Differentially Private Heavy Hitter Detection using Federated Analytics

Karan Chadha † *
Stanford University
knchadha@stanford.edu

Junye Chen
Apple
junyec@apple.com

John Duchi
Stanford University, Apple
john.duchi@apple.com

Vitaly Feldman
Apple
vitalyf@apple.com

Hanieh Hashemi*
Apple
h_hashemi@apple.com

Omid Javidbakht
Apple
omid_j@apple.com

Audra McMillan*
Apple
audra_mcmillan@apple.com

Kunal Talwar
Apple
ktalwar@apple.com

***Abstract*—In this work, we study practical heuristics to improve the performance of prefix-tree based algorithms for differentially private heavy hitter detection. Our model assumes each user has multiple data points and the goal is to learn as many of the most frequent data points as possible across all users' data with aggregate and local differential privacy. We propose an adaptive hyperparameter tuning algorithm that improves the performance of the algorithm while satisfying computational, communication and privacy constraints. We explore the impact of different data-selection schemes as well as the impact of introducing deny lists during multiple runs of the algorithm. We test these improvements using extensive experimentation on the Reddit dataset [Caldas et al., 2018] on the task of learning the most frequent words.**

***Index Terms*—Federated Analytics, Differential Privacy, Frequency Estimation, Heavy Hitter Identification**

I. Introduction

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 [Erlingsson et al., 2014], and PI-RAPPOR [Feldman and Talwar, 2021], 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 [Cormode et al., 2012], [Qardaji et al., 2012], [Song et al., 2013], [Bagdasaryan et al., 2021], [Kim et al., 2021], [McMillan et al., 2022]. 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

† Research was conducted when author was an intern at Apple.
* Authorship is in alphabetical order. Correspondence to: Karan Chadha, Hanieh Hahsmei, Audra McMillan

[1] A potential architecture for running iterative algorithms in this model of privacy is outlined in [McMillan et al., 2022].

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 [Erlingsson et al., 2014]) 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:

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 II we discuss some of the prior works in privacy-preserving heavy hitters detection. Section III explains the privacy primitives we used in this work. In Section IV we elaborate the details of our prefix tree algorithm. Section V explains the post-processing methods and the theoretical analysis behind it. Section VI demonstrates the experimental results and in Section VII we discuss the findings of our experiments.

II. Related Works

Heavy hitters discovery methods have applications in various different domains [Elkordy et al., 2023]. This problem has been studied in both the local model [Apple, 2017], [Wang et al., 2019], [Acharya et al., 2019] and shuffle model [Ghazi et al., 2021] of differential privacy. Furthermore, recently different multi-party-computing [Boneh et al., 2021], [Bell et al., 2022] methods and combination of multi-party-computing and DP techniques [Böhler and Kerschbaum, 2021] 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 [Zhu et al., 2020], 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++` [Cormode and Bharadwaj, 2022] is an extension to `TrieHH`. The authors use the same sampling and threshold algorithm as `TrieHH` to provide the $(\epsilon, \delta)-$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 [Gopi et al., 2020] 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 [Wilson et al., 2020], 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 [Carvalho et al., 2022], 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 [Liu et al., 2021], [Bassily et al., 2020]. 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 [Kim et al., 2021] 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.

III. 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 III.1 (Local Randomizer [Dwork and Roth, 2014], [Kasiviswanathan et al., 2011])**.** Let $\mathcal{A} : D \to \mathcal{Y}$ 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

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

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 III.2 (Local Differential Privacy [Kasiviswanathan et al., 2011])**.** Let $\mathcal{A} : D^n \to \mathcal{Z}$ 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 $\mathcal{A}(d^{(1)}, \cdots, d^{(n)}) = \phi\left(\mathcal{A}_1(d^{(1)}), \cdots, \mathcal{A}_n(d^{(n)})\right)$ where the $\mathcal{A}_i : D \to \mathcal{Y}$ are ϵ-local randomizers for each $i \in [n]$ and $\phi : \mathcal{Y}^n \to \mathcal{Z}$ is some post-processing function of the privatized records $\mathcal{A}_1(d^{(1)}), \cdots, \mathcal{A}_n(d^{(n)})$. 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 [Joseph et al., 2019]. 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 III.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 $\mathcal{A} : \mathcal{D}^n \to \mathcal{Z}$ is (ϵ, δ)-DP in the aggregate model if the following conditions both hold:

- it can be written as $\mathcal{A}(d^{(1)}, \cdots, d^{(n)}) = \phi(\texttt{Aggregator}(g(d^{(1)}), \cdots, g(d^{(n)})))$ where $g : \mathcal{D} \to \mathcal{Z}$ is a randomized function that transforms that data, `Aggregator` is an aggregation protocol, and $\phi : \mathcal{Y}^n \to \mathcal{Z}$ 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.

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 [Erlingsson et al., 2019], [Cheu et al., 2019]. In our experimental results, we will bound the aggregate DP epsilon by the numerical bound on privacy amplification by shuffling due to [Feldman et al., 2023], who provide bounds for both approximate DP and a related privacy notion called Rényi DP.

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,

[2]The aggregate model of DP is a derivative of the more general and common shuffle model of differential privacy introduced in [Erlingsson et al., 2019], [Cheu et al., 2019].

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 [Dwork et al., 2010], [Kairouz et al., 2017] and composition bound in terms of Rényi differential privacy [Abadi et al., 2016], [Mironov, 2017], [Canonne et al., 2020].When the number of iterations is large, a tighter analysis is obtained by computing the composition bound in terms of Rényi differential privacy [Abadi et al., 2016], [Mironov, 2017] then converting this Rényi bound into an (ϵ, δ)-DP bound [Canonne et al., 2020]. 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 (N), the number of iterations (T), and the desired aggregate privacy guarantee (ϵ_{agg}, δ), we use binary search to find the largest per iteration local epsilon that will achieve the given aggregate privacy guarantee upto certain error tolerance. We present this algorithm in Algorithm 1. In the first step ϵ'_r is initialized and in the while loop we adjust this local privacy parameter to be as large as possible while satisfying the aggregated privacy requirement. RenyiShuffleAnalysis computes the Rényi privacy guarantee for amplification by shuffling using Theorem 3.2 of [Feldman et al., 2023] for a set of predefined Rényi parameters (α). In the next step, Composition uses the composition theorem for Rényi DP, and then Conversion converts the Rényi DP guarantees to approximate DP guarantees using Proposition 12 of [Canonne et al., 2020]. Finally, BinarySearchUpdate updates $\epsilon_l{}'$ depending on the calculated aggregate DP values (ϵ'_{agg}) based on current estimate of local DP parameter and the desired aggregate DP values (ϵ_{agg}). These iterations are repeated until a close enough match is found. Since the computations are in floating point, sometimes reaching the exact match is impossible but it is fine to find a close enough match. Binary search error parameter (E) represents the tolerance level toward near-exact searching. In these exepriments $E = 10^{-5}$. We highlight that both ϵ'_r and $\epsilon'_{composition}$ are arrays of Rényi DP parameters of prespecified orders (given by α). Since the aggregate privacy guarantee is our primary privacy guarantee, we do not put an upper bound on our local epsilon.

