where each coordinate corresponds to a word $v \in \mathcal{V}$. The vth coordinate $\psi(x)(v)$ of this vector is the product of two terms: the *term frequency* (TF) $\phi_{\mathrm{tf}}(v, x) = |x|_v$ of v in x (i.e., the number of times v occurs in x) and the *inverse document frequency* (IDF) $\phi_{\mathrm{idf}}(v, Z) = \log((|Z| + 1)/(|Z|_v + 1)) + 1$ of v in Z (where $|Z|_v$ counts the number of documents in the database that contain the word v). By taking the product $\psi(x)(v) = \phi_{\mathrm{tf}}(v, x) \cdot \phi_{\mathrm{idf}}(v, Z)$, the TF-IDF representation ensures that coordinates corresponding to frequent words in the document are larger (TF term), while also down-weighting words that are frequent across the dataset (IDF term) and therefore not representative of a particular document.

Two properties of TF-IDF are particularly relevant to this work. First, since the TF component is zero for all words that do not appear in a document, TF-IDF vectors are very sparse. In Section 3, we use this fact to efficiently compute similarity scores between TF-IDF vectors of documents. Second, observe that the IDF component depends on the whole dataset, not only the document being encoded. It is therefore non-trivial to encode a document without access to the entire database. However, we will see in Section 4 that we can precompute differentially private IDF coefficients, which then can be safely released to allow parties to locally compute TF-IDF embeddings of their documents.

3 Sparse Inner Products and Secure Document Similarity

Similarity computation is a common task in the evaluation of non-parametric models such as k-NN, but also for example in information retrieval, clustering, and collaborative filtering. Common similarity metrics include the Euclidean distance, Pearson's correlation coefficient, or the cosine similarity. All three of these can be computed in two-party settings using only secure inner products. Here, we focus on cosine similarity, which is commonly used for text documents with TF-IDF features [43]:

$$\mathsf{sim}_{\cos}(\mathbf{a}, \mathbf{b}) = \frac{\langle \mathbf{a}, \mathbf{b} \rangle}{\|\mathbf{a}\| \|\mathbf{b}\|}.$$

When each of the feature vectors $\mathbf{a}, \mathbf{b}$ is held by one of the two parties, the denominator can be computed using a single secure multiplication [9, 29], while the numerator requires a secure inner product. As described in Section 2.2, the feature vectors $\mathbf{a}$ and $\mathbf{b}$ are sparse when encoding text documents using TF-IDF. Hence, in this section, we present a secure two-party protocol for inner products that is optimized for *sparse* inputs.

Our protocol takes as input private vectors $\mathbf{a}$ and $\mathbf{b}$ – each provided by one party – and compute an *additive secret share* of their inner product $\langle \mathbf{a}, \mathbf{b} \rangle$. Here, the elements of $\mathbf{a}$ and $\mathbf{b}$ are taken from $\mathbb{Z}_q$, where q is usually chosen as 2^σ for some bit width σ. Rational numbers can be used by relying on an appropriate fixed-point encoding [28, 47]. Since in practice, many similarity scores

need to be computed at once, we generalize our protocol and propose a *batched* version for computing multiple inner products simultaneously; in this case the goal is to compute $\mathbf{AB}$ given private matrices $\mathbf{A}$ and $\mathbf{B}$ which are respectively row and column sparse. Since the single inner product case corresponds to the multiplication of a one-row by a one-column matrix, for simplicity we sometimes use matrix notation to denote properties that apply to both the single and batched inner product case.

As discussed in Section 2.1, we work in the standard simulation-based paradigm of security, and our protocols are secure in the semi-honest model. In the dense case, there are multiple protocols [28, 37, 47] for inner products in this threat model, and we make use of them as a sub-protocol in our sparse version. While our protocol neither reveals the non-zero indexes in each party's inputs, nor their values, we require an upper bound on the *number* of non-zeros to be known. In the next subsection, we will see that such a bound on the sparsity is naturally a available in many real-world scenarios, including text classification.

3.1 Sparsity in Real-World Data

In the context of data analysis, sparsity is frequently induced by the feature representation used. For example, the well-known bag-of-word representations of documents, where a document is represented as a vector of (possibly normalized) word counts, is sparse. The TF-IDF representation is a special case of this. Furthermore, in datasets of genomic variants, individuals are represented as a set of deviations (index-value pairs) from a common reference. Compared to the 3.2 billion base pairs in the human reference genome, these deviations only make up a small fraction (∼5M sites [4]), and so this representation is also very sparse. A third example can be found in recommender systems, where users are represented by the vector of their rated items, which again amount to a tiny percentage of the total number of available items.

In all of the above applications, an upper bound on the sparsity is readily available. For genomics, upper bounds can be derived from public information about the distribution of variants [4]. In recommender systems, the input comes in the form of a list of (item, rating) pairs for each user. The length of a list directly translates to the sparsity of the corresponding vector. We note that existing recommendation protocols based on secure matrix factorization reveal the sparsity as the input size [49, 50]. Finally, for text documents, the input length does not directly give the exact sparsity (words can be repeated in a document), but it provides an upper bound on it.

Our protocol can therefore be applied to all of the settings described here. We now formally describe secret sharing and introduce some notation, before we present our protocol for sparse inner product in Section 3.3.

3.2 Notation and Terminology

Additive Secret Sharing. As mentioned above, our protocols' outputs are secret-shared, namely they produce matrices $\mathbf{C_1}, \mathbf{C_2}$ over $\mathbb{Z}_q$ that are chosen uniformly at random under the constraint that $\mathbf{C_1} + \mathbf{C_2} = \mathbf{AB}$. Since Party i obtains only $\mathbf{C_i}$, and each share individually cannot be distinguished from a random matrix, this does not leak any information about $\mathbf{C} = \mathbf{AB}$. This technique is generally used to hide the values of intermediate results, and it is the basis of various MPC protocols [9, 20, 47]. We will refer to values shared in this way as *additively shared* and call the individual parts *additive shares*.

Index Sets. For any sparse vector $\mathbf{v}$, we write $\mathcal{I}_\mathbf{v}$ to denote the set of indexes where $\mathbf{v}$ is non-zero, and call its cardinality $l_\mathbf{v} := |\mathcal{I}_\mathbf{v}|$. We further denote the k-th element of this set (using canonical ordering) by $(\mathcal{I}_\mathbf{v})_k$.

For a matrix $\mathbf{M}$, we write $\mathsf{Col}_i(\mathbf{M})$ to denote the i-th column vector of $\mathbf{M}$, and $\mathsf{Row}_j(\mathbf{M})$ for the j-th row. In analogy to vectors, we write $\mathcal{I}^{\mathsf{Col}}_\mathbf{M} := \{i \mid \mathsf{Col}_i(\mathbf{M}) \neq \mathbf{0}\}$ for the set of indexes corresponding to non-zero columns of $\mathbf{M}$, likewise for rows.

3.3 Secure Sparse Inner Products

We now present our protocol for computing a sparse inner product. This serves as a stepping stone towards our *batched* variant, which we introduce in Section 3.5 and which is the one that we use in practice.

Here, we consider two sparse vectors $\mathbf{a}, \mathbf{b}$, where $\mathbf{a}$ is owned by Party 1 and $\mathbf{b}$ by Party 2. The goal of our protocol is to compute additive shares $\mathbf{c}_1, \mathbf{c}_2$, such that $\mathbf{c}_1 + \mathbf{c}_2 = \langle \mathbf{a}, \mathbf{b} \rangle$. To develop intuition, let us first discuss an insecure solution where the parties reveal to each other their respective non-zero indexes $\mathcal{I}_\mathbf{a}$ and $\mathcal{I}_\mathbf{b}$. Then each party can simply compute locally the set of common non-zero indexes and construct vectors $\tilde{\mathbf{a}}, \tilde{\mathbf{b}}$ of shorter length $|\mathcal{I}_\mathbf{a} \cap \mathcal{I}_\mathbf{b}|$ containing only the values of common indexes in some canonical order, such that

$\langle \tilde{\mathbf{a}}, \tilde{\mathbf{b}} \rangle = \langle \mathbf{a}, \mathbf{b} \rangle$. Then the inner product of $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ can then be computed using a standard non-sparse MPC protocol [28, 47].

While this insecure approach would greatly increase efficiency by exploiting the sparsity of $\mathbf{a}$ and $\mathbf{b}$, it is also clear that it leaks too much, as $\mathcal{I}_{\mathbf{a}}$ and $\mathcal{I}_{\mathbf{b}}$ are private. Instead, the solution we propose avoids this leakage while retaining the efficiency of this insecure solution by assuming upper bounds on $|\mathcal{I}_{\mathbf{a}}|$ and $|\mathcal{I}_{\mathbf{b}}|$ are public.

For clarity, we now describe our solution assuming the availability of a trusted third party. The parties first create vectors $\hat{\mathbf{a}}, \hat{\mathbf{b}}$ containing the values at indexes in $\mathcal{I}_{\mathbf{a}}$ and $\mathcal{I}_{\mathbf{b}}$, respectively, padded with zeroes to length $l_{\mathbf{a}} + l_{\mathbf{b}}$. Then, the third party receives $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$ from the parties, and returns permutations π_1 and π_2 to party 1 and party 2, respectively, such that $\langle \pi_1(\hat{\mathbf{a}}), \pi_2(\hat{\mathbf{b}}) \rangle = \langle \mathbf{a}, \mathbf{b} \rangle$. Note that padding with zeroes is crucial so that such permutations exist, as intuitively all they have to do is make indexes in $\mathcal{I}_{\mathbf{a}} \cap \mathcal{I}_{\mathbf{b}}$ end up in the same position in both permuted vectors, while all others get matched to a zero. Now, if the permutations π_i are such that they do not reveal anything about $\hat{\mathbf{a}}$ or $\hat{\mathbf{b}}$ to the parties, this protocol would be secure.

While we relied on a trusted third party for explanatory purposes, such a third party is not available in practice. Instead, we will now present a 2PC-sub-protocol that replaces such a party, i.e., it generates *correlated* permutations π_1, π_2 with the required property. We call this functionality $\mathcal{F}^{\text{Perm}}$ and describe it formally in the next section.

3.4 Secure Correlated Permutations

Note that we only require the output to each party alone to be a uniformly random permutation. Thus, $\mathcal{F}^{\text{Perm}}$ can generate one of the permutations randomly, and derive the other from it. A detailed description of $\mathcal{F}^{\text{Perm}}$ is given in Functionality 1.

By the construction of ρ in Step 2, it is clear that the condition from Figure 1, that matching indexes get mapped to the same position, holds for π_1 and π_2. We now prove that the outputs to both parties are indeed random permutations.

Theorem 1. *For any input sets $\mathcal{I}_1$ and $\mathcal{I}_2$ of size l_1 and l_2, and any party $i \in \{1, 2\}$, $\pi_i = \mathcal{F}_i^{\text{Perm}}(\mathcal{I}_1, \mathcal{I}_2)$ is a uniformly random permutation of $\{1, \ldots, l_1 + l_2\}$.*

Proof. If $i = 1$, then by the definition in Step 2 in Functionality 1, ρ is constructed by selecting l_1 different mappings from a uniformly random permutation. Clearly, each of the $(l_1 + l_2)! \,/\, l_2!$ possible functions is selected with equal probability. The extension of ρ in Step 3 reduces to selecting a uniformly random permutation of l_2 elements. Thus, our functionality produces $(l_1 + l_2)! \,/\, l_2! \cdot l_2! = (l_1 + l_2)!$ different permutations, each with equal probability. Together with the observation that there are exactly $(l_1 + l_2)!$ *possible* permutations of $[l_1 + l_2]$, the claim follows. If $i = 2$, the claim follows immediately from Step 1 and $\pi_2 = \pi$. □

Functionality 1: Correlated permutations ($\mathcal{F}^{\text{Perm}}$)

Parties: 1, 2.

Input: Party 1: $\mathcal{I}_1$, Party 2: $\mathcal{I}_2$, public parameters $l_1 = |\mathcal{I}_1|$, $l_2 = |\mathcal{I}_2|$

Output: Permutations π_1 and π_2

1: Choose a permutation π of $\{1, \ldots, l_1 + l_2\}$ uniformly at random.

