PrivTrie: Effective Frequent Term Discovery under Local Differential Privacy

Ning Wang[1,6], Xiaokui Xiao[2], Yin Yang[3], Ta Duy Hoang[4], Hyejin Shin[5], Junbum Shin[5], Ge Yu[1,6,7]

[1,6]*School of Computer Science and Engineering, Northeastern University, China*
[1,6]*Key Laboratory of Medical Image Computing of Ministry of Education, Northeastern University, China*
[2]*School of Computing, National University of Singapore, Singapore*
[3]*College of Science and Engineering, Hamad Bin Khalifa University, Qatar*
[4]*School of Computer Science and Engineering, Hanoi University of Science and Technology, Vietnam*
[5]*Samsung Research, Samsung Electronics, Korea*
[7]*School of Information, Liaoning University, China*
[1]wning1217@gmail.com,[2]xkxiao@ntu.edu.sg,[3]yyang@qf.org.qa,[4]hoangduybk56@gmail.com,
[5]{hyejin1.shin,junbum.shin}@samsung.com, [6]yuge@mail.neu.edu.cn

***Abstract*—A mobile operating system often needs to collect frequent new terms from users in order to build and maintain a comprehensive dictionary. Collecting keyboard usage data, however, raises privacy concerns. Local differential privacy (LDP) has been established as a strong privacy standard for collecting sensitive information from users. Currently, the best known solution for LDP-compliant frequent term discovery transforms the problem into collecting n-grams under LDP, and subsequently reconstructs terms from the collected n-grams by modelling the latter into a graph, and identifying cliques on this graph. Because the transformed problem (i.e., collecting n-grams) is very different from the original one (discovering frequent terms), the end result has poor utility. Further, this method is also rather expensive due to clique computation on a large graph.**

In this paper we tackle the problem head on: our proposal, PrivTrie, directly collects frequent terms from users by iteratively constructing a trie under LDP. While the methodology of building a trie is an obvious choice, obtaining an accurate trie under LDP is highly challenging. PrivTrie achieves this with a novel adaptive approach that conserves privacy budget by building internal nodes of the trie with the lowest level of accuracy necessary. Experiments using real datasets confirm that PrivTrie achieves high accuracy on common privacy levels, and consistently outperforms all previous methods.

I. Introduction

Nowadays most smartphones rely on a software keyboard as the primary means for text input. Typing on such a virtual keyboard is often slow and error-prone. To alleviate the problem, the mobile operating system usually helps the user by suggesting words based on a dictionary. To make accurate word suggestions, it is critical to build a comprehensive dictionary that contains not only traditional words, but new terms that are freshly minted by the users. To learn about such terms, ideally the mobile OS could collect term usage frequency information from the user, and report it to an *aggregator*; the latter then identifies the top-k most frequent new terms among all users, includes them in the mobile keyboard dictionary, and pushes the updates to users' devices. However, collecting term usage data directly would violate users' privacy.

A promising methodology for collecting sensitive data without violating users' privacy is *local differential privacy* (*LDP*), which has been applied in popular software systems such as Apple iOS [1], macOS [2], and Google Chrome browser [3]. For instance, Apple already collects emoji usage data from iOS users under LDP [1], [4]. Specifically, LDP guarantees that no adversary, including the aggregator itself, can possibly infer the exact values of private information with high confidence, regardless of the adversary's background knowledge. This is done by injecting random noise into the data values at each individual user locally, and only transmitting the perturbed version to the aggregator. The goal of LDP mechanism design, therefore, is to maximize the utility of statistical information (e.g., term frequencies) computed from the collected noisy data, while satisfying LDP.

Currently, to our knowledge the best known solution for discovering frequent new terms under LDP is an augmented version of *RAPPOR* [5], which is used in Google's Chrome browser. Hereafter, we refer to this method as A-RAPPOR. The main challenge addressed in A-RAPPOR is that there is an enormous domain for possible new terms. For instance, 8 lower-case English letters can have over 200 billion possible combinations (26^8). Collecting the frequency of each of these possible combinations individually would incur prohibitively high cost and/or result error, as we explain in Section II-B. A-RAPPOR handles this problem by collecting n-grams instead of full terms. After that, A-RAPPOR constructs a graph where each node corresponds to an n-gram, computes cliques on this graph, and reconstructs terms from the obtained cliques, which we elaborate further in Section II-B.

The problem with A-RAPPOR is that the transformed problem (i.e., collecting n-gram frequencies) is very different from the original one (discovering new terms). In particular, cliques obtained from the n-gram graph often do not correspond to real terms, let alone frequent ones. Consequently, the results obtained by A-RAPPOR have rather low utility, as shown in our experiments. In this paper we tackle the original problem head on. In particular, the proposed solution, called *PrivTrie*, estimates frequencies of candidate terms (rather than n-grams) by iteratively constructing a *trie* under LDP. Intuitively, a trie

is a natural choice for identifying frequent terms, since all prefixes of a frequent term must be frequent as well, which enables effective pruning. Interestingly, our experiments reveal that a baseline solution using a simple LDP-compliant trie obtains result accuracy comparable with A-RAPPOR, at a fraction of the latter's computational cost.

The challenge, however, is to achieve high accuracy for new term discovery under LDP, not merely RAPPOR-level accuracy. This is especially difficult under LDP, where the mechanism can only access individual records locally, in contrast to the traditional, centralized setting of differential privacy, where all private data are available. PrivTrie tackles this challenge through a novel LDP-compliant trie-construction algorithm. The key idea is to estimate only a coarse-grained frequency for each prefix (corresponding to an internal node in the trie), such that the estimated frequency is sufficiently accurate to determine whether or not to split the node. This allows PrivTrie to focus most of the privacy budget on estimating the frequencies of actual terms, i.e., leaf nodes in the trie. Extensive experiments using two real datasets confirm that on both datasets, PrivTrie achieves an Fscore of over 0.8 for identifying top-50 frequent new terms with a privacy budget of $\epsilon = 2$ (the ϵ value used in macOS [2]), whereas all previous methods lead to rather low accuracy.

In the following, Section II provides necessary background on LDP and reviews existing solutions. Section III describes a baseline approach using an LDP-compliant trie. Section IV presents PrivTrie. Section V contains an extensive set of experiments, Section VI provides additional related work, and Section VII concludes with directions for future work.

II. Background

Section II-A provides preliminaries on local differential privacy. Section II-B presents A-RAPPOR, the current state of the art for new term discovery under LDP.

A. Local Differential Privacy

The LDP setting involves an aggregator and a number of (say, n) individual users, each of which possesses a data record containing private information. Each user u_i ($1 \leq i \leq n$) locally perturbs her record t_i to obtain a randomized version $\tilde{t_i}$, and sends $\tilde{t_i}$ to the aggregator. The latter then computes statistical results based on the collected noisy records $\tilde{t_1}$, $\tilde{t_2}$, $\cdots$, $\tilde{t_n}$. The perturbation in LDP ensures that the aggregator, or any third party, cannot infer the exact record t_i from the perturbed one $\tilde{t_i}$ with high confidence, which is controlled by a parameter ϵ. Formally, LDP is defined as follows.

Definition II.1 (ϵ-LDP)**.** *A randomized function f satisfies ϵ-LDP if and only if for any two input tuples $t, t' \in D(f)$ (where $D(f)$ denotes the domain of function f), and for any possible output $\tilde{t}$ of f, we have:*

$$\Pr\left[f(t) = \tilde{t}\right] \leq e^{\epsilon} \cdot \Pr\left[f(t') = \tilde{t}\right].$$

Intuitively, in Definition II.1, when ϵ is small, ϵ-LDP ensures that given the perturbed tuple $\tilde{t}$, the adversary cannot infer whether the original tuple is t or t' with high confidence. A smaller ϵ leads to stronger privacy protection, and vice versa.

Since LDP is based on the theory of differential privacy, it inherits the composability properties of the latter. We utilize the sequential composition property of LDP [6], as follows.

Lemma II.1 (Sequential Composition)**.** *Given m independent randomized functions f_1, f_2, $\cdots$, f_m such that each function f_i ($1 \leq i \leq m$) satisfies ϵ_i-LDP, then, any function $g(f_1, ..., f_m)$ satisfies $(\sum_{i=1}^{m} \epsilon_i)$-LDP.*

The above lemma indicates that we can view the privacy parameter ϵ as a budget. Specifically, we can apply a series of LDP-compliant mechanisms, each assigned a portion of the privacy budget ϵ; by sequential composition, the series of mechanisms as a whole satisfies ϵ-LDP.

Fundamental LDP mechanisms. A basic mechanism for enforcing ϵ-LDP is Randomized Response (RR) [7], which deals with the situation that (i) each user possesses a single bit (i.e., either 0 or 1) of information, and (ii) the aggregator aims to compute the number of users who possesses 1, under ϵ-LDP. Algorithm 1 presents a variant of the RR mechanism used in recent work [8], called optimized RR. Specifically, each user u_i executes optimized RR with her private record t_i (i.e., a single bit) as input, and reports the output $\tilde{t_i}$ to the aggregator. The latter then computes $\frac{\sum_i \tilde{t_i} - n \cdot p_0}{p_1 - p_0}$, as an estimate for the target value $\sum_i t_i$, where $p_0 = 1/(e^{\epsilon} + 1)$ and $p_1 = 0.5$. Note that the output of optimized RR can have only two possible values, both of which can be calculated at the aggregator without private information; hence, it suffices for each user to transmit only a single bit to the aggregator, which minimizes communication overhead.