Algorithm 1 Privacy Analysis

Input: ϵ_{agg}, δ: Aggregate privacy budget, T : number of iterations, N: number of devices , α: pre-defined set of Renyi parameter, E: binary search error tolerance
Output: ϵ_l: local privacy budget of each device in each iteration
$\epsilon_l{}' \leftarrow$ Initialization
while $|\epsilon_{agg} - \epsilon'_{agg}| \leq E$ **do**
  $\epsilon'_r \leftarrow \text{RenyiShuffleAnalysis}(\epsilon'_l, \delta, T, N, \alpha)$
  $\epsilon'_{composition} \leftarrow \text{Composition}(\epsilon'_r, T)$ // $\epsilon'_{composition} = T \times \epsilon'_r$
  $\epsilon'_{agg} \leftarrow \text{Conversion}(\epsilon'_{composition}, \delta, \alpha)$
  $\epsilon'_l \leftarrow \text{BinarySearchUpdate}(\epsilon_{agg}, \epsilon'_{agg}, \epsilon_l{}')$
end while
$\epsilon_l \leftarrow \epsilon'_l$
return ϵ_l

IV. 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 V, 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 [Feldman and Talwar, 2021] or Proj-Rappor [Feldman et al., 2022].

Notation. Let N be the total number of users. Let each user $i \in [N]$ have n_i data points denoted by $d_{i,1}, \ldots, d_{i,n_i}$ 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 distribution [3]. Let each data point $d \in A^r$ be of a fixed length r.

A. 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 popular prefixes, and only expanding on these popular prefixes in the next iteration. We give pseudo-code for our proposed algorithm `OptPrefixTree` in Algorithm 2. 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 ($\mathcal{P}_{\text{prefixlist}_{T+1}}$)

[3] In 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 H.

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 provide an example of running two iterations of `OptPrefixTree` in Figure 1. In this example, we set the dimension limit (P) to 4. Consequently, the maximum segment length for iteration 1 (l_1) not to exceed this limit is 2. After aggregation, the prefixes that have counts above threshold are $[00], [10]$. That is why these two prefixes are kept in the prefix list ($\mathcal{P}_{\text{prefixlist}_2}$) and sent to devices for the second iteration. Also, using $l_2 = \max\{\ell \mid |\mathcal{P}_{\text{prefixlist}_2}| \times 2^\ell \leq P\}$, segment length for the second iteration cannot have size larger than 1. In the second iteration, devices that find a match between their local data and the prefix list shared with them, privatize the match (provided it is not in the deny list) and send to the server. They send a privatized special character otherwise. After aggregation, again the prefixes that have counts below threshold are removed and the prefix list of popular items ($[001], [100]$) is formed.

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, we see this when using Huffman encoding.

Algorithm 2 Prefix tree based heavy hitter algorithm (`OptPrefixTree`)

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

B. 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 3 gives more details for the device-side algorithm.

a) 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 [McMillan et al., 2022]. In practice one could use PI-RAPPOR [Feldman and Talwar, 2021] or Proj-RAPPOR [Feldman et al., 2022] for better communication-computation trade-offs (See Section B). 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.

b) 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.

C. 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 4.

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 $\widetilde{f}(d)$ of the number of devices who sent the data point d in the last iteration. The prefix list selection algorithm aims to keep as many of the elements $d \in \mathcal{P}_{\text{prefixlist}_t} \times A^{l_t}$ such that $\widetilde{f}(d) > 0$ as possible, while minimizing the number of "false positives" in the prefix

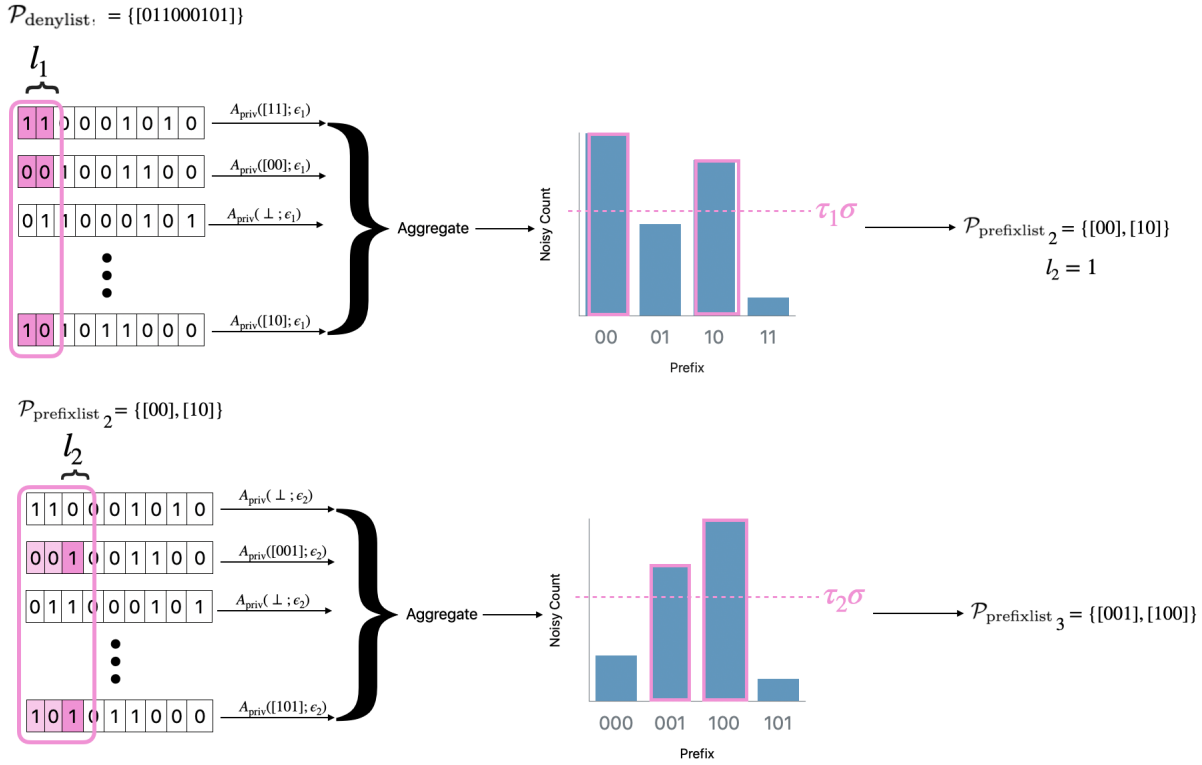


Fig. 1: An example of running two iterations of `OptPrefixTree`

Algorithm 3 Device side algorithm (`DeviceSide`)

Input: ϵ_l: Local privacy parameter, l_{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
Param: D: device dataset
Output: v: Privatized output
$D = \{d \in D \mid d[0 : l_{\text{pref}}] \in \mathcal{P}_{\text{prefixlist}} \wedge d \notin \mathcal{P}_{\text{denylist}}\}$
if $D == \phi$ **then**
  $d \leftarrow \perp$ *// We reserve a special data element for users that have no eligible data points to report.*
else
  $d \leftarrow$ *selectData*(D)
end if
$v \leftarrow A_{\text{priv}}(d[0 : l_{\text{pref}} + l_t]; \epsilon_l)$
return v

list (data elements which do not match the selected data point for *any* device). When using `OHE+2RR`, each estimate $\widetilde{f}(d)$ 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 $\widetilde{f}(d) > 0$, 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 $\widetilde{f}(d) \geq \tau\sigma$. 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 V 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.