2: Compute the function

$$\rho : \{1, \ldots, l_1\} \to \{1, \ldots, l_1 + l_2\},$$
$$\rho(i) = \begin{cases} \pi(j) & \text{if } \big(\mathcal{I}_1\big)_i = \big(\mathcal{I}_2\big)_j, \\ \pi(l_2 + i) & \text{if no such } j \text{ exists.} \end{cases}$$

3: Extend ρ to a random permutation π_1 of $\{1, \ldots, l_1 + l_2\}$ by mapping all elements $i > l_1$ to uniformly random unmapped elements of its codomain.

4: Output π_1 to Party 1 and $\pi_2 = \pi$ to Party 2.

We use Yao's Garbled Circuit protocol [59] to implement $\mathcal{F}^{\text{Perm}}$. Our circuit design for this functionality is inspired by the Private Set Intersection (PSI) protocol of Huang et al. [35], also known as the Sort-Compare-Shuffle approach to PSI. Essentially, our circuit computes the set $\mathcal{I}_1 \cap \mathcal{I}_2$ in $O((l_1 + l_2) \log^2(l_1 + l_2))$ – using Batcher's sorting network [8] – as a Boolean array of length $l_1 + l_2$, and constructs ρ and π_1 from it using a permutation network [56] of size $O((l_1 + l_2) \log(l_1 + l_2))$. Thus, our protocol for the $\mathcal{F}^{\text{Perm}}$ functionality runs in $O((l_1 + l_2) \log^2(l_1 + l_2))$, and exploits efficient implementations of sorting/permutation networks.

Since we operate in the semi-honest model, we can further employ the following optimizations:

1. Instead of choosing π inside the Garbled Circuit, we let Party 2 choose it locally and use it as an input. Note that this is trivially simulatable since $\pi = \pi_2$ is Party 2's output.

2. Similarly, we reveal ρ to Party 1 and let it perform Step 3 locally. Again, simulating ρ it trivial by restricting π_1 to $\{1, \ldots, l_1\}$.

Together with the security of Yao's protocol in the semi-honest model [41], security of our implementation of $\mathcal{F}^{\text{Perm}}$ follows.

An important observation is that the cost of $\mathcal{F}^{\text{Perm}}$ is independent of the range of the values in the vectors u, v in the overall inner product protocol. In the next section, we show how this, and previous observations, extend to matrix multiplications for batched inner products, and provide formal proofs of correctness and security for our two-party protocols.

3.5 From Inner Products to Sparse Matrix Multiplication

We will now consider computing additively shared matrix products $\mathbf{C}_1 + \mathbf{C}_2 = \mathbf{AB}$, where the matrices $\mathbf{A}$ and $\mathbf{B}$ are owned by Party 1 and 2 respectively. We assume these matrices exhibit the sparsity patterns corresponding to batched sparse inner products, i.e., $\mathbf{A}$ has many zero columns, and $\mathbf{B}$ many zero rows. This is a common sparsity pattern in machine learning computations, as, for example, TensorFlow has a dedicated matrix representation for this case, called *IndexedSlices* [5].

A naive solution is to run the protocol from the previous section $|\mathbf{AB}|$ times. In the rest of this section we focus on improving this by leveraging the sparsity patterns in the matrices to avoid the quadratic cost of the naive solution. The advantage of our solution is not only in asymptotic cost, but is also confirmed experimentally in Section 6, using real-world data.

The entire sparse matrix multiplication protocol is depicted in Figure 1, and goes as follows. Party 1 locally computes $\mathcal{I}^{\mathsf{Col}}_{\mathbf{A}} := \{i \mid \mathsf{Col}_i(\mathbf{A}) \neq \mathbf{0}\}$, and Party 2 computes $\mathcal{I}^{\mathsf{Row}}_{\mathbf{B}} := \{j \mid \mathsf{Row}_j(\mathbf{B}) \neq \mathbf{0}\}$. These index sets are then used as inputs to the functionality for generating correlated permutations, $\mathcal{F}^{\text{Perm}}$, which generates two correlated permutations π_1, π_2 that map elements of $\mathcal{I}^{\mathsf{Col}}_{\mathbf{A}} \cap \mathcal{I}^{\mathsf{Row}}_{\mathbf{B}}$ to the same indexes. Note that, up to this point, the secure computation is independent of the inner (resp. outer) dimension of $\mathbf{A}$ (resp. $\mathbf{B}$). This has an important effect on efficiency, and follows from the remark above about $\mathcal{F}^{\text{Perm}}$ in the inner product protocol being independent of the domain size of the values in the vectors. In fact, intuitively we can think of multiplying $\mathbf{A}$ and $\mathbf{B}$ as computing sparse inner product where values are vectors, instead of scalars. Next, $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ are

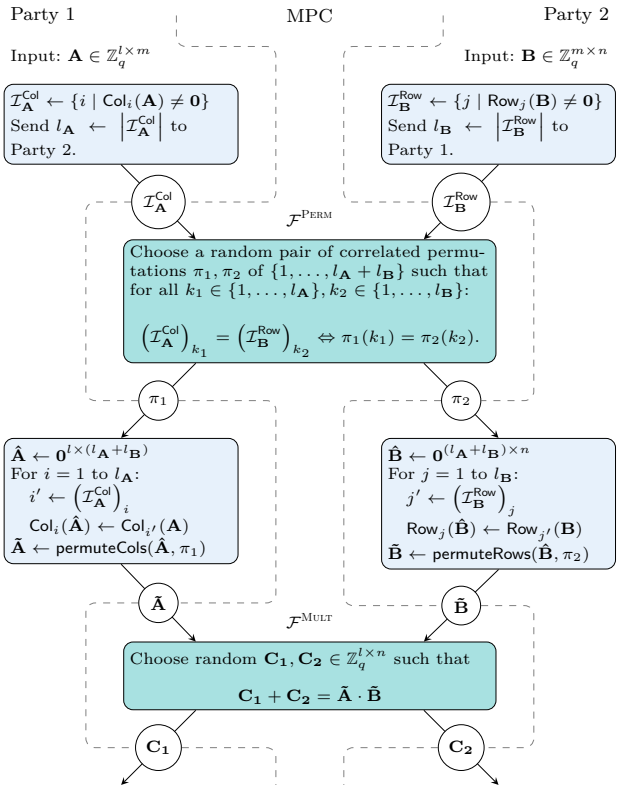


Fig. 1. Secure sparse matrix multiplication. For details on the implementations of $\mathcal{F}^{\text{Mult}}$ and $\mathcal{F}^{\text{Perm}}$, see Appendix B.2 and Section 3.4, respectively.

computed by first padding non-zero columns/rows with zeroes, and then applying π_1 and π_2, just like $\tilde{\mathbf{a}}$ and $\tilde{\mathbf{b}}$ in Section 3.3. The result is again obtained by using a standard MPC protocol for secure matrix multiplication with $\tilde{\mathbf{A}}$ and $\tilde{\mathbf{B}}$ as inputs.

Theorem 2 (Correctness). *For any $\mathbf{A} \in \mathbb{Z}_q^{l \times m}$, $\mathbf{B} \in \mathbb{Z}_q^{m \times n}$, let $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$ be constructed according to the protocol described in Figure 1. Then $\mathbf{AB} = \tilde{\mathbf{A}}\tilde{\mathbf{B}}$.*

Theorem 3 (Security). *Given public sparsity values $l_{\mathbf{A}}, l_{\mathbf{B}}$ and implementations of $\mathcal{F}^{\text{Mult}}$ and $\mathcal{F}^{\text{Perm}}$ that are secure against semi-honest adversaries, the protocol in Figure 1 implements $\mathcal{F}^{\text{Mult}}$ with security against semi-honest adversaries.*

We prove Theorems 2 and 3 in Appendices C and D.

In order to compute similarities between documents using the protocol described in this section, each data holder must know the feature representation of their documents. This becomes a challenge when trying to use data-dependent feature representations such as TF-IDF. We tackle the problem of private feature extraction for this particular case in the next section.

4 Private Feature Extraction

Feature extraction is a fundamental pre-processing step in any data processing pipeline, including those used in machine learning, data mining, information retrieval, computer vision and natural language processing. The goal of feature extraction is to convert raw data (e.g. text documents, RGB images, etc.) into a vectorial format suitable for downstream applications. The same feature representation is often useful for many different applications, and the most effective feature representations are typically tailored to the dataset at hand. This poses a challenge in scenarios where datasets containing sensitive records are distributed among multiple parties, as obtaining an adequate feature representation requires a privacy-preserving distributed computation, and the resulting feature representation might leak information about the dataset on which it was computed.

In this section we address this challenge for the well-known *term frequency–inverse document frequency* (TF-IDF) feature representation for text documents (cf. Section 2.2). Our contribution is a two-party protocol for securely computing IDF coefficients on a distributed document database and releasing them under differential privacy. As in the previous section, we work in the standard simulation-based paradigm of security (see Section 2.1 and Appendix A) and our computation is secure against semi-honest adversaries. At the same time, its output preserves differential privacy with respect to changing one document in the distributed dataset. Using the output of this protocol, multiple parties can *locally* compute a TF-IDF representation of their documents. This representations will be compatible across parties, and can then be used in any subsequent privacy-preserving computation. In Section 5.1, we formalize this as secure computation with differentially-private leakage. Before diving into the details of our protocol, we first review the formal privacy definition in the distributed setting we consider.

4.1 Multi-Party Computational Differential Privacy

Differential privacy (DP) is a technique for privacy-preserving disclosure [23–25]. It prevents a potential adversary observing the output of a computation from recovering information about individual input data points, i.e., individual document in our case. This is made formal by saying that two datasets Z and Z' are neighbors if they only differ in one data point; this relation is denoted by $Z \simeq Z'$. We say that a randomized algorithm $\mathcal{A} : \mathcal{Z} \to \mathcal{W}$ is (ε, δ)-DP if for any indicator function $\chi : \mathcal{W} \to \{0, 1\}$ we have

$$\forall Z \simeq Z' : \quad \mathbb{E}\big[\chi(\mathcal{A}(Z))\big] \leq e^{\varepsilon}\mathbb{E}\big[\chi(\mathcal{A}(Z'))\big] + \delta \ .$$

When $\delta = 0$ we also say that $\mathcal{A}$ is ε-DP. This definition models the setting where a curator owns the input Z, executes the computation $\mathcal{A}$, and discloses the output $\mathcal{A}(Z)$.

For the purpose of providing DP in a multi-party setting one needs to modify the above definition to account for the fact that Z is distributed among several parties. Additionally, implementing DP inside an MPC protocol requires a further modification to account for the information that could be obtained by a coalition of adversarial parties involved in the computation who try to break the cryptography used in the MPC protocol. This leads to the definition of *multi-party computationally differential privacy* [11, 24, 46].

Suppose the input dataset is distributed among n parties $Z = (Z_1, \ldots, Z_n)$ and write $Z \simeq_i Z'$ if $Z_i \simeq Z'_i$ and $Z_j = Z'_j$ for $i \neq j$. Suppose $\mathcal{A} : \mathcal{Z}^n \to \mathcal{W}$ is an n-party protocol and let $\mathsf{view}_{-i}(\mathcal{A}(Z))$ denote the information observed by all parties except the ith one during the execution of $\mathcal{A}(Z)$. Then we say that $\mathcal{A}$ is (ε, δ)-MPC-DP if for all i and all $Z \simeq_i Z'$ we have

$$\mathbb{E}\big[\chi\big(\mathsf{view}_{-i}(\mathcal{A}(Z))\big)\big] \leq e^{\varepsilon}\mathbb{E}\big[\chi\big(\mathsf{view}_{-i}(\mathcal{A}(Z'))\big)\big] + \delta$$

for any $\{0, 1\}$-valued polynomial time algorithm χ. In this work we focus on MPC based on computational security, as opposed to information-theoretic MPC, and hence resort to the variant of MPC-DP studied in [46]. The following key result states that implementing a DP algorithm inside an MPC protocol yields an MPC-DP protocol.