Optimized RR can be extended to applications where each user's record is not a single bit. One such application that is related to our problem is histogram estimation for a categorical attribute, in which each user u_i possesses a categorical value t_i [4]), and the aggregator's goal is to estimate the frequency of each value in the domain, under LDP. A common approach is to transform t_i into a bit vector (e.g., using one-hot encoding [3]), and apply optimized RR to each bit in the vector [8], [9].

Algorithm 1: Optimized Randomized Response(t, ϵ)

Input : vector $t \in \{0, 1\}$ and privacy parameter ϵ
Output: vector $\tilde{t} \in \{0, 1\}$

$p_1 = 0.5$, $p_0 = \frac{1}{e^{\epsilon}+1}$;
if $t = 1$ **then**
  $p = p_1$;
else
  $p = p_0$;
Sample a Bernoulli variable $\tilde{t}$ such that $\Pr[\tilde{t} = 1] = p$;
return $\tilde{t}$;

B. RAPPOR

RAPPOR [3], [5] is Google's solution for collecting data from users of the Chrome browser under LDP [10]. The main

mechanism for enforcing LDP in RAPPOR is RR, explained in Section II-A, with some optimizations leveraging *Bloom filters* [11]. In [3], the authors focus on collecting statistics over simple categorical values, e.g., the user's operating system. A subsequent work [5] augments RAPPOR to estimate frequencies of arbitrary strings such as URLs and image tags, where the domain of such strings is not known in advance. Specifically, in this setting, each user possesses a string of length M; a string shorter than M characters is padded, in order not to reveal the true length of the string to the aggregator. These strings can be arbitrary, and are not known to the aggregator. The goal is to discover frequent strings among the users, while satisfying LDP.

A naive solution would be to directly collect the frequency of each possible string, e.g., using RR described in Section II-A. The problem, however, is that there is a vast space for length-M strings. In particular, let $\mathcal{I}$ be the alphabet and $|\mathcal{I}|$ be its cardinality. Then, the domain size for possible strings is $|\mathcal{I}|^M$. The sheer computation cost required to estimate $|\mathcal{I}|^M$ frequencies is prohibitive, unless $\mathcal{I}$ and M are very small.

A-RAPPOR [5] addresses the above problem by first computing a small candidate set C of potentially frequent strings under LDP, using a portion ϵ_1 of the total privacy budget ϵ; subsequently, the method estimates the frequency of each candidate using the remaining privacy budget $\epsilon - \epsilon_1$. By sequential composition (Lemma II.1), this scheme satisfies ϵ-LDP. To compute C, A-RAPPOR requires that each user u_i randomly sample two n-grams from her string t_i, where $n \ll M$. The aggregator then collects the two n-grams from each user using RAPPOR, and applies a sophisticated algorithm that involves expectation maximization iterations [12] on the collected information to derive C. In particular, A-RAPPOR constructs a graph G where each node corresponds to an n-gram, and each edge connects two n-grams that frequently appear together. From this graph, A-RAPPOR computes K-cliques where $K = M/n$, reconstructs a term from each such clique, and includes the term in the candidate set C.

As we show in the experiments in Section V, the result accuracy of A-RAPPOR is rather unsatisfactory. One reason is that the n-gram graph G constructed by A-RAPPOR is plagued with noise, and hence, the K-cliques corresponding to real terms are often absent from G due to missing edges. Further, A-RAPPOR incurs tremendous computational overhead for the aggregator, due to expensive expectation-maximization iterations and clique computations. Perhaps for these reasons, the implementation of RAPPOR in Chrome does not yet include A-RAPPOR [10].

Finally, we mention that Apple has implemented a proprietary algorithm for new term discovery under LDP in iOS [1] and macOS [2], but the full details of the algorithm are not revealed. Specifically, patents [13], [14] show that the algorithm is based on n-grams, similar to A-RAPPOR. Ref. [2] reports that the algorithm for learning new words is called "CountMedianSketch" in macOS, but does not contain technical details of this algorithm. Hence, A-RAPPOR remains the best known solution for new term discovery under LDP.

C. Concurrent Approaches to Heavy Hitter Computation under Local Differential Privacy

Bassily et al. [15] and Wang et al. [16] independently developed methods for computing heavy hitters from a large domain, under ϵ-LDP. Their main ideas resemble the proposed solutions in that they also involve a prefix tree. Specifically, TreeHist [15] transforms users' items (new terms in our context) to binary strings, and builds a binary prefix tree to compute frequent strings. Users are divided into disjoint groups, each of which are responsible for building one level of the prefix tree, using a technique based on count sketches [17]. In particular, each group of users are further divided into a number of (say, t) subgroups, each of which computes an independent estimate for the support for each node in the corresponding level of the tree, and the count sketch returns the median of these t support values. After that, for each node, TreeHist splits it if its estimated support exceeds a threshold value.

Similar to TreeHist, PEM [16] is also based on a binary prefix tree. Unlike TreeHist, however, in PEM the number of user groups can be smaller than the height of the tree, in which case each group is assigned multiple adjacent levels of the tree. Essentially, this design transforms the binary prefix tree into one with a higher fanout, since multiple consecutive levels are flattened into one, and assigned to the same group of users. Then, PEM optimizes result accuracy by setting the number of groups through a careful theoretical analysis. According to our experiments in Section V, TreeHist and PEM obtain higher accuracy than A-RAPPOR, and their performance is better than our baseline approaches in some settings. Nevertheless, our main proposal PrivTrie consistently outperforms both TreeHist and PEM with clear margins.

III. A Simple Trie-Based Approach

Before presenting our main solution PrivTrie for new term discovery under LDP, we first describe a simple approach called BSL based on an LDP-compliant trie. The main purpose of describing BSL is to separate the basics for utilizing a trie to solve our problem (shared by both BSL and PrivTrie) with the key ideas of PrivTrie, presented in Section IV. Meanwhile, BSL also serves as a more efficient baseline than A-RAPPOR. As we demonstrate in Section V, BSL obtains accuracy comparable to A-RAPPOR (though rather low), and scales to large datasets (where A-RAPPOR fails) with small computation and communication costs. Table I lists frequent notations used throughout the paper.

Our problem setting is similar to that in A-RAPPOR [5]. Specifically, we assume that all users share the same keyboard dictionary, and each user u_i possesses a new term t_i that is not included in the dictionary. Each such term t_i consists of arbitrary characters from a known alphabet $\mathcal{I}$. If u_i possesses multiple such terms, she simply selects one randomly. This design is motivated by two observations: first, the random sampling discounts the influence of users with a large number of new terms, e.g., a user who frequently converses in a different language. Second and more importantly, it avoids splitting the

TABLE I
LIST OF FREQUENT NOTATIONS

Notation	Meaning
n	number of individual users
k	number of most frequent new terms to identify
$\mathcal{I}$	set of possible characters in a term
ℓ	maximum term length (used in BSL but not in PrivTrie)
$p(v)$	prefix associated with node v
$c(v), c'(v)$	exact and estimated support of prefix $p(v)$
θ	support threshold for deciding if a node should split

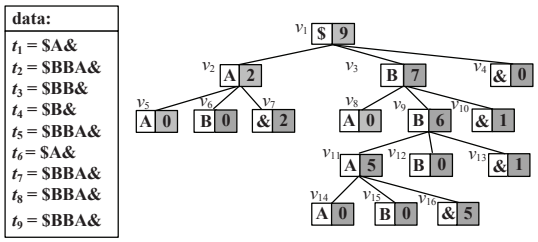


Fig. 1. Example of trie

privacy budget over multiple terms, which negatively affects result accuracy. An aggregator aims to collect top-k most frequent new terms among all users under ϵ-LDP. Formally, given a term t, we define its support $supp(t)$ as the number of users that possess t. Then, the aggregator's goal is to identify k terms with the highest supports among all possible terms.

BSL follows a similar framework as A-RAPPOR described in Section II-B. Specifically, BSL partitions the total privacy budget ϵ into $\epsilon = \epsilon_1 + \epsilon_2$, collects a candidate set C of new terms under ϵ_1-LDP, and selects top-k most frequent candidates in C under ϵ_2-LDP, using optimized RR (described in Section II-A). By sequential composition (Lemma II.1), the two steps as a whole satisfies ϵ-LDP.