V. Adaptive Thresholding and Segmentation

In this section, we will describe our adaptive segmentation and thresholding algorithms. The algorithms aim to keep as

Algorithm 4 Server side algorithm per round (`ServerSide`)

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`
Output: $\mathcal{P}_{\text{prefixlist}}$: Heavy hitters list
$\widetilde{f}(\cdot) \leftarrow A_{\text{privagg}}(V_t, \mathcal{D}_t; \epsilon_l)$ *// Takes the aggregated privatized responses and computes an estimate of the frequency of every data element.*
$\sigma \leftarrow \sqrt{\text{Var}(A_{\text{privagg}})}$ *// Computes an upper bound on the standard deviation of the frequency estimate for d*
$\mathcal{P}_{\text{prefixlist}_{t+1}} \leftarrow \texttt{PruneHH}(\mathcal{D}, \widetilde{f}(\mathcal{D}), \tau_0, FPR, \sigma, \eta)$
$\mathcal{P}_{\text{prefixlist}_{t+1}}, \mathcal{P}_{\text{discoveredlist}} \leftarrow \texttt{RemoveFinished}(\mathcal{P}_{\text{prefixlist}_{t+1}}, \mathcal{P}_{\text{discoveredlist}})$ *// Removes any of the discovered prefixes which are "complete" data points and adds them to the discovered list.*
$l_{t+1} = \max\{\ell \mid |\mathcal{P}_{\text{prefixlist}_{t+1}}| \times 2^\ell \leq P\}$ *// Adaptively chooses the maximum segment length*
return $\mathcal{P}_{\text{prefixlist}_{t+1}}, l_{t+1}, \mathcal{P}_{\text{discoveredlist}}$

many heavy hitters as possible, while maintaining a specific false positive rate, and satisfying the data domain size constraint for the next iteration.

A. 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.

[4]if 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.

Algorithm 5 Pruning algorithm for confident heavy hitter detection (`PruneHH`)

Input: $\mathcal{D}, \widetilde{f}[\mathcal{D}]$: query set and estimated frequencies after aggregation, τ_0: Initialization of threshold, FPR: ratio of expected false positives to the total number of bins, σ: aggregated noise standard deviation, η: step size
$E \leftarrow 1 - \Phi(\tau_0\sigma)$
$\mathcal{P}_{\text{prefixlist}}{}' = \{q \in \mathcal{D} \mid \widetilde{f}[q] > \tau_0 \times \sigma\}$
while $FPR < \frac{E \times |\mathcal{D}|}{|\mathcal{P}_{\text{prefixlist}}{}'|}$ **do**
  $E \leftarrow \eta * E$
  $\tau \leftarrow z_{(1-E)}$
  $\mathcal{P}_{\text{prefixlist}}{}' = \{q \in \mathcal{D} \mid \widetilde{f}[q] > \tau \times \sigma\}$
end while
  return $\mathcal{P}_{\text{prefixlist}}$

B. 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.

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

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

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

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 V.2, which is standard in the differentially private heavy hitters literature [Bassily et al., 2017].

Definition V.2 (λ-accurate)**.** A set of heavy hitters $\mathcal{P}_{\text{prefixlist}_T}$ is said to be (λ, Γ)-accurate if it satisfies the following:

- For all $d \in U$, if $F(d) \geq \Gamma + \lambda$, then $d \in \mathcal{P}_{\text{prefixlist}_T}$.
- For all $d \in U$, if $F(d) < \Gamma - \lambda$, 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 V.3. We first state the result and discuss its implications, deferring the proof to Appendix C.

Proposition V.3. *Let the local randomizer (A_{priv}) and frequency estimator (A_{privagg}) pair satisfy Assumption V.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 (ϵ_{agg}, δ). Then, with probability at least $(1 - \beta)$, $\mathcal{P}_{\text{prefixlist}_1}$ is $(\lambda, \tau\sigma)$-accurate with $\lambda = O\left(\frac{1}{\epsilon_{agg}}\sqrt{\log(1/\delta)\log\left(\frac{|A|^r}{\beta}\right)}\right)$, 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 $\approx \epsilon_{agg}/\sqrt{T}$. 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 D-B 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.

VI. 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 [Caldas et al., 2018]. On the Reddit dataset each device has an average of $1,092$ 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 $H = (x_1, x_2, \ldots, x_{|H|})$ 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 D). We also plot a true histogram line representing $(x^*_{i-W}, x^*_{i-W+1}, \ldots, x^*_i)$ (TruHist line) as a reference of what the true ground truth histogram looks like.

Privacy Analysis: We work with a pre-specified aggregate differential privacy (Definition III.3) constraint and use Algorithm 1 to calculate the largest local privacy parameter ϵ_l such that aggregate differential privacy with desired ϵ_{agg} is satisfied. Table I demonstrates how the local epsilon increases with the number of iterations, and the desired aggregate privacy guarantee. It 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 dataset used for our experiments). We highlight that the local DP parameters change much more slowly than the aggregate DP parameters, for instance having the aggregate DP parameter from 1 to 0.5 reduces the local DP parameter by only ~ 1 for the same number of iterations. This proves useful for running multiple rounds of our algorithm by using results of previous rounds as "deny list" for subsequent rounds and each round doesn't suffer a significant loss in utility even on satisfying a stronger aggregate DP guarantee.

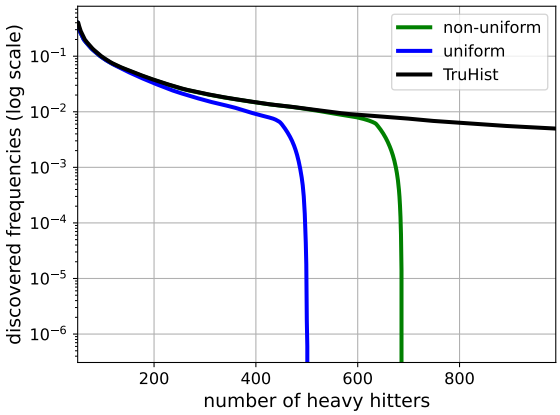
A. Adaptive Segmentation in Single Data Point Setting

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

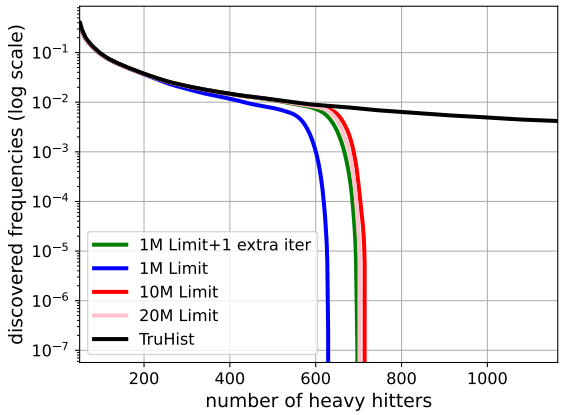
a) Adaptive Segmentation: We demonstrate the benefit of adaptively choosing the segment length, as opposed to using fixed-length uniform segment lengths. For our experimental evaluations in this part we set our data domain limit to $P = 10^7$ which means that the dimension of each payload of size $2^{l_t} \times |\mathcal{P}_{\text{prefixlist}_t}|$ 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 next iteration. Figure 2a 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 Figure 16). We use adaptive thresholding in both of the algorithms, with $FPR = 0.5$.