Theorem 4 (informal)**.** *If $\mathcal{A}$ is $(\varepsilon, \delta_{\mathrm{DP}})$-DP with respect to $Z \simeq Z'$, then an MPC implementation of $\mathcal{A}$ where is Z distributed among n parties is $(\varepsilon, \delta_{\mathrm{DP}} + \delta_{\mathrm{MPC}})$-MPC-DP, where δ_{MPC} is a negligible function of $|Z|$ obtained from standard cryptographic assumptions.*

4.2 Differentially Private IDF Computation

One standard approach for making the output of a computation private is the Laplace mechanism [25]. It works by adding noise drawn from the Laplace distribution

to the output of a deterministic function, where the amount of noise is proportional to the l_1-sensitivity of that function and the inverse of the privacy parameter ε. This poses a challenge when it comes to computing the vector of IDF coefficients $\phi_{\text{idf}}(\cdot, Z) \in \mathbb{R}^{\mathcal{V}}$: since we want to provide privacy to whole documents and there is no a-priori knowledge about which words may occur in an arbitrary document, replacing a document might result in a change in several entries of the vector of IDF coefficients. Thus, for a fixed ε, the amount of noise needed for each IDF value using the Laplace mechanism would need to be proportional to the size of the vocabulary, which can be very large.

To avoid this problem, instead of releasing every single IDF value using the Laplace mechanism, we design our mechanism to only release them where they matter most, and resort to outputting a default value everywhere else. The key observation to justify this approach is that, in a corpus of documents Z, the distribution of the values of $|Z|_v$ for all $v \in \mathcal{V}$ typically follows a power-law distribution [53]. This means there are few very frequent words and lots of infrequent words. The blue bars in Figure 2 exemplify such a typical scenario. The red curve in Figure 2 shows how the IDF values quickly converge to the maximum as the document frequency decreases. We can therefore obtain a good approximation across IDFs over all words by releasing differentially private IDFs for the L most frequent words, and assume a default value c_0 for all other words. In this way the noise added to the IDFs of the most frequent words will only be proportional to L, as opposed to $\mathcal{V}$.

What remains is to make sure the selection of the L most frequent words is also private. To achieve this, we do not release the L most frequent words exactly, but instead release a selection of words that with high probability has a large overlap with the top L. This sampling is done using the exponential mechanism [45], which is a standard construction for differentially private top-L selection [7].

The pseudo-code of our mechanism is given as Functionality 2. It takes as input the absolute frequencies of each word in each party's dataset Z_i. It then proceeds to aggregate these into frequencies across the whole dataset Z, yielding $c_v = |Z|_v$ for each $v \in \mathcal{V}$. The counts are used in a private top-L selection step to find L words with the largest frequencies. The mechanism then releases privatized counts $\tilde{c}_v$ for each of the selected words using the Laplace mechanism. For unselected words the mechanism outputs a default public value $\tilde{c}_v = c_0$ which is independent of the true word count.

Functionality 2: Differentially Private IDFs ($\mathcal{F}^{\text{DP-IDF}}$)

Public Inputs: n, $\mathcal{V}$, c_0, L, ε
Private Inputs: Counts $\{|Z_i|_v\}_{v\in\mathcal{V}}$ for $i \in [n]$
Output: Privatized values $\{\tilde{c}_v\}_{v\in\mathcal{V}}$

foreach $v \in \mathcal{V}$ **do**
  Compute $c_v = \sum_{i=1}^{n} |Z_i|_v$.
end
for $\ell = 1, \ldots, L$ **do**
  Sample $v \in \mathcal{V}$ with probability $\propto \exp\left(\frac{\varepsilon c_v}{2L}\right)$.
  Sample η from $\mathsf{Lap}\left(\frac{2L}{\varepsilon}\right)$.
  Release $\tilde{c}_v = c_v + \eta$.
  Remove v from $\mathcal{V}$.
end
For each $v \in \mathcal{V}$ release $\tilde{c}_v = c_0$.

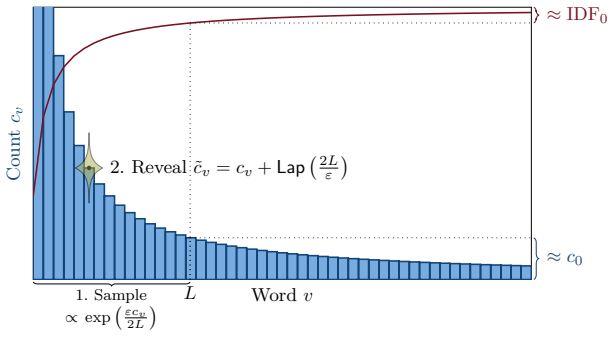


Fig. 2. Graphical representation of our differentially private IDF computation functionality $\mathcal{F}^{\text{DP-IDF}}$. Term counts following a power law distribution are depicted in form of a histogram, and the corresponding IDF values are drawn as a solid line. It can be seen that as c_v decreases, the IDF values quickly converge towards $\text{IDF}_0 = \log(|Z| + 1) + 1$. Steps 1 and 2 are repeated L times in a loop (cf. Functionality 2).

Theorem 5. $\mathcal{F}^{\text{DP-IDF}}$ *(Functionality 2) is* ε*-DP.*

We prove Theorem 5 in Appendix E. Note that using the advanced composition theorem [23, Theorem 3.20] one can also show that $\mathcal{F}^{\text{DP-IDF}}$ is $(O(\varepsilon\sqrt{\log(1/\delta)/L}), \delta)$-DP for any $\delta > 0$. However, in this paper we will stick to the ε-DP guarantee given above for the sake of simplicity.

4.3 Implementing $\mathcal{F}^{\text{DP-IDF}}$

By Theorem 4, we can obtain an MPC-DP protocol from Functionality 2. We propose a circuit-based implementation of $\mathcal{F}^{\text{DP-IDF}}$ which can be ran using any generic circuit-based MPC framework. While our proto-

col in theory supports any number of parties, we limit our implementation (Section 6) and the description in the remainder of this section to two parties.

The main challenge lies in securely generating noise for the Laplace mechanism, and sampling words from the exponential mechanism. Next we describe how to implement both steps.

4.3.1 Laplace Mechanism

For the Laplace Mechanism, we use a standard inversion sampling approach. Given a uniform real number $x \in (0, 1)$, a Laplace sample with mean 0 and scale b can be computed using

$$\mathsf{Lap}(b) = \begin{cases} b\log(2x) & \text{if } x \leq 1/2, \\ -b\log(2-2x) & \text{otherwise.} \end{cases}$$

The required uniform sample can be cheaply computed by adding up two such samples computed locally by each party inside the MPC, and subtracting 1 if the result is larger than 1. Note that this ensures that knowledge of one of the summands reveals nothing about the resulting uniform sample. For all the other operations, including the logarithm, there are circuits that give exact results up to the precision of floating-point numbers. Note that in general, floating-point computation in circuit-based MPC can be quite expensive. However, because we only need to sample L times and $L \ll |\mathcal{V}|$, this only has a minor impact on the running time of our protocol.

4.3.2 Exponential Mechanism

Implementing the exponential mechanism is more challenging since we need to sample words *without replacement*. However, we will see that this can be done in the same asymptotic time as sampling with replacement, using a Bernoulli tree that gets refreshed after each sample. First, we compute the sampling probability $p_v = \exp(\varepsilon_0 c_v)/\sum_v \exp(\varepsilon_0 c_v)$ of each word $v \in \mathcal{V}$ once and write them to the leaves of a balanced binary tree. Next, we traverse this tree bottom-up, labeling each inner node with the sum of the labels of its children. Now, to sample a word $v \in \mathcal{V}$, we traverse the tree starting from the root. At each node, we perform a Bernoulli trial and descend into each sub-tree with probability proportional to the label at the corresponding child (i.e., the sub-tree's root). Once we arrive at a leaf node, we return the word associated with it. Since the binary tree is balanced, it has depth $O(\log|\mathcal{V}|)$, so one sample can be computed using $O(\log|\mathcal{V}|)$ coin tosses and array accesses. The advantage of this approach is that we can refresh our Bernoulli tree using also just $O(\log|\mathcal{V}|)$ steps: after returning a leaf v, we subtract p_v from all nodes on the path from the root and set p_v to 0. This ensures that each leaf is reached at most once. Note that the updated labels do not need to be normalized as we take that into account when descending the tree and compute the probabilities accordingly. An example of our sampling method and the refresh step is shown in Figure 3.

The final piece is the implementation of oblivious reads and writes in the nodes of the Bernoulli tree as a circuit. This is needed in order to hide the order in which nodes are accessed, which could leak their associated probabilities and thus counts of individual words. Here, the asymptotically best choice is a generic ORAM construction, which has poly-logarithmic overhead for each access [57]. However, in terms of concrete efficiency, the optimal choice depends on the level of the tree at which we are reading or writing. In particular for levels with few nodes, asymptotically sub-optimal solutions such as linear scans still outperform generic ORAMs. In practice (cf. Section 6), we switch between linear scans, square-root ORAM [61], and FLORAM [22], depending on the level of the tree we are accessing. The cutoff points between those are informed by the measurements provided in [22].

4.4 Utility Analysis

We now give a utility result about the mechanism in Functionality 2 for computing differentially private IDFs on a dataset of documents. While we motivated our mechanism $\mathcal{F}^{\text{DP-IDF}}$ using the observations that the distributions of words in a document corpus typically follows a power law, we cannot assume this holds for any possible dataset. Thus, in our utility analysis, we make the much weaker assumption that the documents in Z are sampled i.i.d. from some unknown distribution. Here we present only an informal statement of our result. A more concrete statement together with the relevant proofs are provided in Appendix F. The result bounds the relative error between the true vectors of IDFs ϕ_{idf} and the privatized vector $\tilde{\phi}_{\text{idf}}$ computed using the counts released by Functionality 2.

Theorem 6. *For any large enough $m = |Z|$ there exists $c_0 = \Theta(\sqrt{m})$ such that with high probability*

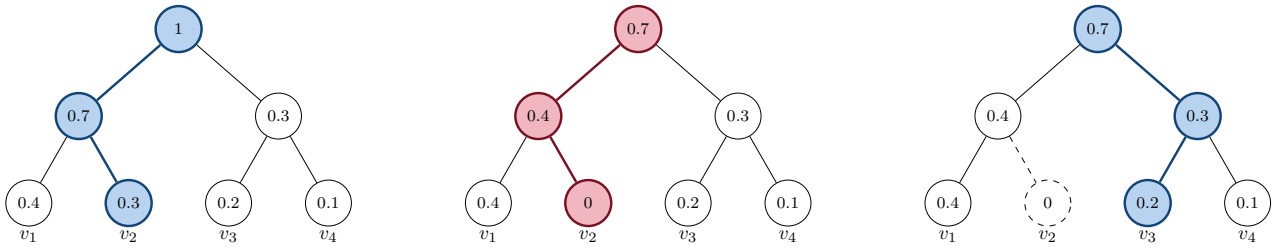


Fig. 3. Example run of our MPC protocol for the exponential mechanism. (Left) The left node gets selected on the first level (probability $0.7/1$), and the right node on the second level (probability $0.3/0.7 \approx 0.43$). The sampled word is v_2. (Middle) Refresh step: $p_{v_2} = 0.3$ is subtracted from all nodes on the path to the root, then p_{v_2} is set to zero. (Right) A second sample is drawn with updated probabilities. On the first level, the right node is selected (probability $0.3/0.7$), on the second level it is the left node (probability $0.2/0.3 \approx 0.67$). The result is v_3.

$$\frac{\|\phi_{\text{idf}} - \tilde{\phi}_{\text{idf}}\|_1}{\|\phi_{\text{idf}}\|_1} \leq \tilde{O}\left(\frac{L^2}{\varepsilon m|\mathcal{V}|} + \left(1 - \frac{L}{|\mathcal{V}|}\right)\log\left(m - \frac{L}{\varepsilon}\right)\right) \quad .$$

Note how this result highlights an important trade-off in the choice of L since the first term grows with L while the second term becomes smaller for larger L. Additionally, the first terms decreases quickly with m, while the second term increases slowly with m. In our experiments we did not observe this growth of the error with m, which suggests that, for well-behaved datasets where the power-law assumption holds, the $O(\log(m))$ term could be removed. We leave this as an open problem for future work.

