Differentially Private Substring and Document Counting

GIULIA BERNARDINI, University of Milan, Italy
PHILIP BILLE, Technical University of Denmark, Denmark
INGE LI GØRTZ, Technical University of Denmark, Denmark
TERESA ANNA STEINER, University of Southern Denmark, Denmark

Differential privacy is the gold standard for privacy in data analysis. In many data analysis applications, the data is a database of *documents*. For databases consisting of many documents, one of the most fundamental problems is that of pattern matching and computing (i) how often a pattern appears as a substring in the database (*substring counting*) and (ii) how many documents in the collection contain the pattern as a substring (*document counting*). In this paper, we initiate the theoretical study of substring and document counting under differential privacy.

We give an ϵ-differentially private data structure solving this problem for all patterns simultaneously with a maximum additive error of $O(\ell \cdot \text{polylog}(n\ell|\Sigma|))$, where ℓ is the maximum length of a document in the database, n is the number of documents, and $|\Sigma|$ is the size of the alphabet. We show that this is optimal up to a $O(\text{polylog}(n\ell))$ factor. Further, we show that for (ϵ, δ)-differential privacy, the bound for document counting can be improved to $O(\sqrt{\ell} \cdot \text{polylog}(n\ell|\Sigma|))$. Additionally, our data structures are efficient. In particular, our data structures use $O(n\ell^2)$ space, $O(n^2\ell^4)$ preprocessing time, and $O(|P|)$ query time where P is the query pattern. Along the way, we develop a new technique for differentially privately computing a general class of counting functions on trees of independent interest.

Our data structures immediately lead to improved algorithms for related problems, such as privately mining frequent substrings and q-grams. For q-grams, we further improve the preprocessing time of the data structure.

CCS Concepts: • **Theory of computation → Data structures design and analysis**; • **Security and privacy**;

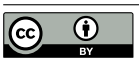
Additional Key Words and Phrases: Differential privacy, Document counting, Substring counting, String mining, Frequent pattern mining

ACM Reference Format:
Giulia Bernardini, Philip Bille, Inge Li Gørtz, and Teresa Anna Steiner. 2025. Differentially Private Substring and Document Counting. *Proc. ACM Manag. Data* 3, 2 (PODS), Article 95 (May 2025), 27 pages. https://doi.org/10.1145/3725232

1 Introduction

Differential privacy [26] is the gold standard for privacy in data analysis. It has been extensively studied in theory and employed on a large scale by companies such as Google [67], Apple [3], and Uber [44], as well as public institutions such as the US Census Bureau [1]. The definition of differential privacy requires that a randomized algorithm must have similar output distributions on similar data sets. This is controlled by a multiplicative parameter ϵ and an additive parameter δ, and an algorithm satisfying differential privacy for these parameters is called (ϵ, δ)-*differentially private*. If $\delta = 0$, then the algorithm is called ϵ-*differentially private*.

Authors' Contact Information: Giulia Bernardini, University of Milan, Milan, Italy, giulia.bernardini@unimi.it; Philip Bille, Technical University of Denmark, Lyngby, Denmark, phbi@dtu.dk; Inge Li Gørtz, Technical University of Denmark, Lyngby, Denmark, inge@dtu.dk; Teresa Anna Steiner, University of Southern Denmark, Odense, Denmark, steiner@imada.sdu.dk.



This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2025 Copyright held by the owner/author(s).
ACM 2836-6573/2025/5-ART95
https://doi.org/10.1145/3725232

One of the most fundamental types of problems in differential privacy is that of *counting queries*, where a data set consists of elements from a predefined universe, and the goal is for a set of properties (i.e., a *query family*), to count the number of elements in the data set which satisfy each property. An example of a family of counting queries is that of histograms, where the query family contains a query for each element in the universe, and the goal is to count how often each element appears in the data set. Another example is the family of threshold functions, where each query corresponds to a threshold, and the goal is to count the number of elements below the threshold. The important question for any given query family is what is the best possible maximum additive error we can get when answering *all* queries in the family while satisfying differential privacy. This is a challenging problem in general: In particular, for any k, there exists a family of k counting queries, such that an error of $\Omega(k)$ is necessary to satisfy ϵ-differential privacy [24, 42]. However, the trade-off can differ significantly for various families of counting queries. See [61] for a comprehensive overview of the complexity of these problems.

In many data analysis applications, the data is a database of sequential data, which we refer to as *documents* or *strings*, for example, trajectory data, DNA data, web browsing statistics, and text protocols for next-word suggestions. Since these documents contain highly confidential information, an important question is whether such collections can be analyzed while preserving differential privacy. In string algorithms research, pattern matching is one of the most fundamental problems with many applications: Given a string S and a pattern P, decide whether P exists in S. A natural way of translating this into a family of counting queries is the following: For a database that is a collection of documents $\mathcal{D} = S_1, \ldots, S_n$, and a query pattern P, how many documents in $\mathcal{D}$ contain P? This problem is called Document Count. Another closely related question is: For a database $\mathcal{D} = S_1, \ldots, S_n$, and a query pattern P, how often does P appear as a substring of a document in $\mathcal{D}$ *in total*? We call this problem Substring Count. The difference between Document Count and Substring Count is that in the latter, if a pattern P occurs multiple times in the same document in $\mathcal{D}$, we count all its occurrences, while in the former, any document in $\mathcal{D}$ can contribute at most one to the count of any pattern. In the non-private setting, these problems have been extensively studied in the data structure setting, where you preprocess the collection $\mathcal{D}$ to be able to answer queries for any pattern P, see e.g., [12, 30, 31, 33–35, 41, 50, 54–56, 58].

In this paper, we initiate a theoretical study of the query families Document Count and Substring Count under differential privacy. Specifically, we consider the problem where we want to be able to answer the above-mentioned counting queries for *all possible patterns* P while preserving differential privacy. The goal is to *minimize the additive error* while preserving differential privacy. While problems of this nature have been studied under differential privacy before from a practical perspective [11, 18, 19, 47, 48, 57, 69], to the best of our knowledge, we are the first to provide a theoretical analysis on the additive error in terms of the problem parameters. We give a new ϵ-differentially private algorithm that works for both Document Count and Substring Count and show that its additive error is tight up to poly-logarithmic factors. Additionally, we show that when relaxing to (ϵ, δ)-differential privacy, we can further reduce the error for Document Count. The differentially private algorithm processes the set of strings into a data structure, which can then be queried ad libitum without further privacy loss. Our algorithms require combining techniques from differential privacy, string algorithms, and tree data structures in a novel way. Our strategy additionally yields new results for differentially privately computing count functions on hierarchical tree data structures of certain properties, which may be of independent interest. As an example, this gives a differentially private data structure for the *colored tree counting* problem.

As an immediate application of our work, we get improved results for problems which have been extensively studied under differential privacy from a practical perspective: namely, the problem of *frequent sequential pattern mining* [11, 19, 47, 57, 69], or the very similar problem of *q-gram*

extraction [18, 48], where a q-gram is a pattern of a fixed length q. The goal in both of these problems is to analyze a collection of strings and to report patterns that either occur in many strings in the collection (i.e., have a high Document Count) or appear often as a substring in the collection in total (i.e., have a high Substring Count). However, existing papers do not give any theoretical guarantees on the accuracy of their algorithms in terms of the problem parameters. As an application of our data structure for Document Count or Substring Count, we can solve these problems by reporting all patterns with an approximate count above a given threshold. We show that the additive error achieved by our ϵ-differentially private algorithm is tight for this problem formulation, up to poly-logarithmic factors. Thus, the problem we study is a natural generalization of these problems. In the related work section, we argue that our tight bounds yield a polynomial improvement of the error compared to prior approaches.

1.1 Setup and Results

In this work, our database is a collection $\mathcal{D}$ of documents (sometimes called strings in this paper) of length at most ℓ drawn from an alphabet Σ of size $|\Sigma|$. That is, we have $\mathcal{D} = S_1, \ldots, S_n$, where $S_i \in \Sigma^{[1,\ell]}$ for all $i = 1, \ldots, n$. Thus, the *data universe* is $X = \Sigma^{[1,\ell]}$ and any database is an element of X^*[1]. The order of the documents in $\mathcal{D}$ is not important, and we can also see $\mathcal{D}$ as a *multiset*. We call two databases $\mathcal{D} \in X^*$ and $\mathcal{D}' \in X^*$ *neighboring* if there exists an $S_i' \in \Sigma^{[1,\ell]}$ such that $\mathcal{D}' = S_1, \ldots, S_{i-1}, S_i', S_{i+1}, \ldots S_n$ for some i (that is, document S_i has been replaced with document S_i'). We also denote the (symmetric) neighboring relation as $\mathcal{D} \sim \mathcal{D}'$.

Definition 1.1 (Differential Privacy). Let χ be a data universe, and $\epsilon > 0$ and $\delta \geq 0$. An algorithm $A : \chi^* \to \text{range}(A)$ is (ϵ, δ)-differentially private, if for all neighboring $\mathcal{D} \in \chi^*$ and $\mathcal{D}' \in \chi^*$ and any set $U \subseteq \text{range}(A)$, we have

$$\Pr[A(\mathcal{D}) \in U] \leq e^{\epsilon} \Pr[A(\mathcal{D}') \in U] + \delta.$$

For $\delta = 0$, the property from Definition 1.1 is also called *ϵ-differential privacy* or *pure differential privacy*. For $\delta > 0$, it is also called *approximate differential privacy*.

Document and Substring Counting. Let P and S be strings over an alphabet Σ. We define $\text{count}(P, S)$ to be the number of occurrences of P in S. If P is the empty string, we define $\text{count}(P, S) = |S|$. Let $\text{count}_\Delta(P, S) = \min(\Delta, \text{count}(P, S))$. We then define $\text{count}_\Delta(P, \mathcal{D}) := \sum_{S \in \mathcal{D}} \text{count}_\Delta(P, S)$ and $\text{count}(P, \mathcal{D}) := \text{count}_\ell(P, \mathcal{D})$. For $\Delta = 1$, this corresponds to problem Document Count, and for $\Delta = \ell$, to problem Substring Count. In general, Δ limits the contribution of any one document (corresponding to one user) to the count of any pattern in the database. Given a database $\mathcal{D} = S_1, \ldots, S_n$ we want to build a data structure that supports the following types of queries:

- Document Count(P): Return the number $\text{count}_1(P, \mathcal{D})$ of documents in $\mathcal{D}$ that contain P.
- Substring Count(P): Return $\text{count}_\ell(P, \mathcal{D})$, i.e., the total number of occurrences of P in the documents in $\mathcal{D}$.

We want to construct a data structure to efficiently answer Document Count and Substring Count queries while preserving differential privacy. For this, we need to introduce an error. We say that a data structure for count_Δ has an *additive error* α, if for any query pattern $P \in \Sigma^{[1,\ell]}$, it produces a value $\text{count}^*_\Delta(P, \mathcal{D})$ such that $|\text{count}_\Delta(P, \mathcal{D}) - \text{count}^*_\Delta(P, \mathcal{D})| \leq \alpha$.

Frequent Substring Mining. Let $\mathcal{D} = S_1, \ldots, S_n$ be a database of documents in $\Sigma^{[1,\ell]}$. We define Substring Mining($\mathcal{D}$, Δ, τ) as follows. Compute a set $\mathcal{P}$ such that for any $P \in \Sigma^{[1,\ell]}$, we have $P \in \mathcal{P}$ if and only if $\text{count}_\Delta(P, \mathcal{D}) \geq \tau$. Similarly, we define q-Gram Mining($\mathcal{D}$, Δ, τ): compute a

[1]Here "$*$" is the Kleene operator: given a set X, X^* is the (infinite) set of all finite sequences of elements in X.

set $\mathcal{P}$ such that for any $P \in \Sigma^q$, we have $P \in \mathcal{P}$ if and only if $\text{count}_\Delta(P, \mathcal{D}) \geq \tau$. That is, in q-Gram Mining we are only interested in substrings of length exactly q.

To obtain a differentially private Substring Mining algorithm we consider the approximate version of the problems. Let $\mathcal{D} = S_1, \ldots, S_n$ be a database of documents from $\Sigma^{[1,\ell]}$.

- α-Approximate Substring Mining($\mathcal{D}$, Δ, τ): Compute a set $\mathcal{P}$ such that for any $P \in \Sigma^{[1,\ell]}$, (1) if $\text{count}_\Delta(P, \mathcal{D}) \geq \tau + \alpha$, then $P \in \mathcal{P}$; and (2) if $\text{count}_\Delta(P, \mathcal{D}) \leq \tau - \alpha$, then $P \notin \mathcal{P}$.
- α-Approximate q-Gram Mining is defined in the same way for $P \in \Sigma^q$.

For the problems α-Approximate Substring Mining and α-Approximate q-Gram Mining, we also refer to α as the *error*.

1.1.1 Main results. In this paper, we give new ϵ-differentially private and (ϵ, δ)-differentially private data structures for count_Δ. We obtain new solutions to frequent substring mining as a corollary. We first state our result for ϵ-differential privacy, which gives a data structure for count_Δ with an additive error at most $O(\ell \cdot \text{polylog}(n\ell + |\Sigma|))$ with high probability.

Theorem 1.2. *Let n and ℓ be integers and Σ an alphabet of size $|\Sigma|$. Let $\Delta \leq \ell$. For any $\epsilon > 0$ and $0 < \beta < 1$, there exists an ϵ-differentially private algorithm, which can process any database $\mathcal{D} = S_1, \ldots, S_n$ of documents in $\Sigma^{[1,\ell]}$ and with probability at least $1 - \beta$ outputs a data structure for count_Δ with additive error $O\left(\epsilon^{-1}\ell \log \ell(\log^2(n\ell/\beta) + \log|\Sigma|)\right)$. The data structure can be constructed in $O(n^2\ell^4)$ time and space, stored in $O(n\ell^2)$ space, and answer queries in $O(|P|)$ time.*

For (ϵ, δ)-differential privacy, we get better logarithmic factors, and additionally improve the linear dependence on ℓ to $\sqrt{\ell\Delta}$.

Theorem 1.3. *Let n and ℓ be integers and Σ an alphabet of size $|\Sigma|$. Let $\Delta \leq \ell$. For any $\epsilon > 0$, $\delta > 0$ and $0 < \beta < 1$, there exists an (ϵ, δ)-differentially private algorithm, which can process any database $\mathcal{D} = S_1, \ldots, S_n$ of documents in $\Sigma^{[1,\ell]}$ and with probability at least $1 - \beta$ outputs a data structure for count_Δ with additive error $O\left(\epsilon^{-1}\sqrt{\ell\Delta\log(1/\delta)}\log\ell\left(\log(n\ell/\beta) + \sqrt{\log|\Sigma|\log\log\ell}\right)\right)$. The data structure can be constructed in $O(n^2\ell^4)$ time and space, stored in $O(n\ell^2)$ space, and answer queries in $O(|P|)$ time.*

Additionally, we give a simpler and more efficient algorithm for α-Approximate q-Gram Mining for fixed q, achieving the bounds of Theorem 1.4.

Theorem 1.4. *Let n, ℓ, and $q \leq \ell$ be integers and Σ an alphabet of size $|\Sigma|$. Let $\Delta \leq \ell$. For any $\epsilon > 0$, $\delta > 0$, and $0 < \beta < 1$, there exists an (ϵ, δ)-differentially private algorithm, which can process any database $\mathcal{D} = S_1, \ldots, S_n$ of documents from $\Sigma^{[1,\ell]}$ and with probability at least $1-\beta$ outputs a data structure for count_Δ for all q-grams with additive error $O\left(\epsilon^{-1}\sqrt{\ell\Delta\log(n\ell)}\log q\left(\epsilon + \log\log q + \log\frac{|\Sigma|}{\delta\beta}\right)\right)$. The data structure can be constructed in $O(n\ell(\log q + \log|\Sigma|))$ time and $O(n\ell)$ space.*

Further down, we give a lower bound which states that the bound from Theorem 1.2 is tight up to $\text{polylog}(n\ell)$ terms, even for the weaker problem formulation of α-Approximate q-Gram Mining.

1.1.2 Counting on Trees. On the way of proving our main result, we develop a new technique to differentially privately compute count functions on a tree, which may be of independent interest. We motivate this using two examples. First, let T be a tree, where every leaf corresponds to an element in the universe. The count of every leaf is the number of times the element appears in a given data set, and the count of a node is the sum of the counts of the leaves below. This problem captures any hierarchical composition of data items (i.e., by zip code, area, state), and has been studied under differential privacy (e.g. [39, 69]). In particular, as noted in [39], it can be solved via a

reduction to differentially private range counting over the leaf counts. For ϵ-differential privacy, the binary tree mechanism by Dwork et al. [27] gives an error of roughly $O(\log^2 u)$ for this problem, where u is the size of the universe. For (ϵ, δ)-differential privacy, the range counting problem has been extensively studied recently [2, 4, 13, 14, 22, 45]. As a second example, let again T be a tree and the leaves the items of the universe, however, we additionally have a *color* for every item of the universe. Now, the goal is to compute for every node the *number of distinct colors* of elements present in the data set corresponding to leaves below the node. We call this problem the *colored tree counting problem*. This is similar to the *colored range counting problem*, which is well-studied in the non-private setting [36, 37, 46], and was also recently studied with differential privacy under the name of "counting distinct elements in a time window" [40]. We give an ϵ-differentially private algorithm that can solve the colored tree counting problem with error $O(\log^2 u \log h)$, where h is the height of the tree. In general, our algorithm can compute any counting function that is (i) monotone, in the sense that any node's count cannot be larger than the sum of the counts of its children, and (ii) has bounded L_1*-sensitivity* on the leaves, i.e., the true counts of the leaves do not differ too much on neighboring data sets (see Definition 2.1 for a formal definition of sensitivity). We next state the lemma in full generality.