BSL differs from A-RAPPOR mainly in the computation of the candidate set C. In particular, BSL iteratively constructs a trie of users' new terms in a top-down manner. Fig. 1 illustrates an example trie, which is a tree structure where (i) the root (i.e., v_1) contains a special symbol not in $\mathcal{I}$ (denoted by "\$"), which indicates the beginning of a term, (ii) each non-leaf node has $|\mathcal{I}| + 1$ children, each of which contains either a character in $\mathcal{I}$, or a special symbol (denoted by "&") that signifies the end of a term. Further, each node v corresponds to a *prefix* $p(v)$, which can be obtained by concatenating the characters of each node along the path from the root to v, e.g., $p(v_9) =$ "$\$BB$". In addition, for each node v, the aggregator aims to estimate the support $c(v)$ of prefix $p(v)$ (shown in shaded boxes in Fig. 1), i.e., the number of users whose term starts with prefix $p(v)$. In the non-private setting, a leaf is either associated with symbol "&", or has support 0. Clearly, a trie indexes terms by grouping those with the same prefix (e.g., t_2, t_3, t_5, t_7, t_8, t_9) under the same branch (rooted at v_9).

In BSL, the aggregator builds a trie under ϵ_1-LDP, and adds each complete term in the trie to the candidate set C. Algorithm 2 summarizes the trie construction procedure at the aggregator. Specifically, the aggregator initializes the trie with a root node (Line 1). Then, starting from the root, the aggregator iteratively splits nodes, as follows. Given a non-leaf node v, the aggregator collects data under LDP to compute an estimate $c'(v)$ of $c(v)$, the support for v's corresponding prefix (Line 9). Then (Lines 10-13), it checks if $c'(v) < \theta$, where θ is a pre-defined threshold that will be discussed later. If so, v is marked as pruned; otherwise, $|\mathcal{I}| + 1$ children are appended to v, each corresponding to either a character in $\mathcal{I}$ or the term-end symbol "&". After that, BSL continues to the next iteration, and attempts to split another non-leaf node in the trie. Finally, the algorithm returns the candidate set C consisting of complete terms in the trie (Lines 14-17).

Algorithm 2: BSL_Build_Trie(ϵ_1, θ, ℓ)

```
1  Initialize the trie T with a single root node v_r containing the
     term-start symbol $, and mark v_r as unvisited;
2  while there exists an unvisited non-leaf node v do
3      Mark v as visited;
4      if length of p(v) ≥ ℓ + 2 then
5          Continue;
6      for each character i ∈ (I ∪ {"&"}) do
7          Add a child v_c of v containing character i;
8          Compute the prefix p(v_c) of v_c;
9          Apply optimized RR to collect information from all
             users to estimate c'(v_c) of c(v_c) = supp(p(v_c)),
             using privacy budget ε1/(ℓ+1);
10         if c'(v_c) < θ then
11             Mark v_c as pruned;
12         else
13             Mark v_c as unvisited;
14 Initialize candidate set C to empty;
15 for each leaf node v_l that contains symbol "&" do
16     Add p(v_l) to C;
17 return C.
```

It remains to clarify (i) the amount of privacy budget spent in each invocation of optimized RR, which estimates the support of a prefix, and (ii) the value of threshold θ. We first focus on (i). BSL requires a pre-defined maximum length of a term ℓ, which can be obtained using a LDP-compliant procedure consuming a part of the privacy budget, i.e., the total privacy budget ϵ is split into $\epsilon_1 + \epsilon_2 + \epsilon_3$, in which ϵ_1 and ϵ_2 are used as described above, and ϵ_3 is for determining ℓ. We discuss the details of this procedure in Appendix. Given ℓ, the maximum depth of the trie is $\ell+2$, since the trie involves two special symbols "\$" and "&". Note that the root's support is simply the total number of users n, which is known to the aggregator in the LDP setting (since the aggregator needs to communicate with each user). Accordingly, BSL allocates to each call of optimized RR a privacy budget of $\frac{\epsilon_1}{\ell+1}$. The correctness of this choice is established by Lemma III.1.

Lemma III.1. *Algorithm 2 satisfies ϵ_1-LDP.*

Proof: Let V_i be the set of nodes on the i-th level in

trie. Then optimized RR will be invoked $|V_i|$ times to collect supports of nodes in V_i from all users. Towards this end, each user u_j submits a binary vector e_j with size of $|V_i|$, and each bit of e_j is associated with one node v in V_i (termed $e_j(v)$). $e_j(v)$ is 1 if $p(v)$ is a prefix of u_j's term. This happens at most one time for nodes in V_i. Other bits in e_j are zero. Assume that the $\frac{\epsilon_1}{\ell+1}$ budget is allocated. Then the process of collecting such binary vector satisfies $\frac{\epsilon_1}{\ell+1}$-LDP as analyzed in Ref. [8]. Further, since the trie has $\ell+1$ levels apart from the root, users are totally applied $\ell+1$ times to collect supports of nodes on such levels. Then we have the claim based on Lemma II.1. ■

Next we clarify the choice of θ, i.e., the support threshold for splitting an internal node in the trie. The main idea is that given a node v and its corresponding prefix $p(v)$, if the estimated support of $p(v)$ is lower than the expected noise level, then v should not be considered further, since its actual support $c(v)$ could be zero. Lemma III.2 provides analysis on the noise level of $c'(v)$, i.e., the estimated support of $p(v)$.

Lemma III.2. *For any node v in the trie, let $c'(v)$ be the estimated support of $p(v)$ obtained in Algorithm 2, and $c(v)$ be the exact value. With at least $1-\beta$ probability, we have:*

$$|c'(v) - c(v)| = O\left(\frac{(\ell+1)\cdot\sqrt{n\ln(1/\beta)}}{\epsilon_1}\right),$$

where n is the total number of users.

Proof: For any user u_i, let $e_i(v)$ equal 1 if $p(v)$ is a prefix of u_i's term, and 0 otherwise. Let $e'_i(v)$ be the output of Algorithm 1 when given $e_i(v)$ and $\epsilon_1/(\ell+1)$ as inputs. Then, the variance of $e'_i - e_i$ is

$$Var[e'_i - e_i] = O\left(\frac{(\ell+1)^2}{\epsilon_1^2}\right).$$

By Bernstein's inequality,

$$\begin{aligned}
&\Pr[|c'(v) - c(v)| \geq t] \\
&\leq 2\cdot\exp\left(-\frac{t^2}{2\sum_{i=1}^{n} Var[e'_i(v) - e_i(v)] + \frac{2t}{3}\frac{e^{\epsilon_1/(\ell+1)}}{e^{\epsilon_1/(\ell+1)}-1}}\right) \\
&= 2\cdot\exp\left(O\left(-\frac{t^2}{n(\ell+1)^2/\epsilon_1^2 + t\ell/\epsilon_1}\right)\right).
\end{aligned}$$

Therefore, there exists

$$t = O\left(\frac{(\ell+1)\sqrt{n}\cdot\sqrt{\ln(1/\beta)}}{\epsilon_1}\right)$$

such that $|c(v) - c'(v)| < t$ holds with at least $1-\beta$ probability. ■

Based on Lemma III.2, we set θ to $\frac{\eta(\ell+1)\cdot\sqrt{n}}{\epsilon_1}$, where η is a system parameter and will be discussed in Section V.

The main drawback of BSL is that each invocation of optimized RR is assigned a rather small portion of the privacy budget (i.e., $\epsilon_1/(\ell+1)$), leading to inaccurate estimates of prefix support values, especially when ℓ is large (i.e., when terms can be long). Further, determining an appropriate value for ℓ is not easy: the method cannot find terms longer than ℓ characters, but a large ℓ leads to low privacy budget used in optimized RR, and, consequently, a noisy trie. Our proposed solution, presented next, addresses these issues to improve accuracy.

Algorithm 3: Improved_BSL_Build_Trie(ϵ_1, θ, ℓ)

```
1  Randomly partition all users into ℓ+1 equal-sized groups G_1, ..., G_{ℓ+1}, with ⌊n/(ℓ+1)⌋ users each;
2  Same as line 1 in Algorithm 2;
3  while there exists an unvisited non-leaf node v do
4      Let d (0 ≤ d ≤ ℓ) be the depth of node v;
5      Same as Lines 3-5 in Algorithm 2;
6      for each character i ∈ (I ∪ {"&"}) do
7          Same as Lines 7-8 in Algorithm 2;
8          Apply Optimized RR to collect information from users
             in group G_{d+1} to compute an estimation c'(v_c) of
             c(v_c) = supp(p(v_c)), using privacy budget ε_1;
9          Same as Lines 10-13 in Algorithm 2;
10 Same as Lines 14-17 in Algorithm 2;
```

IV. PrivTrie

Section IV-A describes a simple and effective way to improve the accuracy of trie construction based on BSL. Section IV-B presents the key idea of PrivTrie, i.e., adaptive trie construction. Section IV-C further improves accuracy by enforcing consistency in the estimated support values.

A. Partitioning Users

The first idea of PrivTrie is *not to split the privacy budget for different levels of the trie, but partition the users instead*. Specifically, PrivTrie randomly partitions the set of users into $\ell+1$ equal-sized groups, one for each level of the trie. As we show soon, this algorithmic change improves the accuracy bound of estimated prefix supports by a factor of $O(\sqrt{\ell})$. We first apply this idea to BSL, leading to an improved baseline method which we call *IBSL*.