5 Secure Document Classification

In Section 3, we have introduced a protocol that can exploit sparse feature representations to compute similarities, and that is particularly efficient when computing many similarities at once. In Section 4, we have shown that such sparse features can be computed even if they depend on a whole database distributed among multiple parties, by revealing differentially private IDF coefficients. While each of these protocols is of independent interest, we will now show how they can be securely composed to form higher-level functionalities. We focus on k-Nearest Neighbors classification for the remainder of the paper. However, we stress that our protocols can also be used to implement other functionalities, for example document rankings.

We assume a two-server setting, where each of the servers holds a database of labeled text documents. The labels are the target classes of the classification task. A third party, the client, holds a single unlabeled document x she wants to classify. A k-Nearest Neighbors classification algorithm can be used to achieve this. In general, it consists of the following steps: (1) for each document j in the full database, compute the similarity score $s_j(x)$ between x and j; (2) compute the labels $\hat{y}_1, \ldots, \hat{y}_k$ corresponding to the documents with the top k scores; and, (3) return the majority vote $\hat{y} = \mathsf{majority}(\hat{y}_1, \ldots, \hat{y}_k)$. This process is formally described in Functionality 3.

5.1 Security with Differentially Private Leakage

We define security of our combined protocol in a similar fashion as previous work [31, 44]. That is, in addition to the output of the ideal functionality, we allow for a randomized leakage $\mathcal{L}$ that depends on the input data. However, said leakage must be differentially private with respect to individual records. Unlike [31, 44], we additionally allow that the output of our functionality may depend on $\mathcal{L}$. This captures the fact that using the differentially private IDFs from Section 4, we do not compute the exact result, but instead an approximation that depends on the privatized IDFs. Note that this setting is also suitable for scenarios where one wants to transfer differentially private hyper-parameter tuning [18, 27] to multi-party learning settings.

Let $\mathcal{F}$ be an n-party functionality with inputs $\bar{x} \in (\{0,1\}^*)^n$, additional input $l \in \{0,1\}^*$, and outputs $\big(\mathcal{F}_1(\bar{x}, l), \ldots, \mathcal{F}_n(\bar{x}, l)\big)$. Let $\mathcal{L}$ denote a randomized leakage function with domain $(\{0,1\}^*)^n$. We write $\tilde{\mathcal{F}}(\bar{x}, \mathcal{L}) = \big(\mathcal{F}(\bar{x}, \mathcal{L}(\bar{x})), \mathcal{L}(\bar{x})\big)$ to denote the function $\mathcal{F}$ *with leakage* $\mathcal{L}$, and for any $I \subseteq [n]$, we write $\tilde{\mathcal{F}}_I(\bar{x}, \mathcal{L}) := \big(\mathcal{F}_I(\bar{x}, \mathcal{L}(\bar{x})), \mathcal{L}(\bar{x})\big)$, where $\mathcal{F}_I$ is defined as in Appendix A. We say a protocol Π *securely computes* $\mathcal{F}$ *with* (ε, δ)*-differentially private leakage* $\mathcal{L}$ if $\mathcal{L}$ is (ε, δ)-differentially private with respect to individual records

Functionality 3: k-NN Classification

Public Inputs: n, $\mathcal{V}$, k, $\{\tilde{c}_v\}_{v\in\mathcal{V}}$
Server i **Inputs:** Document database Z_i, and labels l_x for each $x \in Z_i$
Client Input: Query document q

foreach $x \in \bigcup_{i\in[n]} Z_i \cup \{d\}$ **do**
  Compute $\phi_{\text{tf}}(x, v)$, the number of occurrences of v in x.
end
foreach $v \in \mathcal{V}$ **do**
  Compute IDF coefficient $\phi_{\text{idf}}(v, Z) = \log((|Z| + 1)/(\tilde{c}_v + 1)) + 1$.
end
Compute query vector $\psi(q) = \big(\phi_{\text{tf}}(q, v)\phi_{\text{idf}}(v)\big)_{v\in\mathcal{V}}$.
foreach $x \in \bigcup_{i\in[n]} Z_i$ **do**
  Compute $\psi(x) = \big(\phi_{\text{tf}}(x, v)\phi_{\text{idf}}(v)\big)_{v\in\mathcal{V}}$, and similarity score $s_x = \mathsf{sim}_{\cos}(\psi(x), \psi(q))$.
end
Compute l_q as the label most common among the k documents x with the highest scores s_x.
Reveal l_q to the client.

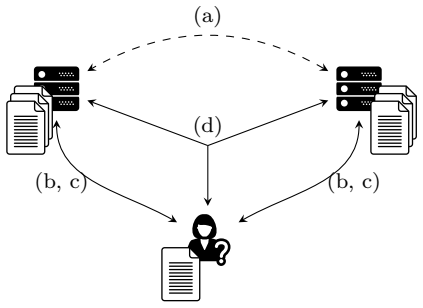


Fig. 4. Diagram of our example application for k-Nearest Neighbors classification of text documents. Each of the two servers holds a collection of labeled text documents, while the client holds an unlabeled document she wants to classify. The protocol has four stages. (a) The servers precompute and release private IDF values for their joint database (Section 4). Note that this is a one-time setup step. (b) With each of the servers, the client performs a secure batched similarity computation via a secure sparse matrix-vector multiplication (Section 3), where the server inputs a sparse matrix with rows corresponding to documents, while the client inputs a single sparse document vector. (c) Using the similarity shares from the previous step, the client computes with each server shares of the labels and similarities of the k most similar documents to her query. (d) The shares from step (c) are used as inputs to a three-party computation that selects the top k documents overall. The label for the query document is computed by a majority vote among those.

in each x_i, and there exists a PPT simulator S such that for any $I \subseteq [n]$:

$$\left\{\left(\mathcal{S}\big(I, x_I, \tilde{\mathcal{F}}_I(\bar{x}, \mathcal{L})\big), \tilde{\mathcal{F}}(\bar{x}, \mathcal{L})\right)\right\}_{\bar{x}\in(\{0,1\}^*)^n} \overset{c}{\equiv} \left\{\left(\mathsf{view}_I^{\Pi}(\bar{x}), \mathsf{output}^{\Pi}(\bar{x})\right)\right\}_{\bar{x}\in(\{0,1\}^*)^n} \quad (1)$$

Note that the additional argument l to $\mathcal{F}$ captures the fact that the output may depend on the leakage. If $\mathcal{F}$ does not depend on l, i.e., $\mathcal{F}(\bar{x}, l) = \mathcal{F}(\bar{x}, \perp)$ for all l, and $n = 2$, we get back the definition from [44]. Also observe that in order to prove security with leakage of a protocol Π, it is enough to define $\tilde{\mathcal{F}}$ and $\mathcal{L}$, show that Π securely computes $\tilde{\mathcal{F}}$ according to the definition in Appendix A, and show that $\mathcal{L}$ is (ε, δ)-differentially private.

In the next section, we describe a 2-server implementation of the k-NN functionality $\mathcal{F}^{k\text{-NN}}$ (Functionality 3) that uses the protocols from Sections 3 and 4 and show that it satisfies this notion of secure computation with differentially private leakage.

5.2 Our Protocol

Figure 4 shows the protocol that we use to implement Functionality 3 in our distributed setting with two servers. There are four phases, two of which correspond to the preceding sections: In a precomputation phase (a), the two servers perform a two-party computation that implements $\mathcal{F}^{\text{DP-IDF}}$ from Section 4.2. Then, the client and each of the servers run the secure matrix multiplication protocol from Section 3 to obtain shares of the similarities of their respective documents. What remains is to select the top k labels and perform a majority vote in a secure way. While so far we were able to split up the entire protocol into two-party components, we need a generic three-party computation for the top-k selection. However, we can ensure its running time only depends on k and not on the number of documents: observe that each document in the top k overall must also be among the top k of the server that owns this particular documents. Therefore, we can compute shares of the top k on each server using cheap two-party computation (c), and then only need $2k$ inputs to the three-party phase.

Theorem 7. *The protocol* $\Pi^{k\text{-NN}}$ *described in Figure 4 securely computes* $\mathcal{F}^{k\text{-NN}}$ *with* ε*-differentially private leakage.*

Proof. Let $\mathcal{L} = \mathcal{F}^{\text{DP-IDF}}$, and let $\tilde{\mathcal{F}}^{k\text{-NN}}$ be defined as the functionality running $\mathcal{L}$ and then using the output

$\{\tilde{c}_v\}_{v\in\mathcal{V}}$ as input to $\mathcal{F}^{k\text{-NN}}$. Note that $\tilde{\mathcal{F}}^{k\text{-NN}}$ has the structure required by eq. (1). By the definition from Section 5.1, it suffices to show that (i) $\mathcal{L}$ is ε-differentially private, and (ii) $\Pi^{k\text{-NN}}$ computes $\tilde{\mathcal{F}}^{k\text{-NN}}$ with security against semi-honest adversaries. (i) follows directly from Theorem 5. As for (ii), observe that for any subset of the parties, intermediate outputs are either secret-shared (and can therefore be simulated by uniformly random values), or part of the final output. Thus, security of $\Pi^{k\text{-NN}}$ reduces to the security of the individual phases (Theorem 3 for Step (b), [41] for (a, c) and [20] for (d)) and modular composition in the semi-honest model (see for example Canetti [16]). □

Note that our security definition does not explicitly capture how $\mathcal{L}$ is implemented, and therefore does not require the notion of MPC-DP (Section 4.1). However, Theorem 4 states that any ε-DP functionality implemented as an MPC protocol yields a (ε, δ_{MPC})-MPC-DP protocol. In our concrete case, this means that protocol $\Pi^{k\text{-NN}}$ securely computes $\mathcal{F}^{k\text{-NN}}$ with ε-DP leakage $\mathcal{F}^{\text{Perm}}$ (as in the definition from Section 5.1), while at the same time the output of the sub-protocol implementing $\mathcal{F}^{\text{Perm}}$ satisfies (ε, δ_{MPC})-MPC-DP.

A distinctive feature of our k-NN application is the fact that our security definition allows for a *dishonest majority*, and thus the client's query remains secure even if both servers collude. This is in stark contrast to previous work, as we discuss in Section 7. We also note that in principle, our protocol can be extended to more than two servers, by implementing Protocol 2 (Step (a) in Figure 4) using a generic semi-honest MPC protocol, as for example given by Ben-Efraim et al. [12].

6 Experiments

We will now experimentally evaluate our protocols. We do so from two perspectives. First, we evaluate the running time for both of our main protocols, secure matrix multiplication (Section 3) and private IDF precomputation (Section 4), in a simple two-party setting. Then, we explore how our protocols scale when applied to a real-world classification task. To that end, we implement the k-nearest neighbors classification protocol from Section 5 and evaluate it on real data. That is, measure the running time taken for similarity computation and top-k selection, taking into account characteristics of real-world data; and we measure the effect our privatized IDF values have on the classification accuracy.

Implementation. We implement our protocols in C++, using Obliv-C [60] and MP-SPDZ [3, 20] for generic two-party and multi-party computation, respectively. We implement the dense matrix multiplication protocol from [47] using Eigen [32] for the online phase, and the EMP toolkit [58] for the offline phase. For the ORAMs needed for the private IDF precomputation, we rely on the implementations of square-root ORAM [61] and FLORAM [22] by Doerner [21]. For our accuracy experiments, we re-implement the private IDF protocol (Section 4) in Python, and evaluate it in the clear using Scikit-Learn [52].

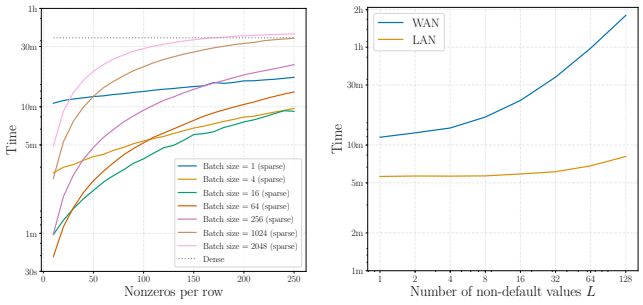
Experimental Setup. Our timings were obtained on Azure DS14v2 instances, each having 32 vCPUs and 110 GB of RAM. For WAN experiments, we placed the instances in two different regions, East US and West Europe. For all of our experiments, we set the number of features (i.e., the inner dimension for matrix multiplication experiments, and the number of words for DP-IDF computation), to 150000. We chose that number because it is about the size of Aspell's `en_US-large` dictionary [1]. For our matrix multiplication experiments, we set the bit width to 64 bits. All the times we report are total running times, i.e., we do not distinguish between offline- and online phase for dense matrix multiplication (cf. Appendix B.2).