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 I: Choices of ϵ_{agg}, T and ϵ_l



(a) Discovered frequencies for different segmentation schemes in `OptPrefixTree`



(b) Discovered frequencies for different dimension limits in `OptPrefixTree`

Fig. 2: Experimental results in the single data point per device setting with $\epsilon_{agg} = 1$, $\delta = 10^{-6}$, and $T = 4$

However, the empirical FPR is even lower with the values of 0.35 and 0.41 for adaptive and uniform segment algorithms, respectively.

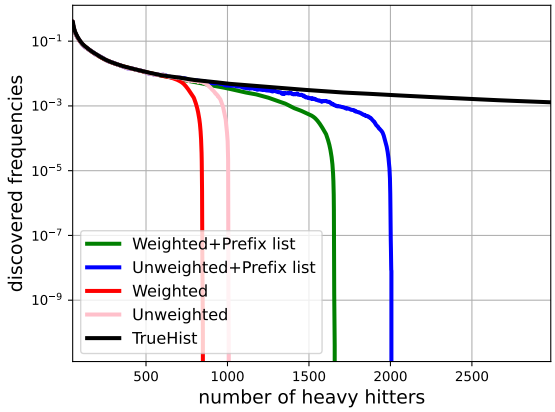
b) Dimension Limitation: In production-scale systems, the computation cost of decompressing and aggregating the responses can be a significant bottleneck when the data comes from a very large domain. Increasing system limits can be expensive and resource intensive. Thus, we explore the effect of different dimension constraints on the performance of `OptPrefixTree`. In order to do so, we used different constraints of $P = 20$ million, 10 million, 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 2b changing the limit to 1 million degrades the utility of the algorithm significantly. This is because of the small number of iterations and small system limit. In the final round, the segment length should be large enough to finish the algorithm in the desired number of iterations. Hence, the prefix list has to be cut to a smaller size not to exceed system dimension limit. 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 I). 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 D. 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 [Zhu and Wang, 2019].

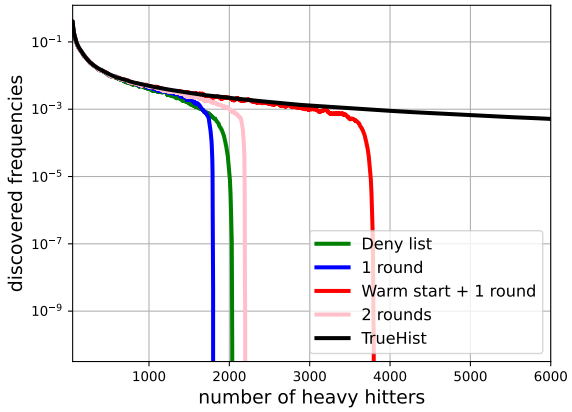
B. Data Selection for Multiple Data Points per Device

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

a) 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 $T = 4$. Figure 3a 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



(a) Discovered frequencies for different data selection schemes in `OptPrefixTree`



(b) Discovered frequencies when adding deny list in different settings to `OptPrefixTree`

Fig. 3: Experimental results in the multiple data points per device setting with $\epsilon_{agg} = 1$, $\delta = 10^{-6}$

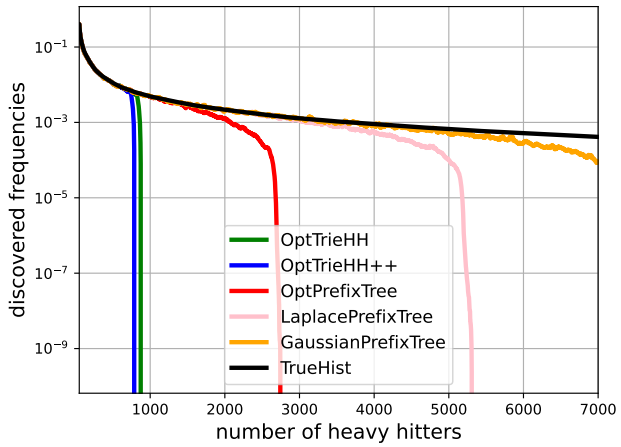


Fig. 4: Discovered frequencies comparison between `OptPrefixTree` and other baselines with $\epsilon_{agg} = 1$, $\delta = 10^{-6}$.

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 Figure 18.

b) 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. In this part we evaluate the effect of different ways of obtaining a deny list, $\mathcal{P}_{\text{denylist}}$ in the utility of the model. In Figure 3b we compare the following different configurations:

- `Deny list`: Here, the deny list comes from an auxiliary public data source (which is not protected with DP). The deny list is chosen to be the top 2000 popular words from Twitter Sentiment140 dataset [Go et al., 2009]. The utility of the deny list $\mathcal{P}_{\text{denylist}}$ on its own (denoted by `deny list` line), highlights the difference between the distribution of the Reddit and Twitter Sentiment 140 dataset. For Twitter Sentiment140 dataset only 0.6% of the data points in this dataset do not belong to Reddit dataset.
- `1 round`: For comparison, we also include the standard algorithm (`OptPrefixTree`) run for a single round, without a deny list. This is denoted by `1 round` line.
- `Warm start + 1 round`: This algorithm uses the auxiliary public as the data deny list and runs one round of `OptPrefixTree` to learn additional popular words. This shows the benefit of using an existing deny list in one round of algorithm execution. In particular, adding warm start leads to discovering 2.1× more heavy hitters in comparison to `1 round` without a deny list.
- `2 rounds`: In this case, we run `OptPrefixTree` twice and use the heavy hitters found in the first round as a deny list for the second round. Note that, here we need to account for the privacy budget of *all* the iteration of *both* rounds, which leads to more noise per iteration. The `2 rounds` algorithm increases the number of discovered heavy hitters by 1.22× in comparison to `1 round`

without a deny list.

For all the 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 Figure 19.

c) 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 $\epsilon_{agg} = 1$ 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 [Zhu et al., 2020] and for `TrieHH++` based on Lemma 3 in [Cormode and Bharadwaj, 2022]. Our analysis in Figure 4 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 E and Appendix F 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 [Dwork et al., 2014], [Dwork, 2006]. As with our previous experiments we compute the privacy guarantee of the composition over multiple rounds using the advanced composition theorem [Dwork et al., 2010], [Kairouz et al., 2017] and the composition bound in terms of Rényi differential privacy [Abadi et al., 2016], [Mironov, 2017]. We compute the Rényi privacy guarantees of both noise addition methods using the bounds provided in Table 2 of [Mironov, 2017]. We finally select the noise power using the tighter bound between these two. As shown in Figure 4 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.

VII. 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

[Abadi et al., 2016] Abadi, M., Chu, A., Goodfellow, I. J., McMahan, H. B., Mironov, I., Talwar, K., and Zhang, L. (2016). Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 308–318.

[Acharya et al., 2019] Acharya, J., Sun, Z., and Zhang, H. (2019). Hadamard response: Estimating distributions privately, efficiently, and with little communication. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pages 1120–1129. PMLR.

[Apple, 2017] Apple (2017). Apple differential privacy technical review.

[Bagdasaryan et al., 2021] Bagdasaryan, E., Kairouz, P., Mellem, S., Gascón, A., Bonawitz, K., Estrin, D., and Gruteser, M. (2021). Towards sparse federated analytics: Location heatmaps under distributed differential privacy with secure aggregation. *arXiv preprint arXiv:2111.02356*.

[Bassily et al., 2020] Bassily, R., Moran, S., and Nandi, A. (2020). Learning from mixtures of private and public populations. *Advances in Neural Information Processing Systems*, 33:2947–2957.