Theorem 1.5. *Let χ be a universe and $T = (V, E)$ a tree height h. Let $L = l_1, \ldots, l_k \subseteq V$ be the set of leaves in T. Let $c : V \times \chi^* \to \mathbb{N}$ be a function, which takes as input a node v from T and a data set $\mathcal{D}$ from χ^*, and computes a count with the following properties:*

- *$c(v, \mathcal{D}) \leq \sum_{u \text{ is a child of } v} c(u, \mathcal{D})$ for all $v \in V \setminus L$ and $\mathcal{D} \in \chi^*$;*
- *$\sum_{i=1}^{k} |c(l_i, \mathcal{D}) - c(l_i, \mathcal{D}')| \leq d$, for all neighboring $\mathcal{D}$ and $\mathcal{D}'$ and some $d \in \mathbb{N}$.*

Then for any $\epsilon > 0$ and $0 < \beta < 1$, there exists an ϵ-differentially private algorithm computing $\hat{c}(v)$ for all nodes in T such that $\max_{v \in V} |\hat{c}(v, \mathcal{D}) - c(v, \mathcal{D})| = O(\epsilon^{-1} d \log |V| \log h \log(hk/\beta))$ with probability at least $1 - \beta$. In particular, if $d = 1$, then the maximum error is at most $O(\epsilon^{-1} \log |V| \log h \log(hk/\beta))$ with probability at least $1 - \beta$.

In Appendix C, we show an improvement to Theorem 1.5 for (ϵ, δ)-differential privacy, if additionally the sensitivity of $c(v, \cdot)$ is small for each node v.

1.1.3 Lower Bound. We note that since Document Count can, in particular, be formulated as a counting query which can distinguish all elements from Σ^ℓ, an $\Omega(\log(|\Sigma|^\ell)) = \Omega(\ell \log |\Sigma|)$ lower bound for ϵ-differentially private algorithms follows from well-established lower bounds (see e.g. the lecture notes by Vadhan [61]; we discuss this in more detail in Appendix D). We extend this lower bound in two ways. (1) We show that it holds even if we only want to output which patterns have a count approximately above a given threshold τ; (2) We show that it holds even if we restrict the output to patterns of a fixed length m, for any $m \geq 2 \log \ell$.

Theorem 1.6. *Let n and ℓ be integers and Σ an alphabet of size $|\Sigma| \geq 4$. Let $m \geq 2\lceil \log \ell \rceil$. Let $\Delta \leq \ell$ and $\epsilon > 0$. Let* Alg *be an ϵ-differentially private algorithm, which takes as input any database $\mathcal{D} = S_1, \ldots, S_n$ of documents in Σ^ℓ and a threshold τ. If* Alg *computes with probability at least $2/3$ a set $\mathcal{P}$ such that for any $P \in \Sigma^m$,*

(1) if $\text{count}_\Delta(P, \mathcal{D}) \geq \tau + \alpha$, then $P \in \mathcal{P}$ and
(2) if $\text{count}_\Delta(P, \mathcal{D}) \leq \tau - \alpha$, then $P \notin \mathcal{P}$,

then $\alpha = \Omega(\min(n, \epsilon^{-1} \ell \log |\Sigma|))$.

1.2 Technical Overview

Simple approach. Before we introduce our algorithm, we will present a simple trie-based approach (a similar strategy was adopted in previous work, e.g. [11, 18, 19, 47, 48, 69]). For simplicity, we

focus on Document Count. A private trie for the data set is constructed top-down. Starting at the root, a new node is created for every letter in the alphabet, and connected by an edge labeled with the corresponding letter. Given a partial trie, each node in the trie represents the string obtained by concatenating the labels from the root to the node. Then, for a leaf in the current trie, its approximate count in the data set $\mathcal{D}$ is computed, i.e., we compute with differential privacy how many documents in $\mathcal{D}$ contain the string represented by the node. If the approximate count exceeds a certain threshold, we expand this node by adding a child for every letter in the alphabet. We continue to do so until no approximate count exceeds the threshold. The main advantage of this top-down approach is to avoid considering the entire universe by excluding patterns that are not frequent as soon as a prefix is not frequent.

To make the algorithm above ϵ-differentially private, we have to add noise to every node count, which scales approximately with the L_1-sensitivity of the computed counts, i.e., how much the true counts of the nodes in the trie can differ in total if we replace a document in the data set. Assume $\mathcal{D}' = \mathcal{D} \setminus \{S\} \cup \{S'\}$. Then, the count of a node is different on $\mathcal{D}$ and $\mathcal{D}'$ if and only if the string represented by the node is a substring of S or S'. However, S and S' can each have up to ℓ^2 different substrings. Thus, this approach yields an error of $\Omega(\ell^2)$.

Idea for improvement. However, an important insight is that while there are $2\ell^2$ nodes that can be influenced by S and S', there is a set of at most 2ℓ paths starting at the root, such that all these nodes lie on one of them: namely, the paths corresponding to the suffixes of S and S'. Our idea is to leverage this property. For this, we make use of the *heavy path decomposition* [59] of a tree, which has the property that any root-to-leaf path in a tree crosses at most a logarithmic number of heavy paths. However, we cannot construct a heavy path decomposition while using the top-down approach above, and if we were to work on the full trie of the universe, then the size of the trie would be $|\Sigma|^\ell$ (so even the logarithm is linear in ℓ).

Our algorithm. We now give an overview of our full algorithm. As a first step, we reduce the universe size from $|\Sigma|^\ell$ to $n^2\ell^3$, by computing a set of *candidate frequent strings* in a differentially private way. To achieve this, we approximate the counts of strings whose length is a power of two and keep only those that appear at least once, with high probability. Note that a document S can have at most ℓ substrings of any given length, and we consider a logarithmic number of lengths. We then use the following observation: any pattern of a given length m which is not a power of two has an overlapping prefix and suffix whose length is a power of two. Thus, our candidate set consists of all strings covered by a power-of-two-length prefix and suffix appearing sufficiently often with high probability. From this set, we build a trie and decompose it into heavy paths. For every heavy path, we compute (i) a differentially private approximation of the count of its root, and (ii) a differentially private estimate of the *prefix sums of the difference sequence of the counts* on the path going down. That is, for every node on the heavy path, we get an approximation of the *difference* between its count and the count of the root. We can compute (ii) with low noise using a generalized variant of the binary tree mechanism [27]. We show that since any root-to-leaf path can cross at most $\log(n^2\ell^3) = O(\log(n\ell))$ heavy paths, and a document S can only influence counts on the ℓ paths given by its suffixes, this can be done with an error which is $O(\ell \cdot \text{polylog}(n\ell))$. This strategy gives our main result (Theorem 1.2).

The general tree counting lemma (Theorem 1.5) is obtained by building a heavy path decomposition on the given tree, and performing steps (i) and (ii). The improvement for (ϵ, δ)-differential privacy comes from the fact that if the L_1-norm of a vector is bounded by L, and the L_∞-norm is bounded by Δ, then by Hölder's inequality, the L_2-norm is bounded by $\sqrt{L\Delta}$. We use this fact several times in the analysis to improve the linear dependency on ℓ to a dependency on $\sqrt{\ell\Delta}$.

The improvement for q-grams stems 1) from the fact that since we only want to compute the counts for patterns of a fixed length, we do not need to compute the full set of candidate strings, and can significantly simplify the algorithm and 2) for (ϵ, δ)-differential privacy, we show that we can avoid computing noisy counts for patterns whose true count is 0, without violating privacy.

1.3 Related Work

Private sequential pattern mining and q-gram extraction. As mentioned above, two very related well-studied problems are those of *frequent sequential pattern mining* [11, 19, 47, 57, 69] and *q-gram extraction* [18, 48]. We also mentioned that these works do not provide theoretical guarantees on the error. We now describe some of the relevant solutions in more detail and argue why our error bounds yield an improvement over previous approaches. In the following, by "error" we either mean the additive error on the counts, or the value of α in the definition of α-Approximate Substring Mining and α-Approximate q-Gram Mining, depending on which problem the paper is considering.

Several papers use a variant of the trie-based approach described in the technical overview [11, 18, 19, 47, 48, 69]. Some of them [11, 19] use a *prefix trie*, i.e., they build a trie of strings that appear frequently as a prefix of a string in the data set. While this can be used for estimating frequent elements for large universes, it does not solve the problem of counting q-grams or substrings, since it is not possible to reconstruct frequent substrings from frequent prefixes (e.g. there could be a frequent substring that is not the suffix of any frequent prefix). Khatri et al. [47] also construct a tree in an online manner (however, appending new letters *in front* instead of at the end), and compute the count of how often a string in the trie appears as a *suffix* as a counting measure, which leads to a similar issue when trying to estimate substring counts. Zhang et al. [69] consider the problem of computing tree counts, where every leaf is an element of the universe, and the count of every node is the number of leaves below which are in the data set. As a first step, they build a pruned tree such that every element in the tree has at least a certain count. As a second step, they compute a differentially private count of every leaf. The count of any node is computed as the sum of the counts of the leaves below. However, when computing the count of a node from the noisy counts of the leaves, the noise values sum up, so this can yield large errors for internal nodes.

Chen et al. [18] and Kim et al. [48] consider the problem of computing frequent q-grams and use a trie-based approach very similar to the simple approach described in the technical overview. While they use different heuristics and parameter tuning to improve accuracy in practice, the worst-case theoretical bound on the error remains $\Omega(\ell^2)$. We note that Kim et al. [48], when computing the frequent q-grams, use a candidate set consisting of any frequent $(q-1)$-gram appended by any frequent 1-gram, and its intersection with the set consisting of any frequent 1-gram appended by any frequent $(q-1)$-gram. We use a similar idea in the pruning process in the first step of our algorithm, except that we *double* the length of the q-grams at every step.

Prado et al. [57] use a graph-based approach for finding frequent patterns. In their work, a bi-partite graph is produced linking users to (frequent) patterns that appear in the user's document; this graph is then privatized using an edge-differentially private algorithm. Note that this algorithm satisfies a weaker privacy guarantee, as one user string can be connected to several patterns.

Further related work. Similar problems which have been extensively studied are frequent itemset mining [20, 23, 64, 65, 68], where the data set consists of a set of items for each person and the goal is to find frequent subsets, and frequent sequence mining [21, 51–53, 66, 70], where the goal is to find frequent *subsequences*, instead of frequent *substrings*. Both of these problems are different from those considered in this work since they do not require a pattern to occur as a consecutive substring. Another related line of work is differentially private pattern counting on a *single* string with different definitions of neighboring (and therefore different privacy guarantees) [17, 32, 60].

The problems of frequent string mining and frequent sequence mining have also been studied in the local model of differential privacy [15, 62, 63].

1.4 Organization of Technical Sections

In the main part of this paper, we focus on ϵ-differential privacy. In Section 2, we collect some background on string data structures and differential privacy. In Section 3, we prove Theorem 1.2, and in Section 3.4, we give a slightly more efficient (and much simpler) algorithm for computing only q-grams with ϵ-differential privacy, for a fixed length q. In Section 4 we give the proof of Theorem 1.5 (counting on trees). The algorithms for approximate differential privacy essentially follow the same steps, so due to lack of space this part has been deferred to Appendix B and C. In Section 5, we describe the fast algorithm for q-grams with approximate differential privacy. Finally, in Appendix D, we prove the lower bound of Theorem 1.6.

2 Preliminaries

In this work, we use log to denote the binary logarithm and ln to denote the natural logarithm. We use $[a, b]$ to denote the interval of integers $\{a, a+1, a+2, \ldots, b-1, b\}$.

String Preliminaries. A string S of length $|S| = \ell$ is a sequence $S[0] \cdots S[\ell-1]$ of ℓ characters drawn from an alphabet Σ of size $|\Sigma|$. The string $S[i] \cdots S[j]$, denoted $S\,[i,\ j]$, is called a *substring* of S; $S\,[0,\ j-1]$ and $S\,[i,\ \ell-1]$ are called the j^{th} *prefix* and i^{th} *suffix* of S, respectively. Let P and S be strings over an alphabet Σ. We say that P *occurs* in S iff there exists an i such that $S\,[i,\ i+|P|-1] = P$. We use $\Sigma^{[a,b]}$ to denote all strings S which satisfy $a \le |S| \le b$. The concatenation of two strings A, B is defined as $A \cdot B = A[0] \cdots A[|A|-1]B[0] \cdots B[|B|-1]$ and is sometimes simply denoted by AB. Given any two sets of strings $\mathcal{A}, \mathcal{B}$, we define $\mathcal{A} \circ \mathcal{B} = \{A \cdot B \mid A \in \mathcal{A}, B \in \mathcal{B}\}$.

A *trie* for a collection of strings $\mathcal{C} = S_1, \ldots, S_n$, denoted $T_{\mathcal{C}}$, is a rooted labeled tree, such that: (1) The label on each edge is a character of one or more S_i. (2) Each string in $\mathcal{C}$ is represented by a path in $T_{\mathcal{C}}$ going from the root down to some node (obtained by concatenating the labels on the edges of the path). (3) Each root-to-leaf path represents a string from $\mathcal{C}$. (4) Common prefixes of two strings share the same path maximally. For a node $v \in T_{\mathcal{C}}$, we let $\mathrm{str}(v)$ denote the string obtained from concatenating the labels on the path's edges from the root to v.

A *compacted trie* is obtained from $T_{\mathcal{C}}$ by dissolving all nodes except the root, the branching nodes, and the leaves, and concatenating the labels on the edges incident to dissolved nodes to obtain *string* labels for the remaining edges. The *string depth* $\mathrm{sd}(v) = |\mathrm{str}(v)|$ of any branching node v is the total length of the strings labeling the path from the root to v. The *frequency* $f(v)$ of node v is the number of leaves in the subtree rooted at v. Throughout the paper, we assume that $f(v)$ and $\mathrm{sd}(v)$ values are stored in each branching node v. This can be obtained in $O(|\mathcal{C}|)$ time with a single trie traversal using $O(|\mathcal{C}|)$ total extra space. The compacted trie can be represented in $O(|\mathcal{C}|)$ space and computed in $O(|\mathcal{C}|)$ time [49].

Let S be a string over an alphabet Σ. The *suffix tree* of a string S, denoted by $\mathrm{ST}(S)$, is the compacted trie of the set of all suffixes of S. Each leaf of $\mathrm{ST}(S)$ is identified by the starting position in S of the suffix it represents. We can construct $\mathrm{ST}(S)$ in $O(\mathrm{sort}(|S|, |\Sigma|)$, where $\mathrm{sort}(x, y)$ is the time to sort $|S|$ integers from an universe of size $|\Sigma|$ [28, 29]. For instance, for any polynomial-size alphabet $\Sigma = [0, |S|^{O(1)}]$ we can construct $\mathrm{ST}(S)$ in $O(|S|)$ time.

Privacy Preliminaries. We collect some important definitions and results here. See Appendix A for additional privacy background. First, we define the notion of sensitivity.

Definition 2.1 (L_p-sensitivity). Let f be a function $f : \chi^* \to \mathbb{R}^k$ for some universe χ. The *L_p-sensitivity of f* is defined as $\max_{\mathcal{D} \sim \mathcal{D}'} ||f(\mathcal{D}) - f(\mathcal{D}')||_p$.

Definition 2.2. The *Laplace distribution* centered at 0 with scale b is the distribution with probability density function $f_{\mathrm{Lap}(b)}(x) = \frac{1}{2b}\exp\left(\frac{-|x|}{b}\right)$. We use $Y \approx \mathrm{Lap}(b)$ or just $\mathrm{Lap}(b)$ to denote a random variable Y distributed according to $f_{\mathrm{Lap}(b)}(x)$.

The following corollary follows from the *Laplace Mechanism* [26] and a Laplace tailbound (see Appendix A, Lemmas A.1 and A.2 for details):

Corollary 2.3. *Let f be a function $f : \chi^* \to \mathbb{R}^k$ for some universe χ with L_1-sensitvity at most Δ_1. Then there exists an ϵ-differentially private algorithm A which for any $\mathcal{D} \in \chi^*$ outputs $A(\mathcal{D})$ satisfying $||A(\mathcal{D}) - f(\mathcal{D})||_\infty \leq \epsilon^{-1}\Delta_1 \ln(k/\beta)$ with probability at least $1-\beta$.*

Lemma 2.4 (Simple Composition [25]). *Let A_1 be an (ϵ_1, δ_1)-differentially private algorithm $\chi^* \to \mathrm{range}(A_1)$ and A_2 an (ϵ_2, δ_2)-differentially private algorithm $\chi^* \times \mathrm{range}(A_1) \to \mathrm{range}(A_2)$. Then $A_1 \circ A_2$ is $(\epsilon_1+\epsilon_2, \delta_1+\delta_2)$-differentially private.*

3 Counting with Pure Differential Privacy

In this section, we prove Theorem 1.2. For simplicity, we prove the theorem for $\Delta = \ell$. Since $\mathrm{count}_\Delta(P,S) \leq \mathrm{count}(P,S)$ holds for all $\Delta \leq \ell$ and all strings $S \in \Sigma^{[1,n]}$, the same proof also works for $\Delta < \ell$. First, we make a simple, but fundamental observation, which we will use repeatedly.