6.1 Running Time

6.1.1 Sparse Matrix Multiplication

In this section, we want to measure the improvement of our sparse matrix multiplication protocol over the dense case, and explore trade-offs that occur by tuning different parameters. As we have seen in Section 3, our sparse matrix multiplication protocol is an extension of our inner product protocol. By processing multiple rows at once, we reduce the number of calls to $\mathcal{F}^{\text{Perm}}$ needed. On the other hand, we possibly increase the number of non-zeros in each such *batch* of rows, as we have to consider all columns that are non-zero in at least one row.

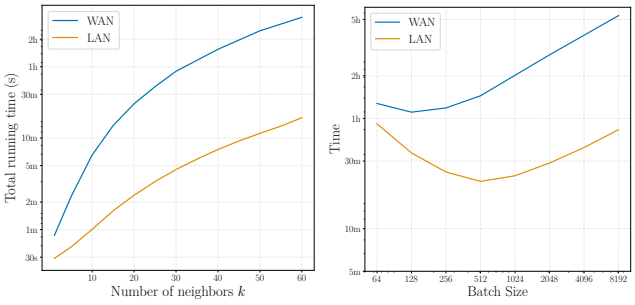
This results in an interesting trade-off between sparsity and batch size, which we explore here. To that end, we fix some of the parameters. In particular, we only evaluate matrix-vector products $\mathbf{Ab}$, and we fix the number of rows of $\mathbf{A}$ to 2048. Then, we measure the time taken for this sparse matrix-vector multiplication using different batch sizes and different sparsity levels.



(a) Running time of a sparse matrix-vector multiplication (LAN).

(b) Running time of the private IDF precomputation.

Fig. 5. Evaluation of the running times of our protocols from Sections 3 and 4.



(a) Running time the top-k selection phase, i.e., Steps (c) and (d) from Figure 4.

(b) Running time of sparse matrix-vector multiplication on real-world data.

Fig. 6. Running times of our private k-NN classification protocol from Section 5. The times correspond to a single classification run with 28000 documents consisting of Amazon product reviews [2]. Overall, we can do a full classification run on this dataset in less than 40 minutes for any $k \in [1, 60]$.

We also measure the time taken using only dense multiplication and use it as a baseline.

The results are shown in Figure 5a. We can see that for the range of parameters we tested, batches of size 16 or 64 give the best running times, depending on the sparsity level. It also becomes apparent that for a suitably chosen batch size, our protocol from Section 3 consistently outperforms the dense baseline by at least an order of magnitude. However, note that the running time of our sparse matrix-vector multiplication inherently depends on the assumed public upper bound on the sparsity (number of nonzeros per row). As the number of nonzeros approaches the total number of columns, the dense baseline will eventually become more efficient than our sparse protocol, due to the overhead of generating correlated permutations in the latter.

6.1.2 Private IDF Precomputation

We also evaluate the time needed for our second protocol, the differentially private IDF generation from Section 4. Recall that this protocol is intended to be used as a one-time precomputation step. Once the private IDF are computed and released, all parties can use them to perform feature extraction locally (cf. Section 5).

For our evaluation, we fix the vocabulary size to 150000, which corresponds to a large english dictionary [1]. We then run the precomputation phase for different values of L, i.e., the number of non-default IDF values selected. The results are shown in Figure 5b. In the LAN setting, the running times stay below 10 minutes. This increases to up to two hours in the WAN setting. Still, as this protocol needs to be only run once per dataset, this is certainly practical in real-world settings.

6.2 Secure Document Classification

Until now, while we chose the dimensions of the inputs in the previous section to match the ones found in text data, we have not used any features of specific datasets in our evaluation. Here, we explore how our protocols scale when applied to real data. To that end, we evaluate our implementation of the k-NN application described in Section 5.

First, we evaluate the time needed for the top-k selection phase (Steps (c) and (d) in Figure 4). We then explore how the sparsity characteristics of real-world data affects the running time of sparse matrix multiplication. And finally, we look at how our differentially private IDF values affect the accuracy of an end-to-end k-NN classification.

6.2.1 Datasets

We used two publicly available datasets to set up a multi-class document classification tasks. The first dataset is a repository of Amazon product reviews spanning May 1996 to July 2014 [2, 33]. We used the *5-core* version of the dataset containing only products with at least five reviews. From the entire dataset we extracted reviews for products in four different categories: “Clothing, Shoes and Jewelry”, “Toys and Games”, “Tools and Home Improvement”, and “Grocery and Gourmet Food”. We use these product categorizations to set up a document classification problem with four classes. To construct the dataset we randomly selected 28K reviews from the four classes with a uniform class distribution.

This resulted in a dataset with approximately 40K distinct words, where documents contain an average of 86 words and a maximum of 3300 words. The second dataset is the RCV1 corpus of Reuters newswire stories produced between August 1996 and August 1997 [38]. Documents in this dataset come annotated with multiple hierarchical labels related to topic, industry and geography. To set up a multi-class classification problem we selected all documents with a single label under topic "Government/Social (GCAT)" and removed any topic with less than five documents, resulting in a total of 19 classes. From the resulting corpus we randomly selected 28K via stratified sampling. Performing this process with the tokenized version of the dataset released with [38] resulted in a dataset with approximately 70K distinct words (tokens), where documents contain an average of 158 words and a maximum of 2746 words.

6.2.2 Running Time of top-k Selection

We implement steps (c) and (d) in Figure 4 using Obliv-C [60] and MP-SPDZ [3]. We then evaluate their running time on the Reviews dataset [2]. The results are shown in Figure 6a. It can be seen that top-k selection does impact the overall running time, while the majority of the computation time for a full classification is still spent on similarity computation.

6.2.3 Effect of Sparsity Distribution on Running Time

For the sparse matrix multiplication experiments in Section 6.1.1, we chose the locations of non-zero values in each row of the matrix uniformly and independently at random. This does not reflect the distribution of words in real-world texts, which usually follows a power-law distribution [53]. Therefore, we re-run our matrix-vector multiplication experiment, this time fixing the sparsity for each batch size to the average sparsity of batches of that size in our first real-world dataset [2]. We also set the number of rows in the matrix to the number of documents in our dataset, i.e., 28000. In Figure 6b, we show the results. While previously (Figure 5a), the optimal batch size was 16 in most cases, it is 512 when considering the distribution of real data. This shows that our protocol is particularly well equipped to handle real-world inputs.

Overall, for any $k \in [1, 60]$, a full classification run on the reviews dataset takes less than 40 minutes in total time. For comparison, the same computation using only dense matrix multiplication would take more than 8 hours, leading to an improvement of at least 12x.

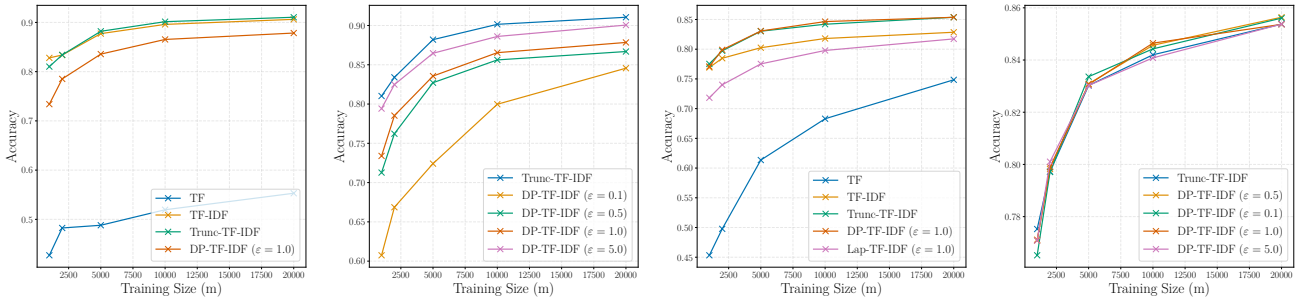
6.2.4 Accuracy of Differentially Private IDFs

To evaluate the effect of DP on the privacy-preserving IDF computation described in Section 4.2 we used the resulting feature representations in two document classification tasks using a k-NN classifier.

Baselines. To quantify the effect on the accuracy of the resulting k-NN classifier of using a TF-IDF feature representation with differentially private IDFs (*DP-TF-IDF*), we compare our approach against the following baselines: *TF* where documents are only represented by their TF vector, which can be computed locally; *TF-IDF* where we use true IDFs computed without DP; *Lap-TF-IDF* where differentially private IDFs are released using the naïve application of the Laplace mechanism sketched at the beginning of Section 4.2; *Trunc-TF-IDF* where truncated IDFs computed as in Functionality 2 but without noise (i.e. the setting $\varepsilon = \infty$).

Hyper-Parameter Tuning. To assess the predictive performance of the different feature representations, we further split the data into 70% for training, 15% for validation, and 15% for testing while maintaining the class proportions in each of the subsets. When testing the effect of the amount of training data on the overall accuracy of the model we further subsample the $\sim$ 20K training examples to obtain a smaller training set. For each training size and privacy parameter, we tune the hyper-parameters of each algorithm separately by optimizing the accuracy on the validation set, and then report the resulting accuracy on the test set. Since differential privacy introduces randomness in the computation, each accuracy measure is obtained by averaging over 20 independent runs. The hyper-parameter ranges over which the optimization is performed are as follows: number of neighbors $k \in [1, 60]$, number of non-default IDFs $L \in \{32, 64, 128\}$, and default values $c_0 \in \{16, 32, 64, 128\}$. These ranges were selected after an initial data exploration phase.

Results. The results of these experiments are displayed in Figure 7. In Figure 7a and 7c we compare the accuracy of k-NN classification as a function of the size of the training dataset using the TF-IDF representation obtained with our method (for $\varepsilon = 1$) and the baselines described above. We observe that IDFs are necessary to obtain good accuracies, as the TF baseline performs poorly on both datasets. Additionally we observe that



(a) Reviews – Baselines Comparison **(b)** Reviews – Effect of ε **(c)** RCV1 – Baselines Comparison **(d)** RCV1 – Effect of ε

Fig. 7. Results of accuracy vs. training size experiments for k-NN with differentially private TF-IDF

Lap-TF-IDF is better than plain TF, but worse than our DP-TF-IDF. On the reviews dataset the Trunc-TF-IDF baseline has the same performance as standard TF-IDF, while in the RCV1 dataset the former is slightly better. Finally, our method is slightly worse than not using DP on the reviews dataset, but performs identically to the best baseline on the RCV1 dataset.

In Figures 7b and 7d we explore the effect of the privacy parameter ε on DP-TF-IDF compared to the best baseline (Trunc-TF-IDF). On the reviews dataset we see that increasing ε leads to a better feature representation, with $\varepsilon = 1$ incurring a 3% accuracy loss with respect to the best non-private feature representation. On the other hand, on the RCV1 dataset, DP-TF-IDF is quite insensitive to the choice of ε and matches the behavior of the best baseline for all the values we tried ($\varepsilon \in \{0.1, 0.5, 1, 5\}$).

7 Related work

As mentioned in the introduction, while several recent works have proposed MPC protocols for machine learning tasks, none of them exploit the input distribution for efficiency. This is in contrast with computation in the clear, where dedicated algorithms and data structures have been developed for different kinds of sparsity patterns. Moreover, all of these protocols assume that feature extraction has been already performed. This is reasonable for settings where that step can be computed locally by each party. However, as in the case of TF-IDF, several powerful feature extraction techniques and normalization steps may require data held by different parties.