[Bassily et al., 2017] Bassily, R., Nissim, K., Stemmer, U., and Guha Thakurta, A. (2017). Practical locally private heavy hitters. *Advances in Neural Information Processing Systems*, 30.

[Bell et al., 2022] Bell, J., Gascon, A., Ghazi, B., Kumar, R., Manurangsi, P., Raykova, M., and Schoppmann, P. (2022). Distributed, private, sparse histograms in the two-server model. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, pages 307–321.

[Böhler and Kerschbaum, 2021] Böhler, J. and Kerschbaum, F. (2021). 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.

[Boneh et al., 2021] Boneh, D., Boyle, E., Corrigan-Gibbs, H., Gilboa, N., and Ishai, Y. (2021). Lightweight techniques for private heavy hitters. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 762–776. IEEE.

[Caldas et al., 2018] Caldas, S., Duddu, S. M. K., Wu, P., Li, T., Konečnỳ, J., McMahan, H. B., Smith, V., and Talwalkar, A. (2018). Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*.

[Canonne et al., 2020] Canonne, C. L., Kamath, G., and Steinke, T. (2020). The discrete gaussian for differential privacy. *Advances in Neural Information Processing Systems*, 33:15676–15688.

[Carvalho et al., 2022] Carvalho, R. S., Wang, K., and Gondara, L. S. (2022). Incorporating item frequency for differentially private set union. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 36, pages 9504–9511.

[Cheu et al., 2019] Cheu, A., Smith, A., Ullman, J., Zeber, D., and Zhilyaev, M. (2019). Distributed differential privacy via shuffling. In Ishai, Y. and Rijmen, V., editors, *Advances in Cryptology – EUROCRYPT 2019*, pages 375–403, Cham. Springer International Publishing.

[Cormode and Bharadwaj, 2022] Cormode, G. and Bharadwaj, A. (2022). Sample-and-threshold differential privacy: Histograms and applications. In *International Conference on Artificial Intelligence and Statistics*, pages 1420–1431. PMLR.

[Cormode et al., 2012] Cormode, G., Procopiuc, C., Srivastava, D., Shen, E., and Yu, T. (2012). Differentially private spatial decompositions. In *2012 IEEE 28th International Conference on Data Engineering*, pages 20–31. IEEE.

[Dwork, 2006] Dwork, C. (2006). 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.

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

[Dwork et al., 2014] Dwork, C., Roth, A., et al. (2014). The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science*, 9(3–4):211–407.

[Dwork et al., 2010] Dwork, C., Rothblum, G. N., and Vadhan, S. P. (2010). Boosting and differential privacy. In *FOCS*.

[Elkordy et al., 2023] Elkordy, A. R., Ezzeldin, Y. H., Han, S., Sharma, S., He, C., Mehrotra, S., Avestimehr, S., et al. (2023). Federated analytics: A survey. *APSIPA Transactions on Signal and Information Processing*, 12(1).

[Erlingsson et al., 2019] Erlingsson, U., Feldman, V., Mironov, I., Raghunathan, A., Talwar, K., and Thakurta, A. (2019). 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. Society for Industrial and Applied Mathematics.

[Erlingsson et al., 2014] Erlingsson, Ú., Pihur, V., and Korolova, A. (2014). Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067.

[Feldman et al., 2023] Feldman, V., McMillan, A., and Talwar, K. (2023). 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.

[Feldman et al., 2022] Feldman, V., Nelson, J., Nguyen, H., and Talwar, K. (2022). Private frequency estimation via projective geometry. In *International Conference on Machine Learning*, pages 6418–6433. PMLR.

[Feldman and Talwar, 2021] Feldman, V. and Talwar, K. (2021). Lossless compression of efficient private local randomizers. In *International Conference on Machine Learning*, pages 3208–3219. PMLR.

[Ghazi et al., 2021] Ghazi, B., Golowich, N., Kumar, R., Pagh, R., and Velingker, A. (2021). 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.

[Go et al., 2009] Go, A., Bhayani, R., and Huang, L. (2009). Twitter sentiment classification using distant supervision. *CS224N project report, Stanford*, 1(12):2009.

[Gopi et al., 2020] Gopi, S., Gulhane, P., Kulkarni, J., Shen, J. H., Shokouhi, M., and Yekhanin, S. (2020). Differentially private set union. In *International Conference on Machine Learning*, pages 3627–3636. PMLR.

[Joseph et al., 2019] Joseph, M., Mao, J., Neel, S., and Roth, A. (2019). The role of interactivity in local differential privacy. In *2019 IEEE 60th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 94–105. IEEE.

[Kairouz et al., 2017] Kairouz, P., Oh, S., and Viswanath, P. (2017). The composition theorem for differential privacy. *IEEE Transactions on Information Theory*, 63(6):4037–4049.

[Kasiviswanathan et al., 2011] Kasiviswanathan, S. P., Lee, H. K., Nissim, K., Raskhodnikova, S., and Smith, A. (2011). What can we learn privately? *SIAM Journal on Computing*, 40(3):793–826.

[Kim et al., 2021] Kim, K., Gopi, S., Kulkarni, J., and Yekhanin, S. (2021). Differentially private n-gram extraction. *ArXiv*, abs/2108.02831.

[Liu et al., 2021] Liu, T., Vietri, G., Steinke, T., Ullman, J., and Wu, S. (2021). Leveraging public data for practical private query release. In *International Conference on Machine Learning*, pages 6968–6977. PMLR.

[McMillan et al., 2022] McMillan, A., Javidbakht, O., Talwar, K., Briggs, E., Chatzidakis, M., Chen, J., Duchi, J., Feldman, V., Goren, Y., Hesse, M., et al. (2022). Private federated statistics in an interactive setting. *arXiv preprint arXiv:2211.10082*.

[Mironov, 2017] Mironov, I. (2017). Rényi differential privacy. In *2017 IEEE 30th computer security foundations symposium (CSF)*, pages 263–275. IEEE.

[Qardaji et al., 2012] Qardaji, W., Yang, W., and Li, N. (2012). Differentially private grids for geospatial data. *Proceedings - International Conference on Data Engineering*.

[Song et al., 2013] Song, S., Chaudhuri, K., and Sarwate, A. D. (2013). Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pages 245–248.

[Wang et al., 2019] Wang, T., Li, N., and Jha, S. (2019). Locally differentially private heavy hitter identification. *IEEE Transactions on Dependable and Secure Computing*, 18(2):982–993.

[Wilson et al., 2020] Wilson, R. J., Zhang, C. Y., Lam, W., Desfontaines, D., Simmons-Marengo, D., and Gipson, B. (2020). Differentially private sql with bounded user contribution. *Proceedings on privacy enhancing technologies*, 2020(2):230–250.

[Zhu et al., 2020] Zhu, W., Kairouz, P., McMahan, B., Sun, H., and Li, W. (2020). Federated heavy hitters discovery with differential privacy. In *International Conference on Artificial Intelligence and Statistics*, pages 3837–3847. PMLR.

[Zhu and Wang, 2019] Zhu, Y. and Wang, Y.-X. (2019). Poission subsampled rényi differential privacy. In *International Conference on Machine Learning*, pages 7634–7642. PMLR.

