

Adaptive Prefix-Tree Algorithms for Federated Heavy Hitter Detection: Optimizing Utility and Privacy

Chris Junchi Li[◊]

Department of Electrical Engineering and Computer Sciences[◊]
University of California, Berkeley

October 16, 2024

Abstract

In this paper, we present optimized prefix-tree based algorithms for differentially private heavy hitter detection in federated settings, with a focus on improving computational efficiency and utility. Our approach integrates adaptive hyperparameter tuning and innovative data selection mechanisms, enabling high-utility detection while preserving user privacy under both aggregate and local differential privacy models. The proposed methods adaptively choose prefix segment lengths and apply deny lists to minimize false positives, allowing the system to focus on the most significant heavy hitters across multiple iterations. Experimental results on the Reddit dataset demonstrate the efficacy of our techniques in balancing communication, computation, and privacy constraints.

Keywords: Federated Learning, Heavy Hitter Detection, Differential Privacy, Prefix-Tree Algorithms, Adaptive Tuning.

1 Introduction

Heavy hitter detection is crucial for analyzing user behavior across distributed data sources, enabling systems to detect popular data points (e.g., words, actions) while preserving the privacy of individual users. With the rise of privacy concerns, differential privacy (DP) has become a standard for safeguarding user data in statistical analyses. While previous methods such as RAPPOR and PI-RAPPOR have been effective for small domains, their computational and communication overheads increase significantly for large data universes, especially in federated learning scenarios.

In this work, we focus on optimizing prefix-tree based heavy hitter detection algorithms under differential privacy constraints. Prefix-tree based methods reduce the communication and computation costs by iteratively narrowing down the data universe, but they also introduce challenges such as managing false positives and selecting optimal segment lengths. We propose adaptive algorithms that tackle these issues through dynamic hyperparameter tuning and deny-list integration, allowing us to achieve higher utility within constrained system resources.

We introduce an adaptive hyperparameter tuning algorithm that dynamically adjusts prefix segment lengths and thresholds to enhance the accuracy of heavy hitter detection while ensuring differential privacy. We propose novel data selection mechanisms, particularly useful when users report multiple data points, and incorporate adaptive deny-list techniques to further optimize utility. Extensive experiments on the Reddit dataset demonstrate the effectiveness of our methods, highlighting the balance between privacy, communication, and computational efficiency, and validating the improved performance of our approach.

Backgrounds. Gaining insight into population trends allows data analysts to make data-driven decisions to improve user experience. Heavy hitter detection, or learning popular data points generated by users, plays an important role in learning about user behavior. A well-known example of this is learning “out-of-vocabulary” words typed on keyboard, which can then be used to improve next word prediction models. This data is often sensitive and the privacy of users’ data is paramount. When the data universe is small, one can obtain private solutions to this problem by directly using private histogram algorithms such as RAPPOR [EPK14], and PI-RAPPOR [FT21], and reading off the heavy-hitters. However, when the data universe is large, as is the case with “out-of-vocabulary” words, these solutions result in algorithms with either very high communication, or very high server side computation, or both. Prefix-tree based iterative algorithms can lower communication and computation costs, while maintaining high utility by efficiently exploring the data universe for heavy hitters. They also offer an additional advantage in the setting where users have multiple data points by refining the query in each iteration using the information learned thus far, allowing each user to select amongst those data points which are more likely to be heavy hitters.

In this work, we consider an iterative federated algorithm for heavy hitter detection in the aggregate model of differential privacy (DP) in the presence of computation or communication constraints. In this setting, each user has a private dataset on their device. In each round of the algorithm, the data analyst sends a query to the set of participating devices, and each participating device responds with a *response*, which is a random function of the private dataset of that user. These *responses* are then summed using a secure aggregation protocol, and reported to the data analyst. The analyst can then choose a query for the next round adaptively, based on the aggregate results they have seen so far. The main DP guarantee is a user-level privacy guarantee on the outputs of the secure aggregator, accounting for the privacy cost of *all* rounds of iteration. Our algorithm will additionally be DP in the local model of DP (with a larger privacy parameter)¹. We do not assume that the set of participating devices is consistent between rounds.

In the central model of DP, there is a long line of work on adaptive algorithms for heavy hitter detection in data with a hierarchical structure such as learning popular n -grams [CPS⁺12, QYL12, SCS13, BKM⁺21, KGKY21, MJT⁺22]. These interactive algorithms all follow the same general structure. Each data point is represented as a sequence of data segments $d = a_1 a_2 \cdots a_r$ and the algorithm iteratively finds the popular values of the first segment a_1 , then finds popular values of $a_1 a_2$ where a_1 is restricted to only heavy hitters found in the previous iteration, and so on. This limits the domain of interest at each round, lowering communication and computation costs. The method of finding the heavy hitters in each round of the algorithm varies in prior work, although is generally based on a DP frequency estimation subroutine. One should consider system constraints (communication, computation, number of participating devices, etc.) and the privacy model when choosing a frequency estimation subroutine. In this work, we will focus on using one-hot encoding with binary randomized response (inspired by RAPPOR [EPK14]) as our DP frequency estimation subroutine. Since we are primarily interested in algorithmic choices that affect the iterative algorithm, we believe our findings should be agnostic to the choice of frequency estimation subroutine used.

We explore the effect on utility of different data selection schemes and algorithmic optimizations. We refer to our algorithm as *Optimized Prefix Tree* (**OptPrefixTree**). Our contributions are summarised below:

¹A potential architecture for running iterative algorithms in this model of privacy is outlined in [MJT⁺22].

Adaptive Segmentation. We propose an algorithm for adaptively choosing the segment length and the threshold for keeping popular prefixes. In contrast to prior works that treat the segment length as a hyperparameter, our algorithm chooses these parameters in response to user data from the previous iteration and attempts to maximize utility (measured as the fraction of the empirical probability distribution across all users captured by the returned heavy hitters), while satisfying any system constraints. We find that our method often results in the segment length varying across iterations, and outperforms the algorithm that uses a constant segment length. We also design a threshold selection algorithm that adaptively chooses the prefix list for the subsequent round. This allows us to control the false positive rate (the likelihood that a data point is falsely reported as a heavy hitter).

Analysis of the effect of on-device data selection mechanisms. We explore the impact of interactivity in the setting where users have multiple data points. We observe empirically that when users have multiple data points, interactivity can improve utility, even in the absence of system constraints. In each round, users choose a single data point from their private data set to (privately) report to the server. The list of heavy hitters in the previous iteration provides a *prefix list*, so users will only choose a data point with one of the allowed prefixes. If a user has several data points with allowed prefixes, then there are several selection rules they may use to choose which data point to report. Each user’s private dataset defines an empirical distribution for that user. We find that when users sample uniformly randomly from the support of their distribution (conditioned on the prefix list) then the algorithm is able to find more heavy hitters than when they sample from their empirical distribution (again conditioned on the prefix list).

Analysis of the impact of inclusion of deny list. Under the constraint of user-level differential privacy, each user is only able to communicate their most frequent data points, and less frequent data points are down weighted. We explore the use of a *deny list* that asks users not to report data points that we already know are heavy hitters. In practice, a deny list may arise from an auxiliary data source, or from a prior run of the algorithm. Our analysis indicates even when the privacy budget is shared between multiple rounds of the algorithm, performing a second round equipped with a deny list improves performance.

The rest of the paper is organized as follows. In Section 2 we discuss some of the prior works in privacy-preserving heavy hitters detection. Section 3 explains the privacy primitives we used in this work. In Section 4 we elaborate the details of our prefix tree algorithm. Section 5 explains the post-processing methods and the theoretical analysis behind it. Section 6 demonstrates the experimental results and in Section 7 we discuss the findings of our experiments.

2 Related Works

Heavy hitters discovery methods have applications in various different domains [EEH⁺23]. This problem has been studied in both the local model [App17, WLJ19, ASZ19] and shuffle model [GGK⁺21] of differential privacy. Furthermore, recently different multi-party-computing [BBCG⁺21] methods and combination of multi-party-computing and DP techniques [BK21] have been proposed to find the top-k heavy hitters in different domains. In this work we focus on large domains and specifically iterative methods that allows us to satisfy system constraints.

In [ZKM⁺20], the authors propose an iterative algorithm to discover heavy hitters in the central model of differential privacy. The general framework of forming a tree-based structure is the same

to our Prefix Tree method except in their algorithm, **TrieHH**, samples a subset of devices ($\gamma\sqrt{N}$) in each iteration and uses the data points of these devices to compute the heavy hitters for the next iteration, without any additional noise and hence does not satisfy local differential privacy. They select the prefix list for the next iteration to be all the prefixes such that more than θ devices send the character in that iteration. The parameters γ and θ are chosen to achieve the required privacy guarantee.

TrieHH++ [CB22] is an extension to **TrieHH**. The authors use the same sampling and threshold algorithm as **TrieHH** to provide the (ϵ, δ) -aggregated differential privacy. However, they are able to support more general applications such as quantile and range queries. In addition to detecting heavy hitters, their method is able to report the frequency of heavy hitters without using additional privacy budget. To achieve their goal, they take advantage of Poisson sampling instead of fixed-size sampling to hide the exact number of samples. Consequently, releasing heavy hitters and their counts does not violate user privacy.

Set Union is a critical operation in many data related applications. Differentially Private Set Union (DPSU) methods are [GGK⁺20] popular for extracting n-grams, which is a common application in NLP algorithms. These methods attempt to find the largest subset of the union while satisfying DP. Authors in [WZL⁺20], samples a specific number of items per user and generates a histogram. Finally the items whose counts are above a certain threshold will be reported. In [CWG22], utility is boosted by privately reporting the frequencies to the server and eliminating the sampling step. They further take advantage of knowledge transfer from public datasets to achieve more accurate frequency estimation. Using public data as prior knowledge for private computation is investigated in various other works [LVS⁺21, BMN20]. In this paper, we use this knowledge transfer to explore the effect of using a deny list on the utility of the algorithm. Authors in [KGKY21] combined DPSU and tree based method to improve the utility of n-gram extraction model. The empirical results of their work imply that selecting more than one data point per device improved performance in the central DP setting. While we focus on data selection mechanisms that select a single data point per user per round, these mechanisms naturally extend to mechanisms that select more than one data point. In order to elaborate the impact of weighted vs. unweighted sampling, we focus on selecting a single data point per device. We leave an exploration of the optimal number of data points per device per iteration in the aggregate DP setting to future work.

3 Differential Privacy

In this work, we will consider an algorithm that satisfies device-level differential privacy (DP) in the aggregate model *and* the local model of differential privacy. We focus on device-level DP, which protects against a user changing all of the data points associated to them. Our primary privacy guarantee is the aggregate privacy guarantee, which will be specified ahead of time. Local differentially private guarantees are achieved locally on a user’s device through the use of a local randomizer. The local privacy guarantee will be set to be the largest epsilon such that the final algorithm satisfies the required aggregate privacy guarantee (i.e. we will not put constraints on the local privacy guarantee).

Local differentially private guarantees are achieved locally on a user’s device through the use of a local randomizer.

Definition 1 (Local Randomizer [DR14a, KLN⁺11]). *Let $\mathcal{A} : D \rightarrow \mathcal{Y}$ be a randomized algorithm mapping a data entry in D to an output space \mathcal{Y} . The algorithm \mathcal{A} is an ϵ -DP local randomizer if*

for all pairs of data entries $d, d' \in D$, and all events $E \subset \mathcal{Y}$, we have

$$-\varepsilon \leq \log \left(\frac{\Pr[\mathcal{A}(d) \in E]}{\Pr[\mathcal{A}(d') \in E]} \right) \leq \varepsilon.$$

The privacy parameter ε captures the *privacy loss* consumed by the output of the algorithm. Differential privacy for an appropriate ε ensures that it is impossible to confidently determine what the individual contribution was, given the output of the mechanism.

In general, differential privacy is defined for algorithms with input databases with more than one record. In the local model of differential privacy, algorithms may only access the data through a local randomizer so that no raw data leaves the device. For a single round protocol, local differential privacy is defined as follows:

Definition 2 (Local Differential Privacy [KLN⁺11]). *Let $\mathcal{A} : D^n \rightarrow \mathcal{Z}$ be a randomized algorithm mapping a dataset with n records to some arbitrary range \mathcal{Z} . The algorithm \mathcal{A} is ϵ -local differentially private if it can be written as $\mathcal{A}(d^{(1)}, \dots, d^{(n)}) = \phi(\mathcal{A}_1(d^{(1)}), \dots, \mathcal{A}_n(d^{(n)}))$ where the $\mathcal{A}_i : D \rightarrow \mathcal{Y}$ are ϵ -local randomizers for each $i \in [n]$ and $\phi : \mathcal{Y}^n \rightarrow \mathcal{Z}$ is some post-processing function of the privatized records $\mathcal{A}_1(d^{(1)}), \dots, \mathcal{A}_n(d^{(n)})$. Note that the post-processing function does not have access to the raw data records.*

We say a multi-round algorithm \mathcal{A} is ϵ -DP in the local model if it is the composition of single round algorithms which are DP in the local model, and the total privacy loss of \mathcal{A} is ϵ -DP. More generally, we can say that an interactive algorithm is locally differentially private if the transcript of all communication between the data subjects and the curator is differentially private [JMNR19]. Since aggregate differential privacy is our primary privacy guarantee, when we refer to local privacy guarantees, they will be for a single round of communication.

In aggregate DP, we assume the existence of an aggregation protocol that sums the local reports before they are released to the analyst. The aggregation protocol guarantees that the analyst does not receive anything about the locally DP reports *except their sum*².