Regarding our more concrete contributions, several secure 2PC protocols for matrix multiplication have been recently proposed [28, 37, 47]. These protocols operate over explicit matrix representation, and thus are not tailored to exploit sparsity. On the other hand, combinations of MPC with DP have been proposed before in the context of limiting leakage of access patterns in secure computation [44], private set intersection [31], and protocols for private record linkage (see [34] and references therein). He et al. [34] use an indistinguishability-based definition that is limited to deterministic functionalities. In contrast, we define security with DP leakage in the simulation-based paradigm that is also used by [31, 44], but unlike these works we allow the output of the final computation to depend on the leakage. The advantage of a simulation-based definition is the fact that it allows for straight-forward composition, which we use in our security proofs (see for example Appendix D).

Regarding our application in private text analysis, related work can be found in the context of *similar document detection* [13, 15, 36, 48]. Another trend of related work is in *privacy-preserving nearest neighbors computation* [19, 39, 54, 55]. However, a remarkable difference between these existing works and ours is in the threat model. In all the contributions mentioned above, either the computation is delegated to two *non-colluding* parties – sometimes referred to as the *two-server model* in MPC – or only involves two parties (for example, one server and one client). In contrast, our threat model allows the database to be distributed among multiple servers who might collude with each other or the client.

8 Conclusion

Our MPC protocol for k-NN classification achieves provable security in the distributed setting with possibly colluding servers, which has not been reported in academic literature before. At the same time, our evaluation shows that it scales to real-world dataset sizes and

is viable in both LAN and WAN settings. We show that by precomputing differentially private statistics, performance can be improved by an order of magnitude, while providing a principled way to trade off between accuracy and privacy.

Apart from classification, our private k-NN algorithm can be easily adapted to support other types of queries on distributed datasets, for example private duplicate detection, or query answering. Additionally, other document similarity measures can be implemented atop our protocol for secure two-party sparse matrix multiplication. Moreover, our protocol for sparse matrix multiplication is general in that it works on arithmetic sharings, and hence can be directly used as a building block in other applications. The latter might be of independent interest to the MPC community. In future work we plan to address some of its applications, and study further improvements using recent results on Private Set Intersection.

Beyond our concrete contributions, this work shows that hybrid solutions combining MPC and DP are a promising venue for privacy-preserving data analysis on distributed data, as carefully designed DP mechanisms for approximated functionalities can enable efficient MPC protocols.

Acknowledgements

Phillipp and Lennart were supported by the German Research Foundation (DFG) through Research Training Group GRK 1651 (SOAMED). Adrià was supported by the EPSRC grant EP/N510129/1, and funding from the UK Government's Defence & Security Programme in support of the Alan Turing Institute. Server and Client icons in Figure 4 are from www.flaticon.com. Server icon designed by Smashicons, Client icon designed by Freepik.

References

[1] Aspell dictionary creation. http://app.aspell.net/create.

[2] Amazon Product Data. http://jmcauley.ucsd.edu/data/amazon/.

[3] Multi-protocol SPDZ. https://github.com/data61/MP-SPDZ, 2018.

[4] 1000 Genomes Project Consortium et al. A global reference for human genetic variation. *Nature*, 526(7571):68, 2015.

[5] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

[6] M. Al-Rubaie, P. Y. Wu, J. M. Chang, and S. Kung. Privacy-preserving PCA on horizontally-partitioned data. In *DSC*, pages 280–287, 2017.

[7] M. Bafna and J. Ullman. The price of selection in differential privacy. In *COLT*, pages 151–168. PMLR, 2017.

[8] K. E. Batcher. Sorting Networks and Their Applications. In *AFIPS Spring Joint Computing Conference*, volume 32, pages 307–314, 1968.

[9] D. Beaver. Efficient Multiparty Protocols Using Circuit Randomization. In *CRYPTO*, pages 420–432, 1991.

[10] J. Beel, B. Gipp, S. Langer, and C. Breitinger. Paper recommender systems: a literature survey. *International Journal on Digital Libraries*, 17(4):305–338, 2016.

[11] A. Beimel, K. Nissim, and E. Omri. Distributed private data analysis: Simultaneously solving how and what. In *CRYPTO*, pages 451–468. Springer, 2008.

[12] A. Ben-Efraim, Y. Lindell, and E. Omri. Optimizing Semi-Honest Secure Multiparty Computation for the Internet. In *ACM Conference on Computer and Communications Security*, pages 578–590, 2016.

[13] C. Blundo, E. D. Cristofaro, and P. Gasti. EsPRESSo: Efficient privacy-preserving evaluation of sample set similarity. In *DPM/SETOP*, pages 89–103, 2012.

[14] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *NDSS*, 2015.

[15] S. Buyrukbilen and S. Bakiras. Secure similar document detection with simhash. In *Secure Data Management*, pages 61–75, 2013.

[16] R. Canetti. Security and composition of multiparty cryptographic protocols. *J. Cryptology*, 13(1):143–202, 2000.

[17] K. Chaudhuri and S. Dasgupta. Rates of convergence for nearest neighbor classification. In *Advances in Neural Information Processing Systems*, pages 3437–3445, 2014.

[18] K. Chaudhuri and S. A. Vinterbo. A stability-based validation procedure for differentially private machine learning. In *NIPS*, pages 2652–2660, 2013.

[19] H. Chen, I. Chillotti, Y. Dong, O. Poburinnaya, I. P. Razenshteyn, and M. S. Riazi. SANNS: scaling up secure approximate k-nearest neighbors search. *IACR Cryptology ePrint Archive*, 2019:359, 2019.

[20] I. Damgård, V. Pastro, N. P. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417, 2012.

[21] J. Doerner. The Absentminded Crypto Kit. URL https://bitbucket.org/jackdoerner/absentminded-crypto-kit/.

[22] J. Doerner and A. Shelat. Scaling ORAM for Secure Computation. In *CCS*, pages 523–535, 2017.

[23] C. Dwork and A. Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, Aug. 2014.

[24] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In *EUROCRYPT*, pages 486–503, 2006.

[25] C. Dwork, F. McSherry, K. Nissim, and A. D. Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*, pages 265–284, 2006.

[26] A. Efros. How to stop worrying and learn to love nearest neighbors. NIPS workshop on Nearest Neighbors for Modern Applications with Massive Data, 2017.

[27] M. Fanaeepour and B. I. P. Rubinstein. Histogramming privately ever after: Differentially-private data-dependent error bound optimisation. In *ICDE*, pages 1204–1207. IEEE Computer Society, 2018.

[28] A. Gascón, P. Schoppmann, B. Balle, M. Raykova, J. Doerner, S. Zahur, and D. Evans. Privacy-Preserving Distributed Linear Regression on High-Dimensional Data. *Proceedings on Privacy Enhancing Technologies*, 2017(4):345–364, Oct. 2017.

[29] N. Gilboa. Two party RSA key generation. In *CRYPTO*, pages 116–129. Springer, 1999.

[30] O. Goldreich. *The Foundations of Cryptography – Volume 2, Basic Applications*. 2004.

[31] A. Groce, P. Rindal, and M. Rosulek. Cheaper private set intersection via differentially private leakage. *PoPETs*, 2019 (3):6–25, 2019.

[32] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[33] R. He and J. J. McAuley. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *WWW*, pages 507–517. ACM, 2016.

[34] X. He, A. Machanavajjhala, C. J. Flynn, and D. Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *ACM Conference on Computer and Communications Security*, pages 1389–1406, 2017.

[35] Y. Huang, D. Evans, and J. Katz. Private set intersection: Are garbled circuits better than custom protocols? In *NDSS*, 2012.

[36] W. Jiang, M. Murugesan, C. Clifton, and L. Si. Similar document detection with limited information disclosure. In *ICDE*, pages 735–743, 2008.

[37] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. Gazelle: A Low Latency Framework for Secure Neural Network Inference. *IACR Cryptology ePrint Archive*, 2018:73, 2018.

[38] D. D. Lewis, Y. Yang, T. G. Rose, and F. Li. Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397, 2004.

[39] F. Li, R. Shin, and V. Paxson. Exploring privacy preservation in outsourced k-nearest neighbors with multiple data owners. In *CCSW*, pages 53–64, 2015.

[40] Y. Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[41] Y. Lindell and B. Pinkas. A Proof of Security of Yao's Protocol for Two-Party Computation. *J. Cryptology*, 22(2): 161–188, 2009.

[42] J. Liu, M. Juuti, Y. Lu, and N. Asokan. Oblivious neural network predictions via minionn transformations. In *CCS*, pages 619–631, 2017.

[43] C. Manning, P. Raghavan, and H. Schütze. Scoring, term weighting and the vector space model. In *Introduction to information retrieval*, pages 100–122. 2008.

[44] S. Mazloom and S. D. Gordon. Secure computation with differentially private access patterns. In *ACM Conference on Computer and Communications Security*, pages 490–507, 2018.

[45] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, pages 94–103, 2007.

[46] I. Mironov, O. Pandey, O. Reingold, and S. P. Vadhan. Computational differential privacy. In *CRYPTO*, pages 126–142, 2009.

[47] P. Mohassel and Y. Zhang. SecureML: A system for scalable privacy-preserving machine learning. In *IEEE Symposium on Security and Privacy*, pages 19–38, 2017.

[48] M. Murugesan, W. Jiang, C. Clifton, L. Si, and J. Vaidya. Efficient privacy-preserving similar document detection. *VLDB J.*, 19(4):457–475, 2010.

[49] K. Nayak, X. S. Wang, S. Ioannidis, U. Weinsberg, N. Taft, and E. Shi. GraphSC: Parallel secure computation made easy. In *IEEE Symposium on Security and Privacy*, pages 377–394. IEEE Computer Society, 2015.

[50] V. Nikolaenko, S. Ioannidis, U. Weinsberg, M. Joye, N. Taft, and D. Boneh. Privacy-preserving matrix factorization. In *ACM Conference on Computer and Communications Security*, pages 801–812. ACM, 2013.

[51] V. Nikolaenko, U. Weinsberg, S. Ioannidis, M. Joye, D. Boneh, and N. Taft. Privacy-preserving ridge regression on hundreds of millions of records. In *IEEE Symposium on Security and Privacy*, pages 334–348, 2013.

[52] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. VanderPlas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[53] D. M. W. Powers. Applications and explanations of zipf's law. In *CoNLL*, pages 151–160, 1998.

[54] M. S. Riazi, B. Chen, A. Shrivastava, D. S. Wallach, and F. Koushanfar. Sub-linear privacy-preserving search with untrusted server and semi-honest parties. *CoRR*, abs/1612.01835, 2016.

[55] H. Rong, H. Wang, J. Liu, and M. Xian. Privacy-preserving k-nearest neighbor computation in multiple cloud environments. *IEEE Access*, 4:9589–9603, 2016.

[56] A. Waksman. A permutation network. *J. ACM*, 15(1): 159–163, 1968.

[57] X. Wang, T.-H. H. Chan, and E. Shi. Circuit ORAM: On Tightness of the Goldreich-Ostrovsky Lower Bound. In *ACM Conference on Computer and Communications Security*, pages 850–861, 2015.

[58] X. Wang, A. J. Malozemoff, and J. Katz. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit, 2016.

[59] A. C.-C. Yao. How to Generate and Exchange Secrets (Extended Abstract). In *FOCS*, pages 162–167, 1986.

[60] S. Zahur and D. Evans. Obliv-C: A Language for Extensible Data-Oblivious Computation. *IACR Cryptology ePrint Archive*, 2015:1153, 2015.

[61] S. Zahur, X. S. Wang, M. Raykova, A. Gascón, J. Doerner, D. Evans, and J. Katz. Revisiting Square-Root ORAM: Efficient Random Access in Multi-party Computation. In *IEEE Symposium on Security and Privacy*, pages 218–234, 2016.

A Security

The following definition of security against semi-honest adversaries for multi-party computation was adapted from [30, Definition 7.5.1]. See also [40].