Lemma 3.1. *For any $S \in \mathcal{D}$ and any $m \leq \ell$, we have $\sum_{P\in\Sigma^m} \mathrm{count}(P,S) \leq \ell$.*

Proof. Since ℓ is the length of S, there are $\ell - m + 1$ possible starting positions in S of substrings of length m, and thus there are exactly $\ell - m + 1$ substrings of length m of S. Therefore the total count is never more than $\ell - m + 1 \leq \ell$. □

Corollary 3.2. *For any $m \leq \ell$, the L_1-sensitivity of $(\mathrm{count}(P,\mathcal{D}))_{P\in\Sigma^m}$ is bounded by 2ℓ.*

Algorithm Overview. Our algorithm underlying Theorem 1.2 consists of four main steps.

Step 1. We run an ϵ-differentially private algorithm which computes a set C of *candidate frequent* strings such that any string not in C has a small count in $\mathcal{D}$ and the set C is not too large (Lemma 3.3).

Step 2. We build the trie T_C of the candidate set C, and construct a *heavy path decomposition* of T_C (Lemma 3.5).

Step 3. For every node r which is the root of a heavy path, we compute a differentially private estimate of $\mathrm{count}(\mathrm{str}(r), \mathcal{D})$ (Corollary 3.8).

Step 4. For every heavy path $p = v_0, v_1, \ldots, v_{|p|-1}$ with root $r = v_0$, we consider the *difference sequence* of counts along the path: i.e., for a node v_i on the path and its parent v_{i-1}, the ith element in the difference sequence is given by $\mathrm{count}(\mathrm{str}(v_i), \mathcal{D}) - \mathrm{count}(\mathrm{str}(v_{i-1}), \mathcal{D})$, for $i = 1, \ldots, |p|-1$. We use the binary tree mechanism to compute a differentially private estimate of the *prefix sum* of the difference sequence (Corollary 3.10).

After these four steps, we can estimate the count of every node v in T_C: If v_i is the ith node on a heavy path $v_0, v_1, \ldots, v_{|p|-1}$, its count can be computed as $\mathrm{count}(\mathrm{str}(v_i), \mathcal{D}) = \mathrm{count}(\mathrm{str}(v_0), \mathcal{D}) + \sum_{j=1}^{i}\left(\mathrm{count}(\mathrm{str}(v_j), \mathcal{D}) - \mathrm{count}(\mathrm{str}(v_{j-1}), \mathcal{D})\right)$. Thus, we can use the results from Step 3 and Step 4 to estimate the counts for every string in C. In the rest of the section, we detail this approach.

3.1 Computing a Candidate Set

As a first step, we show how to compute the candidate set C while satisfying ϵ-differential privacy.

Lemma 3.3. *Let $\mathcal{D} = S_1, \ldots, S_n$ be a database of documents. For any $\epsilon > 0$ and $0 < \beta < 1$ there exists an ϵ-differentially private algorithm, which computes a candidate set $C \subseteq \Sigma^{[1,\ell]}$ enjoying the following two properties with probability at least $1-\beta$:*

- *For any* $P \in \Sigma^{[1,\ell]}$ *not in* C, $\mathrm{count}(P, \mathcal{D}) = O(\epsilon^{-1}\ell \log \ell \log(\max\{\ell^2 n^2, |\Sigma|\}/\beta))$,
- $|C| \leq n^2\ell^3$.

Proof. First, we inductively construct sets $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$, with $j = \lfloor \log \ell \rfloor$, where $\mathcal{P}_{2^k}$ contains only strings of length 2^k which appear sufficiently often as substrings in $\mathcal{D}$. Let $\epsilon_1 = \epsilon/(\lfloor \log \ell \rfloor + 1)$. For every $k = 1, \ldots, \lfloor \log \ell \rfloor$, we use an ϵ_1-differentially private algorithm to construct $\mathcal{P}_{2^k}$ from $\mathcal{P}_{2^{k-1}}$, such that the full algorithm fulfills ϵ-differential privacy. In a post-processing step, we construct sets C_m of strings of length m for every $1 \leq m \leq \ell$, such that every pattern of length m which is not in C_m has a small count with high probability. We define C as the union of all C_m.

Computing $\mathcal{P}_{2^0}$. First, we estimate $\mathrm{count}(\gamma, \mathcal{D})$ of all letters $\gamma \in \Sigma$. By Corollary 3.2, the sensitivity of $(\mathrm{count}(\gamma, \mathcal{D}))_{\gamma \in \Sigma}$ is bounded by 2ℓ. Let $\beta_1 = \beta/(\lfloor \log \ell \rfloor + 1)$. Using the algorithm given by Corollary 2.3, we compute an estimate $\mathrm{count}^*(\gamma, \mathcal{D})$ such that

$$\max_{\gamma \in \Sigma} |\mathrm{count}^*(\gamma, \mathcal{D}) - \mathrm{count}(\gamma, \mathcal{D})| \leq \frac{2\ell}{\epsilon_1} \ln(|\Sigma|/\beta_1) \leq \frac{2\ell}{\epsilon_1} \ln\left(\max\{\ell^2 n^2, |\Sigma|\}/\beta_1\right)$$

with probability at least $1 - \beta_1$, while satisfying ϵ_1-differential privacy. In the following, we let $\alpha = \frac{2\ell}{\epsilon_1} \ln(\max\{\ell^2 n^2, |\Sigma|\}/\beta_1)$. We keep a pruned candidate set $\mathcal{P}_{2^0}$ of strings of length 1 with an approximate count at least $\tau = 2\alpha$, i.e., $\mathcal{P}_{2^0}$ consists of all γ with $\mathrm{count}^*(\gamma, \mathcal{D}) \geq \tau$. If $|\mathcal{P}_{2^0}| > n\ell$, we stop the algorithm and return a fail message.

Computing $\mathcal{P}_{2^k}$ for $k = 1, \ldots, \lfloor \log \ell \rfloor$. Given $\mathcal{P}_{2^{k-1}}$ with $|\mathcal{P}_{2^{k-1}}| \leq n\ell$ and $k \geq 1$, we compute $\mathcal{P}_{2^k}$ as follows: first, we construct the set $\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$, i.e., all strings of length 2^k that are a concatenation of two strings in $\mathcal{P}_{2^{k-1}}$. There are at most $(n\ell)^2$ of these. Again by Corollary 3.2, the L_1-sensitivity of count for all strings in $\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$ is bounded by 2ℓ. We use the algorithm from Corollary 2.3 to estimate the counts of all strings in $\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$ up to an additive error at most $\frac{2\ell}{\epsilon_1} \ln(\ell^2 n^2/\beta_1) \leq \alpha$ with probability at least $1 - \beta_1$ and ϵ_1-differential privacy. The set $\mathcal{P}_{2^k}$ is the set of all strings in $\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$ with an approximate count at least τ. Again, if $|\mathcal{P}_{2^k}| > n\ell$, we stop the algorithm and return a fail message.

Constructing C. From $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$, we now construct, for each m which is not a power of two, a set of candidate patterns C_m, without taking $\mathcal{D}$ further into account. Specifically, for any fixed length m with $2^k < m < 2^{k+1}$, for $k = 0, \ldots, \lfloor \log \ell \rfloor$, we define C_m as the set of all strings P of length m such that $P[0, 2^k - 1] \in \mathcal{P}_{2^k}$ and $P[m - 2^k, m - 1] \in \mathcal{P}_{2^k}$. For $m = 2^k$ for some $k \in \{0, \ldots, \lfloor \log \ell \rfloor\}$, we define $C_m = \mathcal{P}_{2^k}$. We finally define $C = \bigcup_{m=1}^{\ell} C_m$.

Privacy. Since there are $\lfloor \log \ell \rfloor + 1$ choices of k, and for each we run an $\epsilon_1 = \epsilon/(\lfloor \log \ell \rfloor + 1)$-differentially private algorithm, their composition is ϵ-differentially private by Lemma 2.4. Since constructing C from $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^k}$ is post-processing, the entire algorithm is ϵ-differentially private.

Accuracy. Since there are $\lfloor \log \ell \rfloor + 1$ choices of k, by the choice of β_1, and by the union bound, all the error bounds hold together with probability at least $1 - \beta$. Let E be the event that all the error bounds holding simultaneously. In the following, we conditioned on E. Conditioned on E, for any $P \in \mathcal{P}_{2^k}$, $k = 0, \ldots, \lfloor \log \ell \rfloor$, we have that $\mathrm{count}(P, \mathcal{D}) \geq \tau - \alpha \geq \alpha > 1$, i.e., P appears at least once as a substring in $\mathcal{D}$. Note that any string in $\mathcal{D}$ has at most ℓ substrings of length 2^k. Since there are n strings in $\mathcal{D}$, we have $|\mathcal{P}_{2^k}| \leq n\ell$, conditioned on E. Thus, conditioning on E, the algorithm does not abort. Additionally, any P of length 2^k which is not in $\mathcal{P}_{2^k}$ satisfies $\mathrm{count}(P, \mathcal{D}) < \tau + \alpha = 3\alpha$. Now, any pattern P of length $2^k < m < 2^{k+1}$ for some $k \in \{0, \ldots, \lfloor \log \ell \rfloor\}$ satisfies that $\mathrm{count}(P, \mathcal{D}) \leq \mathrm{count}(P[0, 2^k - 1], \mathcal{D})$ and $\mathrm{count}(P, \mathcal{D}) \leq \mathrm{count}(P[m - 2^k, m - 1], \mathcal{D})$. Since $P \notin C_m$ if and only if either $P[0, 2^k - 1] \notin \mathcal{P}_{2^k}$ or $P[m - 2^k, m - 1] \notin \mathcal{P}_{2^k}$, we have that $P \notin C_m$ implies $\mathrm{count}(P, \mathcal{D}) < 3\alpha$, conditioned on E. As $P[0, 2^k - 1]$ and $P[m - 2^k, m - 1]$ cover P completely, we have $|C_m| \leq |\mathcal{P}_{2^k}|^2 \leq (n\ell)^2$. Therefore, $|C| = \sum_{m=1}^{\ell} |C_m| \leq n^2\ell^3$. □

The following lemma provides an algorithm to compute C in polynomial time and space.

Lemma 3.4. *Given a database of n documents $\mathcal{D} = S_1, \ldots, S_n$ over $\Sigma^{[1,\ell]}$, a set C satisfying the properties of Lemma 3.3 can be computed in time $O\left(n^2\ell^3 \log\log(n\ell) + n^2\ell^4\right)$ and space $O(n^2\ell^4)$.*

Proof. In a preprocessing step, we construct the suffix tree ST of the string $S = S_1\$_1S_2\$_2 \ldots S_n\$_n$, where $\mathcal{D} = \{S_1, \ldots, S_n\}$ and $\$_1, \ldots, \$_n \notin \Sigma$, and store, within each branching node v, the ID leaf(v) of the leftmost descending leaf. We also construct a substring concatenation data structure over S. A substring concatenation query consists of four integers i_1, i_2, j_1, j_2. If $S[i_1, j_1] \circ S[i_2, j_2]$ is a substring of S it returns a pair of indices i, j such that $S[i, j] = S[i_1, j_1] \circ S[i_2, j_2]$, together with a pointer to the shallowest (i.e., closest to the root) branching node v of ST such that $S[i, j]$ is a prefix of str(v); it returns a NIL pointer otherwise. Such a data structure can be constructed in $O(n\ell)$ time and space, and answers queries in $O(\log\log(n\ell))$ time [8, 9].

Main algorithm. The algorithm proceeds in phases for increasing values of $k = 0, 1, \ldots, \lfloor \log \ell \rfloor$. In Phase $k = 0$, it computes a noisy counter c^*_γ for each letter $\gamma \in \Sigma$, and stores in $\mathcal{P}_{2^0}$ all and only the letters such that $c^*_\gamma \geq \tau$. To do so, it first computes the true frequency of each letter by traversing the suffix tree ST: the frequency of γ is given by $f(v_\gamma)$, where v_γ is the shallowest branching node of ST such that str(v_γ) starts with γ (the frequency is 0 if no such node exists, i.e., γ does not occur in the database $\mathcal{D}$). Noisy counts c^*_γ are then obtained as in Lemma 3.3. $\mathcal{P}_{2^0}$ is represented by a list of pairs $\langle \gamma, p \rangle$, where p is a pointer to v_γ ($p =$ NIL if v_γ does not exist).

In any phase $k > 0$, the algorithm performs two steps: **(1)**, compute a noisy counter c^*_Q for every string $Q \in \mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$ and store a representation of $\mathcal{P}_{2^k}$ consisting of all and only the strings Q such that $c^*_Q \geq \tau$ (abort the procedure and return FAIL if $|\mathcal{P}_{2^k}| > n\ell$); **(2)**, construct sets C_m for every $2^k < m < 2^{k+1}$. We now describe each of these steps in more detail.

Step 1. $\mathcal{P}_{2^k}$ is represented as a list of pairs $\langle Q, p \rangle$, where Q is a string of length 2^k and p is a pointer to the shallowest node v of ST such that Q is a prefix of str(v) (or $p =$ NIL if no such v exists). Consider all ordered pairs of items from the $\mathcal{P}_{2^{k-1}}$ list. For each such pair $(\langle Q_1, p_1 \rangle, \langle Q_2, p_2 \rangle)$, if $p_1, p_2 \neq$ NIL, let v_1, v_2 be the nodes of ST p_1 and p_2 point to, respectively. We ask a concatenation query with $i_1 =$ leaf(v_1), $i_2 =$ leaf(v_2), $j_1 = i_1 + 2^{k-1} - 1$, $j_2 = i_2 + 2^{k-1} - 1$; if the result is a pointer p to a node v of ST, we compute $c^*_{Q_1Q_2}$ by adding noise to $f(v)$ as in Lemma 3.3, append pair $\langle Q_1Q_2, p \rangle$ to the list of $\mathcal{P}_{2^k}$ if $c^*_{Q_1Q_2} \geq \tau$, or discard it and move on to the next pair of elements from $\mathcal{P}_{2^{k-1}}$ otherwise. In all the other cases, that is, if either $p_1 =$ NIL or $p_2 =$ NIL or the result of the concatenation query is $p =$ NIL, the true frequency of $Q_1 \cdot Q_2$ in $\mathcal{D}$ is 0, thus we compute $c^*_{Q_1Q_2}$ by adding noise to 0 and append $\langle Q_1Q_2, p \rangle$ to the list of $\mathcal{P}_{2^k}$ only if $c^*_{Q_1Q_2} \geq \tau$. We stop the procedure and return FAIL whenever the number of triplets in $\mathcal{P}_{2^k}$ exceeds $n\ell$.

Step 2. For any $2^k < m < 2^{k+1}$, set C_m consists of all strings Q of length m whose prefix $Q_1 = Q[0, 2^k - 1]$ and suffix $Q_2 = Q[m - 2^k, m - 1]$ are both in $\mathcal{P}_{2^k}$. Note that, since $m < 2^{k+1}$, this implies that Q_1 and Q_2 have a suffix/prefix overlap of length $2^{k+1} - m$, or in other words, $Q_1[m - 2^k, 2^k - 1]$ and $Q_2[0, 2^k - 1]$ have a common prefix of length $2^{k+1} - m$. The construction of C_m thus reduces to finding all pairs of strings from $\mathcal{P}_{2^k}$ with a suffix/prefix overlap of length $2^{k+1} - m$. To do this efficiently for all m, we preprocess $\mathcal{P}_{2^k}$ to build in $O(\text{sort}(2^k n\ell, |\Sigma|))$ time a data structure that occupies $O(2^k n\ell)$ space and answers *Longest Common Extension* (LCE) queries in $O(1)$ time [6, 29, 43] (since $2^k n\ell$ upper bounds the total length of the strings in $\mathcal{P}_{2^k}$). An LCE query consists of a pair of strings $Q_1, Q_2 \in \mathcal{P}_{2^k}$ and two positions $q_1, q_2 < 2^k$; the answer $\text{LCE}_{Q_1,Q_2}(q_1, q_2)$ is the length of the longest common prefix of the suffixes $Q_1[q_1, 2^k - 1]$ and $Q_2[q_2, 2^k - 1]$.

Once the LCE data structure is constructed, consider all ordered pairs of items from the list of $\mathcal{P}_{2^k}$. For each pair $(\langle Q_1, p_1 \rangle, \langle Q_2, p_2 \rangle)$ and each $2^k < m < 2^{k+1}$, compute $\text{LCE}_{Q_1,Q_2}(m - 2^k, 0)$: if the result is $2^{k+1} - m$, then Q_1 and Q_2 have a suffix-prefix overlap of length $2^{k+1} - m$, thus the concatenation of Q_1 and the suffix of Q_2 immediately following the overlap, that is $Q = Q_1[0, 2^k - 1] \cdot Q_2[2^{k+1} - m, 2^k - 1]$,

belongs to C_m. Once again, we represent Q_m with a list of pairs $\langle Q, p\rangle$, where p is a pointer to the shallowest node v of ST such that Q is a prefix of $\mathrm{str}(v)$, with $p = \mathsf{NIL}$ if Q does not appear in $\mathcal{D}$ (thus no such node v exists). The pointer p for a string $Q = Q_1[0, 2^k - 1] \cdot Q_2[2^{k+1} - m, 2^k - 1]$ is obtained as the result of the concatenation query with $i_1 = \mathrm{leaf}(v_1)$, $i_2 = \mathrm{leaf}(v_2) + 2^{k+1} - m$, $j_1 = i_1 + 2^k - 1$, $j_2 = i_2 + 2^k - 1$; $p = \mathsf{NIL}$ if either $p_1 = \mathsf{NIL}$ or $p_2 = \mathsf{NIL}$.