Definition 3. *A single round algorithm \mathcal{A} is (ϵ, δ) -DP in the aggregate model if the output of the aggregation protocol on two datasets that differ on the data of a single individual are close. Formally, an algorithm $\mathcal{A} : D^n \rightarrow \mathcal{Z}$ is (ϵ, δ) -DP in the aggregate model if the following conditions both hold:*

- *it can be written as $\mathcal{A}(d^{(1)}, \dots, d^{(n)}) = \phi(\text{Aggregator}(f(d^{(1)}), \dots, f(d^{(n)})))$ where $f : D \rightarrow \mathcal{Z}$ is a randomized function that transforms that data, **Aggregator** is an aggregation protocol, and $\phi : \mathcal{Y}^n \rightarrow \mathcal{Z}$ is some post-processing of the aggregated report*
- *for any pair of datasets D and D' that differ on the data of a single individual, and any event E in the output space,*

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

Note that the post-processing function takes the aggregation as its input and does not have access to the individual reports.

²The aggregate model of DP is a derivative of the more general and common shuffle model of differential privacy introduced in [EFM⁺19, CSU⁺19].

When each user uses a local randomizer with a local DP guarantee to send their data to the aggregation protocol, the privacy guarantee in the aggregation model can be bounded by a quantity that is a function of both ε_l , the privacy guarantee in the local model, and n , the number of users that participate in the aggregation protocol [EFM⁺19, CSU⁺19]. In our experimental results, we will bound the aggregate DP epsilon by the numerical bound on privacy amplification by shuffling due to [FMT23]. Given an expected number of users, the number of iterations, and the desired aggregate privacy guarantee, we solve for the largest local epsilon that will achieve the given aggregate privacy guarantee. Since the aggregate privacy guarantee is our primary privacy guarantee, we do not put an upper bound on our local epsilon.

As with the local model, we will say a multi-round algorithm \mathcal{A} is ε -DP in the aggregate model if it is the composition of single round algorithms which are DP in the aggregate model, and the total privacy loss of \mathcal{A} is ε -DP. In order to analyse the privacy loss over multiple iterations, we will use a combination of the advanced composition theorem [DRV10, KOV17] and composition bound in terms of Rényi differential privacy [ACG⁺16, Mir17, CKS20]. Table 2 in Appendix B demonstrates how the local epsilon increases with the number of iterations, and the desired aggregate privacy guarantee.

4 Algorithm

In this section, we describe our proposed algorithm **OptPrefixTree**. This algorithm privately identifies the heavy hitters using T iterations where each device sends a locally differentially private report in each round and the local reports are aggregated before reaching the server. We will first discuss an outline of the high-level algorithm. In Section 5, we will discuss our proposals for adaptively setting the various parameters and subroutines present in the high-level algorithm. Our focus in the experimental section to follow will be to explore these choices, and provide some guidelines on how they should be chosen. Note that while our algorithm is run over multiple iterations, it is well-suited to the federated setting since it does not require every user to be present at every iteration.

We represent the system constraints as a constraint of the size of the data domain for any single iteration, denoted by P . This bound may be a result of communication constraints, as is the case for the local randomizer we will use (one-hot encoding with binary randomised response), or computational constraints for the server-side algorithm, as in PI-Rappor [FT21] or Proj-Rappor [FNNT22].

Notation. Let N be the total number of users. Let each user $i \in [N]$ have n_i data points denoted by $d_{i,1}, \dots, d_{i,n_i}$ from domain $D \subset U = A^r$, where U denotes the universe of allowable data points, and A denotes an alphabet from which all data points are built. For each user $i \in [N]$, let \mathbb{P}_i denote the empirical distribution of user i 's data and $\mathbb{P} := \frac{1}{N} \sum_{i \in [N]} \mathbb{P}_i$ denote the global empirical distribution³. Let each data point $d \in A^r$ be of a fixed length r .

4.1 Private Heavy Hitters Algorithm

OptPrefixTree proceeds in iterations with the goal of efficiently exploring the large data domain to detect heavy hitters, by exploiting the hierarchical structure by sequentially learning the most

³In the multiple data points per device setting, there are several other natural ways of defining the global empirical distribution. For example, one may define the frequency of a data point d to be the number of users who have the word d in their support. We briefly explore this metric in Appendix I.

Algorithm 1 Prefix tree based heavy hitter algorithm (**OptPrefixTree**)

```
1: Input:  $T$ : number of iterations,  $\epsilon_l$ : Local privacy parameter, selectData: Data selection mechanism,  $P$ : Bound on the dimension,  $FPR$ : false positive ratio,  $\eta$ : extra parameters to pass to ServerSide,  $\mathcal{P}_{\text{denylist}}$ : deny list.  
2: Output:  $\mathcal{P}_{\text{prefixlist}_{T+1}}$ : Set of Heavy Hitters  
3:  $l_1 \leftarrow \lfloor \log(P) \rfloor$ ,  $\mathcal{P}_{\text{prefixlist}_1} = \emptyset$ ,  $\mathcal{P}_{\text{discoveredlist}} = \emptyset$  // Initialize segment length and prefix list  
4: for  $t \in [T]$  do  
5:    $V_t \leftarrow \emptyset$  //  $V_t$  will be the set of all the device responses that is sent to the aggregation protocol  
  
6:   for  $i \in [N]$  do  
7:      $v_i \leftarrow \text{DeviceSide}_i(\epsilon_l, l_t, \mathcal{P}_{\text{prefixlist}_t}, \mathcal{P}_{\text{denylist}}, \text{selectData})$   
8:      $V_t \leftarrow V_t \cup v_i$   
9:   end for  
10:   $V_t \leftarrow \text{AggregationProtocol}(V_t)$  // Device responses are aggregated  
11:   $\mathcal{D}_t \leftarrow \mathcal{P}_{\text{prefixlist}_t} \times A^{l_t}$  // Data domain for iteration t  
12:   $\mathcal{P}_{\text{prefixlist}_{t+1}}, l_{t+1}, \mathcal{P}_{\text{discoveredlist}} \leftarrow \text{ServerSide}(V_t, \mathcal{D}_t, \mathcal{P}_{\text{discoveredlist}}, \epsilon_l, FPR, \eta)$  // Server returns prefix list and segment length for next round  
13:  Send  $\mathcal{P}_{\text{prefixlist}_{t+1}}$ , and  $l_{t+1}$  to all the devices  
14: end for  
15: return  $\mathcal{P}_{\text{prefixlist}_{T+1}} \cup \mathcal{P}_{\text{denylist}} \cup \mathcal{P}_{\text{discoveredlist}}$ 
```

popular prefixes, and only expanding on these popular prefixes in the next iteration. We give pseudo-code for our proposed algorithm **OptPrefixTree** in Algorithm 1. At every iteration t , the server sends the devices a list of live prefixes $\mathcal{P}_{\text{prefixlist}_t}$ of length $l_{\text{pref},t}$, a deny list $\mathcal{P}_{\text{denylist}}$, and a segment length l_t . The devices then use a data selection mechanism to choose a data point that is not in the deny list $\mathcal{P}_{\text{denylist}}$ and whose $l_{\text{pref},t}$ length prefix belongs in $\mathcal{P}_{\text{prefixlist}_t}$, they then (privately) report back the length $l_{\text{pref},t} + l_t (\leq r)$ prefix of the chosen data point. The server uses these local reports to define $\mathcal{P}_{\text{prefixlist}_{t+1}}$ (consisting of prefixes of length $l_{\text{pref},t} + l_t$) and the segment length l_{t+1} for the next round. We will use T to denote the number of iterations and ϵ_l to be the local DP parameter for a single iteration. At the end of T rounds, our algorithm outputs a set of heavy hitters that includes the prefixes found in the last iteration ($\mathcal{P}_{\text{prefixlist}_{T+1}}$) and the contents of $\mathcal{P}_{\text{denylist}}$. We use $(\epsilon_{\text{agg}}, \delta)$ -DP to refer to the privacy parameters of our algorithm in the aggregate model.

We may also remove some prefixes during earlier iterations which we add into the final set of heavy hitters. These data points are stored in the “discovered list”, $\mathcal{P}_{\text{discoveredlist}}$. We’ll discuss this further in a subsequent section, but this turns out to be a useful improvement when the natural encodings of data points can vary in length. For example, when using Huffman encoding.

4.2 Device Algorithm

The device first uses the data selection mechanism *selectData* to choose a data point from their on-device dataset that is not in the deny list, and whose $l_{\text{pref},t}$ -length prefix is in $\mathcal{P}_{\text{prefixlist}}$. Then we pass the $l_{\text{pref},t} + l_t$ -length prefix of the chosen data point to a ϵ_l -local DP algorithm A_{priv} and send the privatized output to the aggregation protocol. Algorithm 2 gives more details for the device-side algorithm.

Algorithm 2 Device side algorithm (**DeviceSide**)

```
1: Input:  $\epsilon_l$ : Local privacy parameter,  $l_{\text{pref}}$ : prefix length,  $l_t$ : segment length,  $\mathcal{P}_{\text{prefixlist}}$ : allowed  
   prefix list,  $\mathcal{P}_{\text{denylist}}$ : deny list, selectData: Function to choose a datapoint from the data  
2: Param:  $D$ : device dataset  
3: Output:  $v$ : Privatized output  
4:  $D = \{d \in D \mid d[0 : l_{\text{pref}}] \in \mathcal{P}_{\text{prefixlist}} \wedge d \notin \mathcal{P}_{\text{denylist}}\}$   
5: if  $D == \phi$  then  
6:    $d \leftarrow \perp$  // We reserve a special data element for users that have no eligible data points to  
   report.  
7: else  
8:    $d \leftarrow \text{selectData}(D)$   
9: end if  
10:  $v \leftarrow A_{\text{priv}}(d[0 : l_{\text{pref}} + l_t]; \epsilon_l)$   
11: return  $v$ 
```

The Local Randomizer In our experiments we use one-hot encoding with asymmetric binary randomized response (denoted by **OHE+2RR**) as the local randomizer. For details of this randomizer, see Appendix A of [MJT⁺22]. In practice one could use PI-RAPPOR [FT21] or Proj-RAPPOR [FNNT22] for better communication-computation trade-offs (See Appendix C). Since the utility guarantees of these mechanisms are very similar to **OHE+2RR**, we expect our findings on **OHE+2RR** to be directly applicable when using PI-Rappor or Proj-Rappor.

Under the constraint of local differential privacy, a participating device must still send a local report to the aggregation protocol, even if the device data contains no data points with a prefix in the prefix list. In our experiments if a device has no data point to communicate, then it encodes this as the all-zeros vector and uses asymmetric randomized response on the coordinates of the all-zeros vector. This has the same effect as adding a special element \perp to the data domain and having devices report the \perp element if they have no data points to report.

Data Selection We consider two data selection mechanisms. In *weighted selection*, each device i selects a data point by sampling from its empirical distribution \mathbb{P}_i conditioned on the datapoint having a prefix in $\mathcal{P}_{\text{prefixlist}_t}$ and not being in $\mathcal{P}_{\text{denylist}}$. Formally, we define the pdf of \mathbb{P}_i as follows: let $f_i(d_j)$ represent the frequency of d_j in the private data set on *device* _{i} so $\sum_{j=0}^{u_i} f_i(d_j) = 1$ where u_i is the number of unique data points on *device* _{i} . In *unweighted/uniform selection*, each device i selects a data point by sampling uniformly from those points in the support of \mathbb{P}_i which have a prefix in $\mathcal{P}_{\text{prefixlist}_t}$ and are not in $\mathcal{P}_{\text{denylist}}$. Note that the data selection mechanism does not impact the privacy guarantees. $d_{i,1}$

4.3 Server Algorithm

The server receives the aggregated privatized responses from the previous iteration, the prefix list and segment lengths from the previous iteration, and the local DP parameter ϵ_l . The general outline of the server-side algorithm is given in Algorithm 3.

The first step of this process is to compute an estimated frequency for the data domain of the last iteration $\mathcal{P}_{\text{prefixlist}_t} \times A^{l_t}$. Given the aggregated privatized results and the local epsilon, for every element, $d \in \mathcal{P}_{\text{prefixlist}_t} \times A^{l_t}$, of the data domain, the server can compute an estimate $\tilde{f}(d)$ of the number of devices who sent the data point d in the last iteration. The prefix list selection algorithm

Algorithm 3 Server side algorithm per round (**ServerSide**)

- 1: **Input:** V_t : Aggregated sum of devices responses, \mathcal{D}_t : Data domain of iteration t , ε_l : Local privacy parameter, FPR : False positive ratio, τ_0 : Initialization of threshold and η : Extra parameters for **PruneHH**
 - 2: **Output:** $\mathcal{P}_{\text{prefixlist}}$: Heavy hitters list
 - 3: $\tilde{f}(\cdot) \leftarrow A_{\text{privagg}}(V_t, \mathcal{D}_t; \varepsilon_l)$ // *Takes the aggregated privatized responses and computes an estimate of the frequency of every data element.*
 - 4: $\sigma \leftarrow \sqrt{\text{Var}(A_{\text{privagg}})}$ // *Computes an upper bound on the standard deviation of the frequency estimate for d*
 - 5: $\mathcal{P}_{\text{prefixlist}_{t+1}} \leftarrow \text{PruneHH}(\mathcal{D}, \tilde{f}(\mathcal{D}), \tau_0, FPR, \sigma, \eta)$
 - 6: $\mathcal{P}_{\text{prefixlist}_{t+1}}, \mathcal{P}_{\text{discoveredlist}} \leftarrow \text{RemoveFinished}(\mathcal{P}_{\text{prefixlist}_{t+1}}, \mathcal{P}_{\text{discoveredlist}})$ // *Removes any of the discovered prefixes which are "complete" data points and adds them to the discovered list.*
 - 7: $l_{t+1} = \max\{\ell \mid |\mathcal{P}_{\text{prefixlist}_{t+1}}| \times 2^\ell \leq P\}$ // *Adaptively chooses the maximum segment length*
 - 8: **return** $\mathcal{P}_{\text{prefixlist}_{t+1}}, l_{t+1}, \mathcal{P}_{\text{discoveredlist}}$
-

aims to keep as many of the elements $d \in \mathcal{P}_{\text{prefixlist}_t} \times A^l$ such that $\tilde{f}(d) > 0$ as possible, while minimizing the number of “false positives” in the prefix list (data elements which do not match the selected data point for *any* device). When using **OHE+2RR**, each estimate $\tilde{f}(d)$ is unbiased and the noise induced by the privatization scheme is approximately Gaussian with standard deviation σ , where σ is a function of ε_l and the number of participating devices. Due to the noise, if we were to define the $\mathcal{P}_{\text{prefixlist}_{t+1}}$ to be all elements such that $\tilde{f}(d) > 0$, the false positive rate would be too high. Instead, we use a threshold multiplier τ such that the prefix list $\mathcal{P}_{\text{prefixlist}_{t+1}}$ contains all the elements such that $\tilde{f}(d) \geq \tau\sigma$. This threshold should be chosen to be as small as possible while ensuring that the fraction of reported elements that are false positives does exceed a specified threshold denoted FPR .