For any number of parties $n \geq 2$, a probabilistic *n-party functionality* is a function $\mathcal{F} : (\{0,1\}^*)^n \to (\{0,1\}^*)^n$. Let $\mathcal{F}_i(x_1, \ldots, x_n)$ denote the i-th element of $\mathcal{F}(x_1, \ldots, x_n)$. For each party $i \in [n]$, $\mathcal{F}_i$ takes all parties' inputs $\bar{x} := (x_1, \ldots, x_n)$ and returns the output $\mathcal{F}_i(\bar{x})$ to Party i. Let Π be a protocol that computes $\mathcal{F}$. Let $\mathsf{output}^\Pi(\bar{x})$ denote the combined output of Π. Additionally, each party has a view on the protocol execution that is denoted by $\mathsf{view}_i^\Pi(\bar{x})$ and contains Party i's inputs, internal random state, and all received messages. For any subset of parties $I = \{i_1, \ldots, i_t\} \subseteq [n]$, let $x_I = (x_{i_1}, \ldots, x_{i_t})$, $\mathcal{F}_I(\bar{x}) := \big(\mathcal{F}_{i_1}(\bar{x}), \ldots, \mathcal{F}_{i_t}(\bar{x})\big)$, and $\mathsf{view}_I^\Pi(\bar{x}) := \big(I, \mathsf{view}_{i_1}^\Pi(\bar{x}), \ldots, \mathsf{view}_{i_t}^\Pi(\bar{x})\big)$.

Now, a *simulator* $\mathcal{S}$ is a probabilistic polynomial-time algorithm that takes as arguments the set I, the inputs $x_{i_1}, \ldots, x_{i_t}$ of all parties in I, and their outputs from the functionality, i. e. $\mathcal{F}_I(\bar{x})$. Using these, $\mathcal{S}$ simulates a view for the parties in I. If such a simulator exists, and for each I satisfies

$$\Big\{\Big(\mathcal{S}\big(I, x_I, \mathcal{F}_I(\bar{x})\big), \mathcal{F}(\bar{x})\Big)\Big\}_{\bar{x}\in(\{0,1\}^*)^n} \overset{c}{\equiv} \Big\{\big(\mathsf{view}_I^\Pi(\bar{x}), \mathsf{output}^\Pi(\bar{x})\big)\Big\}_{\bar{x}\in(\{0,1\}^*)^n} \quad (2)$$

then Π *privately computes* $\mathcal{F}$ [30]. Here, $\overset{c}{\equiv}$ denotes *computational indistinguishability* as defined in [30, 40]. In the two-party case, we write $\mathcal{S}_i\big(x_i, \mathcal{F}(x_1, x_2)\big)$ instead of $\mathcal{S}\big(\{i\}, (x_i), \mathcal{F}(x_1, x_2)\big)$ to denote the simulator for the view of Party $i \in \{1, 2\}$.

B Baseline Protocol

B.1 Remarks on notation

Given a set of parties $S \subseteq \{1, \ldots, n\}$ and a value $v \in \mathbb{Z}_q$, we write $[\![v]\!]^S$ to denote that v is additively shared among the parties in S, as defined in Section 3.2. We identify the shared value with the collection of shares and write $[\![v]\!]^S = ([\![v]\!]_i^S : i \in S)$ where $[\![v]\!]_i^S$ is known only to the ith party. When S is clear from the context we shall just write $[\![v]\!]$ and $[\![v]\!]_i$.

Protocol 4: Dense Mult

Parties: 1, 2.

Input: Party 1: $\mathbf{A} \in \mathbb{Z}_q^{l\times m}$; Party 2: $\mathbf{B} \in \mathbb{Z}_q^{m\times n}$

Output: Party i : $[\![\mathbf{AB}]\!]_i \in \mathbb{Z}_q^{l\times n}$

1: Party 1 computes $\mathbf{U}, [\![\mathbf{UV}]\!]_1 \leftarrow \mathcal{F}^{\text{Off}}(l, m, n)$ and sends $\mathbf{E} = \mathbf{A} - \mathbf{U}$

2: Party 2 computes $\mathbf{V}, [\![\mathbf{UV}]\!]_2 \leftarrow \mathcal{F}^{\text{Off}}(l, m, n)$ and sends $\mathbf{F} = \mathbf{B} - \mathbf{V}$

3: Party 1 sets $[\![\mathbf{AB}]\!]_1 = \mathbf{EF} + \mathbf{UF} + [\![\mathbf{UV}]\!]_1$

4: Party 2 sets $[\![\mathbf{AB}]\!]_2 = \mathbf{EV} + [\![\mathbf{UV}]\!]_2$;

B.2 Dense Matrix Multiplication

We use the matrix multiplication protocol of Mohassel and Zhang [47], which we show in Protocol 4 and state its correctness with respect to the two-party matrix multiplication functionality $\mathcal{F}^{\text{Mult}}$ in Theorem 8.

The protocol works in the so-called preprocessing model, a common paradigm in MPC which delegates part of the computation to a data-independent offline phase, denoted by $\mathcal{F}^{\text{Off}}$. In Protocol 4, this refers to the computation of $\mathbf{U}$, $\mathbf{V}$ and $[\![\mathbf{UV}]\!]_i$, which can be done in advance without knowledge of $\mathbf{A}$ or $\mathbf{B}$.

Theorem 8 ([9, 47])**.** *Assuming a secure implementation of $\mathcal{F}^{\text{Off}}$, Protocol 4 implements $\mathcal{F}^{\text{Mult}}$ with security against semi-honest adversaries.*

C Proof of Theorem 2

Theorem 2. *For any $\mathbf{A} \in \mathbb{Z}_q^{l\times m}$, $\mathbf{B} \in \mathbb{Z}_q^{m\times n}$, let $\tilde{\mathbf{A}}, \tilde{\mathbf{B}}$ be constructed according to the protocol described in Figure 1. Then $\mathbf{AB} = \tilde{\mathbf{A}}\tilde{\mathbf{B}}$.*

Proof. By construction of $\tilde{\mathbf{A}}$, for all $i \in \{1, \ldots, l\}$ and all $j \in \{1, \ldots, n\}$,

$$(\tilde{\mathbf{A}}\tilde{\mathbf{B}})_{ij} = \sum_{k=1}^{l_\mathbf{A}+l_\mathbf{B}} \tilde{a}_{ik}\tilde{b}_{kj} = \sum_{k=1}^{l_\mathbf{A}+l_\mathbf{B}} \hat{a}_{i\pi_1^{-1}(k)}\hat{b}_{\pi_2^{-1}(k)j}.$$

From the definition of $\mathcal{F}^{\text{Perm}}$, one of the following cases holds for any pair $(k_1, k_2) := \big(\pi_1^{-1}(k), \pi_2^{-1}(k)\big), k \in \{1, \ldots, l_\mathbf{A} + l_\mathbf{B}\}$:

Case 1 $k_1 \le l_{\mathbf{A}}$ and $k_2 \le l_{\mathbf{B}}$. Then there is a unique $k' \in \{1,\dots,m\}$ such that $\left(\mathcal{I}_{\mathbf{A}}^{\mathsf{Col}}\right)_{k_1} = \left(\mathcal{I}_{\mathbf{B}}^{\mathsf{Row}}\right)_{k_2} = k'$ and $\hat{a}_{ik_1}\hat{b}_{k_2 j} = a_{ik'}b_{k'j}$.

Case 2 $k_1 > l_{\mathbf{A}}$ or $k_2 > l_{\mathbf{B}}$. Then $\hat{a}_{ik_1}$ or $\hat{b}_{k_2 j}$ are zero, and thus $\hat{a}_{ik_1}\hat{b}_{k_2 j} = 0$.

On the other hand, for any $k' \in \{1,\dots,m\}$ with $a_{ik'}b_{k'j} \neq 0$, there is a pair (k_1,k_2) with $(\mathcal{I}_{\mathbf{A}}^{\mathsf{Col}})_{k_1} = (\mathcal{I}_{\mathbf{B}}^{\mathsf{Row}})_{k_2} = k'$ and thus a unique $k \in \{1,\dots,l_{\mathbf{A}}+l_{\mathbf{B}}\}$ s.t. $\pi_1(k_1) = \pi_2(k_2) = k$. Therefore,

$$(\tilde{\mathbf{A}}\tilde{\mathbf{B}})_{ij} = \sum_{k' \in \mathcal{I}_{\mathbf{A}}^{\mathsf{Col}} \cap \mathcal{I}_{\mathbf{B}}^{\mathsf{Row}}} a_{ik'}b_{k'j} = (\mathbf{AB})_{ij}.$$

□

D Proof of Theorem 3

Theorem 3. *Given public sparsity values $l_{\mathbf{A}}, l_{\mathbf{B}}$ and implementations of $\mathcal{F}^{\text{Mult}}$ and $\mathcal{F}^{\text{Perm}}$ that are secure against semi-honest adversaries, the protocol in Figure 1 implements $\mathcal{F}^{\text{Mult}}$ with security against semi-honest adversaries.*

Proof. We only give the proof for the view of Party 1. By symmetry, the proof for Party 2 follows analogously. Our proof uses the standard simulation paradigm for cryptographic protocols (see [30, 40] and Appendix A). For a functionality $\mathcal{F}$ and a protocol Π, we denote the output to player i of an execution with inputs x, y by $\mathcal{F}_i(x,y)$ and $\mathsf{output}_i^{\Pi}(x,y)$, respectively.

We present our proof in the $(\mathcal{F}^{\text{Perm}}, \mathcal{F}^{\text{Mult}})$-hybrid model. That is, we construct a simulator $\mathcal{S}_1^{\Pi}$ for the view of party 1 assuming ideal functionalities for $\mathcal{F}^{\text{Perm}}$ and $\mathcal{F}^{\text{Mult}}$. Security in the standard model then follows immediately from the security of the corresponding protocols and the modular composition theorem [16].

Our simulator $\mathcal{S}_1^{\Pi}$ in the ideal model simulates the view of Party 1 on the Protocol in Figure 1, i.e.,

$$\mathsf{view}_1^{\Pi}(\mathbf{A},\mathbf{B}) = \left(\mathbf{A}, \mathcal{F}_1^{\text{Perm}}(\mathcal{I}_{\mathbf{A}}^{\mathsf{Col}}, \mathcal{I}_{\mathbf{B}}^{\mathsf{Row}}), \mathcal{F}_1^{\text{Mult}}(\tilde{\mathbf{A}},\tilde{\mathbf{B}})\right).$$

Upon receiving input $\mathbf{A}$ and output $\mathcal{F}_1^{\text{Mult}}(\mathbf{A},\mathbf{B})$, the simulator $\mathcal{S}_1^{\Pi}$:

1. samples a permutation π_1' of $\{1,\dots,l_{\mathbf{A}}+l_{\mathbf{B}}\}$ uniformly at random,
2. outputs $\left(\mathbf{A}, \pi_1', \mathcal{F}_1^{\text{Mult}}(\mathbf{A},\mathbf{B})\right)$.

By Theorem 1, $\mathcal{F}_1^{\text{Perm}}(\mathcal{I}_{\mathbf{A}}^{\mathsf{Col}}, \mathcal{I}_{\mathbf{B}}^{\mathsf{Row}})$ outputs a uniformly random permuation of $l_{\mathbf{A}}+l_{\mathbf{B}}$. Similarly, $\mathcal{F}_1^{\text{Mult}}(\tilde{\mathbf{A}},\tilde{\mathbf{B}})$ is a uniformly random matrix. Finally, by Theorem 2, $\mathcal{F}_1^{\text{Mult}}(\tilde{\mathbf{A}},\tilde{\mathbf{B}})\big)$ is identically distributed to $\mathcal{F}^{\text{Mult}}(\mathbf{A},\mathbf{B})$. Therefore

$$\Big(\mathsf{view}_1^{\Pi}(\mathbf{A},\mathbf{B}), \mathsf{output}^{\Pi}(\mathbf{A},\mathbf{B})\Big) \equiv \Big(S_1^{\Pi}\big(\mathbf{A}, \mathcal{F}_1^{\text{Mult}}(\mathbf{A},\mathbf{B})\big), \mathcal{F}^{\text{Mult}}(\mathbf{A})\Big).$$

□

In practice, we implement $\mathcal{F}^{\text{Mult}}$ using the multiplication protocol of Mohassel and Zhang [47] (where the offline phase is provided by Gilboa's multiplication protocol [29]), and $\mathcal{F}^{\text{Perm}}$ using Yao's Garbled Circuits [41].