Algorithm 3 shows the trie construction procedure of IBSL[1]. Compared with Algorithm 2, the main change is that when attempting to split an internal node v, the aggregator in IBSL collects information from v's corresponding user group (i.e., the d_v-th group, where d_v is the depth of v with root being depth 0), using privacy budget ϵ_1. Note that in order to estimate the support of v among all n users (i.e., not just in one group), the aggregator needs to scale the estimated support collected by a factor of $\ell+1$. The following lemma establishes the correctness of IBSL.

Lemma IV.1. *Algorithm 3 satisfies ϵ_1-LDP.*

Proof: On the i-th level in trie, let V_i be the set of nodes and G_i be the users assigned. Following the proof of Lemma III.1, we can easily infer that it satisfies ϵ_1-LDP when applying the optimized RR on users in G_i with ϵ_1 budget to compute supports of nodes in V_i. In addition, Algorithm 3 shows that

[1] For ease of presentation, here we assume that the privacy budget ϵ is still split into ϵ_1 for trie construction and ϵ_2 for candidate refinement. In our implementation, we actually split the users into two disjoint groups for these tasks, each with full privacy budget ϵ.

Algorithm 4: PrivTrie_Build_Trie(ϵ_1)

Same as Line 1 in Algorithm 2;
Set available user set $U(v_r)$ for root v_r to $\{u_1, u_2, \cdots, u_n\}$;
while *there exists an unvisited non-leaf node* v **do**
    Mark v as visited;
    Let $U(v)$ be the set of available users for node v;
    for *each character* $i \in (\mathcal{I} \cup \{\text{“\&”}\})$ **do**
        Same as Lines 7-8 in Algorithm 2, which add a new node v_c as a child of v;
        Initialize estimated support $c'(v_c)$ to 0;
        Initialize available user set $U(v_c)$ to $U(v)$;
        while $U(v_c)$ *is not empty* **do**
            Randomly sample a batch of users U' from $U(v_c)$;
            Apply optimized RR to collect information from U' using privacy budget ϵ_1, in order to incrementally refine the estimated support $c'(v_c)$ of v_c;
            Update θ according to Equation (1);
            if $i \neq$ “&” *and* $c'(v_c) \geq \theta$ **then**
                Mark v_c as unvisited;
                Break;
            Remove all users in U' from $U(v_c)$;
        if $U(v_c)$ *is empty* **then**
            Mark v_c as pruned;
Same as Lines 14-17 in Algorithm 2;

we use disjoint users to generate the supports of nodes from different levels. Then the terms of any user in G_i are irrelevant to the estimation of supports associated with nodes in other levels. Therefore, this lemma is proved. ∎

Compared to BSL, IBSL obtains an improved accuracy bound with a reduction by a factor of $\sqrt{\ell + 1}$, according to the following lemma.

Lemma IV.2. *For any node v in the trie, let $c'(v)$ be the estimated support of $p(v)$ obtained in Algorithm 3, and $c(v)$ be the exact support of $p(v)$. With at least $1 - \beta$ probability,*

$$|c'(v) - c(v)| = O\left(\frac{\sqrt{n(\ell+1)\ln(1/\beta)}}{\epsilon_1}\right).$$

We can prove this lemma like Lemma III.2.

Based on Lemma IV.2, in IBSL, we set the support threshold θ to $\frac{\eta\sqrt{(\ell+1)n}}{\epsilon_1}$. Although IBSL obtains improved accuracy compared to BSL, it still suffers from the latter's main drawbacks, i.e., (i) each run of optimized RR is assigned a rather small portion of the user population, and (ii) ℓ needs to be pre-defined. Next, we address these problems with a novel adaptive trie construction approach.

B. Adaptive Trie Construction

The adaptive trie construction algorithm in PrivTrie is based on three important observations. First, some nodes in the trie, especially those at higher levels, have high supports for their corresponding prefixes. For instance, in Fig. 1, the node v_3 has a true support of 7. For such a node v, its estimated support $c'(v)$ is likely to be large, even with heavy perturbation. This suggests that we could decide to split v with only a coarse-grained estimate of its support. In other words, *we could assign fewer users to estimate the support for v*. On the other hand, it is non-trivial to determine how many users are needed to estimate the support of a given node, since it is not known beforehand whether the node has high support or not. For instance, nodes at a high level of the trie do not always have high support, e.g., v_2 in Fig. 1.

The second important observation made in PrivTrie is that *it is not necessary to assign each user to a single level of the trie*, as is done in IBSL described in Section IV-A. In fact, a user u_i can be involved in the support estimation process for an arbitrary set of nodes V_i, each with privacy budget ϵ_1 as in IBSL, so long as no node in V_i is an ancestor of another. This is formally established by the following lemma.

Lemma IV.3. *Given any user u_i ($1 \leq i \leq n$), suppose that (i) the aggregator collects information from u_i to estimate the support values of a set of nodes V_i, each using optimized RR with privacy budget ϵ_1 and (ii) V_i satisfies that $\nexists v, w \in V_i$ such that v is an ancestor of w. Then, the data collection for the entire V_i satisfies ϵ_1-LDP with respect to user u_i.*

Proof: Let t_i be the term that user u_i has. Since $\nexists v, w \in V_i$ such that v is an ancestor of w, there is only one node v in V_i with $p(v)$ being a prefix of t_i. Hence, t_i can contribute to the prefix support of at most one node in V_i. As such, similar to the proof of Lemma III.1, it can then be verified that, the data collection for the entire V_i by applying optimized RR with ϵ_1 budget ensures ϵ_1-LDP. ∎

Accordingly, given a node v in the trie, we define its *available user set* $U(v)$ as the set of users who have not participated in the estimation of the support value for any of v's ancestor nodes. For example, in Fig. 1, the available user set $U(v_{11})$ of node v_{11} consists of users who have not contributed to data collection related to v_{11}'s ancestors, i.e., v_3 and v_9 (the root does not need data collection). Our third observation is that *for every leaf node v_l in the trie, we can simply use its entire available user set $U(v_l)$ to estimate its support $c'(v_l)$*. Such a leaf node could be one containing the end-of-term character “&”, or a node whose support is lower than its expected estimation error, as in BSL and IBSL. Since v_l has no descendents, and all of its ancestors have already been explored due to our top-down approach to trie construction, using of $U(v_l)$ does not affect data collection for any unvisited nodes. For example, consider node v_5 in Fig. 1. If v_5 is eventually pruned due to insufficient support, then we could use all of its available users (i.e., those who have not contributed to the support estimation of node v_2) to estimate $c'(v_5)$; these users are still able to participate in data collection related to other unvisited nodes in the trie, by Lemma IV.3.

Based on the above observations, we design the adaptive trie construction algorithm of PrivTrie, shown in Algorithm 4. The main idea is that for each internal node v_c, we do not decide in advance the number of users involved in estimating the support for v_c; instead, the aggregator asks available users

from $U(v_c)$ one batch at a time (Lines 10-17)[2], each time refining the estimated support $c'(v_c)$. In the full version [18], we discuss the setting of the batch size. This terminates when either (i) $c'(v)$ exceeds our threshold θ (Lines 14-16), meaning v_c should be split or (ii) the aggregator has asked all available users in $U(v_c)$, and yet $c'(v_c) < \theta$, which indicates that v should not be split (Lines 18-19).

Further, the support threshold θ should also be adaptive, as the aggregator incrementally refines the support of a node by asking more available users. To determine the appropriate value for θ after the aggregator finishes collecting data from a new batch of users, we first analyze the error in the estimated support value with a given number of users, in Lemma IV.4.

Lemma IV.4. *For any node v in the trie, let $c'(v)$ be the estimated support of $p(v)$ obtained in Algorithm 4 after collecting data from m users, and $c(v)$ be the exact support of $p(v)$. With at least $1-\beta$ probability,*

$$|c'(v) - c(v)| = O\left(\frac{n\sqrt{\ln(1/\beta)}}{\epsilon_1\sqrt{m}}\right).$$

We can prove Lemma IV.4 like Lemma III.2.

Therefore, in Line 13 of Algorithm 4, we iteratively update θ with the following equation:

$$\theta = \frac{\eta \cdot n}{\epsilon_1\sqrt{m}}, \tag{1}$$

where m is the number of users that have been sampled from $U(v_c)$, and η is a system parameter discussed in Section V.

Lemma IV.5 establishes the correctness of Algorithm 4.

Lemma IV.5. *Algorithm 4 satisfies ϵ_1-LDP.*

Proof: Let V_i be the set of nodes with support computations involving user u_i in Algorithm 4. According to Lemma IV.3, we know data collection for the entire V_i satisfies ϵ_1-LDP with respect to user u_i. Clearly, Algorithm 4 processing all users also satisfies ϵ_1-LDP. ∎

Compared to Algorithms 2 and 3, Algorithm 4 achieves adaptive estimation of node support values, and avoids predefining the maximum term length ℓ. In particular, by Lemmas III.2 and IV.4, if Algorithm 4 estimates the support of a node v with m users, then the error in the estimated support would be roughly $\sqrt{\frac{n}{\ell m}}$ times the error that Algorithm 3 would incur for the same node. For those nodes with small supports, the adaptiveness of Algorithm 4 (due to Lines 10-17 and Equation 1) is likely to set $m > \frac{n}{\ell}$, thus reducing the error in the estimated support $p(v)$ and leading a more accurate decision regarding whether v should be split. As we show in Section V, Algorithm 4 achieves significantly higher accuracy than both Algorithms 2 and 3.