In the setting where there are natural encodings of the data for which different data points have different lengths (e.g. when an encoding scheme such as Huffman encoding is used) we note a improvement that can be made when choosing the prefix list $\mathcal{P}_{\text{prefixlist}_{t+1}}$. In these cases, some data points may be complete before the end of the algorithm. To avoid unnecessary communication and utilize the system capacity, an end character symbol can be used at the end of each encoding so that the server can detect when a data point is “complete”. After aggregating the data on the server if there are prefixes that reach the end character, they are added to the list of already discovered prefixes and removed from the prefix list sent to the devices for the next iteration. They are then added to the set of heavy hitters in the final output.

In most of the prior works, the segment length and threshold τ are treated as hyperparameters that need to be tuned. Tuning hyperparameters is notoriously hard in the federated setting. In Section 5 we will discuss our adaptive algorithms for choosing these parameters. Our proposed algorithms choose these parameters in response to user data, without using additional privacy budget.

5 Adaptive Thresholding and Segmentation

In this section, we will describe our adaptive segmentation and thresholding algorithms. The algorithms aim to keep as many heavy hitters as possible, while maintaining a specific false positive rate, and satisfying the data domain size constraint for the next iteration.

Algorithm 4 Pruning algorithm for confident heavy hitter detection ([PruneHH](#))

```

1: Input:  $\mathcal{D}, \tilde{f}[\mathcal{D}]$ : query set and estimated frequencies after aggregation,  $\tau_0$ : Initialization of
   threshold,  $FPR$ : ratio of expected false positives to the total number of bins,  $\sigma$ : aggregated
   noise standard deviation,  $\eta$ : step size
2:  $E \leftarrow 1 - \Phi(\tau_0 \sigma)$ 
3:  $\mathcal{P}_{\text{prefixlist}}' = \{q \in \mathcal{D} \mid \tilde{f}[q] > \tau_0 \times \sigma\}$ 
4: while  $FPR < \frac{E \times |\mathcal{D}|}{|\mathcal{P}_{\text{prefixlist}}'|}$  do
5:    $E \leftarrow \eta * E$ 
6:    $\tau \leftarrow z_{(1-E)}$ 
7:    $\mathcal{P}_{\text{prefixlist}}' = \{q \in \mathcal{D} \mid \tilde{f}[q] > \tau \times \sigma\}$ 
8: end while
9: return  $\mathcal{P}_{\text{prefixlist}}$ 

```

5.1 Adaptive Thresholding

Let us first discuss our proposal for how to choose the threshold τ . Given a threshold τ and standard deviation σ , let E denotes the probability that a data point with true count zero (i.e. no device contributes this data point) has an estimated count above $\tau\sigma$. That is, the probability of a false positive. As discussed earlier, for any d , the estimate $\tilde{f}(d)$ is approximately Gaussian with standard deviation σ so we can compute E based on the Gaussian approximation, i.e. $E = 1 - \Phi(\tau\sigma)$, where $\Phi()$ denotes the gaussian CDF (with mean 0 and standard deviation σ)⁴.

In the while loop (line 4 to 7), we first compute the expected number of false positive bins by $|\mathcal{D}| \times E$ where $|\mathcal{D}|$ represents the aggregated data domain size. We then make sure that the ratio of the expected number of false positives to the number of data points such that $\tilde{f}(d) \geq \tau\sigma$ (represented by $|\mathcal{P}_{\text{prefixlist}}'|$) does not go above a specified threshold. If the ratio of the expected number of false positives that threshold τ to the number of data elements with estimated frequency above $\tau\sigma$ exceeds the specified false positive ratio, we change the confidence level to the point that we make sure the algorithm satisfies this parameter.

5.2 Adaptive Segmentation

Given the prefix list, the segment length is adaptively chosen to be as large as possible while maintaining the dimension constraint, $l_{t+1} = \arg \max_{\ell} \{\ell \mid |\mathcal{P}_{\text{prefixlist}_{t+1}}| \cdot 2^{\ell} \leq P\}$. In the single data point per device setting, intuitively, there are two opposing factors in the performance of the private heavy hitters algorithm — the privacy budget per iteration (which decreases with increase in T) and the size of the total search space (which is smaller for algorithms with smaller segmentation lengths and hence more iterations). In this section, we outline an argument illustrating that the effect of decreasing in privacy budget per iteration dominates and it is better to minimize the number of iterations. We also illustrate this via experiments. Thus, we choose each segment length to be as large as possible while retaining as many popular prefixes (with suitable confidence) as possible and maintaining the dimension constraint.

⁴if d has true count 0, then the distribution of $\tilde{f}(d)$ is actually a shifted, scaled Binomial distribution. If n is small enough that the Gaussian approximation is not accurate, then one can use high probability bounds on the Binomial.

Intuitive theoretical analysis of OptPrefixTree for single datapoint In this part, we try to obtain guidelines for how to set the segment length in the a single data point per device setting. We provide analysis for running OptPrefixTree for one round ($T = 1$) searching over the whole high dimensional universe A^r . While this may be impractical to implement, it provides us with setting of hyperparameters in the case of no computation and communication constraints. We analyze the algorithm assuming good performance of the local randomizer A_{priv} , frequency estimator A_{privagg} pair. We formalize this assumption as follows:

Assumption 1. For any $\beta \in [0, 1]$, for any element x in the domain, with probability at least $1 - \beta$, we have,

$$|F(x) - \tilde{f}(x)| \leq C_1 \sqrt{\frac{ne^{\epsilon_l}}{(e^{\epsilon_l} - 1)^2} \log\left(\frac{1}{\beta}\right)} = C_2 \frac{\sqrt{\log(1/\delta) \log(1/\beta)}}{\epsilon_{\text{agg}}},$$

where C_1 and C_2 are absolute constants, F is the global empirical distribution and \tilde{f} is the estimated empirical distribution.

We note that this assumption is satisfied by OHE+2RR, as well as other common frequency estimation algorithms such as PI-Rappor. For the purpose of this analysis, we will use a different metric to the metric that will be the main focus in our experimental results. We will measure the performance of OptPrefixTree using a metric we define in Definition 4, which is standard in the differentially private heavy hitters literature [BNSGT17].

Definition 4 (λ -accurate). A set of heavy hitters $\mathcal{P}_{\text{prefixlist}_T}$ is said to be (λ, A) -accurate if it satisfies the following:

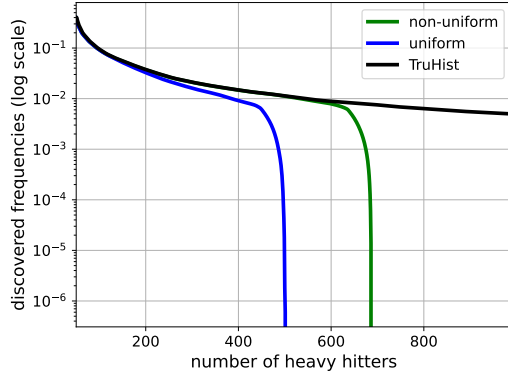
- For all $d \in U$, if $F(d) \geq A + \lambda$, then $d \in \mathcal{P}_{\text{prefixlist}_T}$.
- For all $d \in U$, if $F(d) < A - \lambda$, then $d \notin \mathcal{P}_{\text{prefixlist}_T}$.

We now prove the utility of OptPrefixTree when we have one iteration with $l = r$ in Proposition 1. We first state the result and discuss its implications, deferring the proof to Appendix D.

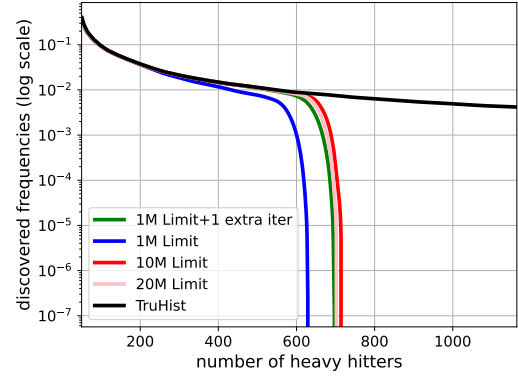
Proposition 1. Let the local randomizer (A_{priv}) and frequency estimator (A_{privagg}) pair satisfy Assumption 1 and let $\mathcal{P}_{\text{prefixlist}_1}$ be the output of OptPrefixTree when run for $T = 1$ round with $l = r$ and the query set $\mathcal{D} = A^r$ and aggregate DP parameters $(\epsilon_{\text{agg}}, \delta)$. Then, with probability at least $(1 - \beta)$, $\mathcal{P}_{\text{prefixlist}_1}$ is $(\lambda, \tau\sigma)$ -accurate with $\lambda = O\left(\frac{1}{\epsilon_{\text{agg}}} \sqrt{\log(1/\delta) \log\left(\frac{|A|^r}{\beta}\right)}\right)$, where $\tau\sigma$ is the final threshold.

While this only give us upper bound on the performance of the single iteration algorithm, it does help us gain some intuition into how to set the segmentation length. Suppose we were to run the algorithm for T iterations with even segmentation (i.e. $l_t = r/T$ for all t). Then, at each iteration, we would need the aggregate privacy guarantee for that round to be $\approx \epsilon_{\text{agg}}/\sqrt{T}$. If the data domain size per iteration was exactly $|A|^{r/T}$ then the impact of T on the error would approximately cancel (ignoring logarithmic terms that arise from ensuring true heavy hitters survive at each iteration). However, the data domain at each round is *greater* than $|A|^{r/T}$ pushing us towards using a single round.

This intuition does not generalize to the multiple data points per device setting as it is possible for iterations allows us to more intelligently select the data points that users contribute. This is aligned with our empirical results in Appendix E.2 which indicates sending one character at a time improves the utility. However, we conjecture that reducing the number of iterations per round, and including multiple rounds with deny-lists will improve utility.



(a) Discovered frequencies for different segmentation schemes in [OptPrefixTree](#)



(b) Discovered frequencies for different dimension limits in [OptPrefixTree](#)

Figure 1. Experimental results in the single data point per device setting with $\varepsilon_{agg} = 1$, $\delta = 10^{-6}$, and $T = 4$

6 Experiments

Evaluation Dataset: For our evaluations, we use the Reddit public dataset which contains the data of 1.6 million users’ comments posted on social media in December 2017 [CDW⁺18]. On the Reddit dataset each device has an average of 1092 words and an average of 379 unique words. For investigations on the single data point per device setting, each device i samples a single data point from \mathbb{P}_i . This data point remains fixed during the multiple iterations of the algorithm. Unless stated otherwise, we use a Huffman encoding to translate the words into binary strings. One token is reserved for unknown characters and we have an end character encoding at the end of each bit stream. We set the system constraint $P = 10^7$ and $r = 60$ in all experiments.

Evaluation metric: Let $H = (x_1, x_2, \dots, x_{|H|})$ denote the set of heavy hitters output by an algorithm ordered by the empirical global frequency distribution \mathbb{P} . For our evaluations in this section, we report the frequencies (according to \mathbb{P}) of heavy hitters output by the algorithm. To aid with visualization in our plots, for a window size $W = 50$ and a given heavy hitter set H , we plot for each i , the sum of the probabilities (according to \mathbb{P}) of the heavy hitters in the sliding window (more details on plots are in Appendix E).

6.1 Adaptive Segmentation in Single Data Point Setting

In this section, we focus on the simpler single data point per device setting. For these evaluations we used $\varepsilon_{agg} = 1$ and $\delta = 10^{-6}$.

Adaptive Segmentation We demonstrate the benefit of adaptively choosing the segment length, as opposed to using fixed-length uniform segment lengths. For our experimental evaluations in this part we set our data domain limit to $P = 10^7$ which means that the dimension of each payload of size $2^{l_t} \times |\mathcal{P}_{\text{prefixlist}_t}|$ should not exceed this limit. For both of the configurations in this part, we limit $T = 4$. In one configuration we used the fixed segment length of 15 for all the iterations. On the other hand, with [OptPrefixTree](#) in each iteration we select the largest possible segmentation length for the next iteration based on the dimension limitation and the size of the prefix list for the

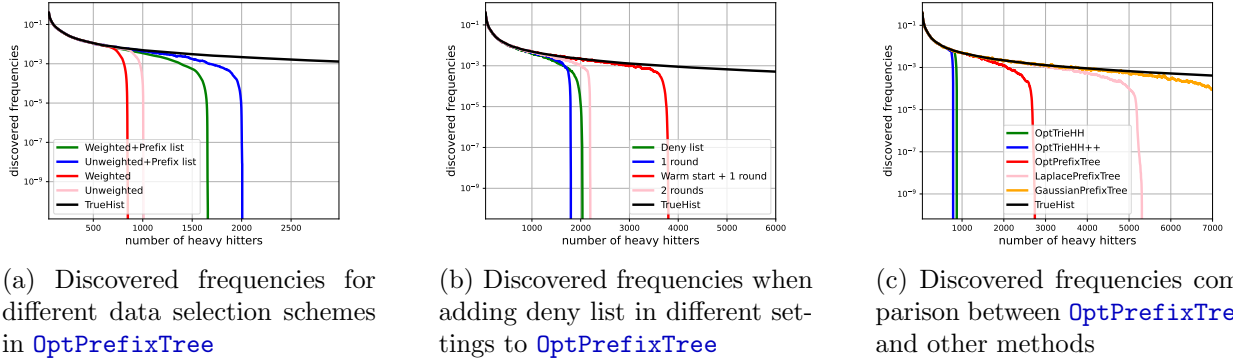


Figure 2. Experimental results in the multiple data points per device setting with $\varepsilon_{agg} = 1$, $\delta = 10^{-6}$

next iteration. Fig. 1a compares the discovered normalized frequencies for the algorithm which uses uniform segment lengths ($l_t = 15$ for all t) to the algorithm which selects the segments adaptively, which leads to segment lengths [23, 14, 11, 12]. Our adaptive scheme is able to discover 40% more heavy hitters (more plots in Fig. 14). We use adaptive thresholding in both algorithms, with $FPR = 0.5$. However, the empirical FPR is even lower with the values of 0.35 and 0.41 for adaptive and uniform segment algorithms, respectively.