Time and space analysis. Consider Step 1 and focus first on the space occupancy. By construction, at any phase $k = 0, \ldots, \lfloor \log \ell \rfloor$, $\mathcal{P}_{2^k}$ is represented by a list of $O(n\ell)$ pairs, each consisting of a string of length 2^k and a pointer. Storing the list in Phase k thus requires space $O(2^k n\ell)$, implying a total space $O(n\ell^2)$ to represent all the lists. As for the running time, in each phase $k = 1, \ldots, \lfloor \log \ell \rfloor$ we consider $O(n^2\ell^2)$ pairs of elements from $\mathcal{P}_{2^{k-1}}$; for each pair, we ask one concatenation query, requiring $O(\log\log(n\ell))$ time (as to ask the query we do not need to read the string of the pair), compute a noisy counter in $O(1)$ time, and write a new pair consisting of a string of length 2^{k+1} and a pointer if the counter exceeds τ. Since we abort the procedure if the number of written pairs exceeds $n\ell$, the total writing time in Phase k is $O(2^{k+1}n\ell)$, thus $O\left(\sum_{k=0}^{\lfloor \log \ell \rfloor} 2^{k+1} n\ell\right) = O(n\ell^2)$ over all phases. Therefore, the total time for Step 1 is dominated by the concatenation queries, and summing over the $\lfloor \log \ell \rfloor + 1$ phases it is $O(n^2\ell^2 \log\log(n\ell) \log \ell)$.

Let us now focus on Step 2, and consider first the space occupancy. For each phase $k = 0, \ldots, \lfloor \log \ell \rfloor$, we construct an LCE data structure that occupies $O(2^k n\ell)$ space; however, at the end of Phase k, this data structure can be discarded, thus the space it occupies at any point of the algorithm is $O\left(n\ell^2\right)$. For each $2^k < m < 2^{k+1}$, we represent C_m with a list of up to $n^2\ell^2$ pairs consisting of one string of size m and one pointer, thus using space $O\left(mn^2\ell^2\right)$. The total space to store all lists is thus $O\left(\sum_{m=1}^{\ell} mn^2\ell^2\right) = O(n^2\ell^4)$, which dominates the space occupancy of the algorithm. The running time of Step 2 is as follows. For each k, we construct the LCE data structure for $\mathcal{P}_{2^k}$ in time $O\left(\mathrm{sort}(2^k n\ell, |\Sigma|)\right)$, thus the total time for all phases is $O\left(\sum_{k=0}^{\lfloor \log \ell \rfloor} \left(\mathrm{sort}(2^k n\ell, |\Sigma|)\right)\right) = O\left(\mathrm{sort}(n\ell^2, |\Sigma|)\right)$. To compute the list representation of C_m, we ask $O\left(n^2\ell^2\right)$ LCE and concatenation queries in total $O\left(n^2\ell^2 \log\log(n\ell)\right)$ time; and we use $O\left(mn^2\ell^2\right)$ time to write all the list pairs. Summing over all $m = 1, \ldots, \ell$, we obtain a total time $O\left(n^2\ell^4\right)$ to write all the lists and $O\left(n^2\ell^3 \log\log(n\ell)\right)$ total time for the LCE and concatenation queries. The total running time for Step 2 is thus $O\left(\mathrm{sort}(n\ell^2, |\Sigma|) + n^2\ell^3 \log\log(n\ell) + n^2\ell^4\right) = O\left(n^2\ell^3 \log\log(n\ell) + n^2\ell^4\right)$. □

The next step towards Theorem 1.2 is to arrange C in a trie and compute its heavy path decomposition, as we explain in Section 3.2.

3.2 Heavy Path Decomposition and Properties

In the following, let T_C be the trie of a set C fulfilling the properties of Lemma 3.3. Note that the number of nodes in T_C is bounded by the total length of strings in C, which is $n^2\ell^4$. In the following, let $|T_C|$ denote the number of nodes in T_C. A *heavy path decomposition* of a tree T is defined as follows. Every edge is either *light* or *heavy*. There is exactly one heavy edge outgoing from every node except the leaves, defined as the edge to the child with the largest subtree (i.e., the subtree containing most nodes). Ties are broken arbitrarily. The longest *heavy path* is obtained by following the heavy edges from the root of T to a leaf: note that all edges hanging off this path are light. The other heavy paths are obtained by recursively decomposing all subtrees hanging off a heavy path. We call the topmost node of a heavy path (i.e., the only node of the path which is not reached by a heavy edge) its *root*. The heavy path decomposition of a tree with N nodes can be constructed in $O(N)$ time and has the following important property:

Lemma 3.5 (Sleator and Tarjan [59]). *Given a tree T with N nodes and a heavy path decomposition of T, any root-to-leaf path in T contains at most $\lceil \log N \rceil$ light edges.*

Difference sequence. For any path $p = v_0, v_1, \ldots, v_{|p|-1}$ in the trie T_C, the *difference sequence of* count *on* p is the $|p|-1$ dimensional vector $\mathrm{diff}_p(\mathcal{D})[i] = \mathrm{count}(\mathrm{str}(v_i), \mathcal{D}) - \mathrm{count}(\mathrm{str}(v_{i-1}), \mathcal{D})$ for $i = 1 \ldots |p|-1$. In the following, we collect some useful lemmas about the sensitivity of the counts of the roots and the difference sequences on the heavy paths.

Lemma 3.6. *Let $\mathcal{D}$ and $\mathcal{D}'$ be neighboring data sets, such that $\mathcal{D}' = \mathcal{D} \setminus \{S\} \cup \{S'\}$.*

(1) For any node v in the trie T_C, $|\mathrm{count}(\mathrm{str}(v), \mathcal{D}) - \mathrm{count}(\mathrm{str}(v), \mathcal{D}')| = |\mathrm{count}(\mathrm{str}(v), S) - \mathrm{count}(\mathrm{str}(v), S')|$.

(2) For any path p with root r in T_C, $||\mathrm{diff}_p(\mathcal{D}) - \mathrm{diff}_p(\mathcal{D}')||_1 \leq \mathrm{count}(\mathrm{str}(r), S) + \mathrm{count}(\mathrm{str}(r), S')$.

Proof. Item 1 follows from the definition of count. For item 2 we have

$$
\begin{aligned}
&||\mathrm{diff}_p(\mathcal{D}) - \mathrm{diff}_p(\mathcal{D}')||_1 \\
&= \sum_{i=1}^{|p|-1} |\mathrm{count}(\mathrm{str}(v_i), S) - \mathrm{count}(\mathrm{str}(v_{i-1}), S) - (\mathrm{count}(\mathrm{str}(v_i), S') - \mathrm{count}(\mathrm{str}(v_{i-1}), S'))| \\
&\leq \sum_{i=1}^{|p|-1} |\mathrm{count}(\mathrm{str}(v_i), S) - \mathrm{count}(\mathrm{str}(v_{i-1}), S)| + \sum_{i=1}^{|p|-1} |\mathrm{count}(\mathrm{str}(v_i), S') - \mathrm{count}(\mathrm{str}(v_{i-1}), S')|.
\end{aligned}
$$

Note that if v' is a descendant of v, then $\mathrm{str}(v)$ is a prefix of $\mathrm{str}(v')$ and therefore $\mathrm{count}(\mathrm{str}(v), S) \geq \mathrm{count}(\mathrm{str}(v'), S)$. Since $\mathrm{count}(\mathrm{str}(v_i), S)$ is monotonically non-increasing in i for $p = v_0, v_1, \ldots, v_{|p|-1}$ with $v_0 = r$, from $0 \leq \mathrm{count}(\mathrm{str}(v_i), S) \leq \mathrm{count}(\mathrm{str}(r), S)$, for any descendant v_i of r, it follows that $\sum_{i=1}^{|p|-1} |\mathrm{count}(\mathrm{str}(v_i), S) - \mathrm{count}(\mathrm{str}(v_{i-1}), S)| \leq \mathrm{count}(\mathrm{str}(r), S)$. The same argument applies for S' and thus item 2 follows. □

Lemma 3.7. *Let $r_0, r_1, \ldots, r_k$ be the roots of the paths of the heavy path decomposition of T_C, where r_0 is the root of T_C. Then for any string S of length at most ℓ, we have $\sum_{i=0}^{k} \mathrm{count}(\mathrm{str}(r_i), S) \leq \ell(\lceil \log |T_C| \rceil + 1) = O(\ell \log(n\ell))$.*

Proof. For any suffix S_i of S, let v_i be the node in T_C such that $\mathrm{str}(v_i)$ is the longest prefix of S_i which is in C. We say that the path from the root of T_C to v_i *corresponds* to S_i. For any heavy path root r, note that $\mathrm{count}(\mathrm{str}(r), S)$ is exactly the number of suffixes of S that begin with $\mathrm{str}(r)$. Further, if a suffix S_i has $\mathrm{str}(r)$ as a prefix, then the path corresponding to S_i goes through r. By the property of the heavy path decomposition (Lemma 3.5), the path corresponding to S_i contains at most $\lceil \log |T_C| \rceil$ light edges, and as such, at most $\lceil \log |T_C| \rceil + 1$ heavy path roots. Thus, any suffix S_i of S contributes at most $\lceil \log |T_C| \rceil + 1$ to the total sum. Since we have at most ℓ suffixes, this gives us $\sum_{i=0}^{k} \mathrm{count}(\mathrm{str}(r_i), S) \leq \ell(\lceil \log |T_C| \rceil + 1) = O(\ell \log(n\ell))$. □

Lemma 3.6.1 and Lemma 3.7 give that the L_1-sensitivity of $(\mathrm{count}(\mathrm{str}(r_i), \mathcal{D}))_{i=0}^{k}$ is bounded by $\ell(\lceil \log |T_C| \rceil + 1) = O(\ell \log(n\ell))$. Additionally, note that since T_C has at most $n^2\ell^3$ leaves, there are at most $n^2\ell^3$ heavy paths. Together with Corollary 2.3, this gives:

Corollary 3.8. *Let $r_0, \ldots, r_k$ be the roots of the heavy paths of T_C. For any $\epsilon > 0$ and $0 < \beta < 1$, there exists an ϵ-differentially private algorithm, which estimates $\mathrm{count}(\mathrm{str}(r_i), \mathcal{D})$ for the heavy path roots $r_0, \ldots, r_k$ up to an additive error at most $O(\epsilon^{-1}\ell \log(n\ell) \ln(k/\beta)) = O(\epsilon^{-1}\ell \log^2(n\ell/\beta))$ with probability at least $1 - \beta$.*

Additionally, Lemma 3.6.2 and Lemma 3.7 give that the *sum of* L_1-sensitivities of the difference sequences *over all heavy paths* is bounded by $2\ell(\lceil \log |T_C| \rceil + 1)$. As the next step, we show how to

compute all *prefix sums* of the difference sequences of all heavy paths with an error $O(\ell\lceil\log|T_C|\rceil)$ up to polylogarithmic terms. To do this, we show that we can estimate the prefix sums of k sequences while preserving differential privacy up to an additive error which is roughly the sum of their L_1-sensitivities. The algorithm builds a copy of the binary tree mechanism [27] for each of the k sequences. Since the binary tree mechanism, to the best of our knowledge, has not been analyzed for this problem formulation, we give the following lemma.

Lemma 3.9. *Let χ^* be any universe of possible data sets and $T \in \mathbb{N}$. Let $a^{(1)}, \ldots, a^{(k)}$ be k functions, where the output of every function is a sequence of length T, i.e. $a^{(i)} : \chi^* \to \mathbb{N}^T$ for all $i \in [1,k]$. Let L be the sum of L_1-sensitivities of $a^{(1)}, \ldots, a^{(k)}$, that is, let $L = \max_{x\sim x'} \sum_{i=1}^{k}\sum_{j=1}^{T} |a^{(i)}(x)[j] - a^{(i)}(x')[j]|$. For any $\epsilon > 0$ and $0 < \beta < 1$ there exists an ϵ-differentially private algorithm computing for every $i \in [1,k]$ all prefix sums of $a^{(i)}(x)$ with additive error at most $O(\epsilon^{-1}L\log T\log(Tk/\beta))$ with probability at least $1-\beta$.*

We defer the proof to the full version [7]. Lemma 3.9 gives the following corollary with $L = \ell(\lceil\log|T_C|\rceil + 1) = O(\ell\log(n\ell))$, $k \le n^2\ell^3$, and $T = \ell$.

Corollary 3.10. *Let $p_0, \ldots, p_k$ be all heavy paths of T_C. For any $\epsilon > 0$ and $0 < \beta < 1$, there exists an ϵ-differentially private algorithm, which estimates $\sum_{j=1}^{i} \mathrm{diff}_{p_m}(\mathcal{D})[j]$ for all $i = 1, \ldots, |p_m|$ and all $m = 0, \ldots, k$ up to an additive error at most*

$$O\left(\epsilon^{-1}\ell\log(n\ell)\log\ell\log(\ell k/\beta)\right) = O\left(\epsilon^{-1}\ell\log\ell\log^2(n\ell/\beta)\right)$$

with probability at least $1-\beta$.

Lemma 3.11 below provides a procedure to compute T_C efficiently.

Lemma 3.11. *Given the representation of C output by the algorithm of Lemma 3.4 for a database $\mathcal{D}$ of n documents from $\Sigma^{[1,\ell]}$, a trie T_C and its heavy path decomposition, noisy counts of the heavy path roots and noisy prefix sums on the heavy paths, can be computed in $O(n^2\ell^4)$ time and space.*

Proof. Let $\mathcal{D} = S_1, \ldots, S_n$. Given the representation of C output by the algorithm of Lemma 3.4 with the suffix tree of $S_1\$_1 \ldots S_n\$_n$, we can build the trie augmented with the true counts for each node in $O(n^2\ell^4)$ time and space: we just insert each string letter by letter, and at the same time traverse the suffix tree to compute the counts for each prefix. Given the trie, we can compute the heavy path decomposition in linear time of the size of the trie, i.e. in $O(n^2\ell^4)$ time. Given the counts for each node, we can compute the noisy counts for all roots of the heavy paths in $O(n^2\ell^3)$ time. Further, for every heavy path of length h, we can compute the noisy prefix sums of the difference sequences in $O(h)$ time and space: note that we divide the length of the path into dyadic intervals, and need to compute a noisy count for each dyadic interval. Constructing these partial sums bottom-up can be done in $O(h)$ time. Since all heavy paths lengths summed up are bounded by the size of the tree, the total time is $O(n^2\ell^4)$. □

3.3 Putting It Together

Our full algorithm runs the algorithms given by Lemma 3.4, Corollary 3.8 and Corollary 3.10 in sequence with privacy parameter $\epsilon' = \epsilon/3$ and failure probability $\beta' = \beta/3$. For a heavy path $p = v_0, v_1, \ldots, v_{|p|-1}$ with root $r = v_0$, let $\mathrm{count}^*(\mathrm{str}(r), \mathcal{D})$ be the approximate counts computed by the algorithm from Corollary 3.8, and let $\mathrm{sums}^*_p[i]$ be the approximate estimate of $\sum_{j=1}^{i}\mathrm{diff}_p(\mathcal{D})[j]$. Then for any node v_i on the path with $i > 0$, we compute $\mathrm{count}^*(\mathrm{str}(v_i), \mathcal{D}) = \mathrm{count}^*(\mathrm{str}(r), \mathcal{D}) + \mathrm{sums}^*_p[i]$. Let α be the sum of the error bounds for Corollary 3.8 and Corollary 3.10 which each hold with probability $1-\beta'$. We have $\alpha = O(\epsilon^{-1}\ell\log\ell\log^2(n\ell/\beta))$. As a final step, we prune T_C by traversing the trie from the root, and for any node v with $\mathrm{count}^*(\mathrm{str}(v_i), \mathcal{D}) < 2\alpha$, we

delete v and its subtree. We return the resulting pruned trie T^* together with approximate counts $\text{count}^*(\text{str}(v), \mathcal{D})$ for every node $v \in T^*$.

By the composition theorem (Lemma 2.4), this algorithm is ϵ-differentially private. By a union bound, all its error guarantees hold together with probability $1 - 3\beta' = 1 - \beta$. Thus, we get with probability $1 - \beta$ a trie T^*, with the following properties:

- For each node $v \in T^*$, by Corollaries 3.8 and 3.10,
$$|\text{count}^*(\text{str}(v), \mathcal{D}) - \text{count}(\text{str}(v), \mathcal{D})| \leq \alpha = O\left(\epsilon^{-1}\ell \log \ell \log^2(n\ell/\beta)\right)$$
- Every string $P \in \Sigma^{[1,\ell]}$ which is not present T^* was either: (i) not present in C, in which case $\text{count}(P, \mathcal{D}) = O(\epsilon^{-1}\ell \log \ell(\log(n\ell/\beta) + \log|\Sigma|))$ by Lemma 3.3; or (ii) deleted in the pruning process of T_C, in which case $\text{count}(P, \mathcal{D}) < 3\alpha = O(\epsilon^{-1}\ell \log \ell \log^2(n\ell/\beta))$.
- Any string $P \in \sum^{[1,\ell]}$ which is present in T^* has a count at least $2\alpha - \alpha > 1$. Therefore, T^* has at most $O(n\ell^2)$ nodes.

To query the resulting data structure for a pattern P, we match P in the trie, and if there is a node v with $\text{str}(v) = P$, we return the approximate count $\text{count}^*(\text{str}(v), \mathcal{D})$ saved at v. If P is not present, we return 0. This requires $O(|P|)$ time. In summary, we have obtained Theorem 1.2.

3.4 Faster Algorithm for Fixed-Length q-grams and Pure Differential Privacy

If we only care about counting q-grams for a fixed length q, the algorithm can be simplified significantly, improving the construction time significantly and the error slightly.