APPENDIX A
NOTATION TABLE

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
$\widetilde{f}(x)$	estimated frequency of data point x

TABLE II: Notations

APPENDIX B
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 each element queried (usually the data domain in each round $\mathcal{P}_{\text{prefixlist}_{t-1}} \times A^{l_t}$). 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 [Feldman and Talwar, 2021], 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 [Feldman and Talwar, 2021]). 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 [Feldman and Talwar, 2021] gives the PI-RAPPOR local randomizer algorithm which can be used as A_{priv} and Algorithms 4 and 5 of [Feldman and Talwar, 2021] 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 $\widetilde{f}[w]$ as defined below:

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

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 [Feldman and Talwar, 2021], $\tilde{f}$ is an unbiased estimator of f with the minimum possible variance for locally private estimators. Algorithms 4 and 5 in [Feldman and Talwar, 2021] output the same estimate but only differ in terms of their computational complexity. Please See [Feldman and Talwar, 2021] 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.

a) 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 $K = \max\{e^{\epsilon_l}+1, 2^l\}$. We also let $c = \lfloor \frac{q}{e^{\epsilon_l}+1} \rfloor$ and denote $\{0,1,\ldots,c\}$ 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 [Feldman and Talwar, 2021] 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 $\frac{e^{\epsilon_l}+1}{\epsilon_l}$.

APPENDIX C
PROOF OF PROPOSITION V.3

Proof of Proposition V.3. In a single round, we query each element in the data domain of size $|A|^r$. Thus using Assumption V.1, with $\beta' = \frac{\beta}{|A|^r}$ and using a union bound, we have that with probability $1-\beta$, $\max_{d \in U} |F[d] - \widetilde{f}[d]| \leq C\sqrt{\frac{ne^{\epsilon_l}}{(e^{\epsilon_l}-1)^2} \log(\frac{|A|^r}{\beta})}$.

Now, we prove that this algorithm is λ-accurate (Definition V.2) with $\lambda = C\sqrt{\frac{ne^{\epsilon_l}}{(e^{\epsilon_l}-1)^2} \log(\frac{|A|^r}{\beta})}$. Let $d \in A^r$ with $\widetilde{f}[d] \geq \tau\sigma + \lambda$. 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 $\lambda = C\sqrt{\frac{ne^{\epsilon_l}}{(e^{\epsilon_l}-1)^2} \log(\frac{|A|^r}{\beta})}$.

□

APPENDIX D
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 $H = (x_1, x_2, \ldots, x_{|H|})$ 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).

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.

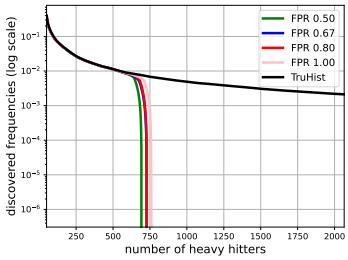
For marginal figures, to aid with visualisation in our plots, for a window size $W = 50$ and a given heavy hitter set H, we plot for each i, the sum of the probabilities (according to $\mathbb{P}$) of the heavy hitters in the sliding window $(x_{i-W}, x_{i-W+1}, \ldots, x_i)$. We also plot a true histogram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

A. Adaptive Segmentation for Single Data Point

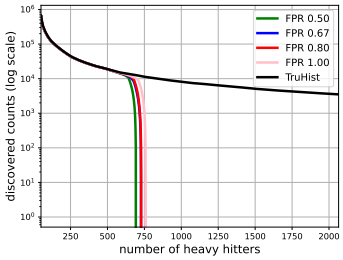
In this section, we explore the effect of different parameters in the utility of `OptPrefixTree`. For simplicity in this section, we assume each device has a single datapoint.

a) 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.

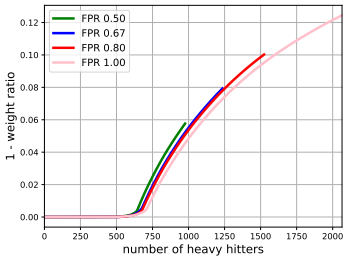
As observed increasing the FPR from 0.5 to 1 increases the number of heavy hitters detected slightly [660, 656, 678.0, 695] but increasing FPR also increases the number of discovered false positives significantly [357, 636, 828, 1400]. Based on Figure 5c, although lower FPR detects fewer true bins to 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.



(a) Marginal discovered frequencies



(b) Marginal discovered counts



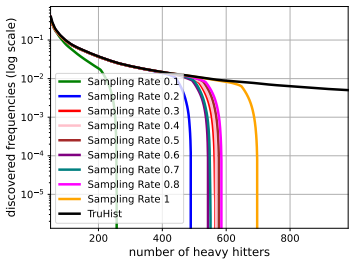
(c) Total WR Loss

Fig. 5: Effect of the different FPR parameters for single data point setting on `OptPrefixTree` utility ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$ and $T = 4$)

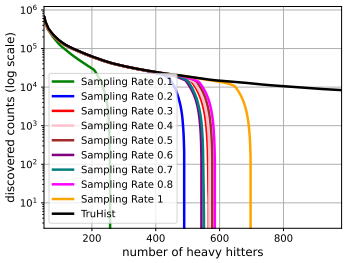
T	3										4										5									
sampling rate	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1
ϵ_l	8.42	8.26	8.19	8.12	8.1	7.92	7.81	7.73	7.62	7.73	8.32	8.19	8.12	7.97	7.87	7.72	7.59	7.52	7.4	7.58	8.27	8.12	8.1	7.86	7.6	7.35	7.19	7.2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

TABLE III: Sampling rate effect on the ϵ_l for $\epsilon_{agg} = 1$ and $\delta = 10^{-6}$

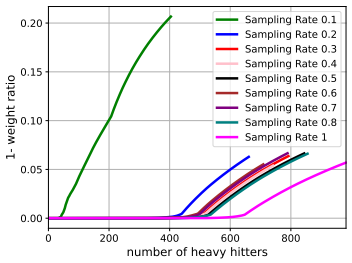
b) Number of Devices Limitation: In this part we analyze the effect of having constraints on the number of devices on the utility of the model. In a production-scale model with billions of devices sending data, dimension can grow extremely large. One way to get around this issue is to use sampling. By 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 Ber(γ) where γ is the sub-sampling rate. We use theorem 5 described in [Zhu and Wang, 2019] for estimating the upper-bound of ϵ_c. The effect of different sub-sampling rates on the value of ϵ_l when $\epsilon_c = 1$ and $N = 1.6 * (10)^6$ is shown in Table III. 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 6c, 6b, 6a 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.



(a) Marginal discovered frequencies



(b) Marginal discovered counts



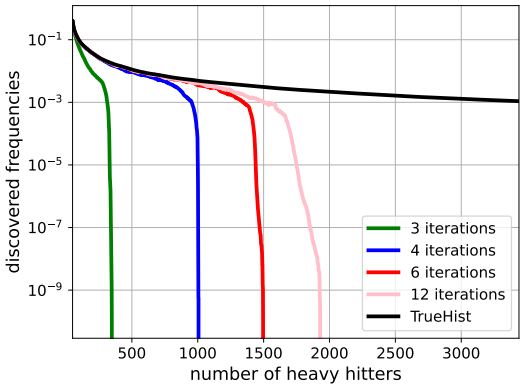
(c) Total WR Loss

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

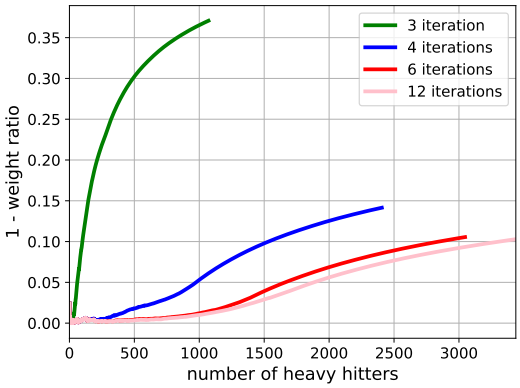
B. Multiple Data point Adaptive Segmentation

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.