Dimension Limitation In production-scale systems the computation cost of decompressing and aggregating the responses can be a significant bottleneck when the data comes from a very large domain. Increasing system limits can be expensive and resource intensive. Thus, we explore the effect of different dimension constraints on the performance of **OptPrefixTree**. In order to do so, we used different constraints of $P = 20, 10$, and 1 million. This limitation in iteration t specifies an upper bound on $2^{l_{t+1}} \times |\mathcal{P}_{\text{prefixlist}_t}|$. First for all the configurations we set the number of iterations to 4. As illustrated in Figure 1b changing the limit to 1 million degrades the utility of the algorithm significantly. This is because the small number of iterations and small system limit means that in the final round, the prefix list has to be smaller than desired in order to set the segment length large enough to finish the algorithm in the desired number of iterations. One way to compensate this degradation is to allow the algorithm to run in 5 iterations instead of 4. To account for the total privacy budget in all the iterations, by having one more iteration, the per iteration local epsilon ε_l decreases (check Table 2). However, with 5 iterations, the algorithm is able to detect 15% more heavy hitters in comparison to the same dimension limit of 1 million when using 4 iterations. For these experiments we set the $FPR = 0.5$. However the empirical FPR is 0.42, 0.46, 0.33, and 0.42 for $1M + 1$ extra iteration, $1M$, $10M$, and $20M$ dimension limit, respectively.

We explored other constraints effect on the utility of the algorithm in Appendix E. One other important constraint is the number of payloads a system can receive in a single iteration. In order to limit the number of payloads, we use Poisson sampling on the devices at each iteration, so each device tosses a coin and decides to participate with probability p . This reduces the number of devices participating, and allows us to take advantage of privacy amplification by sampling. For the evaluations, we use the privacy amplification by sampling for Rényi DP bounds due to [ZW19].

6.2 Data Selection for Multiple Data Points per Device

Now we will explore the setting where each device has multiple data points. In these experiments we will set $\varepsilon_{agg} = 1$ and $\delta = 10^{-6}$.

Effect of Data Selection First, we evaluate the effect of different data selection schemes, specifically focusing on weighted vs unweighted data selection. In order to highlight the benefit of conditioning on the prefix list during data selection, we also compare against the version of these schemes that do not take the prefix list into account when selecting a data point. We will refer to the version where the selected data point is conditioned to belong in the prefix list as weighted selection + prefix list and unweighted selection + prefix list, and the versions where the selected data point is not forced to be in the prefix list as weighted selection and unweighted selection. In this experiment we used $T = 4$. Fig. 2a shows the experimental results using the different data selection schemes. Perhaps surprisingly, unweighted sampling outperforms weighted sampling in both the with and without prefix list experiments. One explanation for this observation is that in unweighted selection, moderately frequent words are given more opportunity to participate to the final output, while in weighted selection, the most frequent words are selected by all the users. That is, perhaps unweighted selection allows us to explore further into the tails of the distributions. As expected, conditioning on the prefix list has a significant impact. For the rest of this paper, we use unweighted sampling conditioned on the prefix list for on-device data selection. For these experiments we set the $FPR = 0.5$. However the empirical FPR is 0.30, 0.35, 0.35, and 0.38 for weighted selection + prefix list, unweighted selection + prefix list, weighted selection, and unweighted selection respectively. For more evaluations please refer to Fig. 16.

Effect of adding a deny list In practice, it is possible that an analyst has prior knowledge about some of the heavy hitters before the start of the algorithm. In such cases, we can take advantage of a deny list which includes this prior knowledge. When sending query to the devices, we can also send the deny list and ask the devices to exclude these already discovered data points during data selection. We explore two ways of obtaining a deny list, $\mathcal{P}_{denylist}$. In the first case, the deny list comes from an auxiliary public data source (which is not protected with DP), we will refer this as a **warm start**. The second is when we run the algorithm twice, with the heavy hitters from the first round forming the deny list of the second round (referred to as **2 rounds**). In **2 rounds** case, we need to account for the privacy budget of *both* rounds, which leads to more noise per iteration. In Fig. 2b we compare the different configurations. The warm start is initiated with the top 2000 popular words from Twitter Sentiment140 dataset [GBH09]. We also present the utility of the warm start $\mathcal{P}_{denylist}$ on its own (denoted by **deny list** line), showing the difference between the distribution of the two datasets. For comparison, we also include the standard algorithm run for a single round, without a deny list (denoted by **1 round** line). We further show the benefit of using deny list in one round of algorithm execution. Finally, we first execute one round of algorithm without a warm start and form a deny list out of discovered prefixes, which is then used in a second round. Fig. 2b shows that adding a deny list significantly improves performance. Adding warm start leads to discovering $2.1\times$ more heavy hitters. The 2 round algorithm increases the number of discovered heavy hitters by $1.22\times$. For twitter sentiment140 dataset only 0.006 of the data points in this dataset do not belong to Reddit dataset. For the rest of configurations, we set the $FPR = 0.5$. For **1 round** the empirical FPR is 0.45 while for **1 round + warm start** it's 0.37. For **2 rounds** of algorithm 0.39 of discovered bins are false positives. For more evaluations regarding deny list, please refer to Fig. 17.

Comparison to prior works We compare with `TrieHH` (`OptTrieHH`) and `TrieHH ++` (`OptTrieHH ++`), Central Laplace (`LaplacePrefixTree`), and Central Gaussian (`GaussianPrefixTree`) with the optimisation that we use our adaptive segmentation to determine the segment length at each round, in the setting where each device has multiple data points. For all of the algorithms we use the unweighted data selection, which performed the best in our previous experiments. To have a fair comparison, the same binary encoding is used for all of the models (5 bits per character, not Huffman encoding) and we used 12 iterations (1 character per iteration) that shows the best performance for this encoding for all the methods.

In `OptTrieHH` and `OptTrieHH ++`, for $\varepsilon_{agg} = 1$ and $\delta = 10^{-6}$, we set the threshold for the number of reports that needs to be received for a word to be a part of the prefix list (denoted by θ in their paper) to 10. Accordingly we set the sampling rate based for `TrieHH` based on Corollary 1 in [ZKM⁺20] and for `TrieHH ++` based on Lemma 3 in [CB22]. Our analysis in Fig. 2c indicates `OptPrefixTree` is able to discover 3.2 times more heavy hitters for the same number of iterations and same dimension constraint. One explanation for this performance difference is that `OptTrieHH` and `OptTrieHH ++` use sampling and thresholding to achieve the aggregated privacy guarantee, without adding any local differential privacy noise. When we set the threshold (θ) to 10, the sampling rate is 0.0079 and 0.0071 for `TrieHH` and `TrieHH ++` respectively. Hence, the low sampling rates required to achieve the privacy guarantee results in sampling error in the distribution that is larger than the noise injected by our mechanism. In Appendix F and Appendix G we explore the effect of different numbers of iterations in the utility of `TrieHH` and `TrieHH ++` for both single and multiple data points setting.

We further compare our method with a state-of-art central differential privacy methods. The assumption here is that a trusted curator is able to collect the device’s data and add central noise. We used two types of central noise for our analysis. For providing the central DP privacy guarantee we add Gaussian or Laplace noise to each dimension of the histogram [DR⁺14b, Dwo06]. As with our previous experiments we compute the privacy guarantee of the composition over multiple rounds using the advanced composition theorem [DRV10, KOV17] and the composition bound in terms of Rényi differential privacy [ACG⁺16, Mir17]. We compute the Rényi privacy guarantees of both noise addition methods using the bounds provided in Table 2 of [Mir17]. We finally select the noise power using the tighter bound between these two. As shown in Fig. 2c both of the central DP scenarios outperform `OptPrefixTree`. While `GaussianPrefixTree` can be implemented in our model (by distributing the Gaussian noise among the participating devices), it suffers from a significantly worse local privacy guarantee.

7 Conclusion

In this work we shed light on the importance of adaptive segmentation and intelligent data selection in heavy hitter detection algorithms. We conducted various experiments to find the optimum adaptive segmentation scheme based on the computation and communication constraints. In addition to comparing different data selection schemes, we demonstrated the benefit of using a prefix list and deny list for improving the utility of `OptPrefixTree`. Moreover, `OptPrefixTree` is designed to satisfy both local and aggregated differential privacy.

References

- [ACG⁺16] Martín Abadi, Andy Chu, Ian J. Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 308–318, 2016.
- [App17] Apple. Apple differential privacy technical review, 2017.
- [ASZ19] Jayadev Acharya, Ziteng Sun, and Huanyu Zhang. Hadamard response: Estimating distributions privately, efficiently, and with little communication. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1120–1129. PMLR, 2019.
- [BBCG⁺21] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776. IEEE, 2021.
- [BK21] Jonas Böhler and Florian Kerschbaum. Secure multi-party computation of differentially private heavy hitters. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pages 2361–2377, 2021.
- [BKM⁺21] Eugene Bagdasaryan, Peter Kairouz, Stefan Mellem, Adrià Gascón, Kallista Bonawitz, Deborah Estrin, and Marco Gruteser. Towards sparse federated analytics: Location heatmaps under distributed differential privacy with secure aggregation. *arXiv preprint arXiv:2111.02356*, 2021.
- [BMN20] Raef Bassily, Shay Moran, and Anupama Nandi. Learning from mixtures of private and public populations. *Advances in Neural Information Processing Systems*, 33:2947–2957, 2020.
- [BNSGT17] Raef Bassily, Kobbi Nissim, Uri Stemmer, and Abhradeep Guha Thakurta. Practical locally private heavy hitters. *Advances in Neural Information Processing Systems*, 30, 2017.
- [CB22] Graham Cormode and Akash Bharadwaj. Sample-and-threshold differential privacy: Histograms and applications. In *International Conference on Artificial Intelligence and Statistics*, pages 1420–1431. PMLR, 2022.
- [CDW⁺18] Sebastian Caldas, Sai Meher Karthik Duddu, Peter Wu, Tian Li, Jakub Konečný, H Brendan McMahan, Virginia Smith, and Ameet Talwalkar. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.
- [CKS20] Clément L Canonne, Gautam Kamath, and Thomas Steinke. The discrete gaussian for differential privacy. *Advances in Neural Information Processing Systems*, 33:15676–15688, 2020.
- [CPS⁺12] Graham Cormode, Cecilia Procopiuc, Divesh Srivastava, Entong Shen, and Ting Yu. Differentially private spatial decompositions. In *2012 IEEE 28th International Conference on Data Engineering*, pages 20–31. IEEE, 2012.
- [CSU⁺19] Albert Cheu, Adam Smith, Jonathan Ullman, David Zeber, and Maxim Zhilyaev. Distributed differential privacy via shuffling. In Yuval Ishai and Vincent Rijmen, editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 375–403, Cham, 2019. Springer International Publishing.
- [CWG22] Ricardo Silva Carvalho, Ke Wang, and Lovedeep Singh Gondara. Incorporating item frequency for differentially private set union. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9504–9511, 2022.
- [DMNS06] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*, 2006.
- [DR14a] Cynthia Dwork and Aaron Roth. The Algorithmic Foundations of Differential Privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3–4):211–407, 2014.
- [DR⁺14b] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407, 2014.