Lemma 3.12. *Let n, ℓ, and $q \leq \ell$ be integers and Σ an alphabet of size $|\Sigma|$. Let $\Delta \leq \ell$. For any $\epsilon > 0$ and $0 < \beta < 1$, there exists an ϵ-differentially private algorithm, which can process any database $\mathcal{D} = S_1, \ldots, S_n$ of documents from $\Sigma^{[1,\ell]}$ and with probability at least $1 - \beta$ output a data structure for computing count_Δ for all q-grams with additive error $O\left(\epsilon^{-1}\ell \log \ell(\log(n\ell/\beta) + \log|\Sigma|)\right)$. The data structure can be stored in $O(n\ell^2)$ space and answer queries in $O(|P|)$ time. It can be constructed in $O(n^2\ell^2 \log q \log\log(n\ell) + n^2\ell^3)$ time using $O(n^2\ell^3)$ space.*

Proofsketch. The algorithm constructs $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$, where $j = \lfloor \log q \rfloor$, as in the proof of Lemma 3.3, with privacy parameter $\epsilon' = \epsilon/2$. We can then construct C_q, again as in Lemma 3.3. We compute noisy counts of every string in the set C_q with the Laplace mechanism (Lemma A.1) and privacy parameter $\epsilon/2$. As a post-processing step, we throw out all elements from C_q whose noisy count is below 2α, where α is the error of the Laplace mechanism. We call the resulting set $\mathcal{P}_q$ and arrange its elements into a trie together with the corresponding noisy counts. By composition, this is ϵ-differentially private. By a similar argument as Lemma 3.3, with probability at least $1 - \beta$, the noisy counts of any element in the trie has an error at most $O\left(\epsilon^{-1} \log \ell \left(\log(n\ell/\beta) + \log(|\Sigma|)\right)\right)$, and everything not in the trie has a count at most $O\left(\epsilon^{-1} \log \ell \left(\log(n\ell/\beta) + \log(|\Sigma|)\right)\right)$. Again by a similar argument as Lemma 3.3, with probability at least $1 - \beta$, we have $|\mathcal{P}_q| \leq n\ell$, and therefore the trie has size at most $n\ell^2$.

As shown in Step 1 of the proof of Lemma 3.4, $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$ can be constructed using total time $O(n^2\ell^2 \log q \log\log(n\ell))$ and $O(n\ell^2)$ space. The set C_q and the true count of every string in C_q can be computed in an additional $O(n^2\ell^2 \log\log(n\ell) + n^2\ell^3 + \text{sort}(n\ell^2, |\Sigma|)) = O(n^2\ell^2 \log\log(n\ell) + n^2\ell^3)$ time and $O(n^2\ell^3)$ space. The trie can be constructed in time $O(n^2\ell^3)$. □

4 Counting Functions on Trees

In this section, we prove Theorem 1.5. The strategy is essentially the same algorithm as the one applied to T_C in the previous sections.

Proof of Theorem 1.5. The algorithm starts by computing a heavy path decomposition of T. Then, we use the Laplace mechanism to compute the counts of all heavy path roots with $\epsilon/2$-differential privacy. Furthermore, for every heavy path $p = v_0, \ldots, v_{|p|-1}$, we compute the prefix sums of the difference sequence $\text{diff}_p(\mathcal{D})[i] = c(v_i, \mathcal{D}) - c(v_{i-1}, \mathcal{D})$ via Lemma 3.9 with $\epsilon/2$-differential privacy. Let $r = v_0$ be the root of the heavy path. For any node v_i on heavy path p, we can compute $\hat{c}(v_i)$ from $\hat{c}(r)$ plus the approximate value of $\sum_{j=1}^{i} \text{diff}_p(\mathcal{D})$.

To analyze the accuracy, let $\mathcal{D}$ and $\mathcal{D}'$ be neighboring and let l be a leaf such that $|c(l, \mathcal{D}) - c(l, \mathcal{D}')| = b \leq d$. Note that $c(l, \mathcal{D})$ contributes to $c(v, \mathcal{D})$ if and only if l is below v. In particular, by Lemma 3.5, it can contribute at most b to the function c of at most $O(\log |V|)$ heavy path roots. Thus, for the roots $r_1, \ldots, r_k$ of heavy paths, we have $\sum_{i=1}^{k} |c(r_i, \mathcal{D}) - c(r_i, \mathcal{D}')| = O(d \log |V|)$. Thus, by Lemma A.1, we can estimate $c(r_1, \mathcal{D}), \ldots, c(r_k, \mathcal{D})$ up to an error of $O(\epsilon^{-1} d \log |V| \ln(k/\beta))$ with probability at least $1 - \beta/2$. Similarly, l can only contribute by at most b to the sensitivity of the difference sequence of any heavy path that has a root above l. Thus, using Lemma 3.9 with $L = O(d \log |V|)$, we can estimate all prefix sums of diff_p for all k heavy paths p with an error at most $O(\epsilon^{-1} d \log |V| \log h \log(hk/\beta))$ with probability at least $1 - \beta/2$. □

5 Faster Algorithm for Fixed-Length q-grams and Approximate Differential Privacy

In this section, we give an algorithm for (ϵ, δ)-differentially private q-grams, which can be made to run in $O(n\ell(\log q + \log |\Sigma|))$ time and $O(n\ell)$ space. The idea is that for this less stringent version of approximate differential privacy, we can avoid computing the noisy counts of substrings that do not appear in the database, by showing that with high probability, these strings will not be contained in the result, anyway. Specifically, we will use the following lemma.

Lemma 5.1. *Let* $\text{Alg}_1 : \chi^* \to \mathcal{R}$ *and* $\text{Alg}_2 : \chi^* \to \mathcal{R}$ *be two randomized algorithms. Let* $\epsilon > 0$, $\delta \in [0, 1)$ *and* $\gamma \leq \frac{\delta}{3e^\epsilon}$. *Let* Alg_1 *be* (ϵ, δ')*-differentially private, where* $\delta' \leq \gamma$. *Assume that for every data set* $\mathcal{D} \in \chi^*$, *there exists an event* $E(\mathcal{D}) \subseteq \mathcal{R}$ *such that for all* $U \subseteq \mathcal{R}$

$$\Pr[\text{Alg}_1(\mathcal{D}) \in U | \text{Alg}_1(\mathcal{D}) \in E(\mathcal{D})] = \Pr[\text{Alg}_2(\mathcal{D}) \in U]$$

and $\Pr[\text{Alg}_1(\mathcal{D}) \in E(\mathcal{D})] \geq 1 - \gamma$. *Then* Alg_2 *is* (ϵ, δ)*-differentially private.*

Proof. Let $\mathcal{D}$ and $\mathcal{D}'$ be neighbouring data sets and let $(\text{Alg}_1(\mathcal{D}) \in E(\mathcal{D})) = E$ and $(\text{Alg}_1(\mathcal{D}') \in E(\mathcal{D}')) = E'$. We have, for any $U \in \mathcal{R}$:

$$\begin{aligned}
\Pr[\text{Alg}_2(\mathcal{D}) \in U] = \Pr[\text{Alg}_1(\mathcal{D}) \in U | E] &\leq \frac{\Pr[\text{Alg}_1(\mathcal{D}) \in U]}{1 - \gamma} \leq \frac{e^\epsilon \Pr[\text{Alg}_1(\mathcal{D}') \in U] + \delta'}{1 - \gamma} \\
&\leq \frac{e^\epsilon (\Pr[\text{Alg}_1(\mathcal{D}') \in U | E'](1 - \gamma) + \gamma) + \delta'}{1 - \gamma} \\
&\leq e^\epsilon \Pr[\text{Alg}_1(\mathcal{D}') \in U | E'] + \frac{e^\epsilon \gamma + \delta'}{1 - \gamma} \\
&\leq e^\epsilon \Pr[\text{Alg}_1(\mathcal{D}') \in U | E'] + \delta \\
&= e^\epsilon \Pr[\text{Alg}_2(\mathcal{D}') \in U] + \delta
\end{aligned}$$

The third inequality holds since $\Pr[E'] \geq 1 - \gamma$ and the last inequality holds since

$$\frac{e^\epsilon \gamma + \delta'}{1 - \gamma} \leq \frac{2e^\epsilon \gamma}{1 - \gamma}.$$

Setting $\frac{2e^\epsilon \gamma}{1-\gamma} \leq \delta$ gives $(2e^\epsilon + \delta)\gamma \leq \delta$ and therefore it is enough to choose $\gamma \leq \frac{\delta}{3e^\epsilon} \leq \frac{\delta}{2e^\epsilon + \delta}$. □

Lemma 5.2. *Let* n, ℓ, *and* $q \leq \ell$ *be integers and* Σ *an alphabet of size* $|\Sigma|$. *Let* $\Delta \leq \ell$. *For any* $\epsilon > 0$, $\delta > 0$, *and* $0 < \beta < 1$, *there exists an* (ϵ, δ)*-differentially private algorithm, which can process any database*

$\mathcal{D} = S_1, \ldots, S_n$ of documents from $\Sigma^{[1,\ell]}$ and with probability at least $1-\beta$ output a data structure for count_Δ *for all q-grams with additive error* $O\left(\epsilon^{-1}\sqrt{\ell\Delta\log(n\ell)}\log q\left(\epsilon + \log\log q + \log\frac{|\Sigma|}{\delta\beta}\right)\right)$.

Proof. The idea is to design two algorithms, Alg_1 and Alg_2, where Alg_2 is the algorithm that we will run, and Alg_1 is an algorithm which we show to be differentially private with appropriate parameters. We then argue that the two algorithms behave the same conditioned on a high-probability event. We set $\epsilon_1 = \epsilon/(\lfloor\log q\rfloor + 2)$, $\beta_1 = \min(\beta/(\lfloor\log q\rfloor + 2), \delta/(3e^\epsilon(\lfloor\log q\rfloor + 2)))$, $\delta_1 \le \beta_1$. Let $j = \lfloor\log q\rfloor$ and $\mathrm{Alg}_1 = \mathrm{Alg}_1^{(j+1)} \circ \cdots \circ \mathrm{Alg}_1^{(0)}$.

- $\mathrm{Alg}_1^{(0)}$ computes $\mathcal{P}_{2^0}$ as follows: it adds noise scaled with $N(0,\sigma^2)$ to the count of each letter, where $\sigma = 2\epsilon_1^{-1}\sqrt{2\ell\Delta\ln(2/\delta_1)}$. Let $\alpha = 2\epsilon_1^{-1}\sqrt{2\ell\Delta\ln(2/\delta_1)\ln(2\max\{\ell^2n^2, |\Sigma|\}/\beta_1)}$. The set $\mathcal{P}_{2^0}$ is the set of all letters with a noisy count at least 2α. We stop if $|\mathcal{P}_{2^0}| > n\ell$.
- $\mathrm{Alg}_1^{(k)}$, for $0 < k < j+1$, computes $\mathcal{P}_{2^k}$ by adding noise scaled with $N(0,\sigma^2)$ to the count of every string in $\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$. $\mathcal{P}_{2^k}$ is the set of all strings of length 2^k with noisy count at least 2α. We stop if $|\mathcal{P}_{2^k}| > n\ell$.
- $\mathrm{Alg}_1^{(j+1)}$ computes $\mathcal{P}_q$ by adding noise scaled with $N(0,\sigma^2)$ to the count of every string P of length q such that $P[0\ldots 2^j-1] \in \mathcal{P}_{2^j}$ and $P[q-2^j\ldots q-1] \in \mathcal{P}_{2^j}$. $\mathcal{P}_q$ is the set of all such strings with a noisy count at least 2α and we output it together with the noisy counts.

Let $\mathcal{Z}_m(\mathcal{D})$ be the set of all strings P of length m which satisfy $\mathrm{count}_\Delta(P,\mathcal{D}) = 0$. Alg_2 is $\mathrm{Alg}_2 = \mathrm{Alg}_2^{(j+1)} \circ \cdots \circ \mathrm{Alg}_2^{(0)}$, where for every $0 \le k \le j+1$, the algorithm $\mathrm{Alg}_2^{(k)}$ is the same as $\mathrm{Alg}_1^{(k)}$, except it does not take into account strings whose true count is zero. In detail:

- $\mathrm{Alg}_2^{(0)}$ constructs $\mathcal{P}_{2^0}$ by adding noise scaled with $N(0,\sigma^2)$ to the count of every letter in $\Sigma \setminus \mathcal{Z}_1(\mathcal{D})$, where $\sigma = 2\epsilon_1^{-1}\sqrt{2\ell\Delta\ln(2/\delta_1)}$. Let $\alpha = 2\epsilon_1^{-1}\sqrt{2\ell\Delta\ln(2/\delta_1)\ln(2\max\{\ell^2n^2, |\Sigma|\}/\beta_1)}$: $\mathcal{P}_{2^0}$ is the set of all letters with a noisy count at least 2α.
- $\mathrm{Alg}_2^{(k)}$, for $0 < k < j+1$, constructs $\mathcal{P}_{2^k}$ by adding noise scaled with $N(0,\sigma^2)$ to the count of every string in $(\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}) \setminus \mathcal{Z}_{2^k}(\mathcal{D})$. The set $\mathcal{P}_{2^k}$ contains all strings with noisy count at least 2α.
- $\mathrm{Alg}_2^{(j+1)}$ constructs $\mathcal{P}_q$ by adding noise scaled with $N(0,\sigma^2)$ to the count of every string $P \in \Sigma^q \setminus \mathcal{Z}_q(\mathcal{D})$ of length q such that $P[0\ldots 2^j-1] \in \mathcal{P}_{2^j}$ and $P[q-2^j\ldots q-1] \in \mathcal{P}_{2^j}$. $\mathcal{P}_q$ is the set of all such strings with a noisy count at least 2α together with the noisy counts.

Privacy of Alg_2. Every $\mathrm{Alg}_1^{(k)}$ for $0 \le k \le j+1$ is (ϵ_1,δ_1)-differentially private by Corollary B.2 and Lemma A.6. We use Lemma 5.1 on $\mathrm{Alg}_1^{(k)}$ and $\mathrm{Alg}_2^{(k)}$, for every $0 \le k \le j+1$. We define $E^{(k)}(\mathcal{D})$ as the event that none of the noises added by $\mathrm{Alg}_1^{(k)}$ to strings in $\mathcal{Z}_k$ exceeds an absolute value of α. By Lemma A.7, this is true with probability at least $1-\beta_1$. Let $\mathcal{R}^{(k)}$ be the range of $\mathrm{Alg}_1^{(k)}$ and $\mathrm{Alg}_2^{(k)}$. Now, for any $U \subseteq \mathcal{R}^{(k)}$ and since all the added noises are independent, $\Pr[\mathrm{Alg}_1^{(k)} \in U | E^{(k)}(\mathcal{D})]$ only depends on the distribution of the noises added to the counts of strings in $(\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}) \setminus \mathcal{Z}_{2^k}(\mathcal{D})$ and is thus equal to $\Pr[\mathrm{Alg}_2^{(k)} \in U]$. Since $\delta_1 \le \beta_1$ and $\beta_1 \le \frac{\delta}{3e^\epsilon(\lfloor\log q\rfloor+2)}$, by Lemma 5.1, we have that $\mathrm{Alg}_2^{(k)}$ is $(\epsilon_1, \delta/(\lfloor\log q\rfloor+2))$ differentially private. By Lemma 2.4, we have that Alg_2 is (ϵ,δ)-differentially private.

Accuracy. The accuracy proof proceeds similarly to the proof of Lemma 3.3 (or Lemma B.3, which is its equivalent for (ϵ,δ)-differential privacy). By the same arguments, with probability at least $1-\beta$, every pattern of length q not in $\mathcal{P}_q$ has a count at most 3α, and we compute the counts of patterns in $\mathcal{P}_q$ with error at most α, where $\alpha = 2\epsilon_1^{-1}\sqrt{2\ell\Delta\ln(2/\delta_1)\ln(2\max\{\ell^2n^2, |\Sigma|\}/\beta_1)} = O(\epsilon^{-1}\sqrt{\ell\Delta\log(n\ell)}\log q(\epsilon + \log\log q + \log\frac{|\Sigma|}{\delta\beta}))$. □

Lemma 5.3 provides an efficient procedure to construct sets $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$ and $\mathcal{P}_q$ for Alg_2. The efficiency of this procedure relies on the fact that Alg_2 ignores all substrings that do not occur in any document of $\mathcal{D}$, in contrast with the algorithms underlying Theorem 1.2. This crucial difference allows us to compute (noisy) counts only for carefully selected substrings of $\mathcal{D}$, effectively avoiding the computational bottleneck of the algorithm of Theorem 1.2. Lemmas 5.2 and 5.3 together give Theorem 1.4.

Lemma 5.3. *Given a database of n documents $\mathcal{D} = S_1, \ldots, S_n$ from $\Sigma^{[1,\ell]}$, and an integer $q \leq \ell$, a data structure satisfying the properties of Lemma 5.2 that answers q-gram counting queries in $O(q)$ time can be computed in $O(n\ell(\log q + \log|\Sigma|))$ time and $O(n\ell)$ space.*

Proof. Our main tool is the suffix tree ST of the string $S = S_1\$_1 S_2\$_2 \cdots S_n\$_n$, with $\mathcal{D} = \{S_1, S_2, \ldots, S_n\}$ and $\$_1, \$_2, \ldots, \$_n \notin \Sigma$, preprocessed to answer weighted ancestor queries in $O(1)$ time. A weighted ancestor query consists of two integers $p, \ell > 0$, where p is the ID of a leaf of ST; the answer is the farthest (closest to the root) ancestor v of leaf p with string depth at least ℓ. Any such query can be answered in $O(1)$ time after a linear-time preprocessing to build a linear-space additional data structure [5, 38]. We also store, within each branching node v of ST, the ID $\mathrm{leaf}(v)$ of the leftmost descending leaf. This extra information can be computed in linear time with a DFS of ST, and requires linear extra space. The ID stored at a node v gives, together with the string depth of v, a *witness occurrence* in some document from $\mathcal{D}$ of the substring represented by v.