[2]For ease of presentation, Algorithm 4 explicitly maintains the available user set $U(v)$ for each node v, which would require a large amount of memory. In our implementation, we avoid materializing $U(v)$ by generating a random permutation of all users in advance, and maintaining only a pointer to the random user sequence instead of actual user sets. More details can be found in the full version [18].

Finally, we analyze the communication cost of Algorithm 4. Observe that if a user is involved in the estimation of a node's support, then the user needs to submit one bit to the server (see Algorithm 1). Therefore, the total communication cost incurred for a user is β bits if the user participates in the estimation of β nodes' supports. Although β can be large in the worst case, we observe that it tends to be small in practice. In particular, in our experiment, the value of β is only a few thousands. The reason is that (i) Algorithm 4 examines a node's support only when the prefix of the node has a chance to be frequent, and (ii) there is only a moderate number of such nodes.

Next we further optimize the accuracy of the estimated node support values by enforcing consistency.

C. Enforcing Consistency on Node Support Values

Let C be the set of candidate terms identified by Algorithm 4, and $\mathcal{T}$ be the trie constructed. Let V be set of nodes in $\mathcal{T}$, and V_l be the set of leaf nodes in $\mathcal{T}$. For convenience, we abuse notation and use C to denote the set of nodes that correspond to the candidate terms.

We reserve a set of n_2 users that are not involved in the construction of the trie (see Footnote 1), and we apply optimized RR $|C|$ times on these users, each of which obtains an estimation of the frequency of a term in C. As such, we have $|V| + |C|$ support estimations for the nodes in $\mathcal{T}$. In particular, Algorithm 4 generates one noisy estimation $c'(v)$ for each node $v \in V$, while the reserved users produce an additional estimation $c''(v)$ for each node $v \in C$. These estimations are unbiased, and their expected values satisfy the following *consistency* constraints:

1) For each node $v \in C$, its two estimations $c'(v)$ and $c''(v)$ have the same expectation.
2) If a node v has a set S of children, then

$$\mathbb{E}[c'(v)] = \sum_{u \in S} \mathbb{E}[c'(u)].$$

Hay et al. [19] has demonstrated that the above constraints could be exploited to *refine* the noisy estimations to obtain better accuracy. The basic idea is to derive a new estimation $\hat{c}(v)$ for each node v, such that all $\hat{c}(v)$ satisfy the consistency constraint while minimizing their L_2 distance to the noisy estimations. However, Hay et al.'s algorithm cannot be directly applied in our setting, since it assumes that (i) each node in the tree has only one estimation, and (ii) each estimation has the same variance. In what follows, we extend Hay et al.'s algorithm to tackle our problem.

For each node v, let $\sigma'(v)$ and $\sigma''(v)$ be the variances of $c'(v)$ and $c''(v)$, respectively, and $child(v)$ be the set of v's children (if any). We aim to obtain an estimation $\hat{c}(v)$ for the support of each node v, such that

$$\sum_{v \in V}\left(\frac{\hat{c}(v) - c'(v)}{\sigma'(v)}\right)^2 + \sum_{v \in C}\left(\frac{\hat{c}(v) - c''(v)}{\sigma''(v)}\right)^2 \tag{2}$$

is minimized subject to the constraint that

$$\forall v \in V,\ \hat{c}(v) = \sum_{u \in child(v)} \hat{c}(u). \tag{3}$$

Similar to Hay et al.'s algorithm [19], our derivation of $\hat{c}(v)$ consists of two phases. First, in the *bottom-up* phase, we examine the nodes in $\mathcal{T}$ in a bottom-up manner and compute four intermediate results $\sigma(v)$, $r(v)$, $s(v)$, and $z(v)$ for each node v as follows.

$$\sigma(v) = \begin{cases} \frac{\sigma'(v) \cdot \sigma''(v)}{\sqrt{(\sigma'(v))^2 + (\sigma''(v))^2}}, & \text{if } v \in C \\ \sigma'(v), & \text{otherwise} \end{cases} \tag{4}$$

$$r(v) = \begin{cases} \sigma^2(v), & \text{if } v \text{ is a leaf} \\ \frac{1}{1 + \sum_{u \in child(v)} \frac{s(u)}{\sigma^2(v)}}, & \text{otherwise} \end{cases} \tag{5}$$

$$s(v) = \begin{cases} r(v), & \text{if } v \text{ is a leaf} \\ r(v) \cdot \sum_{u \in child(v)} s(u), & \text{otherwise} \end{cases} \tag{6}$$

$$z(v) = \begin{cases} r(v) \cdot \left(\frac{c'(v)}{(\sigma'(v))^2} + \frac{c''(v)}{(\sigma''(v))^2} + \sum_{u \in ancestor(v)} \frac{c'(u)}{\sigma^2(u)} \right), \\ \qquad \text{if } v \in C \\ r(v) \cdot \sum_{u = v \vee u \in ancestor(v)} \frac{c'(u)}{\sigma^2(u)}, \\ \qquad \text{if } v \text{ is a leaf} \\ r(v) \cdot \sum_{u \in child(v)} z(u), \quad \text{otherwise} \end{cases} \tag{7}$$

where $ancestor(v)$ denotes the set of ancestors of v in $\mathcal{T}$.

After that, in the *top-down* phase, we perform a breath-first search from the root of $\mathcal{T}$ to derive the refined estimation $\hat{c}(v)$ for each node v:

$$\hat{c}(v) = \begin{cases} z(v), & \text{if } v \text{ is the root} \\ z(v) - s(v) \sum_{u \in ancestor(v)} \frac{\hat{c}(u)}{\sigma^2(u)}, & \text{otherwise} \end{cases} \tag{8}$$

The correctness of the above method is established in the following lemma. Its proof appears in the full version [18].

Lemma IV.6. *The estimations $\hat{c}(\cdot)$ computed by the two-phase method minimizes Equation (2) while satisfying Equation (3).*

V. Experiments

In this section, we evaluate our proposals to show their effectiveness. All experiments were performed on an Intel Xeon 3.4GHz CPU with 16 GBytes of memory.

Competitors. We have implemented all proposed methods BSL (Algorithm 2), IBSL (Algorithm 3) and PrivTrie (Algorithm 4) in C++. We also obtained the source code of A-RAPPOR from its Github repository mentioned in [5]. In addition, we have implemented TreeHist [15] and PEM [16], described in Section II, and adapted them to our problem, as follows. First, both TreeHist and PEM require a fixed-length, binary string representation of each item, i.e., a term in our context. Since terms have variable lengths in our setting, we estimate a maximum term length ℓ using procedure *estimate_l* (presented in Appendix), and then fix the length of each term to ℓ with truncation and padding with a special symbol "&". Second, on some datasets (e.g., *Reddit* and *Twitter*, described later), TreeHist sometimes outputs no frequent term at all; manual inspection reveals that in these settings, its support threshold (denoted as f in [15]) is on the same order of magnitude as the number of users. As a fix, we reserve top-k frequent nodes at each level, as is done in PEM [16]. Finally, we found that PEM tends to use an overly large value for its parameter α, i.e., the number of tree levels assigned to each user group, which leads to poor performance. Trough manual tuning (without privacy considerations, which favors PEM), we found that $\alpha = 5$ usually yields good accuracy for PEM; hence, we set $\alpha = 5$ throughout our experiments.

Parameter configurations. In addition to PrivTrie, we also optimized BSL and IBSL using the consistency enforcing approach described in Section IV-C. In the following, we give the default parameter values of our methods; a detailed parameter study is presented in Section V-C. We fix parameter η (for computing frequency threshold θ) to 4 in all our solutions BSL, IBSL and PrivTrie. Regarding privacy parameters, for BSL, we set $\epsilon_1 = 0.7\epsilon$, $\epsilon_2 = 0.2\epsilon$ and, additionally, $\epsilon_3 = 0.1\epsilon$ for determining the maximum term length ℓ, elaborated further in Appendix. Similarly, in IBSL, for the same purposes we use three subsets of users with cardinality $n_1 = 0.7n$, $n_2 = 0.2n$ and $n_3 = 0.1n$, as explained in Footnote 1. For PrivTrie, we set $n_1 = 0.8n$ for trie generation and $n_2 = 0.2n$ for refining candidates. Note that PrivTrie does not need to determine the maximum term length as in the BSL and IBSL.