- [DRV10] Cynthia Dwork, Guy N. Rothblum, and Salil P. Vadhan. Boosting and differential privacy. In *FOCS*, 2010.
- [Dwo06] Cynthia Dwork. Differential privacy. In *Automata, Languages and Programming: 33rd International Colloquium, ICALP 2006, Venice, Italy, July 10-14, 2006, Proceedings, Part II 33*, pages 1–12. Springer, 2006.
- [EEH⁺23] Ahmed Roushdy Elkordy, Yahya H Ezzeldin, Shanshan Han, Shantanu Sharma, Chaoyang He, Sharad Mehrotra, Salman Avestimehr, et al. Federated analytics: A survey. *APSIPA Transactions on Signal and Information Processing*, 12(1), 2023.
- [EFM⁺19] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '19, page 2468–2479, USA, 2019. Society for Industrial and Applied Mathematics.
- [EPK14] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.
- [FMT23] Vitaly Feldman, Audra McMillan, and Kunal Talwar. Stronger privacy amplification by shuffling for rényi and approximate differential privacy. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 4966–4981. SIAM, 2023.
- [FNNT22] Vitaly Feldman, Jelani Nelson, Huy Nguyen, and Kunal Talwar. Private frequency estimation via projective geometry. In *International Conference on Machine Learning*, pages 6418–6433. PMLR, 2022.
- [FT21] Vitaly Feldman and Kunal Talwar. Lossless compression of efficient private local randomizers. In *International Conference on Machine Learning*, pages 3208–3219. PMLR, 2021.
- [GBH09] Alec Go, Richa Bhayani, and Lei Huang. Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009, 2009.
- [GGK⁺20] Sivakanth Gopi, Pankaj Gulhane, Janardhan Kulkarni, Judy Hanwen Shen, Milad Shokouhi, and Sergey Yekhanin. Differentially private set union. In *International Conference on Machine Learning*, pages 3627–3636. PMLR, 2020.
- [GGK⁺21] Badih Ghazi, Noah Golowich, Ravi Kumar, Rasmus Pagh, and Ameya Velingker. On the power of multiple anonymous messages: Frequency estimation and selection in the shuffle model of differential privacy. In *Advances in Cryptology–EUROCRYPT 2021: 40th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, October 17–21, 2021, Proceedings, Part III*, pages 463–488. Springer, 2021.
- [JMNR19] Matthew Joseph, Jieming Mao, Seth Neel, and Aaron Roth. The role of interactivity in local differential privacy. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 94–105. IEEE, 2019.
- [JRSS21] Palak Jain, Sofya Raskhodnikova, Satchit Sivakumar, and Adam D. Smith. The price of differential privacy under continual observation. *ArXiv*, abs/2112.00828, 2021.
- [KGKY21] Kunho Kim, Sivakanth Gopi, Janardhan Kulkarni, and Sergey Yekhanin. Differentially private n-gram extraction. *ArXiv*, abs/2108.02831, 2021.
- [KLN⁺11] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.
- [KOV17] Peter Kairouz, Sewoong Oh, and Pramod Viswanath. The composition theorem for differential privacy. *IEEE Transactions on Information Theory*, 63(6):4037–4049, 2017.

- [LVS⁺21] Terrance Liu, Giuseppe Vietri, Thomas Steinke, Jonathan Ullman, and Steven Wu. Leveraging public data for practical private query release. In *International Conference on Machine Learning*, pages 6968–6977. PMLR, 2021.
- [Mir17] Ilya Mironov. Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pages 263–275. IEEE, 2017.
- [MJT⁺22] Audra McMillan, Omid Javidsbakht, Kunal Talwar, Elliot Briggs, Mike Chatzidakis, Junye Chen, John Duchi, Vitaly Feldman, Yusuf Goren, Michael Hesse, et al. Private federated statistics in an interactive setting. *arXiv preprint arXiv:2211.10082*, 2022.
- [QYL12] Wahbeh Qardaji, Weining Yang, and Ninghui Li. Differentially private grids for geospatial data. *Proceedings - International Conference on Data Engineering*, 09 2012.
- [SCS13] Shuang Song, Kamalika Chaudhuri, and Anand D. Sarwate. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 245–248, 2013.
- [WLJ19] Tianhao Wang, Ninghui Li, and Somesh Jha. Locally differentially private heavy hitter identification. *IEEE Transactions on Dependable and Secure Computing*, 18(2):982–993, 2019.
- [WZL⁺20] Royce J Wilson, Celia Yuxin Zhang, William Lam, Damien Desfontaines, Daniel Simmons-Marengo, and Bryant Gipson. Differentially private sql with bounded user contribution. *Proceedings on privacy enhancing technologies*, 2020(2):230–250, 2020.
- [ZKM⁺20] Wennan Zhu, Peter Kairouz, Brendan McMahan, Haicheng Sun, and Wei Li. Federated heavy hitters discovery with differential privacy. In *International Conference on Artificial Intelligence and Statistics*, pages 3837–3847. PMLR, 2020.
- [ZW19] Yuqing Zhu and Yu-Xiang Wang. Poission subsampled rényi differential privacy. In *International Conference on Machine Learning*, pages 7634–7642. PMLR, 2019.

Notation	Definition
N	Number of users
$d_{i,j}$	j^{th} Datapoint of user i
n_i	Number of datapoints of user i
\mathbb{P}	True underlying distribution of users
D	Datapoint domain
A	Alphabet domain
U	Universe of all possible datapoints
T	Number of unknown dictionary rounds
l_t	Segment length in iteration t
r	Fixed total length of all the words
\mathcal{H}_t	histogram in iteration t
$\mathcal{P}_{\text{prefixlist}_t}$	Prefix list after iteration t
$v_{i,t}$	private data of user i after iteration t
τ	threshold multiplier
P	Dimension limit
σ	standard deviation of the noise
FPR	Ratio of number of the false positives to total discovered
$\tilde{f}(x)$	estimated frequency of data point x

Table 1: Notations

A Notation table

B Differential Privacy

In this work, we will consider an algorithm that satisfies two levels of privacy protection appropriate for federated learning; differential privacy in the aggregate model and differential privacy in the local model. For more details on a potential system for achieving these guarantees please see [MJT⁺22]. In the federated setting where users may have more than one data point, there are two main choices for the granularity of the privacy guarantee: device-level DP and event (or data point)-level DP. We will focus on the stronger of these two guarantees, device-level DP, which protects against a user changing all of the data points associated to them. We will introduce these two types of privacy guarantees in this section. Throughout the remainder of this section, when we refer to a user’s data point, we are referring to their set of data points.

B.1 Aggregate Differential Privacy

In the aggregate model of differential privacy, we assume the existence of an aggregation protocol that sums the local reports before they are released to the analyst. The analyst still interacts with the clients in a federated manner to perform the algorithm, but the aggregation protocol guarantees that the analyst does not receive anything about the local reports *except their sum*. The aggregate model of DP is a derivative of the more general and more common than aggregate DP shuffle model of differential privacy introduced in [EFM⁺19, CSU⁺19].

Definition 5. A single round algorithm \mathcal{A} is (ϵ, δ) -DP in the aggregate model if the output of the aggregation protocol on two datasets that differ on the data of a single individual are close.

Formally, an algorithm $\mathcal{A} : \mathcal{D}^n \rightarrow \mathcal{Z}$ is (ε, δ) -DP in the aggregate model if the following conditions both hold:

- it can be written as $\mathcal{A}(d^{(1)}, \dots, d^{(n)}) = \phi(\text{Aggregator}(f(d^{(1)}), \dots, f(d^{(n)})))$ where $f : \mathcal{D} \rightarrow \mathcal{Z}$ is a randomized function that transforms that data, **Aggregator** is an aggregation protocol, and $\phi : \mathcal{Y}^n \rightarrow \mathcal{Z}$ is some post-processing of the aggregated report
- for any pair of datasets D and D' that differ on the data of a single individual, and any event E in the output space,

$$\Pr(\mathcal{A}(D) \in E) \leq e^\varepsilon \Pr(\mathcal{A}(D') \in E) + \delta.$$

Note that the post-processing function takes the aggregation as its input and does not have access to the individual reports.

When $\delta > 0$, we call this approximate DP. When each user uses a local randomizer (i.e. the functions f in Definition 5 are local randomizers), the privacy guarantee in the aggregation model can be bounded by a quantity that is a function of both ε_0 , the privacy guarantee in the local model, and n , the number of users that participate in the aggregation protocol [EFM⁺19, CSU⁺19]. As the number of users increases, the privacy guarantee on the output of the aggregation protocol gets stronger; essentially each user gets “lost in the crowd”. In this work, we will bound aggregate DP guarantee by the numerical bound on privacy amplification by shuffling due to [FMT23], who provide bounds for both approximate DP and a related privacy notion called Rényi DP.

A multi-round algorithm \mathcal{A} is (ε, δ) -DP in the aggregate model if it is the composition of single round algorithms which are DP in the aggregate model, and the total privacy loss of \mathcal{A} is (ε, δ) -DP. One can formulate a version of Definition 5 specifically for multi-round algorithms, for a more in-depth discussion see [JRSS21]. There are a number of standard theorems for analysing the privacy guarantee of composing multiple differentially private algorithms [DMNS06, DRV10]. When the number of iterations is small, the advanced composition theorem [DRV10, KOV17] provides a tight analysis. When the number of iterations is large, a tighter analysis is obtained by computing the composition bound in terms of Rényi differential privacy [ACG⁺16, Mir17] then converting this Rényi bound into an (ε, δ) -DP bound [CKS20]. In our experiments, we compute the composed privacy guarantee using both of these methods, then select the tighter bound.

Given an expected number of users, the number of iterations, and the desired aggregate privacy guarantee, we can use binary search to approximate the largest per iteration local epsilon that will achieve the given aggregate privacy guarantee. This algorithm is given in Algorithm 5 where `RenyiShuffleAnalysis` computes the Rényi privacy guarantee for amplification by shuffling, `Composition` uses the composition theorem for Rényi DP, `Conversion` converts the Rényi DP guarantees to approximate DP guarantees and `BinarySearch` makes the decision on whether to increase or decrease ε_l' . Since the aggregate privacy guarantee is our primary privacy guarantee, we do not put an upper bound on our local epsilon. Table 2 demonstrates how the local epsilon increases with the number of iterations, and the desired aggregate privacy guarantee. Table 2 shows different values of ε_l and ε_{agg} depending on the number of iterations for $N = 1.6 \times 10^6$ devices (number of users in the Reddit data set used for our experiments).

C PI-RAPPOR

In this section, we describe the PI-RAPPOR local randomizer that may be used as the local randomizer (A_{priv}) and two algorithms (A_{privagg}) we can use to get the private frequency estimates

Algorithm 5 Privacy Analysis

```

1: Input:  $\epsilon_{agg}, \delta$ : Aggregate privacy budget,  $T$ : number of iterations,  $N$ : number of devices,  $\alpha$ :
   pre-defined set of Renyi parameter,  $E$ : binary search error tolerance
2: Output:  $\epsilon_l$ : local privacy budget of each device in each iteration
3:  $\epsilon_l' \leftarrow$  Initialization
4: while  $|\epsilon_{agg} - \epsilon_{agg}'| \leq E$  do
5:    $\epsilon_r' \leftarrow$  RenyiShuffleAnalysis( $\epsilon_l', \delta, T, N, \alpha$ ) // Theorem 3.2 of [FMT23]
6:    $\epsilon_{composition}' \leftarrow$  Composition( $\epsilon_r', T$ ) //  $\epsilon_{composition}' = T \times \epsilon_r'$ 
7:    $\epsilon_{agg}' \leftarrow$  Conversion( $\epsilon_{composition}', \delta, \alpha$ ) // Proposition 12 of [CKS20]
8:    $\epsilon_l' \leftarrow$  BinarySearch( $\epsilon_{agg}, \epsilon_{agg}', \epsilon_l'$ )
9: end while
10:  $\epsilon_l \leftarrow \epsilon_l'$ 
11: return  $\epsilon_l$ 

```

ϵ_{agg}	0.25						0.5						1					
T	1	2	3	4	5	6	1	2	3	4	5	6	1	2	3	4	5	6
ϵ_l	6.36	6.05	5.79	5.63	5.35	5.31	7.18	6.96	6.73	6.48	6.33	6.26	8.03	7.73	7.58	7.39	7.03	7.018

Table 2: Choices of ϵ_{agg} , T and ϵ_l

each element queried (usually the data domain in each round $\mathcal{P}_{\text{prefixlist}_{t-1}} \times A^t$). We also discuss the computation and communication costs for the device side and both the server side algorithms, provide recommendations for speeding up the algorithms by a factor of $e^{\epsilon_l} + 1$, and provide guidelines for practitioners on how to choose one amongst the two frequency estimation algorithms based on the computation costs.

As defined in [FT21], we define two constants α_0 and α_1 which we set suitably to satisfy deletion ϵ_l - DP ($\alpha_0 = 1 - \alpha_1 = \frac{1}{e^{\epsilon_l} + 1}$) and replacement ϵ_l - DP ($\alpha_0 = \frac{1}{e^{\epsilon_l} + 1}, \alpha_1 = \frac{1}{2}$), respectively. Let q be a prime power so that $\alpha_0 q$ is an integer (the case when $\alpha_0 q$ is not an integer incurs a small additional error as described in Lemma 4.7 of [FT21]). We let F_1 denote the $\alpha_0 q$ smallest elements of the field and let $\text{bool}(z)$ denote the indicator of the event $z \in F_1$. Algorithm 3 of [FT21] gives the PI-RAPPOR local randomizer algorithm which can be used as A_{priv} and Algorithms 4 and 5 of [FT21] are two frequency estimation algorithms, which use the outputs of A_{priv} applied to a datapoint from all users and estimate the frequency of elements in the data domain. The frequency estimate output of either of these algorithms of an element w is a random variable $\tilde{f}[w]$ as defined below:

$$\tilde{f}[w] = \frac{\text{Bin}(f[w], \alpha_1) + \text{Bin}(n - f[w], \alpha_0) - n\alpha_0}{\alpha_1 - \alpha_0}, \quad (1)$$

where $f[w]$ is the true frequency of w and $\text{Bin}((k), \alpha)$ denotes a binomial random variable with parameters k (number of experiments) and α (success probability).

As shown in Lemma 4.2 of [FT21], \tilde{f} is an unbiased estimator of f with the minimum possible variance for locally private estimators. Algorithms 4 and 5 in [FT21] output the same estimate but only differ in terms of their computational complexity. Please See [FT21] for a discussion. While both the algorithms find the frequency of all elements in the domain, they can be easily modified to find the frequency of only subset of the elements in the domain and the computational complexity correspondingly depends on this subset size linearly.