Our procedure computes an implicit representation of sets $\mathcal{P}_{2^k}$ for increasing values of $k \in [0, j]$, where $j = \lfloor \log q \rfloor$; it returns a compacted trie of the elements of $\mathcal{P}_q$ with a noisy counter for each element. For each phase $k \leq j$, we perform the following steps. (i) Traverse ST to find the set of branching nodes whose string depth is at least 2^k and such that the string depth of their parent node is strictly smaller than 2^k. We call such nodes *2^k-minimal*. (ii) For $k = 0$, add noise to the frequency stored in each 1-minimal node as in Lemma 5.2, and mark as belonging to $\mathcal{P}_{2^0}$ the nodes whose noisy count exceeds 2α. In Phase $k > 0$, we ask two weighted ancestor queries for each 2^k-minimal node v: one with $p = \mathrm{leaf}(v)$ and $\ell = 2^{k-1}$, one with $p = \mathrm{leaf}(v) + 2^{k-1}$ and $\ell = 2^{k-1}$. If the answer to both queries is a node marked as belonging to $\mathcal{P}_{2^{k-1}}$, we add noise to the frequency of v and mark it as belonging to $\mathcal{P}_{2^k}$ if the noisy count exceeds 2α; we proceed to the next node otherwise. Note that at the end of Phase k we can remove from ST the markers for $\mathcal{P}_{2^{k-2}}$. Finally, to compute $\mathcal{P}_q$, for each q-minimal node v we ask weighted ancestor queries with $p = \mathrm{leaf}(v)$ and $\ell = 2^j$ and with $p = \mathrm{leaf}(v) + q - 2^j + 1$ and $\ell = 2^j$. If the answer to both queries is a node marked for $\mathcal{P}_{2^j}$, we add noise to the frequency of v; if it exceeds 2α, we insert $\mathrm{str}(v)[1, q]$ in the output trie and store this noisy counter within the corresponding leaf. To answer a query for a q-gram P, we spell P from the root of the trie in $O(q)$ time. We return its noisy counter if we find it, 0 otherwise.

Time and space analysis. For each value of k, the algorithm traverses ST in $O(n\ell)$ time; since the 2^k-minimal nodes partition the leaves of ST, it asks up to 2 $O(1)$-time weighted ancestor queries per leaf, thus requiring $O(n\ell)$ total time. Since ST can be constructed in $O(n\ell \log|\Sigma|)$ time [28] and k takes $\lfloor \log q \rfloor$ distinct values, the whole procedure requires $O(n\ell(\log q + \log|\Sigma|))$ time. The space is $O(n\ell)$ because ST and the additional data structure for weighted ancestor queries occupy $O(n\ell)$ words of space [5], and at any phase k, each node stores at most two markers. Finally, the output compacted trie occupies space $O(n\ell)$ because (i), since $\mathcal{D}$ contains at most $O(n\ell)$ distinct q-grams it has at most $O(n\ell)$ leaves, and (*ii*), since all the q-grams in the trie occur in $\mathcal{D}$, the edge labels can be compactly represented with intervals of positions over $\mathcal{D}$, similar to the edge labels of ST. □

Acknowledgments

This work is supported by Danish Research Council grant DFF-8021-002498 and Villum Fonden grant VIL51463.

References

[1] John M. Abowd et al. 2020. The modernization of statistical disclosure limitation at the US Census Bureau. *Census Working Papers* (2020).

[2] Noga Alon, Roi Livni, Maryanthe Malliaris, and Shay Moran. 2019. Private PAC learning implies finite Littlestone dimension. In *Proc. 51st STOC*. 852–860. https://doi.org/10.1145/3313276.3316312

[3] Apple Differential Privacy Team. 2017. Learning with privacy at scale. *Apple Machine Learning* (2017).

[4] Amos Beimel, Kobbi Nissim, and Uri Stemmer. 2016. Private Learning and Sanitization: Pure vs. Approximate Differential Privacy. *Theory Comput.* 12, 1 (2016), 1–61. https://doi.org/10.4086/TOC.2016.V012A001

[5] Djamal Belazzougui, Dmitry Kosolobov, Simon J. Puglisi, and Rajeev Raman. 2021. Weighted Ancestors in Suffix Trees Revisited. In *Proc. 32nd CPM (LIPIcs, Vol. 191)*. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 8:1–8:15. https://doi.org/10.4230/LIPICS.CPM.2021.8

[6] Michael A. Bender and Martin Farach-Colton. 2000. The LCA Problem Revisited. In *Proc. 4th LATIN (Lecture Notes in Computer Science, Vol. 1776)*. Springer, 88–94. https://doi.org/10.1007/10719839_9

[7] Giulia Bernardini, Philip Bille, Inge Li Gørtz, and Teresa Anna Steiner. 2024. Differentially Private Substring and Document Counting. arXiv:2412.13813 [cs.DS] https://arxiv.org/abs/2412.13813

[8] Philip Bille, Anders Roy Christiansen, Patrick Hagge Cording, Inge Li Gørtz, Frederik Rye Skjoldjensen, Hjalte Wedel Vildhøj, and Søren Vind. 2018. Dynamic Relative Compression, Dynamic Partial Sums, and Substring Concatenation. *Algorithmica* 80, 11 (2018), 3207–3224. https://doi.org/10.1007/S00453-017-0380-7

[9] Philip Bille, Inge Li Gørtz, Hjalte Wedel Vildhøj, and Søren Vind. 2014. String Indexing for Patterns with Wildcards. *Theory Comput. Syst.* 55, 1 (2014), 41–60.

[10] Avrim Blum, Cynthia Dwork, Frank McSherry, and Kobbi Nissim. 2005. Practical privacy: the SuLQ framework. In *Proc. 24th PODS*, Chen Li (Ed.). ACM, 128–138. https://doi.org/10.1145/1065167.1065184

[11] Luca Bonomi and Li Xiong. 2013. A two-phase algorithm for mining sequential patterns with differential privacy. In *Proc. 22nd CIKM*. 269–278. https://doi.org/10.1145/2505515.2505553

[12] Prosenjit Bose, Meng He, Anil Maheshwari, and Pat Morin. 2009. Succinct orthogonal range search structures on a grid with applications to text indexing. In *Proc. 11th WADS*. 98–109.

[13] Mark Bun, Cynthia Dwork, Guy N. Rothblum, and Thomas Steinke. 2018. Composable and versatile privacy via truncated CDP. In *Proc. 50th STOC*. 74–86. https://doi.org/10.1145/3188745.3188946

[14] Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil P. Vadhan. 2015. Differentially Private Release and Learning of Threshold Functions. In *Proc. 56th FOCS*. 634–649. https://doi.org/10.1109/FOCS.2015.45

[15] Karan N. Chadha, Junye Chen, John C. Duchi, Vitaly Feldman, Hanieh Hashemi, Omid Javidbakht, Audra McMillan, and Kunal Talwar. 2024. Differentially Private Heavy Hitter Detection using Federated Analytics. In *Proc. SaTML 2024*. 512–533. https://doi.org/10.1109/SATML59370.2024.00032

[16] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. 2011. Private and Continual Release of Statistics. *ACM Trans. Inf. Syst. Secur.* 14, 3 (2011), 26:1–26:24.

[17] Huiping Chen, Changyu Dong, Liyue Fan, Grigorios Loukides, Solon P. Pissis, and Leen Stougie. 2021. Differentially Private String Sanitization for Frequency-Based Mining Tasks. In *Proc. 21st ICDM*. 41–50. https://doi.org/10.1109/ICDM51629.2021.00014

[18] Rui Chen, Gergely Ács, and Claude Castelluccia. 2012. Differentially private sequential data publication via variable-length n-grams. In *Proc. 19th CCS*. 638–649. https://doi.org/10.1145/2382196.2382263

[19] Rui Chen, Benjamin C. M. Fung, Bipin C. Desai, and Nériah M. Sossou. 2012. Differentially private transit data publication: a case study on the montreal transportation system. In *Proc. 18th KDD*. 213–221. https://doi.org/10.1145/2339530.2339564

[20] Xiang Cheng, Sen Su, Shengzhi Xu, and Zhengyi Li. 2015. DP-Apriori: A differentially private frequent itemset mining algorithm based on transaction splitting. *Comput. Secur.* 50 (2015), 74–90. https://doi.org/10.1016/J.COSE.2014.12.005

[21] Xiang Cheng, Sen Su, Shengzhi Xu, Peng Tang, and Zhengyi Li. 2015. Differentially private maximal frequent sequence mining. *Comput. Secur.* 55 (2015), 175–192. https://doi.org/10.1016/J.COSE.2015.08.005

[22] Edith Cohen, Xin Lyu, Jelani Nelson, Tamás Sarlós, and Uri Stemmer. 2023. Optimal Differentially Private Learning of Thresholds and Quasi-Concave Optimization. In *Proc. 55th STOC*. 472–482. https://doi.org/10.1145/3564246.3585148

[23] Xiaofeng Ding, Long Chen, and Hai Jin. 2017. Mining Representative Patterns Under Differential Privacy. In *Proc. 18th WISE*. 295–302. https://doi.org/10.1007/978-3-319-68786-5_23

[24] Irit Dinur and Kobbi Nissim. 2003. Revealing information while preserving privacy. In *Proc. 22nd PODS*. 202–210. https://doi.org/10.1145/773153.773173

[25] Cynthia Dwork and Jing Lei. 2009. Differential privacy and robust statistics. In *Proc. 41st STOC*. 371–380. https://doi.org/10.1145/1536414.1536466

[26] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam D. Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *Proc. 3rd TCC*, Vol. 3876. Springer, 265–284.

[27] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential privacy under continual observation. In *Proc. 42nd STOC*. 715–724.
[28] Martin Farach. 1997. Optimal Suffix Tree Construction with Large Alphabets. In *Proc. 38th FOCS*. IEEE Computer Society, 137–143. https://doi.org/10.1109/SFCS.1997.646102
[29] Martin Farach-Colton, Paolo Ferragina, and Shanmugavelayutham Muthukrishnan. 2000. On the sorting-complexity of suffix tree construction. *J. ACM* 47, 6 (2000), 987–1011.
[30] Paolo Ferragina and Giovanni Manzini. 2005. Indexing compressed text. *J. ACM* 52, 4 (2005), 552–581.
[31] Paolo Ferragina, Giovanni Manzini, Veli Mäkinen, and Gonzalo Navarro. 2007. Compressed representations of sequences and full-text indexes. *ACM Trans. Algorithms* 3, 2 (2007), 20.
[32] Hendrik Fichtenberger, Monika Henzinger, and Jalaj Upadhyay. 2023. Constant Matters: Fine-grained Error Bound on Differentially Private Continual Observation. In *Proc. 40th ICML*. 10072–10092. https://proceedings.mlr.press/v202/fichtenberger23a.html
[33] Travis Gagie and Juha Kärkkäinen. 2011. Counting colours in compressed strings. In *Proc. 22nd CPM*. 197–207.
[34] Travis Gagie, Juha Kärkkäinen, Gonzalo Navarro, and Simon J Puglisi. 2013. Colored range queries and document retrieval. *Theoret. Comput. Sci.* 483 (2013), 36–50.
[35] Travis Gagie, Gonzalo Navarro, and Nicola Prezza. 2020. Fully functional suffix trees and optimal text searching in BWT-runs bounded space. *J. ACM* 67, 1 (2020), 1–54.
[36] Younan Gao. 2023. Adaptive Data Structures for 2D Dominance Colored Range Counting. In *Proc. 18th WADS*. 460–473. https://doi.org/10.1007/978-3-031-38906-1_30
[37] Younan Gao and Meng He. 2021. Space Efficient Two-Dimensional Orthogonal Colored Range Counting. In *Proc. 29th ESA*. 46:1–46:17. https://doi.org/10.4230/LIPICS.ESA.2021.46
[38] Pawel Gawrychowski, Moshe Lewenstein, and Patrick K. Nicholson. 2014. Weighted Ancestors in Suffix Trees. In *Proc. 22th ESA (Lecture Notes in Computer Science, Vol. 8737)*. Springer, 455–466. https://doi.org/10.1007/978-3-662-44777-2_38
[39] Badih Ghazi, Pritish Kamath, Ravi Kumar, Pasin Manurangsi, and Kewen Wu. 2023. On Differentially Private Counting on Trees. In *Proc. 50th ICALP*, Vol. 261. 66:1–66:18. https://doi.org/10.4230/LIPICS.ICALP.2023.66
[40] Badih Ghazi, Ravi Kumar, Jelani Nelson, and Pasin Manurangsi. 2023. Private Counting of Distinct and k-Occurring Items in Time Windows. In *Proc. 14th ITCS*. 55:1–55:24. https://doi.org/10.4230/LIPIcs.ITCS.2023.55
[41] Prosenjit Gupta, Ravi Janardan, and Michiel Smid. 1995. Further results on generalized intersection searching problems: counting, reporting, and dynamization. *Journal of Algorithms* 19, 2 (1995), 282–317.
[42] Moritz Hardt and Kunal Talwar. 2010. On the geometry of differential privacy. In *Proc. 42nd STOC*. 705–714. https://doi.org/10.1145/1806689.1806786
[43] Dov Harel and Robert E. Tarjan. 1984. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.* 13, 2 (1984), 338–355.
[44] Noah Johnson, Joseph P Near, and Dawn Song. 2018. Towards practical differential privacy for SQL queries. *Proceedings of the VLDB Endowment* 11, 5 (2018), 526–539.
[45] Haim Kaplan, Katrina Ligett, Yishay Mansour, Moni Naor, and Uri Stemmer. 2020. Privately Learning Thresholds: Closing the Exponential Gap. In *Proc. 33rd COLT*. 2263–2285. http://proceedings.mlr.press/v125/kaplan20a.html
[46] Haim Kaplan, Natan Rubin, Micha Sharir, and Elad Verbin. 2008. Efficient Colored Orthogonal Range Counting. *SIAM J. Comput.* 38, 3 (2008), 982–1011. https://doi.org/10.1137/070684483
[47] Tanya Khatri, Gaby G. Dagher, and Yantian Hou. 2019. Privacy-Preserving Genomic Data Publishing via Differentially-Private Suffix Tree. In *Proc. 15th EAI*. 569–584. https://doi.org/10.1007/978-3-030-37228-6_28
[48] Kunho Kim, Sivakanth Gopi, Janardhan Kulkarni, and Sergey Yekhanin. 2021. Differentially Private n-gram Extraction. In *Proc. 34th NeurIPS*. 5102–5111. https://proceedings.neurips.cc/paper/2021/hash/28ce9bc954876829eeb56ff46da8e1ab-Abstract.html
[49] Gad M Landau, Toru Kasai, Gunho Lee, Hiroki Arimura, Setsuo Arikawa, and Kunsoo Park. 2001. Linear-Time Longest-Common-Prefix Computation in Suffix Arrays and Its Applications. In *Proc. 12th CPM*. 181–192.
[50] Kasper Green Larsen and Freek Van Walderveen. 2013. Near-optimal range reporting structures for categorical data. In *Proc. 24th SODA*. 265–276.
[51] Hieu Hanh Le, Muneo Kushima, Kenji Araki, and Haruo Yokota. 2019. Differentially private sequential pattern mining considering time interval for electronic medical record systems. In *Proc. 23rd IDEAS*. 13:1–13:9. https://doi.org/10.1145/3331076.3331098
[52] Yanhui Li, Guoren Wang, Ye Yuan, Xin Cao, Long Yuan, and Xuemin Lin. 2018. PrivTS: Differentially Private Frequent Time-Constrained Sequential Pattern Mining. In *Proc. 23rd DASFAA*. 92–111. https://doi.org/10.1007/978-3-319-91458-9_6
[53] Wenjuan Liang, Wenke Zhang, and Caihong Yuan. 2023. Privately vertically mining of sequential patterns based on differential privacy with high efficiency and utility. *Sci Rep* 13 (2023), 17866. https://doi.org/10.1038/s41598-023-43030-z