E Proof of Theorem 5

Theorem 5. $\mathcal{F}^{\text{DP-IDF}}$ *(Functionality 2) is ε-DP.*

Proof. Let $\varepsilon_0 = \varepsilon/(2L)$. Note that for any pair of neighboring datasets $Z \simeq Z'$ and any word $v \in \mathcal{V}$ we have $|c_v - c_v'| \le 1$. Thus, the analysis of the exponential mechanism [23, Theorem 3.6] implies that releasing each selected word v is ε_0-DP. Furthermore, the analysis of the Laplace mechanism [23, Theorem 3.10] implies that releasing $\tilde{c}_v$ for each selected word is ε_0-DP. Note also that the values released for the words which are not selected are independent of the dataset Z. Thus, the result follows by applying the standard composition theorem of differential privacy $2L$ times [23, Theorem 3.14]. □

F Proof of Theorem 6

Theorem 6. *For any large enough $m = |Z|$ there exists $c_0 = \Theta(\sqrt{m})$ such that with high probability*

$$\frac{\|\phi_{\text{idf}} - \tilde{\phi}_{\text{idf}}\|_1}{\|\phi_{\text{idf}}\|_1} \le \tilde{O}\left(\frac{L^2}{\varepsilon m|\mathcal{V}|} + \left(1 - \frac{L}{|\mathcal{V}|}\right)\log\left(m - \frac{L}{\varepsilon}\right)\right) .$$

Throughout the proof we write $V = |\mathcal{V}|$ and $\varepsilon_0 = \varepsilon/2L$ for convenience. Recall that as input our mechanism receives counts $\{c_v : v \in \mathcal{V}\}$ estimated on a Z database with m labeled documents $x_1,\dots,x_m$ over vocabulary $\mathcal{V}$ of size $V = |\mathcal{V}|$. For the utility analysis we will assume a distribution $\mathcal{D}$ over documents such that the x_i are i.i.d. Using $\mathcal{D}$ we can define the word occurrence probabilities $p_v = \mathbb{P}_{x\sim\mathcal{D}}[v \in x]$ so that the expected counts can be written as $\mathbb{E}[|Z|_v] = \mathbb{E}[c_v] = mp_v$. Using these probabilities we can define an order on the vocabulary $\mathcal{V} = \{v_1,\dots,v_V\}$ in such a way that

$p_{v_1} \geq p_{v_2} \geq \cdots \geq p_{v_V}$. To simplify our notation throughout the proof we define $\phi_v = \phi_{\text{idf}}(v, Z)$ and $\tilde{\phi}_v = \tilde{\phi}_{\text{idf}}(v, Z)$.

We start by splitting the vocabulary $\mathcal{V}$ into two parts: the set of words $\mathcal{V}_s$ selected by the exponential mechanism, and the set of not selected words $\mathcal{V}_{\bar{s}} = \mathcal{V} \backslash \mathcal{V}_s$. Note that by definition we have $|\mathcal{V}_s| = L$. The main idea behind our analysis of the error between the private and the non-private IDFs is to consider the words in $\mathcal{V}_s$ and $\mathcal{V}_{\bar{s}}$ separately. The error for words in the former depends on Laplace noise added to c_v, while the error for words in the latter depends on the difference between the default value c_0 and the true count c_v. The first source of error can be controlled by bounding the noise added by the Laplace mechanism. To control the second source of error we will need to make sure that most of the words in $\mathcal{V}_{\bar{s}}$ have counts not too far from c_0.

We start by recalling well-known facts about the Laplace and the exponential mechanism.

Lemma 1. *With probability at least $1-\gamma_l$ we have $|c_v - \tilde{c}_v| \leq \Delta_l$ simultaneously for all words $v \in \mathcal{V}_s$, where $\Delta_l = \log(L/\gamma_l)/\varepsilon_0$.*

Proof. See [23, Theorem 3.8]. □

Lemma 2. *Let $c_{(L)}$ denote the Lth greatest word count in $\{c_v : v \in \mathcal{V}\}$. With probability at least $1-\gamma_e$, if $v \in \mathcal{V}_s$, then $c_v \geq c_{(L)} - \Delta_e$, where $\Delta_e = 2\log(LV/\gamma_e)/\varepsilon_0$.*

Proof. The proof follows the same structure as the classical utility analysis for the exponential mechanism [45, Lemma 7]. □

Lemma 3. *With probability at least $1 - \gamma_L$ we have $c_{(L)} \geq mp_{v_L} - \Delta_L$, where $\Delta_L = \sqrt{2m\log(L/\gamma_L)}$.*

Proof. First note that by definition of $c_{(L)}$ we have

$$\begin{aligned}\mathbb{P}[c_{(L)} < mp_{v_L} - \Delta_L] &\leq \mathbb{P}\left[\min_{i\in[L]} c_{v_i} < mp_{v_L} - \Delta_L\right] \\ &\leq \sum_{i\in[L]} \mathbb{P}[c_{v_i} < mp_{v_L} - \Delta_L] \ .\end{aligned}$$

Since $\mathbb{E}[c_{v_i}] = mp_{v_i} \geq mp_{v_L}$ for any $i \in [L]$, by the Chernoff bound we have

$$\begin{aligned}\mathbb{P}[c_{v_i} < mp_{v_L} - \Delta_L] &\leq \mathbb{P}[c_{v_i} < mp_{v_i} - \Delta_L] \\ &\leq \exp\left(-\frac{\Delta_L^2}{2mp_{v_i}}\right) \leq \frac{\gamma_L}{L} \ . \qquad \square\end{aligned}$$

Lemma 4. *Suppose we have $c_v \geq mp_{v_L} - \Delta_L - \Delta_e$ for every word $v \in \mathcal{V}_s$. Then with probability at least $1 - \gamma_l$ we have the following:*

$$\sum_{v\in\mathcal{V}_s} |\phi_v - \tilde{\phi}_v| \leq \frac{L\Delta_l}{mp_{v_L} - \Delta_L - \Delta_e - \Delta_l} \ .$$

Proof. Note that by the expression for the smoothed IDFs, for any $v \in \mathcal{V}$ we have $|\phi_v - \tilde{\phi}_v| = \left|\log\left(\frac{\tilde{c}_v+1}{c_v+1}\right)\right|$. Now, by combining the assumption $c_v \geq mp_{v_L} - \Delta_L - \Delta_e$ and Lemma 1 we see that with probability at least $1-\gamma_l$ the following is simultaneously satisfied for all $v \in \mathcal{V}_s$:

$$\begin{aligned}\tilde{c}_v &\geq c_v - \Delta_l \geq c_{(L)} - \Delta_e - \Delta_l \\ &\geq mp_{v_L} - \Delta_L - \Delta_e - \Delta_l \ .\end{aligned}$$

Thus, using that for $0 \leq y < x$ we have $\log(x) - \log(x-y) \leq y/(x-y)$, we get

$$\begin{aligned}\left|\log\left(\frac{\tilde{c}_v+1}{c_v+1}\right)\right| &\leq \log\left(\frac{c_v+1}{c_v-\Delta_l+1}\right) \leq \frac{\Delta_l}{c_v-\Delta_l} \\ &\leq \frac{\Delta_l}{mp_{v_L} - \Delta_L - \Delta_e - \Delta_l} \ .\end{aligned}$$

The result follows from noting that $|\mathcal{V}_s| = L$. □

Lemma 5. *Suppose m is large enough to satisfy the following inequality:*

$$\begin{aligned}m \geq \Bigg(\frac{1}{p_{v_L} - p_{v_{L+1}}}\Big(&\sqrt{3p_{v_{L+1}}\log((V-L)/\gamma_{\bar{s}})} \\ &+ \sqrt{2\log(L/\gamma_L)} \\ &+ 2\log(LV/\gamma_e)/\varepsilon_0\sqrt{m}\Big)\Bigg)^2 \ .\end{aligned}$$

With probability at least $1 - \gamma_{\bar{s}}$ we have $c_{v_i} < mp_{v_L} - \Delta_L - \Delta_e$ for every $i > L$.

Proof. First note that an application of the Chernoff bound together with a union bound shows that simultaneously for all $L < i \leq V$ we have, with probability at least $1 - \gamma_{\bar{s}}$:

$$c_{v_i} < mp_{v_i} + \Delta_{\bar{s}} \leq mp_{v_{L+1}} + \Delta_{\bar{s}} \ ,$$

where $\Delta_{\bar{s}} = \sqrt{3mp_{v_{L+1}}\log((V-L)/\gamma_{\bar{s}})}$. The result follows by plugging in the definitions of Δ_L and Δ_e and observing that the constraint on m implies

$$mp_{v_{L+1}} + \Delta_{\bar{s}} \leq mp_{v_L} - \Delta_L - \Delta_e \ . \quad \square$$

Note that the constraint on m above is satisfied by taking

$$m = \tilde{\Omega}\left(\frac{\log(LV)}{\varepsilon_0(p_{v_L} - p_{v_{L+1}})^2}\right) \ , \tag{3}$$

where the notation $\tilde{\Omega}(\cdot)$ hides constants and logarithmic terms in $1/\gamma_{\bar{s}}$, $1/\gamma_L$, and $1/\gamma_e$.

Lemma 6. *Suppose we have* $c_v < mp_{v_L} - \Delta_L - \Delta_e$ *for every word* $v \in \mathcal{V}_{\bar{s}}$. *Then* $\sum_{v \in \mathcal{V}_{\bar{s}}} |\phi_v - \tilde{\phi}_v|$ *is bounded by*

$$O\left((V-L)\max\left\{\log(c_0), \log\left(\frac{mp_{v_L} - \Delta_L - \Delta_e}{c_0}\right)\right\}\right) .$$

Proof. Recall that for every word $v \in \mathcal{V}_{\bar{s}}$ the mechanism outputs $\tilde{c}_v = c_0$. Furthermore, by assumption we have $0 \leq c_v < mp_{v_L} - \Delta_L - \Delta_e$, which yields:

$$\begin{aligned}
|\phi_v - \tilde{\phi}_v| &= \left|\log\left(\frac{c_0+1}{c_v+1}\right)\right| \\
&\leq \max\left\{\log(c_0+1), \log\left(\frac{mp_{v_L} - \Delta_L - \Delta_e + 1}{c_0+1}\right)\right\} \\
&= O\left(\max\left\{\log(c_0), \log\left(\frac{mp_{v_L} - \Delta_L - \Delta_e}{c_0}\right)\right\}\right) .
\end{aligned}$$

The result follows by noting that $|\mathcal{V}_{\bar{s}}| = V - L$. □

Theorem 9. *Suppose* m *satisfies* (3) *and* $c_0 = \sqrt{mp_{v_L} - \Delta_L - \Delta_e}$. *Let* $\gamma = \gamma_l + \gamma_e + \gamma_L + \gamma_{\bar{s}}$. *With probability at least* $1 - \gamma$ *we have*

$$\frac{\|\phi_{\text{idf}} - \tilde{\phi}_{\text{idf}}\|_1}{\|\phi_{\text{idf}}\|_1} \leq \tilde{O}\left(\frac{L}{V}\frac{1}{\varepsilon_0 m} + \left(1 - \frac{L}{V}\right)\log\left(m - \frac{1}{\varepsilon_0}\right)\right) .$$

Proof. By decomposing the L_1 distance according to the partition $\mathcal{V} = \mathcal{V}_s \cup \mathcal{V}_{\bar{s}}$ and plugging the bounds from Lemmas 4 and 6 we get

$$\begin{aligned}
\|\phi_{\text{idf}} - \tilde{\phi}_{\text{idf}}\|_1 &= \sum_{v \in \mathcal{V}_s} |\phi_v - \tilde{\phi}_v| + \sum_{v \in \mathcal{V}_{\bar{s}}} |\phi_v - \tilde{\phi}_v| \\
&\leq \frac{L\Delta_l}{mp_{v_L} - \Delta_L - \Delta_e - \Delta_l} \\
&\quad + (V-L)O\big(\log(mp_{v_L} - \Delta_L - \Delta_e)\big).
\end{aligned}$$

Note that the conditions to apply these lemmas hold simultaneously with probability at least $1 - \gamma$ by Lemmas 1, 2, 3, and 5. Next we observe that by the definition of $\phi_{\text{idf}}(v, Z)$ we have $\|\phi_{\text{idf}}\|_1 \geq V$. The result now follows by plugging the expression for Δ_l, Δ_e, and Δ_L into the bound above, and using the notation $\tilde{O}(\cdot)$ to hide constants and logarithmic terms not involving m. □