Speedups in the decompressor for prefix based algorithms: When the data domain is a contiguous set of elements, we can precisely calculate the values of w for which it evaluates to $\text{bool}(v(w)) = 1$ and use it to speed up computation by a factor of roughly $\frac{1}{\alpha_0}$. For a given segment length l , we choose q to be the smallest prime power bigger than $K = \max\{e^{\varepsilon_l} + 1, 2^l\}$. We also let $c = \lfloor \frac{q}{e^{\varepsilon_l} + 1} \rfloor$ and denote $\{0, 1, \dots, c\}$ by $[c]$. The data domain is of the form $\mathcal{P}_{\text{prefixlist}} \times \mathbb{F}_q$ (mapped to \mathbb{F}_q^d) where the elements of $\mathcal{P}_{\text{prefixlist}}$ are mapped to elements in the first $d - 1 = \lceil \log_q |\mathcal{P}_{\text{prefixlist}}| \rceil$ dimensions and the last dimension is assigned to all possible completions for a given prefix of length l .

Both algorithms 4 and 5 of [FT21] calculate $\text{bool}(v^i(w))$ where $v^i(w) = v_0^i + \sum_{j \in [d]} v_j^i w_j$ for all w in the data domain, which in our case is $\mathcal{P}_{\text{prefixlist}} \times \mathbb{F}_q$. Instead of calculating the product $v^i(w) = v_0^i + \sum_{j \in [d]} v_j^i w_j$ for all $w \in \mathcal{P}_{\text{prefixlist}} \times \mathbb{F}_q$, we calculate $v_{-1}^i(h) = v_0^i + \sum_{j \in [d-1]} v_j^i h_j$ for all prefixes $h \in \mathcal{P}_{\text{prefixlist}}$. Then for each element (say g) in $[c]$, we calculate $(v_d^i)^{-1}(g - v_{-1}^i(h))$ to get back precisely the element $w_d \in \mathbb{F}_q$ for which $\text{bool}(v^i(w))$ will evaluate to 1. The inverse operation can be computed using a lookup table that can be calculated ahead of time in $\log(q)$ time. Thus, instead of searching over all elements in the data domain $\mathcal{P}_{\text{prefixlist}} \times \mathbb{F}_q$ in constant time, we search over all possible cutoffs and prefixes in $\log(q)$ time each making the effective computational complexity linear in $c|\mathcal{P}_{\text{prefixlist}}| \log(q)$ instead of linear in $|\mathcal{P}_{\text{prefixlist}}|q$, which is a speedup of size roughly $\frac{e^{\varepsilon_l} + 1}{\varepsilon_l}$.

D Proof of Proposition 1

Proof of Proposition 1. In a single round, we query each element in the data domain of size $|A|^r$. Thus using Assumption 1, with $\beta' = \frac{\beta}{|A|^r}$ and using a union bound, we have that with probability $1 - \beta$, $\max_{d \in U} |F[d] - \tilde{f}[d]| \leq C \sqrt{\frac{ne^{\varepsilon_l}}{(e^{\varepsilon_l} - 1)^2} \log(\frac{|A|^r}{\beta})}$.

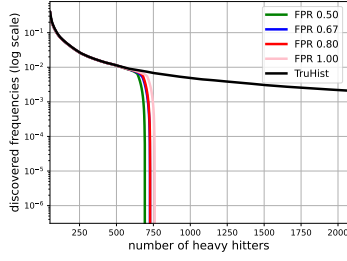
Now, we prove that this algorithm is λ -accurate (Definition 4) with $\lambda = C \sqrt{\frac{ne^{\varepsilon_l}}{(e^{\varepsilon_l} - 1)^2} \log(\frac{|A|^r}{\beta})}$. Let $d \in A^r$ with $\tilde{f}[d] \geq \tau\sigma + \lambda$. Then, $f[d] \geq \tau\sigma$ and hence $d \in \mathcal{P}_{\text{prefixlist}_1}$. Next, let $d \in A^r$ such that $f[d] < \tau\sigma - \lambda$, then $f[d] < \tau\sigma$ and we have $d \notin \mathcal{P}_{\text{prefixlist}_1}$. This shows that $\mathcal{P}_{\text{prefixlist}_1}$ is λ -accurate with $\lambda = C \sqrt{\frac{ne^{\varepsilon_l}}{(e^{\varepsilon_l} - 1)^2} \log(\frac{|A|^r}{\beta})}$. \square

E Adaptive Segmentation Exploration

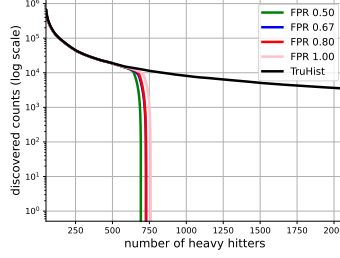
Here, in addition to showing discovered frequencies and discovered counts, we show another metric which we refer to as utility loss.

Let $H = (x_1, x_2, \dots, x_{|H|})$ denote the set of heavy hitters output by an algorithm ordered by the empirical global frequency distribution \mathbb{P} , and let x_i^* denote the i^{th} most frequent element according to the global empirical distribution \mathbb{P} , i.e. the true i^{th} heavy hitter. Then, we evaluate an algorithm with output H by how close the total mass of H is to the total mass of the true top $|H|$ heavy hitters.

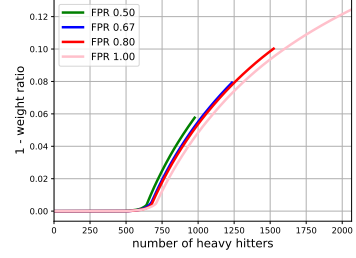
Utility Loss: Define the weight ratio as $WR(H) = \frac{\sum_{x \in H} \mathbb{P}(x)}{\sum_{i \in [|H|]} \mathbb{P}(x_i^*)}$, i.e. the ratio of total probability mass of private heavy hitters H over the probability mass of the actual top $|H|$ heavy hitters. The goal is to maximize the WR to minimize the loss (1-WR).



(a) Marginal discovered frequencies



(b) Marginal discovered counts



(c) Total WR Loss

Figure 3. Effect of the different FPR parameters for single data point setting on `OptPrefixTree` utility ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$ and $T = 4$)

One other potential metric for evaluating these models is to use precision and recall or different versions of their combination for instance F1-Score. However, these metrics do not take into the account the frequency of discovered items. Meaning that if any iterative algorithm discovers the most frequent heavy hitters, but some of them are not in the actual most frequent heavy hitters because of a small frequency difference, precision/recall metrics are not able to capture that. In other words, they capture those as a miss which is not fair to these algorithms.

For marginal figures, to aid with visualisation in our plots, for a window size $W = 50$ and a given heavy hitter set H , we plot for each i , the sum of the probabilities (according to \mathbb{P}) of the heavy hitters in the sliding window $(x_{i-W}, x_{i-W+1}, \dots, x_i)$. We also plot a true histogram line representing $(x_{i-W}^*, x_{i-W+1}^*, \dots, x_i^*)$ (TruHist line) as a reference of what true histogram looks like.

E.1 Adaptive Segmentation for Single Data Point

In this section, we explore the effect of different parameters in the utility of `OptPrefixTree`. For simplicity in this section, we assume each device has a single datapoint.

False Positives Ratio In these experiments we investigate the effect of different FPR parameters on the utility of the final model. False positives ratio can be defined for each application. Depending on how sensitive the application is, FPR determines ratio of number of expected false positives to the total discovered that we keep in each iteration. Please note that this parameter is application dependent. For the applications that are more tolerant to false positives this ratio can be higher. In this part we set the $P = 10^7$ and `OptPrefixTree` finishes in 4 iterations.

As observed increasing the FPR from 0.5 to 1 increases the number of heavy hitters detected slightly [660, 656, 678.0, 695] but increasing FPR also increases the number of discovered false positives significantly [357, 636, 828, 1400]. Based on Figure 3c, although lower FPR detects fewer true bins to ensure it includes fewer false positives, it detects the top most-frequent bins correctly as the loss value shown on y -axis is negligible. We also remark that $FPR = 1$ does not imply that all bins are included since the threshold is based on the expected expected value of false positives for every confidence whereas the true number typically fluctuates around the expectation.

Number of Devices Limitation In this part we analyze the effect of having constraints on the number of devices on the utility of the model. In a production-scale model with billions of devices sending data, dimension can grow extremely large. One way to get around this issue is

T	3										4										5									
sampling rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
ε_l	8.42	8.26	8.19	8.12	8.1	7.92	7.81	7.73	7.62	7.73	8.32	8.19	8.12	7.97	7.87	7.72	7.59	7.52	7.4	7.58	8.27	8.12	8.1	7.86	7.6	7.35	7.19	7.2	7.02	7.39

Table 3: Sampling rate effect on the ε_l for $\varepsilon_{agg} = 1$ and $\delta = 10^{-6}$

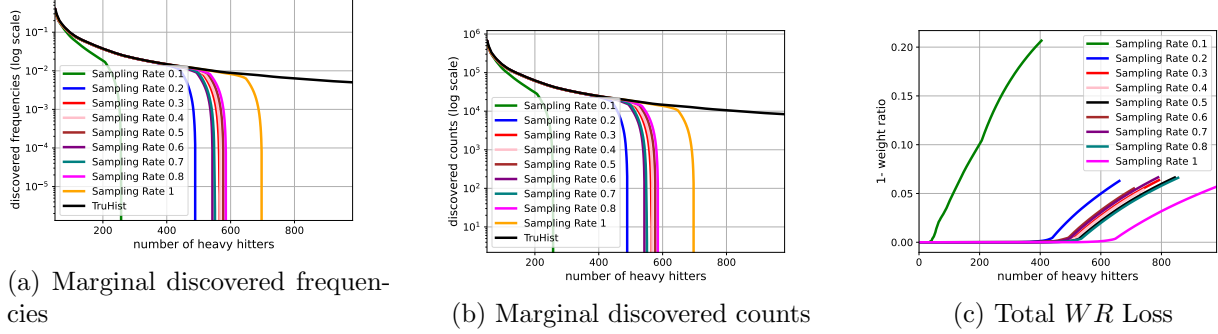


Figure 4. Effect of the different sampling rates for single data point setting on **OptPrefixTree** utility ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$ and $T = 4$)

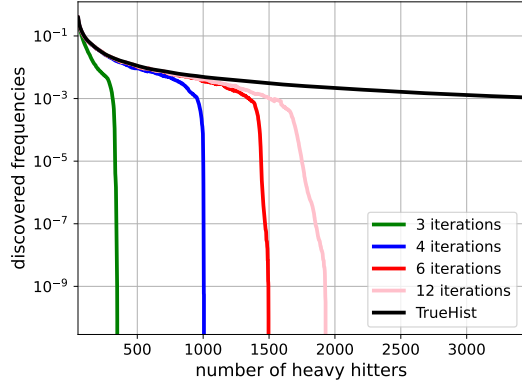
ε_{agg}	1				0.5				0.25			
T	12	6	4	3	12	6	4	3	12	6	4	3
Sampling Rate	0.0079	0.0153	0.0221	0.0283	0.0040	0.0079	0.0117	0.0153	0.0020	0.0040	0.0060	0.0079

Table 4. Number of rounds effect on the sampling rate of **TrieHH** ($\delta = 10^{-6}$, $N = 1.6 \times 10^6$, $\theta = 10$)

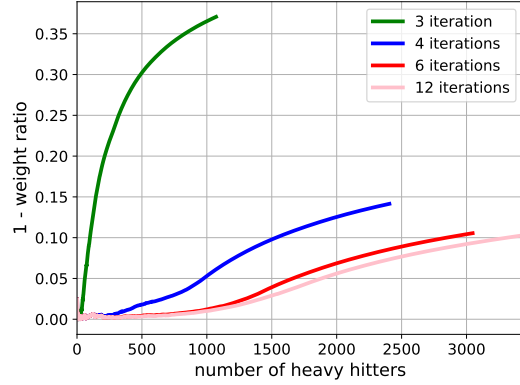
to use sampling. By sampling in each iteration only a sub-set of devices receive the query and contribute to the algorithm. In this section we discuss the effect of having different sampling rates on the utility of the model. Each device can participate in an iteration with a $\text{Ber}(\gamma)$ where γ is the sub-sampling rate. We use theorem 5 described in [ZW19] for estimating the upper-bound of ε_c . The effect of different sub-sampling rates on the value of ε_l when $\varepsilon_c = 1$ and $N = 1.6 * (10)^6$ is shown in Table 3. As shown small sampling rate, increase the ε_l by adding to the randomness. However, as shown in the table, no sampling leads to higher ε_l in comparison to the moderate sampling rates ε_l . The reason is when the number of devices increases, the privacy guarantee on the output of the aggregation protocol gets stronger (“lost in the crowd”). Hence, for moderate sampling rates having less number of users cancels the benefit of using sampling. Figure 4c, 4b, 4a demonstrates the effect of different sampling rates on the utility of **OptPrefixTree**. In addition to ε_l difference of using different methods, having smaller number of devices can affect the utility by eliminating some part of the distribution. Consequently, we avoid using sampling if the dimension constraint allows.

E.2 Multiple Data point Adaptive Segmentation

Segmentation Size In this analysis, we used binary encoding of the data. We set the dimension limitation to $P = 10^7$. We ran the experiment for the utilized setting in which we have unweighted sampling and prefix list. We used both weighted and unweighted metric. As demonstrated increasing the number of iterations from 3 to 12 improves the utility of the algorithm.