[54] Veli Mäkinen, Gonzalo Navarro, Jouni Sirén, and Niko Välimäki. 2010. Storage and retrieval of highly repetitive sequence collections. *J. Comp. Biology* 17, 3 (2010), 281–308.
[55] S. Muthukrishnan. 2002. Efficient algorithms for document retrieval problems.. In *Proc. 13th SODA*, Vol. 2. 657–666.
[56] Gonzalo Navarro. 2014. Spaces, trees, and colors: The algorithmic landscape of document retrieval on sequences. *ACM Comput. Surv.* 46, 4 (2014), 1–47.
[57] Miguel Nunez-del Prado, Yoshitomi Maehara-Aliaga, Julián Salas, Hugo Alatrista-Salas, and David Megías. 2022. A Graph-Based Differentially Private Algorithm for Mining Frequent Sequential Patterns. *Applied Sciences* 12, 4 (2022). https://doi.org/10.3390/app12042131
[58] Kunihiko Sadakane. 2007. Succinct data structures for flexible text retrieval systems. *J. Discrete Algorithms* 5, 1 (2007), 12–22.
[59] Daniel D. Sleator and Robert Endre Tarjan. 1983. A data structure for dynamic trees. *J. Comput. Syst. Sci.* 26, 3 (1983), 362–391.
[60] Teresa Anna Steiner. 2024. Differentially Private Approximate Pattern Matching. In *Proc. 15th ITCS*. 94:1–94:18. https://doi.org/10.4230/LIPICS.ITCS.2024.94
[61] Salil P. Vadhan. 2017. The Complexity of Differential Privacy. In *Tutorials on the Foundations of Cryptography*, Yehuda Lindell (Ed.). Springer International Publishing, 347–450. https://doi.org/10.1007/978-3-319-57048-8_7
[62] Ning Wang, Xiaokui Xiao, Yin Yang, Ta Duy Hoang, Hyejin Shin, Junbum Shin, and Ge Yu. 2018. PrivTrie: Effective Frequent Term Discovery under Local Differential Privacy. In *Proc. 34th ICDE*. 821–832. https://doi.org/10.1109/ICDE.2018.00079
[63] Teng Wang and Zhi Hu. 2022. Local differential privacy-based frequent sequence mining. *J. King Saud Univ. Comput. Inf. Sci.* 34, 6 Part B (2022), 3591–3601. https://doi.org/10.1016/J.JKSUCI.2022.04.013
[64] Tianhao Wang, Ninghui Li, and Somesh Jha. 2018. Locally Differentially Private Frequent Itemset Mining. In *Proc. 39th SP*. 127–143. https://doi.org/10.1109/SP.2018.00035
[65] Xinyu Xiong, Fei Chen, Peizhi Huang, Miaomiao Tian, Xiaofang Hu, Badong Chen, and Jing Qin. 2018. Frequent Itemsets Mining With Differential Privacy Over Large-Scale Data. *IEEE Access* 6 (2018), 28877–28889. https://doi.org/10.1109/ACCESS.2018.2839752
[66] Shengzhi Xu, Sen Su, Xiang Cheng, Zhengyi Li, and Li Xiong. 2015. Differentially private frequent sequence mining via sampling-based candidate pruning. In *Proc. 31st ICDE*, Johannes Gehrke, Wolfgang Lehner, Kyuseok Shim, Sang Kyun Cha, and Guy M. Lohman (Eds.). IEEE Computer Society, 1035–1046. https://doi.org/10.1109/ICDE.2015.7113354
[67] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903* (2018).
[68] Chen Zeng, Jeffrey F. Naughton, and Jin-Yi Cai. 2012. On differentially private frequent itemset mining. *Proc. VLDB Endow.* 6, 1 (2012), 25–36. https://doi.org/10.14778/2428536.2428539
[69] Jun Zhang, Xiaokui Xiao, and Xing Xie. 2016. PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions. In *Proc. SIGMOD Conference 2016*. 155–170. https://doi.org/10.1145/2882903.2882928
[70] Fengli Zhou and Xiaoli Lin. 2018. Frequent Sequence Pattern Mining with Differential Privacy. In *Proc. 14th ICIC*. 454–466. https://doi.org/10.1007/978-3-319-95930-6_42

A Additional Privacy and Probability Background

Lemma A.1 (Laplace Mechanism [26]). *Let f be any function $f : \chi^* \to \mathbb{R}^k$ with L_1-sensitivity Δ_1. Let $Y_i \approx \text{Lap}(\Delta_1/\epsilon)$ for $i \in [k]$. The mechanism is defined as $A(\mathcal{D}) = f(\mathcal{D}) + (Y_1, \ldots, Y_k)$ satisfies ϵ-differential privacy.*

Lemma A.2 (Laplace tailbound). *If $Y \approx \text{Lap}(b)$, then $\Pr[|Y| \geq tb] = e^{-t}$.*

Definition A.3 (k-neighboring). Let χ be a data universe. Two data sets $\mathcal{D}$ and $\mathcal{D}'$ are called k-neighboring, if there exists a sequence $\mathcal{D} = \mathcal{D}_1 \sim \mathcal{D}_2 \sim \cdots \sim \mathcal{D}_{k-1} \sim \mathcal{D}_k = \mathcal{D}'$.

Fact 1 (Group Privacy). *Let χ be a data universe, and $\epsilon > 0$. If an algorithm $A : \chi^* \to \text{range}(A)$ is ϵ-differentially private, then for all k-neighboring $\mathcal{D} \in \chi^*$ and $\mathcal{D}' \in \chi^*$ and any set $U \in \text{range}(A)$, we have $\Pr[A(\mathcal{D}) \in U] \leq e^{k\epsilon} \Pr[A(\mathcal{D}') \in U]$.*

Lemma A.4 (Sum of Laplace Variables). *Let $Y_1, \ldots, Y_k$ be independent variables with distribution $\text{Lap}(b)$ and let $Y = \sum_{i=1}^k Y_i$. Let $0 < \beta < 1$. Then*

$$\Pr\left[|Y| > 2b\sqrt{2\ln(2/\beta)} \max\left\{\sqrt{k}, \sqrt{\ln(2/\beta)}\right\}\right] \leq \beta.$$

Proof. Apply Corollary 2.9 in [16] to $b_1 = \cdots = b_k = b$. □

Definition A.5 (Normal Distribution). The *normal distribution* centered at 0 with variance σ^2 is the distribution with the probability density function

$$f_{N(0,\sigma^2)}(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{x^2}{2\sigma^2}\right)$$

We use $Y \approx N(0, \sigma^2)$ or sometimes just $N(0, \sigma^2)$ to denote a random variable Y distributed according to $f_{N(0,\sigma^2)}$.

Lemma A.6 (Gaussian mechanism [10]). *Let f be any function $f : \chi^* \to \mathbb{R}^k$ with L_2-sensitivity Δ_2. Let $\epsilon \in (0, 1)$, $c^2 > 2\ln(1.25/\delta)$, and $\sigma \geq c\Delta_2(f)/\epsilon$. Let $Y_i \approx N(0, \sigma^2)$ for $i \in [k]$. Then the mechanism defined as $A(\mathcal{D}) = f(\mathcal{D}) + (Y_1, \ldots, Y_k)$ satisfies (ϵ, δ)-differential privacy.*

Lemma A.7 (Gaussian Tailbound). *Let $Y \approx N(\mu, \sigma^2)$. Then $\Pr[|Y - \mu| \geq t] \leq 2e^{-\frac{t^2}{2\sigma^2}}$.*

Corollary A.8. *Let f be a function $f : \chi^* \to \mathbb{R}^k$ for some universe χ with L_2-sensitvity at most Δ_2. Then there exists an (ϵ, δ)-differentially private algorithm A which for any $\mathcal{D} \in \chi^*$ outputs $A(\mathcal{D})$ satisfying $||A(\mathcal{D}) - f(\mathcal{D})||_\infty \leq 2\epsilon^{-1}\Delta_2\sqrt{\ln(2/\delta)\ln(2k/\beta)}$ with probability at least $1 - \beta$.*

Fact 2 (Sum of Normal Distributions). *Let $X_1 \approx N(0, \sigma_1^2)$ and $X_2 \approx N(0, \sigma_2^2)$ be independently drawn random variables. Then $Y = X_1 + X_2$ fulfills $Y \approx N(0, \sigma_1^2 + \sigma_2^2)$.*

B Counting with Approximate Differential Privacy

In this section, we prove Theorem 1.3. The algorithm proceeds in the same four main steps as the ϵ-differentially private algorithm. To analyze this strategy for (ϵ, δ)-differential privacy, we need the following lemma:

Lemma B.1. *Let $v \in \mathbb{R}^k$ be a k-dimensional vector such that:*

- $||v||_1 = |v[1]| + \cdots + |v[k]| \leq M$
- $|v[i]| \leq \Delta$, *for all* $i \in [1, k]$.

Then $||v||_2 \leq \sqrt{M\Delta}$.

Proof. The proof follows directly from Hölder's inequality. The inequality gives that for any $f, g \in \mathbb{R}^k$, and any $p, q \in [1, \infty]$ with $1/p + 1/q = 1$ it holds that $||f \cdot g||_1 \leq ||f||_p ||g||_q$. If we now set $p = \infty$ and $q = 1$ and $f = g = v$, we get $||v||_2^2 = ||v \cdot v||_1 \leq ||v||_\infty ||v||_1 = M\Delta$ and thus $||v||_2 \leq \sqrt{M\Delta}$. □

This together with Corollary 3.2 implies the following bound.

Corollary B.2. *For any $m \leq \ell$, the L_2-sensitivity of $(\mathrm{count}_\Delta(P, \mathcal{D}))_{P \in \Sigma^m}$ is bounded by $\sqrt{2\ell\Delta}$.*

In the following, we give the details and analysis of our approach for (ϵ, δ)-differential privacy.

B.1 Computing a Candidate Set

We show how to compute the candidate set C while satisfying (ϵ, δ)-differential privacy:

Lemma B.3. *Let $\mathcal{D} = S_1, \ldots, S_n$ be a collection of strings. For any $\epsilon > 0$, $\delta > 0$, and $0 < \beta < 1$, there exists an (ϵ, δ)-differentially private algorithm, which computes a candidate set $C \subseteq \Sigma^{[1,\ell]}$ with the following two properties, with probability at least $1 - \beta$:*

- *For any $P \in \Sigma^{[1,\ell]}$ not in C, we have*

$$\mathrm{count}_\Delta(P, \mathcal{D}) = O\left(\epsilon^{-1} \log \ell \sqrt{\ell\Delta \log(\log \ell/\delta) \log(\max\{\ell^2 n^2, |\Sigma|\}/\beta)}\right).$$

- $|C| \leq n^2\ell^3$.

Proof. First, we inductively construct sets $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$ for $j = \lfloor \log \ell \rfloor$, where $\mathcal{P}_{2^k}$ contains only strings of length 2^k which have a sufficiently high count in $\mathcal{D}$. For every k, we use an (ϵ_1, δ_1)-differentially private algorithm to construct $\mathcal{P}_{2^k}$ from $\mathcal{P}_{2^{k-1}}$, where $\epsilon_1 = \epsilon/(\lfloor \log \ell \rfloor + 1)$ and $\delta_1 = \delta/(\lfloor \log \ell \rfloor + 1)$, such that the full algorithm fulfills (ϵ, δ)-differential privacy. In a post-processing step, we construct sets C_m of strings of length m for every $1 \leq m \leq \ell$, such that every pattern of length m which is not in C_m has a small count with high probability. We define C as the union of all C_m.

Computing $\mathcal{P}_{2^0}$. First, we estimate $\mathrm{count}_\Delta(\gamma, \mathcal{D})$ of all letters $\gamma \in \Sigma$. By Corollary B.2, the sensitivity of $(\mathrm{count}_\Delta(\gamma, \mathcal{D}))_{\gamma \in \Sigma}$ is bounded by $\sqrt{2\ell\Delta}$. Let $\beta_1 = \beta/(\lfloor \log \ell \rfloor + 1)$. Using the algorithm given by Corollary A.8, we compute an estimate $\mathrm{count}^*_\Delta(\gamma, \mathcal{D})$ such that

$$\max_{\gamma \in \Sigma} |\mathrm{count}^*_\Delta(\gamma, \mathcal{D}) - \mathrm{count}_\Delta(\gamma, \mathcal{D})| \leq 2\epsilon_1^{-1} \sqrt{2\ell\Delta \ln(2/\delta_1) \ln(2|\Sigma|/\beta_1)}$$

with probability at least $1 - \beta_1$, while satisfying (ϵ_1, δ_1)-differential privacy. In the following, let $\alpha = 2\epsilon_1^{-1}\sqrt{2\ell\Delta \ln(2/\delta_1) \ln(2 \max\{\ell^2 n^2, |\Sigma|\}/\beta_1)}$. We keep a pruned candidate set $\mathcal{P}_{2^0}$ of strings of length 1 with an approximate count at least $\tau = 2\alpha$, i.e., we keep all γ with $\mathrm{count}^*_\Delta(\gamma, \mathcal{D}) \geq \tau$. If $|\mathcal{P}_{2^0}| > n\ell$, we stop the algorithm and return a fail message.

Computing $\mathcal{P}_{2^k}$ for $k = 1, \ldots, \lfloor \log \ell \rfloor$. Given $\mathcal{P}_{2^{k-1}}$ with $|\mathcal{P}_{2^{k-1}}| \leq n\ell$ and $k \geq 1$, compute $\mathcal{P}_{2^k}$ as follows: first, construct the set $\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$, i.e., all strings that are a concatenation of two strings in $\mathcal{P}_{2^{k-1}}$. There are at most $(n\ell)^2$ of these. Again by Corollary B.2, the L_2-sensitivity of count_Δ for all strings in $\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$ is bounded by $\sqrt{2\ell\Delta}$. We use the algorithm from Corollary A.8 to estimate count_Δ for all strings in $\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$ up to an additive error at most $2\epsilon_1^{-1}\sqrt{2\ell\Delta \ln(2/\delta_1) \ln(2\ell^2 n^2/\beta_1)} \leq \alpha$ with probability at least $1 - \beta_1$ and (ϵ_1, δ_1)-differential privacy. The set $\mathcal{P}_{2^k}$ is the set of all strings in $\mathcal{P}_{2^{k-1}} \circ \mathcal{P}_{2^{k-1}}$ with an approximate count at least τ. Again, if $|\mathcal{P}_{2^k}| > n\ell$, we stop the algorithm and return a fail message.

Constructing C. From $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$, we now construct, for each m which is not a power of two, a set of candidate patterns C_m, without taking $\mathcal{D}$ further into account. Specifically, for any fixed length m with $2^k < m < 2^{k+1}$, for $k = 0, \ldots, \lfloor \log \ell \rfloor$, we define C_m as the set of all strings P of length

m such that $P[0, 2^k - 1] \in \mathcal{P}_{2^k}$ and $P[m - 2^k, m - 1] \in \mathcal{P}_{2^k}$. For $m = 2^k$ for some $k \in \{0, \ldots, \lfloor \log \ell \rfloor\}$, we define $C_m = \mathcal{P}_{2^k}$. We define $C = \bigcup_{m=1}^{\ell} C_m$.

Privacy. Since there are $\lfloor \log \ell \rfloor + 1$ choices of j, and for each we run an (ϵ_1, δ_1)-differentially private algorithm for $\epsilon = \epsilon/(\lfloor \log \ell \rfloor + 1)$ and $\delta_1 = \delta/(\lfloor \log \ell \rfloor + 1)$, their composition is (ϵ, δ)-differentially private by Lemma 2.4. Since constructing C from $\mathcal{P}_{2^0}, \ldots, \mathcal{P}_{2^j}$ is post-processing, the entire algorithm is (ϵ, δ)-differentially private.

Accuracy. Since there are $\lfloor \log \ell \rfloor + 1$ choices of k, by the choice of β_1, and by the union bound, all the error bounds hold together with probability at least $1 - \beta$. Let E be the event that all the error bounds holding simultaneously. In the following, we conditioned on E. Conditioned on E, for any $P \in \mathcal{P}_{2^k}$, $k = 0, \ldots, \lfloor \log \ell \rfloor$, we have that $\text{count}_\Delta(P, \mathcal{D}) \geq \tau - \alpha \geq \alpha > 1$, i.e., it appears at least once as a substring in $\mathcal{D}$. Note that any string in $\mathcal{D}$ has at most ℓ substrings of length 2^k. Since there are n strings in $\mathcal{D}$, we have $|\mathcal{P}_{2^k}| \leq n\ell$, conditioned on E. Thus, conditioned on E, the algorithm does not abort. Additionally, any P of length 2^k which is not in $\mathcal{P}_{2^k}$ satisfies $\text{count}_\Delta(P, \mathcal{D}) < \tau + \alpha = 3\alpha$. Now, any pattern P of length $2^k < m < 2^{k+1}$ for some $k \in \{0, \ldots, \lfloor \log \ell \rfloor\}$ satisfies that $\text{count}_\Delta(P, \mathcal{D}) \leq \text{count}_\Delta(P[0, 2^k - 1], \mathcal{D})$ and $\text{count}_\Delta(P, \mathcal{D}) \leq \text{count}_\Delta(P[m - 2^k, m - 1], \mathcal{D})$. Since $P \notin C_m$ if and only if either $P[0, 2^k - 1] \notin \mathcal{P}_{2^k}$ or $P[m - 2^k, m - 1] \notin \mathcal{P}_{2^k}$, we have that $P \notin C_m$ implies $\text{count}_\Delta(P, \mathcal{D}) < 3\alpha$, conditioned on E. As $P[0, 2^k - 1]$ and $P[m - 2^k, m - 1]$ cover P completely, we have $|C_m| \leq |\mathcal{P}_{2^k}|^2 \leq (n\ell)^2$. Therefore, $|C| = \sum_{m=1}^{\ell} |C_m| \leq n^2\ell^3$. □

B.2 Heavy Path Decomposition and Properties

In the following, let T_C be the trie of a set C fulfilling the properties of Lemma B.3. As a next step, we build the heavy path decomposition for T_C, as defined in Section 3.2. We redefine the difference sequence on a path for count_Δ (in Section 3.2 we only defined it for count).

Difference sequence. For any path $p = v_0, v_1, \ldots, v_{|p|-1}$ in the trie T_C, the *difference sequence of* count_Δ *on* p is given by the $|p| - 1$ dimensional vector $\text{diff}_p(\mathcal{D})[i] = \text{count}_\Delta(\text{str}(v_i), \mathcal{D}) - \text{count}_\Delta(\text{str}(v_{i-1}), \mathcal{D})$ for $i = 1 \ldots |p| - 1$.