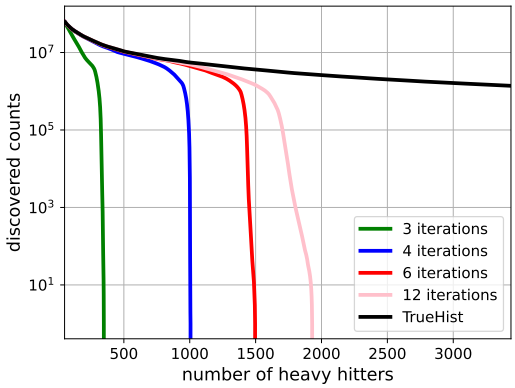


(a) Marginal discovered frequencies (weighted metric)

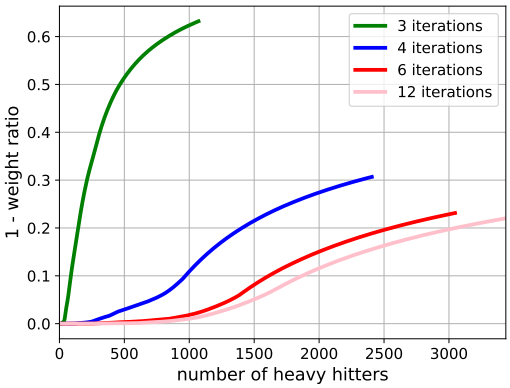


(b) Total WR loss (weighted metric)

Fig. 7: Effect of the segment size for multiple data points setting on `OptPrefixTree` utility ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$)



(a) Marginal discovered counts (unweighted metric)



(b) Total WR Loss (unweighted metric)

Fig. 8: Effect of the segment size for multiple data points setting on `OptPrefixTree` utility ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$)

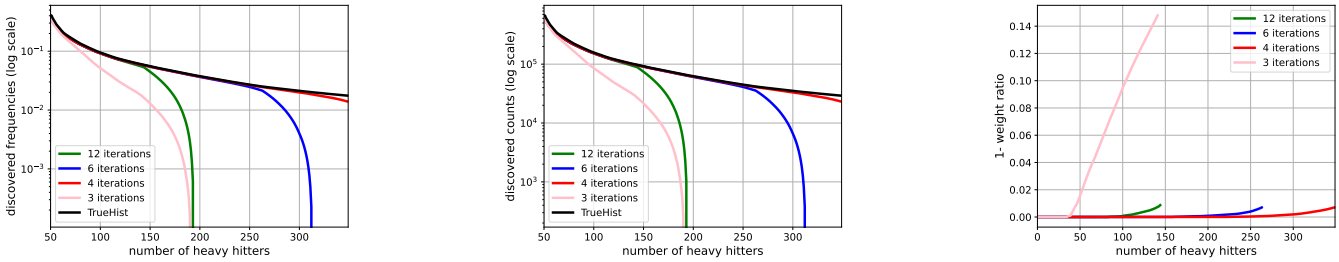
APPENDIX E
TRIEHH

A. *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` [Zhu et al., 2020]. In Figure 9a, 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 $\epsilon_{agg} = 1$ and sampling rates are set based on table IV. 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 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.

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 IV: Number of rounds effect on the sampling rate of `TrieHH` ($\delta = 10^{-6}$, $N = 1.6 \times 10^6, \theta = 10$)



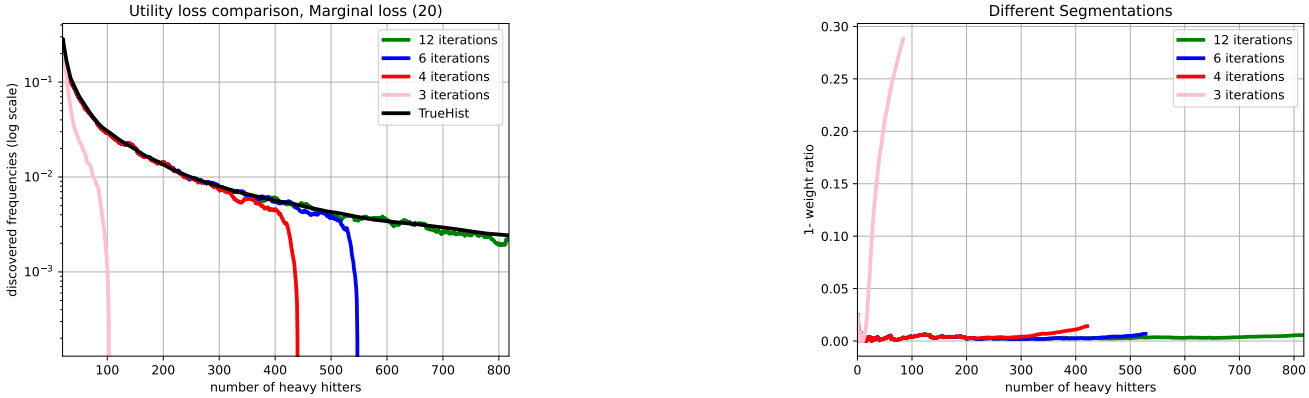
(a) Marginal discovered frequencies (b) Marginal discovered counts (c) Total WR loss

Fig. 9: The effect of different number of iterations on `TrieHH` for single data point setting ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$)

B. 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 10a and 10b we used weighted sampling described in the original paper.



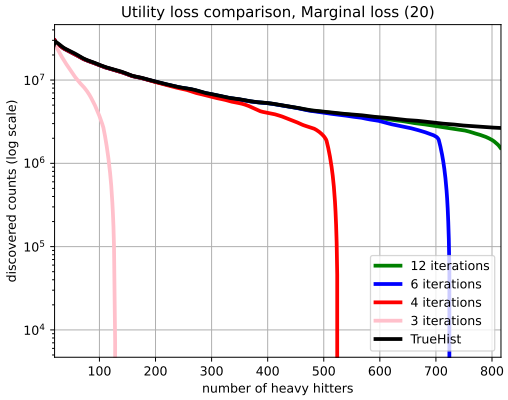
(a) Marginal discovered frequencies (b) Total WR loss

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

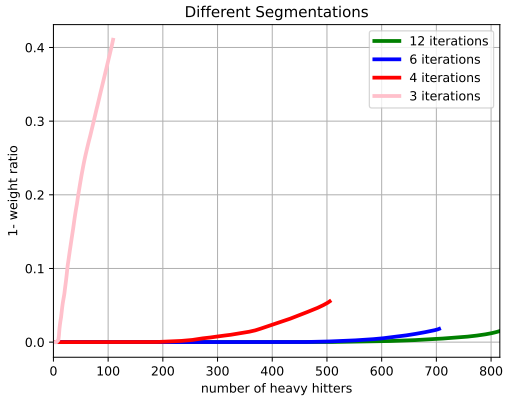
To further improve the utility of `TrieHH`, we used unweighted sampling in another set of experiments. This new sampling scheme leads to finding $[816, 529, 422, 85]$ heavy hitters when having 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char) iterations respectively. Figure 11a and 11b shows the loss and 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`.