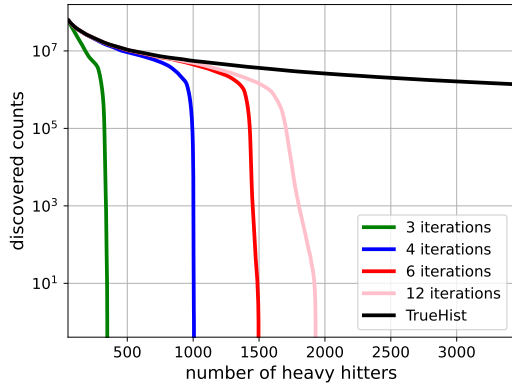


(a) Marginal discovered frequencies (weighted metric)

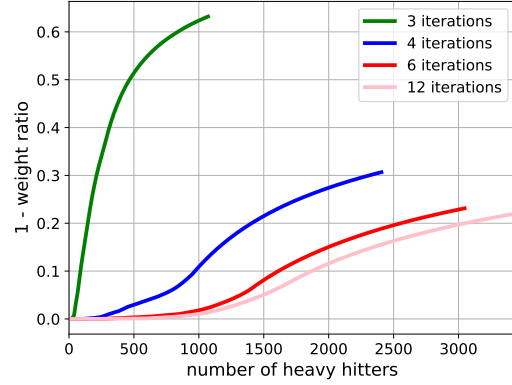


(b) Total WR loss (weighted metric)

Figure 5. Effect of the segment size for multiple data points setting on `OptPrefixTree` utility ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$)



(a) Marginal discovered counts (unweighted metric)



(b) Total WR Loss (unweighted metric)

Figure 6. Effect of the segment size for multiple data points setting on `OptPrefixTree` utility ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$)

F TrieHH

F.1 Single Data Point Setting for TrieHH

In this section we discuss the effect of number of iterations on the utility of a single data point setting for `TrieHH` [ZKM⁺20]. In Figure 7a, we show the effect of different segmentation on the utility of the algorithm for a single data point per device setting. For these experiments we used $\varepsilon_{agg} = 1$ and sampling rates are set based on table 4. To evaluate the effect of different segmentation in this part we used the $P = 10^7$ on the dimension. The number of heavy hitters detected by `TrieHH` algorithm, when the number of iterations are 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char) are [142, 261, 355, 135]. Initially having larger segments help with the algorithm since less number of iterations are required and consequently sampling rate becomes larger. However, by increasing the segment size to certain point, the utility drops. The reason is by enlarging the segment length the

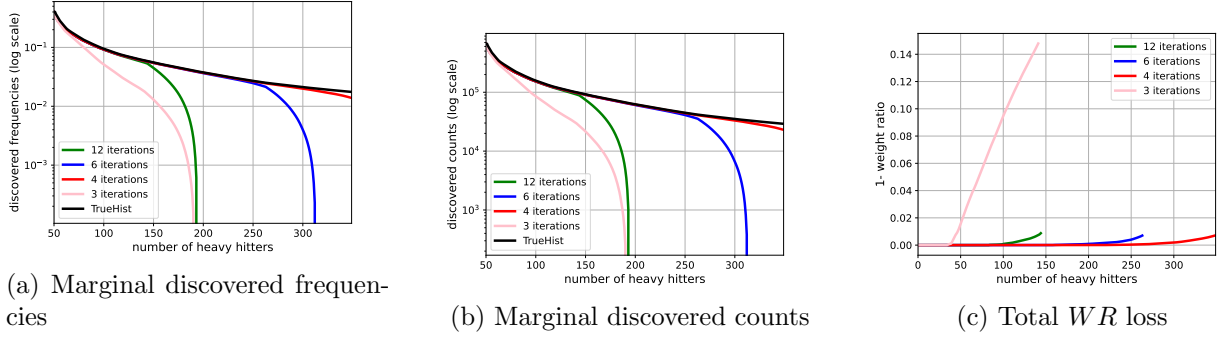


Figure 7. The effect of different number of iterations on **TrieHH** for single data point setting ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$)

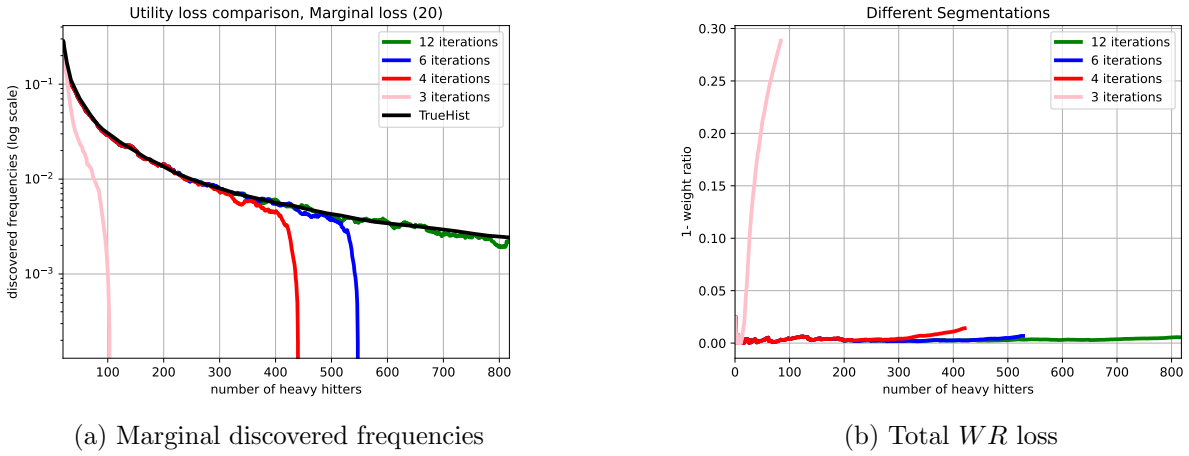


Figure 8. The effect of different number of iterations on **TrieHH** for multiple data points setting with weighted sampling ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$)

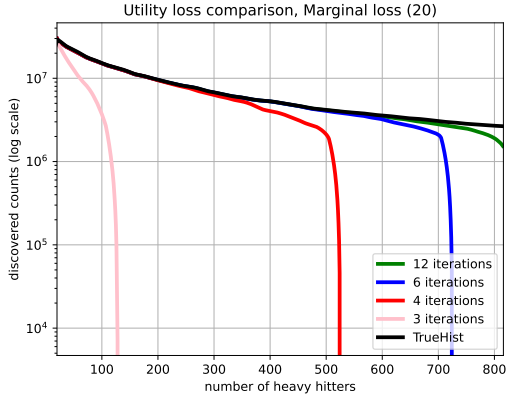
number of prefixes in each iteration reduces because of the dimension constraint. Hence, for the comparisons with **OptPrefixTree** we used the best configurations which is having 4 iterations.

F.2 Multiple Data Points Setting for TrieHH

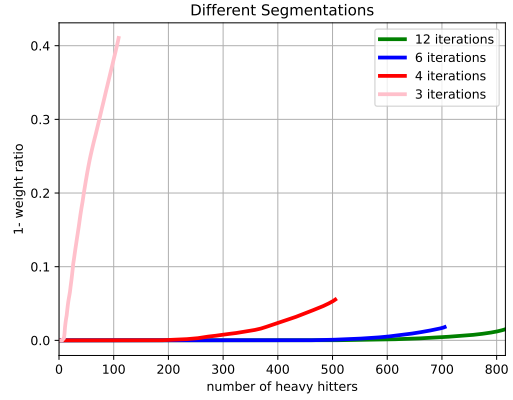
We further analyze the multiple data points setting. To optimize the algorithm we took advantage of a prefix list for each iteration. Devices send their data only if they find a match with a prefix in the prefix list. We also use an end character symbol to indicate the end of string. If end character symbol is observed in a prefix at the end of an iteration, the corresponding prefix will be excluded from the prefix list. Therefore, users can send other unfinished prefixes. Also, for our evaluation, we used binary encoding which uses 5 bits to represent each character.

The total number of heavy hitters detected by **TrieHH** algorithm, when the number of iterations are 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char), are [816, 706, 506, 110] respectively. In this setting, 12 iterations shows the best utility. In Figure 8a and 8b we used weighted sampling described in the original paper.

To further improve the utility of **TrieHH**, we used unweighted sampling in another set of experiments. This new sampling scheme leads to finding [816, 529, 422, 85] heavy hitters when having



(a) Marginal discovered counts



(b) Total WR loss

Figure 9. The effect of different number of iterations on **TrieHH** for multiple data points setting with unweighted sampling ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$)

ϵ_{agg}	1				0.5				0.25			
T	12	6	4	3	12	6	4	3	12	6	4	3
Sampling Rate	0.0071	0.0129	0.0193	0.0255	0.0032	0.0067	0.0102	0.0138	0.0016	0.0034	0.0053	0.0071

Table 5. Number of rounds effect on the sampling rate of **TrieHH** ++ ($\delta = 10^{-6}$, $N = 1.6 \times 10^6$, $\theta = 10$)

ϵ_{agg}	1				0.5				0.25			
T	12	6	4	3	12	6	4	3	12	6	4	3
Sampling Rate	0.0153	0.0305	0.0449	0.0589	0.0078	0.0159	0.0239	0.0319	0.0039	0.0082	0.0123	0.0166

Table 6. Number of rounds effect on the sampling rate of **TrieHH** ++ ($\delta = 10^{-6}$, $N = 1.6 \times 10^6$, $\theta = 20$)

12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char) iterations respectively. Figure 9a and 9b shows the loss and marginal discovered counts based on the unweighted sampling scheme. As demonstrated in these figures, unweighted scheme is able to find more heavy hitters and cause less utility degradation. Also using both sampling schemes, 12 iterations shows the highest utility. Thus, for the comparisons with **OptPrefixTree** we use unweighted sampling, prefix-list and 12 iterations for **OptTrieHH**.

G TrieHH++

Based on Lemma 3 of [CB22], **TrieHH** ++ achieves (ϵ, δ) differential privacy when sampling rate $p_s = \alpha(1 - e^{-\epsilon})$ where $0 < \alpha \leq 1$ and $\epsilon < 1$ for $\delta = e^{-C_\alpha \theta}$, where $C_\alpha = \log 1/\alpha - 1/(1 + \alpha)$. The ϵ here is for one iteration. We used advanced composition in theorem 3.4 of [KOV17] to find the optimal ϵ per iteration which gives us ϵ_{agg} of 1. We set the α parameter so that $\delta = 10^{-6}$. **TrieHH** ++ provide the analysis that shows the trade off between sampling and threshold values. We change the threshold θ from 10 to 20 and its effect on sampling rate are reported in Table 5 and 6.

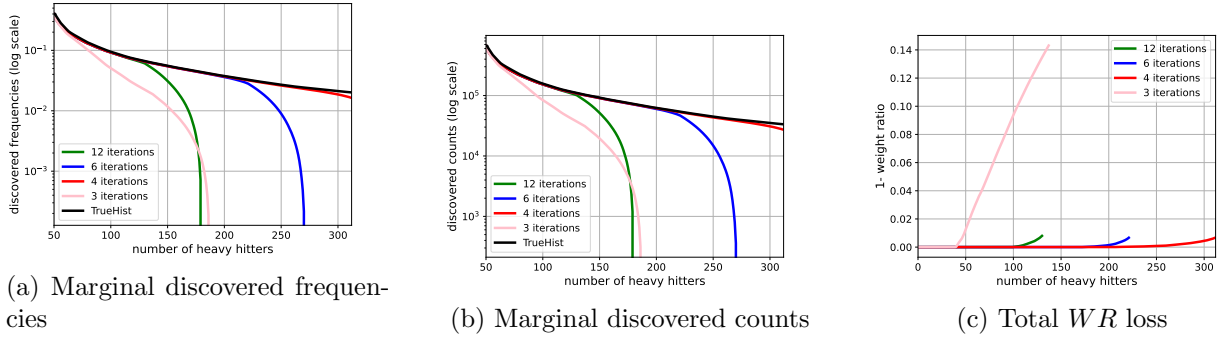


Figure 10. The effect of different number of iterations on **TrieHH++** for single data point setting ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$)

G.1 Single Data Point Setting for TrieHH++

In this section we discuss the effect of number of iterations on the utility of a single data point setting for **TrieHH++** [CB22]. In Figure 10, we show the effect of different segmentation on the utility of the algorithm for a single data point per device setting. For these experiments we used $\varepsilon_{agg} = 1$ and sampling rates are set based on table 5. To evaluate the effect of different segmentation in this part we used the $P = 10^7$ on the dimension. The number of heavy hitters detected by **TrieHH++** algorithm, when the number of iterations are 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char) are [124, 223, 314, 137]. Similar to **TrieHH** having larger segments help with the algorithm since less number of iterations are required and consequently sampling rate becomes larger. However, by increasing the segment size to a certain point, the utility drops. The reason is by enlarging the segment length the number of prefixes that can be kept in each iteration reduces because of the dimension constraint. Thus, for the comparisons with **OptPrefixTree** we used the best configurations which is having 4 iterations.

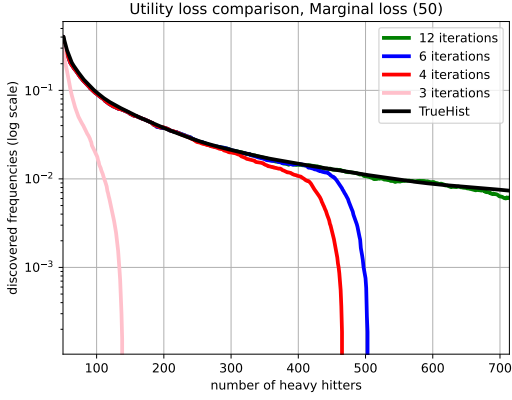
G.2 Multiple Data Points Setting for TrieHH++

For our evaluation, we used the same binary encoding we described before. The total number of heavy hitters detected by **TrieHH++** algorithm, when the number of iterations are 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char), are [714, 455, 417, 90] respectively. As shown in Figure 11, in this setting, 12 iterations shows the best utility.