We need the following generalizations of Lemmas 3.6 and 3.7:

Lemma B.4. *Let $\mathcal{D}$ and $\mathcal{D}'$ be neighboring data sets, such that $\mathcal{D}' = \mathcal{D} \setminus \{S\} \cup \{S'\}$.*

(1) For any node v in the trie T_C, we have

$$|\text{count}_\Delta(\text{str}(v), \mathcal{D}) - \text{count}_\Delta(\text{str}(v), \mathcal{D}')| = |\text{count}_\Delta(\text{str}(v), S) - \text{count}_\Delta(\text{str}(v), S')|.$$

(2) For any path p with root r in the trie T_C, we have

$$||\text{diff}_p(\mathcal{D}) - \text{diff}_p(\mathcal{D}')||_1 \leq \text{count}_\Delta(\text{str}(r), S) + \text{count}_\Delta(\text{str}(v), S').$$

Proof. Item 1 follows from the definition of count_Δ. For item 2, note that if v' is a descendant of v, then $\text{str}(v)$ is a prefix of $\text{str}(v')$ and therefore $\text{count}_\Delta(\text{str}(v), S) \geq \text{count}_\Delta(\text{str}(v'), S)$. Now item 2 follows from the fact that $\text{count}_\Delta(\text{str}(v_i), S)$ is monotonically non-increasing in i for $p = v_0, v_1, \ldots, v_{|p|-1}$ with $v_0 = r$, and from $0 \leq \text{count}_\Delta(\text{str}(v_i), S) \leq \text{count}(\text{str}(r), S)$, for any descendant v_i of r. The same argument applies for S' and thus item 2 follows. □

Lemma B.5. *Let $r_0, r_1, \ldots, r_k$ be the roots of the paths of the heavy path decomposition of T_C, where r_0 is the root of T_C. Then for any string S of length at most ℓ, we have $\sum_{i=0}^{k} \text{count}_\Delta(\text{str}(r_i), S)) \leq \ell(\lceil \log |T_C| \rceil + 1) = O(\ell \log(n\ell))$.*

Proof. This lemma is a direct Corollary of Lemma 3.7. □

By Lemma B.4.1 and Lemma B.5, the L_1-sensitivity of $(\text{count}_\Delta(\text{str}(r_i), \mathcal{D}))_{i=0}^{k}$ is bounded by $2\ell(\lceil \log |T_C| \rceil + 1) = O(\ell \log(n\ell))$. Since also $\text{count}_\Delta(\text{str}(r_i), S) \leq \Delta$ for every $i = 0, \ldots, k$ and any

string S, we have that the L_2-sensitivity of $(\mathrm{count}_\Delta(\mathrm{str}(r_i), \mathcal{D}))_{i=0}^k$ is bounded by $O(\sqrt{\Delta\ell \log(n\ell)})$ by Lemma B.1. Additionally, note that since T_C has at most $n^2\ell^3$ leaves, there are at most $n^2\ell^3$ heavy paths. Together with Corollary A.8, this gives:

Corollary B.6. *Let $r_0, \ldots, r_k$ be the roots of the heavy paths of T_C. For any $\epsilon > 0$, $\delta > 0$, and $0 < \beta < 1$, there exists an (ϵ, δ)-differentially private algorithm, which estimates $\mathrm{count}_\Delta(\mathrm{str}(r_i), \mathcal{D})$ for the heavy path roots $r_0, \ldots, r_k$ up to an additive error at most $O(\epsilon^{-1}\sqrt{\Delta\ell \log(n\ell) \ln(1/\delta) \ln(k/\beta)}) = O(\epsilon^{-1}\sqrt{\Delta\ell \ln(1/\delta)} \ln(n\ell/\beta))$ with probability at least $1 - \beta$.*

Additionally, Lemma B.5 and Lemma B.4.2 give that the *sum of* L_1-sensitivities of the difference sequences *over all heavy paths* is bounded by $2\ell(\lceil \log |T_C| \rceil + 1)$, and Lemma B.4.2 gives that the L_1-sensitivity of the difference sequence on *one* heavy path is bounded by Δ. As a next step, we show how we can compute all *prefix sums* of the difference sequences of all heavy paths with an error roughly $O(\sqrt{\ell(\lceil \log |T_C| \rceil + 1)\Delta})$ (up to poly-logarithmic terms). To do this, we extend Lemma 3.9 to (ϵ, δ)-differential privacy.

Lemma B.7. *Let χ^* be any universe of possible data sets. Let $a^{(1)}, \ldots, a^{(k)}$ be k functions, where the output of every function is a sequence of length T, i.e. $a^{(i)} : \chi^* \to \mathbb{N}^T$ for all $i \in [1, k]$. If $a^{(1)}, \ldots, a^{(k)}$ fulfill the following properties for any two neighboring x and x' from χ^*:*

- $\sum_{j=1}^T |a^{(i)}(x)[j] - a^{(i)}(x')[j]| \leq \Delta$ *for all* $i = 1, \ldots, k$,
- $\sum_{i=1}^k \sum_{j=1}^T |a^{(i)}(x)[j] - a^{(i)}(x')[j]| \leq L$,

then for any $\epsilon > 0$, any $\delta > 0$, and $0 < \beta < 1$, there exists an (ϵ, δ)-differentially private algorithm computing for every $i \in [1, k]$ all prefix sums of $a^{(i)}(x)$ with additive error at most $O(\epsilon^{-1}\sqrt{L\Delta \ln(1/\delta) \log(Tk/\beta)} \log T)$ with probability at least $1 - \beta$.

We defer the proof to the full version [7]. Lemma B.7 now gives the following corollary with $L = 2\ell(\lceil \log |T_C| \rceil + 1) = O(\ell \log(n\ell))$, $k \leq n^2\ell^3$, and $T = \ell$.

Corollary B.8. *Let $p_0, \ldots, p_k$ be all heavy paths of T_C. For any $\epsilon > 0$, $\delta > 0$, and $0 < \beta < 1$, there exists an (ϵ, δ)-differentially private algorithm, which estimates $\sum_{j=1}^i \mathrm{diff}_{p_m}(\mathcal{D})[j]$ for all $i = 1, \ldots, |p_m|$ and all $m = 0, \ldots, k$ up to an additive error at most*

$$O\left(\epsilon^{-1}\sqrt{\Delta\ell \log(n\ell) \ln(1/\delta) \log(\ell k/\beta)} \log \ell\right) = O\left(\epsilon^{-1}\sqrt{\Delta\ell \ln(1/\delta)} \log(n\ell/\beta) \log \ell\right).$$

B.3 Putting It Together

Our full algorithm runs the algorithms given by Lemma B.3, Corollary B.6 and Corollary B.8 in sequence with privacy parameters $\epsilon' = \epsilon/3$, $\delta' = \delta/3$, and failure probability $\beta' = \beta/3$. For a heavy path $p = v_0, v_1, \ldots, v_{|p|-1}$ with root $r = v_0$, let $\mathrm{count}^*_\Delta(\mathrm{str}(r), \mathcal{D})$ be the approximate counts computed by the algorithm from Corollary B.6, and let $\mathrm{sums}^*_p[i]$ be the approximate estimate of $\sum_{j=1}^i \mathrm{diff}_p(\mathcal{D})[j]$. Then for any node v_i on the path with $i > 0$, we compute $\mathrm{count}^*_\Delta(\mathrm{str}(v_i), \mathcal{D}) = \mathrm{count}^*_\Delta(\mathrm{str}(r), \mathcal{D}) + \mathrm{sums}^*_p[i]$. Let α be the sum of the error bounds for Corollary B.6 and Corollary B.8 which each hold with probability $1 - \beta'$. We have $\alpha = O\left(\epsilon^{-1}\sqrt{\Delta\ell \ln(1/\delta)} \log(n\ell/\beta) \log \ell\right)$. As a final step, we prune T_C by traversing the trie from the root, and for any node v with $\mathrm{count}^*_\Delta(\mathrm{str}(v_i), \mathcal{D}) < 2\alpha$, we delete v and its subtree. We return the resulting pruned trie T^* together with approximate counts $\mathrm{count}^*_\Delta(\mathrm{str}(v), \mathcal{D})$ for every node $v \in T^*$.

By the composition theorem (Lemma 2.4), this algorithm is (ϵ, δ)-differentially private. By a union bound, all its error guarantees hold together with probability $1 - 3\beta' = 1 - \beta$. Thus, we get with probability $1 - \beta$ a trie T^*, with the following properties:

- By Corollaries B.6 and B.8, for each node $v \in T^*$

$$|\text{count}^*_\Delta(\text{str}(v), \mathcal{D}) - \text{count}_\Delta(\text{str}(v), \mathcal{D})| \leq \alpha = O\left(\epsilon^{-1}\sqrt{\Delta \ell \ln(1/\delta)} \log(n\ell/\beta) \log \ell\right)$$

- Every string $P \in \Sigma^{[1,\ell]}$ which is not present T^* was either: (i) not present in C, in which case

$$\text{count}_\Delta(P, \mathcal{D}) = O\left(\epsilon^{-1} \log \ell \sqrt{\ell \Delta \log(\log \ell/\delta) \log(\max\{\ell^2 n^2, |\Sigma|\}/\beta)}\right)$$

  by Lemma B.3; or (ii) deleted in the pruning process of T_C, in which case

$$\text{count}_\Delta(P, \mathcal{D}) < 3\alpha = O\left(\epsilon^{-1}\sqrt{\Delta \ell \ln(1/\delta)} \log(n\ell/\beta) \log \ell\right).$$

- Any string $P \in \sum^{[1,\ell]}$ which is present in T^* has a count at least $2\alpha - \alpha > 1$. Therefore, T^* has at most $O(n\ell^2)$ nodes.

To query the resulting data structure for a pattern P, we match P in the trie, and if there is a node v with $\text{str}(v) = P$, we return the approximate count $\text{count}^*_\Delta(\text{str}(v), \mathcal{D})$ saved at v. If P is not present, we return 0. This requires $O(|P|)$ time. Together, this gives Theorem 1.3.

C Counting Functions on Trees

In this section, show an improvement of Theorem 1.5 for (ϵ, δ)-differential privacy if the sensitivity of the count function for every node is bounded.

Theorem C.1. *Let χ be a universe and $T = (V, E)$ a tree height h. Let $L = l_1, \ldots, l_k \subseteq V$ be the set of leaves in T. Let $c : V \times \chi^* \to \mathbb{N}$ a function, which takes as input a node v from T and a data set $\mathcal{D}$ from χ^*, and computes a count with the following properties:*

- *$c(v, \mathcal{D}) \leq \sum_{u \text{ is a child of } v} c(u, \mathcal{D})$ for all $v \in V \setminus L$ and $\mathcal{D} \in \chi^*$;*
- *$\sum_{i=1}^k |c(l_i, \mathcal{D}) - c(l_i, \mathcal{D}')| \leq d$, for all neighboring $\mathcal{D}$ and $\mathcal{D}'$ and some $d \in \mathbb{N}$.*
- *$|c(v, \mathcal{D}) - c(v, \mathcal{D}')| \leq \Delta$, for all $v \in T$ and all neighboring $\mathcal{D}$ and $\mathcal{D}'$.*

Then for any $\epsilon > 0$, $\delta > 0$, and $0 < \beta < 1$, there exists an (ϵ, δ)-differentially private algorithm computing $\hat{c}(v)$ for all nodes in T such that

$$\max_{v \in V} |\hat{c}(v, \mathcal{D}) - c(v, \mathcal{D})| = O(\epsilon^{-1}\sqrt{d\Delta \log |V| \log(1/\delta) \log(hk/\beta)} \log h)$$

with probability at least $1 - \beta$.

We defer the proof to the full version [7].

D Lower Bound

We first state a theorem from [61] (Theorem 7.5.14), which immediately implies a lower bound on the additive error of any ϵ-differentially private algorithms solving Document Count and Substring Count. In Theorem 1.6, we extend this lower bound to also hold for the easier problem where we want to find patterns of a fixed length which have a count above a given threshold (i.e., for q-Gram Mining). In the following, for any query $q : \chi \to \{0, 1\}$, the corresponding *counting query* $q : \chi^n \to [0, 1]$ is defined as $q(x) = \frac{1}{n}\sum_{i=1}^n q(x_i)$.

Lemma D.1 ([61], Theorem 7.5.14). *Let $Q = q : \chi \to \{0, 1\}$ be any class of counting queries that can distinguish all elements from χ. That is, for all $w \neq w' \in \chi$, there exists a query $q \in Q$ such that $q(w) \neq q(w')$. Suppose $\text{Alg} : \chi^n \to \mathbb{R}^Q$ is an (ϵ, δ)-differentially private algorithm that with high probability answers every query in Q with error at most α. Then*

$$\alpha = \Omega\left(\min\left(\frac{\log|\chi|}{n\epsilon}, \frac{\log(1/\delta)}{n\epsilon}, 1/2\right)\right).$$

For ϵ-differential privacy, $\delta = 0$, and therefore the second bound in the minimum is always ∞. For the universe $X = \Sigma^\ell$, let Q be the family of point functions over X, i.e., Q contains for every $S \in X$ a query q_S such that $q_S(S') = 1$ if and only if $S' = S$, and 0 otherwise. Clearly, Q distinguishes all elements from Σ^ℓ. We note that an algorithm solving Document Count and Substring Count can answer every query in Q on a data set X^n, by reporting the counts of every string of length ℓ and dividing the count by n. Thus, the ϵ-differentially private algorithms must have an error of $\Omega(\epsilon^{-1} \log X) = \Omega(\min(\epsilon^{-1}\ell \log|\Sigma|, n))$. In the following, we extend this lower bound in two ways:

(1) We show that it holds even if we only want to output which patterns have a count approximately above a given threshold τ;
(2) We show that it holds even if we restrict the output to patterns of a fixed length m, for any $m \geq 2 \log \ell$.

Proof of Theorem 1.6. Let Alg be the assumed ϵ-differentially private algorithm with error at most α. Assume $\alpha < n/2$, else we have $\alpha = \Omega(n)$. Let $B = 2\alpha \leq n$. Fix two symbols $0, 1 \in \Sigma$ and let $\hat{\Sigma} = \Sigma \setminus \{0, 1\}$. For any set of k patterns $P_1, \ldots, P_k$ from $\hat{\Sigma}^{m/2}$, where $k = \frac{\ell}{m}$, we construct a string $S_{P_1,\ldots,P_k}$ as follows: For any $i \leq \ell$, let c_i be the $m/2$-length binary code of i. Note that $m \geq 2\lceil \log \ell \rceil$. We define $S_{P_1,\ldots,P_k} = P_1 c_1 P_2 c_2 \ldots P_k c_k$.

For a string of this form, we define $\mathcal{D}(P_1, \ldots, P_k)$ as the data set which contains B copies of $S_{P_1,\ldots,P_k}$ and $n - B$ copies of 0^ℓ. For any two sets of patterns $P_1, \ldots, P_k$ from $\hat{\Sigma}^{m/2}$ and $P'_1, \ldots, P'_k$ from $\hat{\Sigma}^{m/2}$, the data sets $\mathcal{D}(P_1, \ldots, P_k)$ and $\mathcal{D}(P'_1, \ldots, P'_k)$ are B-neighboring (see Definition A.3).

Set $\tau = B/2$. We call $\mathcal{E}(P_1, \ldots, P_k)$ the event that the output $\mathcal{P}$ of the algorithm contains the strings $P_1c_1, P_2c_2, \ldots, P_kc_k$, and it does not contain any other strings suffixed by $c_1, c_2, \ldots, c_k$. For $\mathcal{D}(P_1, \ldots, P_k)$, we have that $\text{count}_\Delta(P_i c_i, \mathcal{D}(P_1, \ldots, P_k)) = B \geq \tau + \alpha$, for all $i = 1, \ldots, k$, independent of Δ. Further, any other string with suffix c_i for any $i = 1, \ldots, k$ has a count of $0 \leq \tau - \alpha$. Thus

$$\Pr[\text{Alg}(\mathcal{D}(P_1, \ldots, P_k)) \in \mathcal{E}(P_1, \ldots, P_k)] \geq 2/3.$$

Since we assume that Alg is ϵ-differentially private, this gives by Fact 1 (see appendix), for any other set $P'_1, \ldots, P'_k$ of patterns from $\hat{\Sigma}^{m/2}$,

$$\Pr[\text{Alg}(\mathcal{D}(P'_1, \ldots, P'_k)) \in \mathcal{E}(P_1, \ldots, P_k)] \geq e^{-B\epsilon} 2/3.$$

Note that by construction, $\mathcal{E}(P_1, \ldots, P_k) \cap \mathcal{E}(P'_1, \ldots, P'_k) = \emptyset$ for $(P_1, \ldots, P_k) \neq (P'_1, \ldots, P'_k)$. Thus, we have for a fixed set $(P'_1, \ldots, P'_k)$ of patterns,

$$\begin{aligned} 1 &\geq \sum_{(P_1,\ldots,P_k) \in (\hat{\Sigma}^{m/2})^k} \Pr[\text{Alg}(\mathcal{D}(P'_1, \ldots, P'_k)) \in \mathcal{E}(P_1, \ldots, P_k)] \\ &\geq \sum_{(P_1,\ldots,P_k) \in (\hat{\Sigma}^{m/2})^k} e^{-B\epsilon} 2/3 = (|\Sigma| - 2)^{mk/2} e^{-B\epsilon} 2/3. \end{aligned}$$

Solving for B, this gives

$$B \geq \frac{1}{\epsilon}\left(\frac{mk}{2} \log(|\Sigma| - 2) + \log(2/3)\right) = \Omega(\epsilon^{-1}\ell \log|\Sigma|).$$

The theorem follows since $\alpha = B/2$. □

Received December 2024; revised February 2025; accepted March 2025