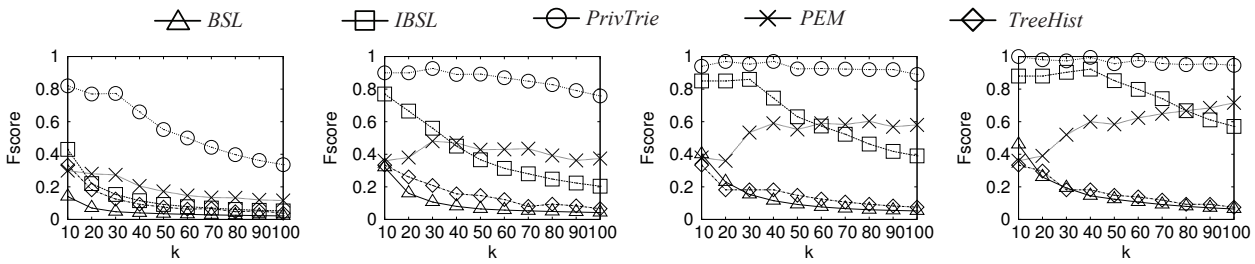
For A-RAPPOR, we follow the defaults in its source code, which is equivalent to setting $\epsilon_1 = \frac{2}{3}\epsilon$ for collecting n-grams where $n = 2$ and $\epsilon_2 = \frac{1}{3}\epsilon$ for candidate refinement. A-RAPPOR also requires a pre-defined maximum term length, but does not contain an algorithm to determine it. In our experiments, we simply provide the exact maximum term length in the data to A-RAPPOR without deducting any privacy budget, which strongly favors A-RAPPOR. Finally, For both TreeHist and PEM, we use 0.1ϵ to compute the maximum term length, as explained above. Meanwhile, we set $\lambda = 0.85$ for all testing cases. These parameter choices are further elaborated in Section V-C.

Real datasets. We use two real datasets: *Reddit*[3] and *Twitter*[4]. *Reddit* contains one month's public comments made on Reddit, while *Twitter* contains approximately 1% of the messages posted on Twitter in June 2017. We preprocess each dataset to retain only the "new" terms that (i) consist of English characters and (ii) do not appear in an English dictionary[5]. After preprocessing, *Reddit* (resp. *Twitter*) contains 7.6M non-empty comments with 4,585 new terms (resp. 2.5M non-empty messages with 3,185 new terms). We regard each comment

[3]https://www.reddit.com/r/datasets/comments/3bxlg7/i_have_every_publicly_available_reddit_comment/

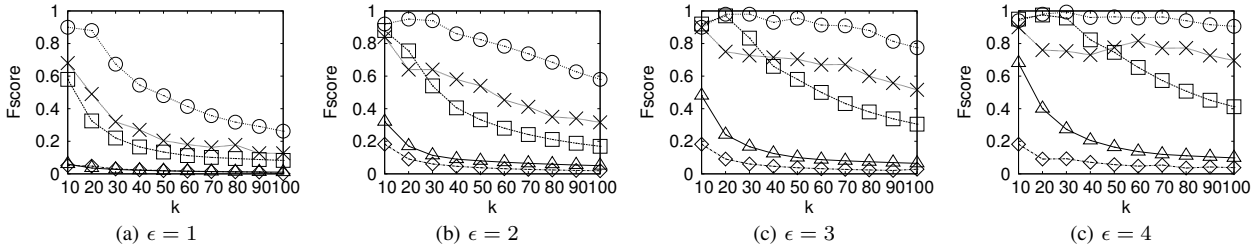
[4]https://archive.org/details/archiveteam-twitter-stream-2017-06

[5]https://github.com/dwyl/english-words



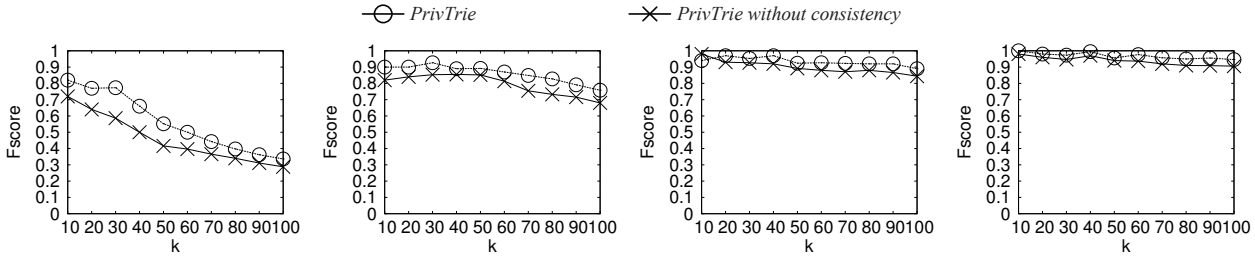
(a) $\epsilon = 1$ (b) $\epsilon = 2$ (c) $\epsilon = 3$ (c) $\epsilon = 4$

Fig. 2. Fscore results on the *Reddit* dataset.



(a) $\epsilon = 1$ (b) $\epsilon = 2$ (c) $\epsilon = 3$ (c) $\epsilon = 4$

Fig. 3. Fscore results on the *Twitter* dataset.



(a) $\epsilon = 1$ (b) $\epsilon = 2$ (c) $\epsilon = 3$ (c) $\epsilon = 4$

Fig. 4. Effect of consistency enforcement on Fscore with the *Reddit* dataset.

(resp. message) as the private text of a user.

Synthetic dataset. A-RAPPOR does not terminate within 48 hours on either of the two real datasets, which are relatively large. In order to compare with A-RAPPOR, we generate a smaller synthetic dataset based on *Reddit* in a similar way as one of the experiments in the A-RAPPOR paper [5]. In particular, we retrieve the terms in *Reddit*, sort them in descending order of their frequencies, and then sample 100k terms from a Zipf distribution over the terms such that the frequency order of the terms remain unchanged. We regard each of the 100k terms as a user's private text.

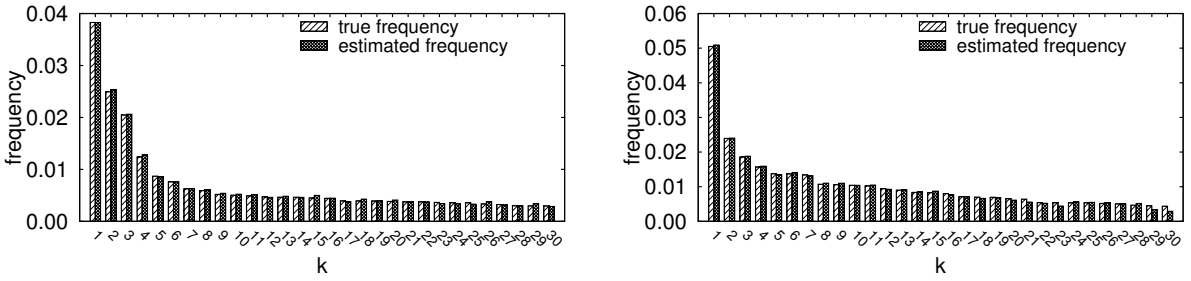
Utility Metric. We report result utility in terms of Fscore [20]. Specifically, let W' and W be the estimated and actual top-k frequent new term sets, respectively. Fscore is simply the harmonious mean of precision and recall, calculated by comparing W' against W. Additional results on utility metrics beyond Fscore can be found in the full version [18].

A. Evaluation Results on Real Data

We first present results on the two real datasets *Reddit* and *Twitter*. Because A-RAPPOR fails to terminate within 48 hours on these datasets, we exclude it in the results. For every setting, each of the remaining methods BSL, IBSL, PrivTrie, PEM and TreeHist requires less than 1 minute CPU time. Considering that these datasets involve millions of users, such computation cost is negligible compared to the communication overhead.

Fig. 2 plots the Fscore as a function of k, the number of top frequent new terms to report. The results include 4 different values for ϵ: 1, 2, 3 and 4, among which 2 is reportedly used in both macOS [2] and Chrome [3]. On all settings, PrivTrie significantly outperforms all the other methods, with a clear performance gap. Notably, when $\epsilon = 2$, PrivTrie is the only solution that obtains a practical level of accuracy for moderately large values of k, e.g., over 0.8 Fscore when $k = 50$. In the same setting, the Fscores for others are rather low, i.e., less than 0.5. The good performance of PrivTrie is mainly due to the effective adaptive trie construction algorithm, since all methods are optimized with the consistency enforcing module.

For larger values of ϵ, the accuracy of IBSL improves, but is still consistently below that of PrivTrie. Note that $\epsilon = 4$ is a rather large value, since (i) the privacy guarantee of differential privacy loosens exponentially with ϵ and (ii) RAPPOR aims to limit a user's lifetime privacy budget to 8-9 [3]. On the other hand, the accuracy of IBSL drops rapidly with increasing k, whereas PrivTrie is considerably more resilient against a large k. Comparing BSL and IBSL, the latter is consistently more accurate than the former, and the performance gap expands with increasing ϵ. This is expected, as IBSL achieves a factor



(a) *Reddit* (b) *T*witter

Fig. 5. Frequencies of true top-30 frequent words and estimated frequencies output by PrivTrie with ϵ = 2.

of $O(\sqrt{\ell})$ reduction on the accuracy bound.

Regarding TreeHist and PEM, the former performs rather poorly in all settings, whose accuracy is usually lower than that of IBSL. The main reason is that the use of the count sketch leads to numerous subgroups with relatively few users each, as described in Section II, leading to high error in the estimated support values. PEM, on the other hand, outperforms IBSL in settings with larger values of k. This is mainly due to the different ways that IBSL and PEM judge whether a prefix is frequent. Note that PrivTrie outperforms PEM in all settings.