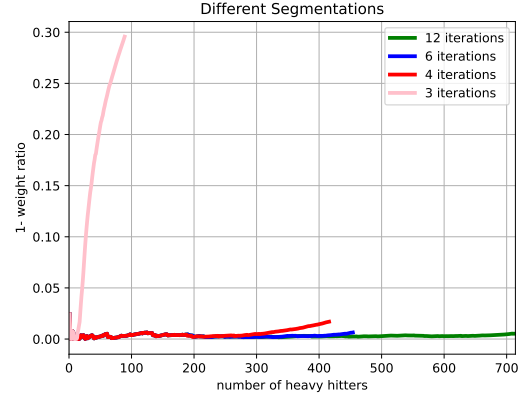
To further improve the utility of **TrieHH++**, we used unweighted sampling in another set of experiments. This sampling scheme leads to finding [704, 610, 484, 102] heavy hitters when having 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char) iterations respectively. Figure 12a and 12b shows the loss and marginal discovered counts based on the unweighted sampling scheme. As demonstrated in these figures, unweighted scheme is able to find more heavy hitters and cause less utility degradation. Also using both sampling schemes, 12 iterations shows the highest utility. Thus, for the comparisons with **OptPrefixTree** we use unweighted sampling, and 12 iterations for **TrieHH++** and we refer to it as **OptTrieHH++**.

H Comparison of OptTrieHH, OptTrieHH++, and OptPrefixTree

Zhu et al.’s primary experiments are on learning n -grams (words or length n sentences) where they propose setting the segmentation length to be a single character. However, our analysis indicates

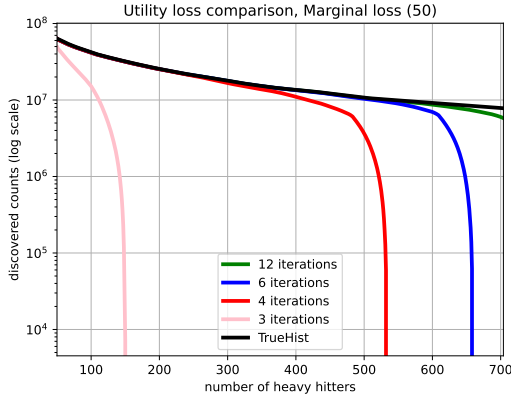


(a) Marginal discovered frequencies

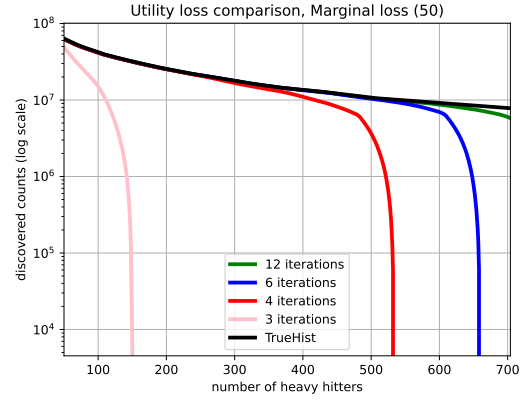


(b) Total WR loss

Figure 11. The effect of different number of iterations on $\text{TrieHH}++$ for multiple data points setting with weighted sampling ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$)

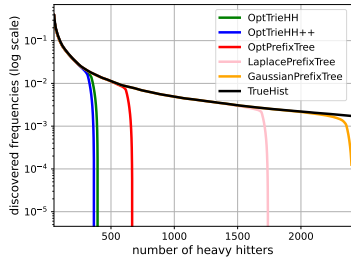


(a) Marginal discovered counts

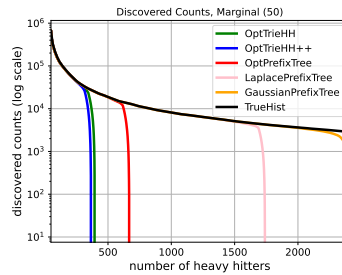


(b) Total WR loss

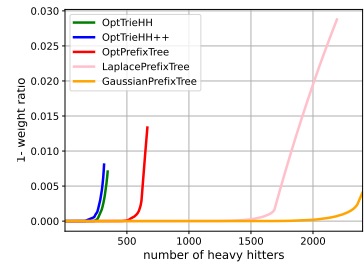
Figure 12. The effect of different number of iterations on $\text{TrieHH}++$ for multiple data points setting with unweighted sampling ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$)



(a) Marginal discovered frequencies

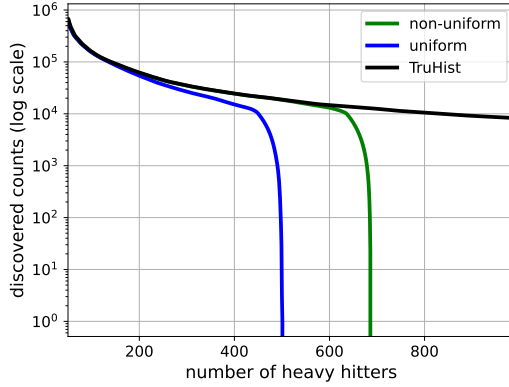


(b) Marginal discovered counts

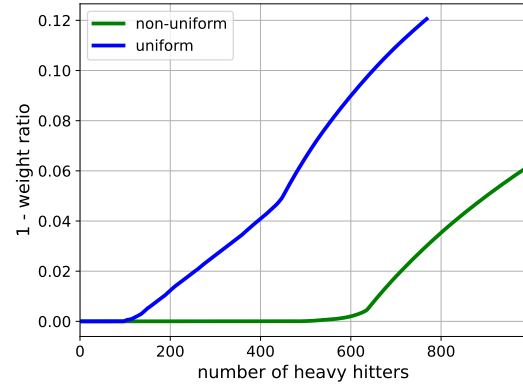


(c) Total WR loss

Figure 13. Utility comparison between OptTrieHH and OptPrefixTree for single data point setting ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)



(a) Marginal discovered counts comparison



(b) Total WR loss

Figure 14. Effect of different segmentation on `OptPrefixTree` for single data point setting ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)

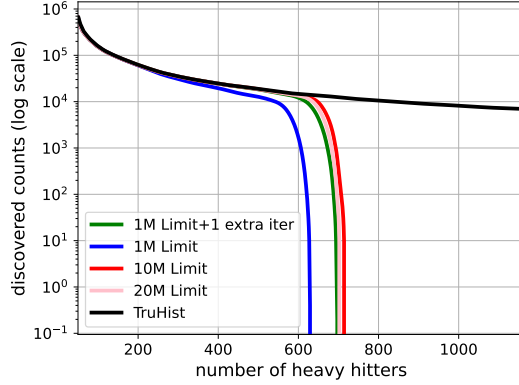
that, in the single data point setting, larger segments provide the higher utility. Earlier in section F we showed the effect of different number of iterations on the sampling rate and utility of `TrieHH`. In this set of experiments we used 4 iterations that shows the highest utility for `TrieHH` based on the dimension limitations ($P = 10^7$). We refer to this optimized version of `TrieHH` as `OptTrieHH`. The same observation holds for `TrieHH ++` and hence we use the same configuration for this algorithm. We refer to the optimized version of `TrieHH ++` as `OptTrieHH ++`.

To have a fair comparison, we use the same binary encoding for all the three models. This binary encoding uses 5 bits to convert English letters to a binary representation. In these experiments, $r = 60$. We set $FPR = 2$ for `OptPrefixTree`. Using this method, `OptPrefixTree` needs 4 iteration of unknown dictionary and uses the segmentation of [23, 13, 12, 12]. Figure 13a shows the marginal frequencies of discovered bins on y-axis and number of heavy hitters in x-axis. This figure shows the marginal value with sliding window of 50 on y-axis. In conclusion, with the same dimension limit and binary encoding, our method is able to outperform `OptTrieHH` and `OptTrieHH ++` by finding 1.85X and 2.1X more heavy hitters. Figure 13b shows the same plot but in the y-axis we have the marginal counts of discovered bins. Also, Figure 13c shows the total utility loss.

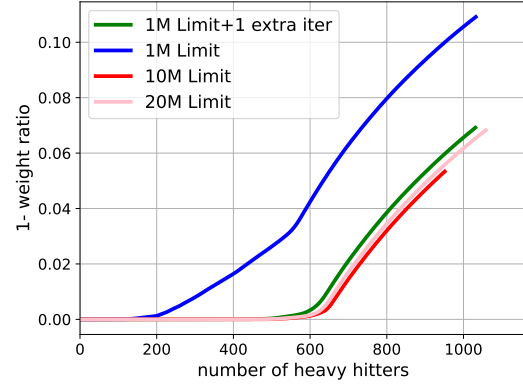
I Additional Experimental Results

Effect of Uniform vs Non-uniform Segmentation in Single Data Point Setting As explained in 6 using our adaptive segmentation algorithm helps discovering more heavy hitters. In Figures 14a and 14b we show the discovered counts and total utility loss comparison of this uniform and non-uniform segmentation.

Effect of Dimension Limitation in Single Data Point Setting In Figures 15a and 15b we evaluated the effect of different dimension limitation parameters in the utility of the model. As shown, reducing the dimension limitation below a certain point, causes a significant utility drop. However, allowing the algorithm to have one extra iteration can help recovering top heavy hitters.

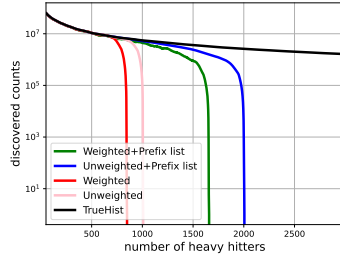


(a) Marginal discovered counts

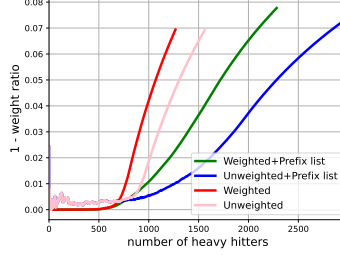


(b) Total WR Loss

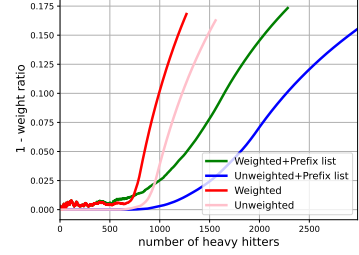
Figure 15. Effect of dimension limitation for single data point setting on `OptPrefixTree` utility ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$ and $T = 4$)



(a) Marginal discovered counts



(b) Total WR loss (weighted metric)



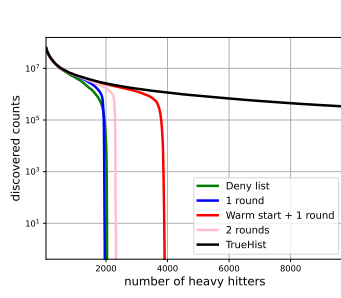
(c) Total WR loss (unweighted metric)

Figure 16. Comparing different data selection schemes in `OptPrefixTree` for multiple data points ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)

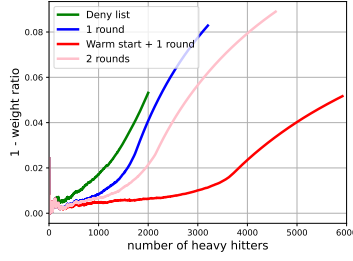
Effect of Data Selection in Multiple Data Points Setting There are different ways to measure the frequency of data points. In Section 4, we discuss how averaging the distribution of words overall devices can be used to define the global frequency of words (weighted metric). Figure 16b shows the total utility loss when using this distribution.

One other way to measure the frequency is the percentage of devices who has the word in their support (unweighted metric). Figure 16a shows the global number of discovered bins based on how many devices have the word. Figure 16c demonstrates the total utility loss when using the average of users who has the data point as the frequency of words. As depicted using both distributions, unweighted data selection outperforms weighted data selection regardless of the frequency computation technique. Also prefix list benefits both of the data selection schemes.

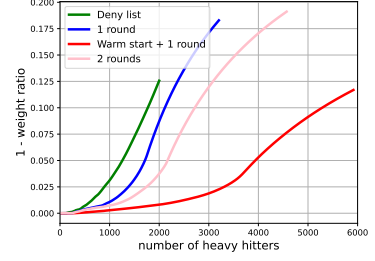
Effect of adding a deny list in Multiple Data Points Setting Figure 17b shows the total utility loss when using the frequency of the words in devices for extracting the global distribution. In Figure 17a and 17c we use the number of devices with the word to demonstrate the true counts of the discovered words and total utility loss.



(a) Marginal discovered counts

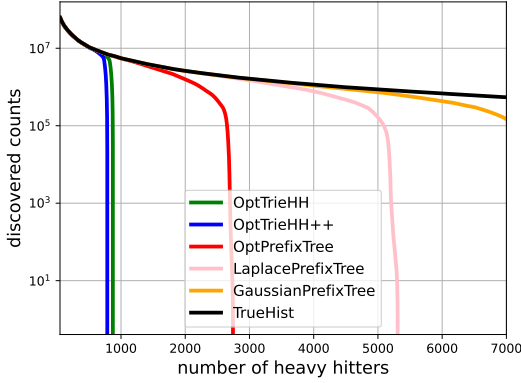


(b) Total WR loss (weighted metric)

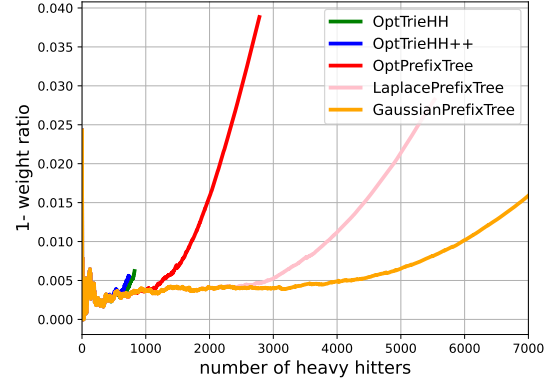


(c) Total WR loss (unweighted metric)

Figure 17. Effect of adding deny list to **OptPrefixTree** for multiple data points setting ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)



(a) Marginal discovered counts



(b) Total WR loss

Figure 18. Comparing **OptPrefixTree**, **OptTrieHH**, **OptTrieHH++** for multiple data points setting ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)

Comparison with previous works As illustrated in section 6 **OptPrefixTree** outperforms **OptTrieHH** under the same constraints. In Figure 18a and 18b, we show a comparison of **OptPrefixTree** and **OptTrieHH** for the multiple data points setting.