Fig. 3 repeats the same experiments on the *Twitter* dataset. The relative performance of the three methods remain similar as on the *Reddit* dataset. The absolute Fscores are generally lower on *Twitter* than on *Reddit*. This is because *Reddit* is a much larger dataset involving over 7.6 million users, whereas *Twitter* has around 2.5 million users. Nevertheless, PrivTrie still obtains good accuracy on *Twitter*.

Next we investigate the effect of consistency enforcement (described in Section IV-C) on the accuracy of PrivTrie. Fig. 4 shows the Fscore results on *Reddit*. The results on *Twitter* lead to similar conclusions, and are omitted for brevity. Overall, consistency enforcement improves the Fscore by about 0.1, which is considerable. In the meantime, the accuracy improvement achieved by consistency enforcement is also clearly lower than that achieved by the adaptive trie construction module.

Finally, we compare true frequencies of top-30 terms with the estimated ones output by PrivTrie, with privacy budget $\epsilon = 2$. Fig. 5 shows the results on both real datasets. Clearly, the estimated frequencies are very close to their true values.

B. Evaluation Results on Synthetic Data

Next we use the synthetic data to compare A-RAPPOR against the other methods. Fig. 6 shows the result utility in terms of Fscore. Again, PrivTrie is the clear winner compared with other methods except for PEM, though its absolute performance is not as high as on the two real datasets, since the synthetic dataset involves only 100k users, as opposed millions of users in the real datasets. Note that the baseline approach BSL obtains comparable accuracy performance as A-RAPPOR; in fact, when ϵ is smaller (i.e., 1 and 2), BSL clearly outperforms A-RAPPOR, since the latter's accuracy is zero as it fails to identify any frequent terms. This suggests A-RAPPOR's accuracy is probably also rather low on the real datasets (once it finishes running), where BSL performs poorly. TreeHist has rather poor performance overall, obtaining zero Fscore in all settings of ϵ. PEM, on the other hand, achieves good performance, sometimes even better than PrivTrie. This is because PEM employs a different method to judge whether a prefix is frequent. However, the accuracy of PEM is sensitive to various factors like k and the input dataset, whereas PrivTrie's performance is stable and predictable.

Besides, we also test running time of PrivTrie and A-Rappor. PrivTrie requires about 30s CPU time on *Reddit* and 20s CPU time on *Twitter*, while A-RAPPOR does not terminate within 48 hours.

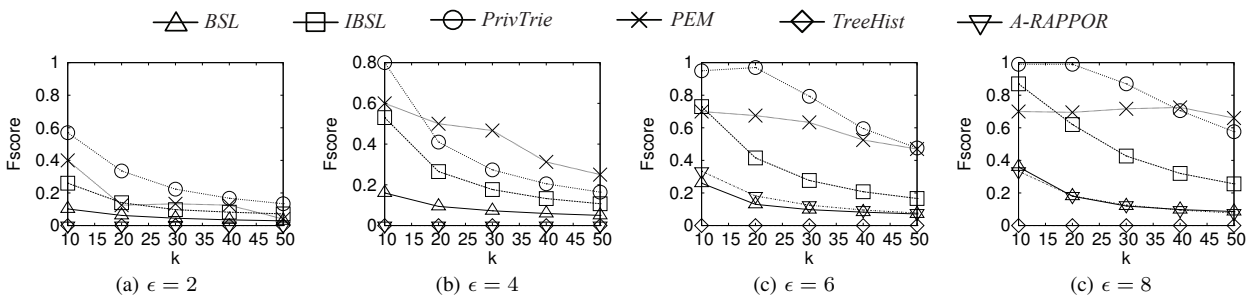
C. Choosing Parameter Values

Next we focus on choosing appropriate parameter values for the proposed methods. These parameters include η (for computing support threshold θ), the allocation policy for n_1-n_3 (ϵ_1-ϵ_3 in BSL), and the truncation ratio λ. In order not to cherrypick parameter values for PrivTrie on real data, we perform the parameter study using IBSL and the synthetic dataset with $\epsilon = 2$. The parameter values chosen from this study turn out to generalize well, i.e., they also lead to good performance on PrivTrie on other datasets and settings.

We first study the impact of η, after fixing other parameters to their defaults, i.e., $\lambda = 0.85$, $n_1 = 0.7n$, $n_2 = 0.2n$ and $n_3 = 0.1n$. We observe that smaller values of η generally lead to high accuracy, but also long running time. In particular, when $\eta < 2$, IBSL does not terminate after 3 hours. Hence, we choose a relatively small $\eta = 4$ as the default value. We omit detailed results due to space limitations, which can be found in the full version [18].

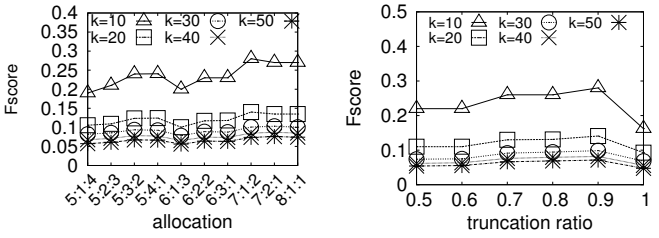
Next we explore how the allocation of users (i.e., n_1-n_3) affects performance, after fixing η ad λ to their defaults. Figure 7(a) shows the results. We omit the results when $n_1 < 5$, which lead to very low Fscores. Based on the results, we select the default values of n_1-n_3 to have ratio 7:2:1, which resides in a plateau of good parameter values. Notice PrivTrie involves only n_1 and n_2. We initialize these two parameters based on the best ratio, and shift users assigned for n_3 to n_1.

Finally, Figure 7(b) plots the Fscore results against varying truncation ratio λ, with other parameters fixed to their defaults. The Fscore first increases and then decreases with growing λ, since a larger truncation ratio reduces information loss, at the cost of finer-grained user partitioning. The default value



(a) $\epsilon = 2$ (b) $\epsilon = 4$ (c) $\epsilon = 6$ (c) $\epsilon = 8$

Fig. 6. Fscore results on the synthetic dataset.



(a) allocation policies (b) truncation ratio λ

Fig. 7. Impact of allocation policies and truncation ratio

$\lambda = 0.85$ generally strikes a good balance, which agrees with the recommendation in Ref. [21]. Note that this parameter is not used in PrivTrie.

VI. Other Related Work

The works most related to ours are RAPPOR [3] and its extended version [5], discussed in Section II. In what follows, we review other existing solutions that are related to differential privacy and sequence publishing.

First, there exist several algorithms [22]–[27] for publishing sequence data under *global differential privacy* [28], which assumes that a data curator collects unfettered data from all users and aims to release a modified version of the data that preserves privacy. In particular, Bonomi and Xiong [24] as well as Xu et al. [26] propose differentially private methods for publishing frequent subsequences, while Cheng et al. [25] develop a technique for mining maximal frequent subsequences under differential privacy. Chen et al. [22] devise a differentially private algorithm for generating synthetic sequence data, which is subsequently improved in [23], [27]. None of the above algorithms, however, can be extended to ensure local differential privacy.

Meanwhile, existing techniques for LDP have focused on tasks rather different from sequence mining, with A-RAPPOR [5] being the only exception. In particular, Duchi et al. [29], [30] propose LDP methods to estimate the mean values of numerical attributes. Bassily and Smith [31] address the problem of frequency estimation over categorical data, which is further studied in [8], [32], [33]. Ren et al. [34] investigate the problem of frequency distribution on multiple attributes, while Chen et al. [35] present a generalized version of LDP that is parameterized with individual-specific privacy guarantees for publishing location data. Besides, there exist some works [9], [15], [16], [36] focusing on publishing heavy hitters. Specifically, with the assumption that some of users trust the curator, Avent et al. [36] propose to first find heavy hitters based on such users under centralized DP, and then transmit hitters to users not trusting the curator to compute statistics (such as probabilities and variances) under local DP. However, their hybrid model (on top of centralized DP and local DP) does not apply to our scenario where the privacy model is pure LDP. It is technically orthogonal to our solutions. Bassily et al. [15] and Wang et al. [16] publish data relying solely on local DP. The proposed solution PrivTrie outperforms both approaches as shown in Section V, thanks to the former's adaptive trie construction algorithm.

Finally, we note that the way that PrivTrie identifies frequent terms is somewhat similar to the *Apriori* approach used by non-private algorithms (e.g., [37]) to derive frequent itemsets from market basket data. In particular, Apriori identifies frequent itemsets in increasing order of their sizes, utilizing the property that a frequent itemset's subsets must all be frequent. This is similar to PrivTrie's strategy, which exploits the fact that a frequent term's prefixes should also be frequent.

VII. Conclusion

This paper investigates the problem of collecting top-k frequent new terms from users under ϵ-local differential privacy. Our main proposal, PrivTrie, achieves high accuracy on real datasets under a common privacy level $\epsilon = 2$. This is done mainly through an adaptive algorithm for building a trie under LDP, which clearly outperforms baseline solutions with simple trie construction strategies. As future work, we plan to explore language modeling under LDP, which promises to further improve user experience on the mobile keyboard, e.g., by predicting the next word.

Acknowledgment

This research is supported by grants (61433008, U143520006, 61472071) from National Natural Science Foundation of China, a grant from Samsung GRO, and NPRP9-466-1-103 from QNRF, Qatar.

References

[1] "Apple's 'differential privacy' is about collecting your data – but not *your* data," 2018, https://www.wired.com/2016/06/apples-differential-privacy-collecting-data/.

[2] J. Tang, A. Korolova, X. Bai, X. Wang, and X. Wang, "Privacy loss in apple's implementation of differential privacy on macos 10.12," *arXiv preprint 1709.02753*, 2017.

[3] Ú. Erlingsson, V. Pihur, and A. Korolova, "Rappor:randomized aggregatable privacy-preserving ordinal response," in *CCS*, 2014, pp. 1054–1067.
[4] A. Thakurta, A. Vyrros, U. Vaishampayan, G. Kapoor, J. Freudinger, V. Prakash, A. Legendre, and S. Duplinsky, "Emoji frequency detection and deep link frequency," Jul. 11 2017, uS Patent 9,705,908. [Online]. Available: https://www.google.com/patents/US9705908
[5] G. C. Fanti, V. Pihur, and Ú. Erlingsson, "Building a RAPPOR with the unknown: Privacy-preserving learning of associations and data dictionaries," *PoPETs*, vol. 2016, no. 3, pp. 41–61, 2016.
[6] F. McSherry, "Privacy integrated queries: an extensible platform for privacy-preserving data analysis," in *SIGMOD*, 2009, pp. 19–30.
[7] S. L. Warner, "Randomised response: a survey technique for eliminating evasive answer bias," *Journal of the American Statistical Association*, vol. 60, no. 309, pp. 63–69, 1965.
[8] T. Wang, J. Blocki, N. Li, and S. Jha, "Locally differentially private protocols for frequency estimation," in *USENIX Security Symposium*, 2017, pp. 729–745.
[9] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren, "Heavy hitter estimation over set-valued data with local differential privacy," in *CCS*, 2016, pp. 192–203.
[10] "Rappor (randomized aggregatable privacy preserving ordinal responses)," 2018, https://www.chromium.org/developers/design-documents/rappor.
[11] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, 1970.
[12] A. P. Dempster, N. M. Laird, and D. B. Rubin, "Maximum likelihood from incomplete data via the em algorithm," *Journal of the royal statistical society. Series B (methodological)*, pp. 1–38, 1977.
[13] A. Thakurta, A. Vyrros, U. Vaishampayan, G. Kapoor, J. Freudiger, V. Sridhar, and D. Davidson, "Learning new words," Mar. 14 2017, uS Patent 9,594,741. [Online]. Available: https://www.google.com/patents/US9594741
[14] ——, "Learning new words," May 9 2017, uS Patent 9,645,998. [Online]. Available: https://www.google.com/patents/US9645998
[15] R. Bassily, K. Nissim, U. Stemmer, and A. G. Thakurta, "Practical locally private heavy hitters," in *NIPS*, 2017, pp. 2285–2293.
[16] T. Wang, N. Li, and S. Jha, "Locally differentially private heavy hitter identification," *CoRR*, vol. abs/1708.06674, 2017.
[17] M. Charikar, K. C. Chen, and M. Farach-Colton, "Finding frequent items in data streams," in *ICALP*, 2002, pp. 693–703.
[18] "Privtrie: Effective frequent term discovery under local differential privacy (technical report)," 2018, https://sites.google.com/site/privtrie/tr.
[19] M. Hay, V. Rastogi, G. Miklau, and D. Suciu, "Boosting the accuracy of differentially private histograms through consistency," *PVLDB*, vol. 3, no. 1, pp. 1021–1032, 2010.
[20] C. J. V. Rijsbergen, "A theoretical basis for the use of cooccurrence data in information retrieval," *Journal of Documentation*, vol. 33, no. 2, pp. 106–119, 1977.
[21] C. Zeng, J. F. Naughton, and J. Cai, "On differentially private frequent itemset mining," *PVLDB*, vol. 6, no. 1, pp. 25–36, 2012.
[22] R. Chen, B. C. M. Fung, and B. C. Desai, "Differentially private transit data publication: A case study on the montreal transportation system," in *KDD*, 2012, pp. 213–221.
[23] R. Chen, G. Acs, and C. Castelluccia, "Differentially private sequential data publication via variable-length n-grams," in *CCS*, 2012, pp. 638–649.
[24] L. Bonomi and L. Xiong, "A two-phase algorithm for mining sequential patterns with differential privacy," in *CIKM*, 2013, pp. 269–278.
[25] X. Cheng, S. Su, S. Xu, P. Tang, and Z. Li, "Differentially private maximal frequent sequence mining," *Comput. Secur.*, vol. 55, pp. 175–192, 2015.
[26] S. Xu, X. Cheng, S. Su, K. Xiao, and L. Xiong, "Differentially private frequent sequence mining," *IEEE Trans. Knowl. Data Eng.*, vol. 28, no. 11, pp. 2910–2926, 2016.
[27] J. Zhang, X. Xiao, and X. Xie, "Privtree: A differentially private algorithm for hierarchical decompositions," in *SIGMOD*, 2016, pp. 155–170.
[28] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *TCC*, 2006, pp. 265–284.
[29] J. C. Duchi, M. I. Jordan, and M. J. Wainwright, "Privacy aware learning," in *NIPS*, 2012, pp. 1439–1447.
[30] ——, "Local privacy and statistical minimax rates," in *FOCS*, 2013, pp. 429–438.

Algorithm 5: Estimate_$\ell(\epsilon_3, \tau)$

Input : privacy budget ϵ_3, upper bound of term length τ
Output: term length threshold ℓ

1 Initialize a vector L of size τ;
2 **for** $i = 1$ *to* τ **do**
3 Apply optimized RR with privacy budget ϵ_3 to collect information from all users to obtain an estimation x_i of the number of users whose term lengths equal i;
4 $L[i] \leftarrow x_i$;
5 $\ell \leftarrow$ the smallest integer such that $\frac{\sum_{i=1}^{\ell} L[i]}{\sum_{i=1}^{\tau} L[i]} \geq \lambda$;
6 **return** ℓ.

[31] R. Bassily and A. D. Smith, "Local, private, effcient protocols for succinct histograms," in *STOC*, 2015, pp. 127–135.
[32] P. Kairouz, S. Oh, and P. Viswanath, "Extremal mechanisms for local differential privacy," in *NIPS*, 2014, pp. 2879–2887.
[33] P. Kairouz, K. Bonawitz, and D. Ramage, "Discrete distribution estimation under local privacy," in *ICML*, 2016, pp. 2436–2444.
[34] X. Ren, C. Yu, W. Yu, S. Yang, X. Yang, and J. A. McCann, "High-dimensional crowdsourced data distribution estimation with local privacy," in *CIT*, 2016, pp. 226–233.
[35] R. Chen, H. Li, A. K. Qin, S. P. Kasiviswanathan, and H. Jin, "Private spatial data aggregation in the local setting," in *ICDE*, 2016, pp. 289–300.
[36] B. Avent, A. Korolova, D. Zeber, T. Hovden, and B. Livshits, "BLENDER: enabling local search with a hybrid differential privacy model," in *USENIX Security*, 2017, pp. 747–764.
[37] R. Agrawal and R. Srikant, "Fast algorithms for mining association rules in large databases," in *VLDB*, 1994, pp. 487–499.

APPENDIX

An LDP Method for Deciding ℓ. Recall that Algorithm 2 involves a parameter ℓ, which is the maximum length of the terms that can be captured by the trie. Since users' maximum term lengths are sensitive, we present an LDP mechanism to estimate ℓ, presented in Algorithm 5. The algorithm takes as input a privacy budget ϵ_3 (such that $\epsilon = \epsilon_1 + \epsilon_2 + \epsilon_3$) and an upper bound τ of the length of the users' term, e.g., $\tau = 45$ for English terms (https://en.wikipedia.org/wiki/Longest_word_in_English). The algorithm first initializes a vector L of size τ (Line 1). After that, for each $i \in [1, \tau]$, it applies optimized RR on all users to estimate the number of users whose term lengths equal i, and it sets the i-th element in L to the estimated value (Lines 2-4). Finally, it scans through L and identifies the smallest integer ℓ such that the estimated number of users with term lengths at most ℓ is at least λ, the truncation ratio (Line 5), and then returns ℓ as output. The rationale is that we want ℓ to be reasonably large without being overly susceptible to outliers and estimation errors. The following lemma shows the correctness of the algorithm.

Lemma A.1. *Algorithm 5 satisfies ϵ_3-LDP.*

Proof: For any user u_i, let L_i be a τ-size vector such that the j-th element in L_i equals 1 if u_i's term has length j, and 0 otherwise. Referring the proof of Lemma III.1, it can be easily verified that applying optimized RR to collect L_i from any user u_i satisfies ϵ_3-LDP. ■