APPENDIX F
TRIEHH++

Based on Lemma 3 of [Cormode and Bharadwaj, 2022], `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 $C_\alpha = \ln 1/\alpha - 1/(1 + \alpha)$. The ϵ here is for one iteration. We used advanced composition in theorem 3.4 of [Kairouz et al., 2017] 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 V and VI.



(a) Marginal discovered counts



(b) Total WR loss

Fig. 11: 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

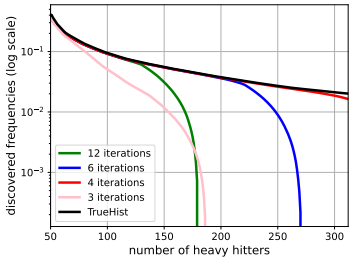
TABLE V: Number of rounds effect on the sampling rate of `TrieHH++` ($\delta = 10^{-6}$, $N = 1.6 \times 10^6, \theta = 10$)

A. *Single Data Point Setting for* `TrieHH++`

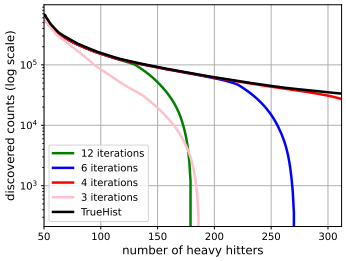
In this section we discuss the effect of number of iterations on the utility of a single data point setting for `TrieHH++` [Cormode and Bharadwaj, 2022]. In Figure 12, we show the effect of different segmentation on the utility of the algorithm for a single data point per device setting. For these experiments we used $\epsilon_{agg} = 1$ and sampling rates are set based on table V. 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.

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

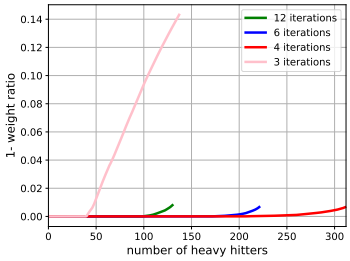
TABLE VI: Number of rounds effect on the sampling rate of `TrieHH++` ($\delta = 10^{-6}$, $N = 1.6 \times 10^6, \theta = 20$)



(a) Marginal discovered frequencies

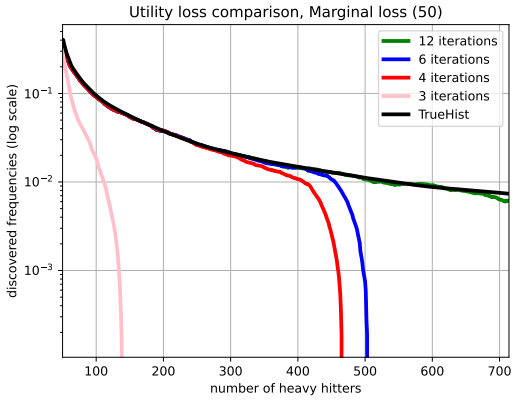


(b) Marginal discovered counts

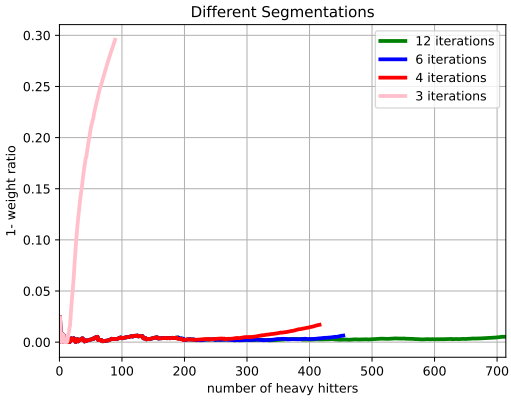


(c) Total WR loss

Fig. 12: The effect of different number of iterations on `TrieHH++` for single data point setting ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$)

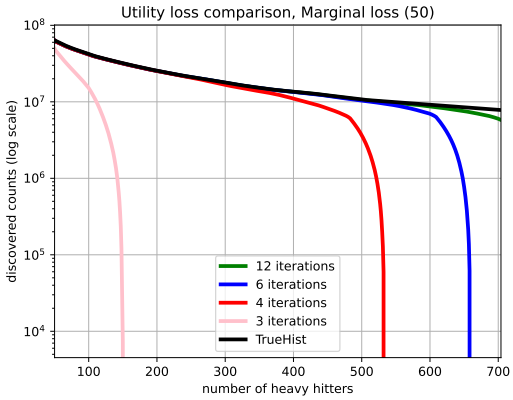


(a) Marginal discovered frequencies

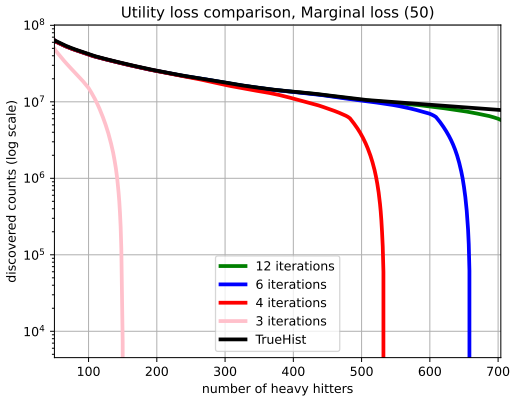


(b) Total WR loss

Fig. 13: The effect of different number of iterations on $\texttt{TrieHH}++$ for multiple data points setting with weighted sampling($\epsilon_{agg} = 1$, $\delta = 10^{-6}$)



(a) Marginal discovered counts



(b) Total WR loss

Fig. 14: The effect of different number of iterations on $\texttt{TrieHH}++$ for multiple data points setting with unweighted sampling ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$)

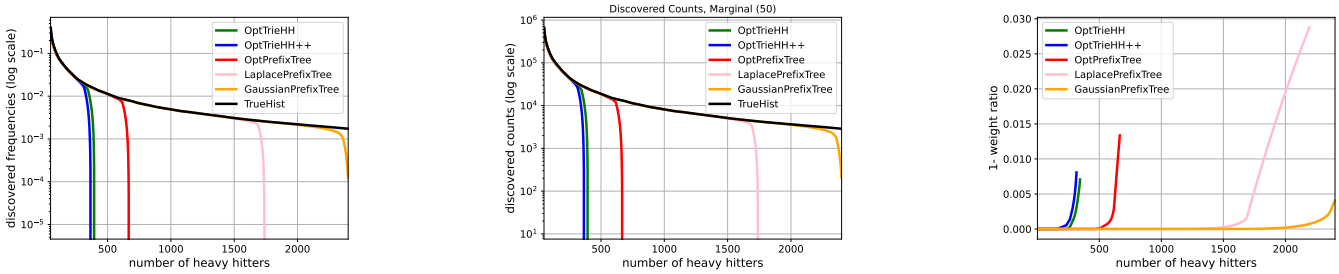
B. Multiple Data Points Setting for $\texttt{TrieHH}++$

For our evaluation, we used the same binary encoding we described before. The total number of heavy hitters detected by $\texttt{TrieHH}++$ 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 13, in this setting, 12 iterations shows the best utility.

To further improve the utility of $\texttt{TrieHH}++$, 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 14a and 14b 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 $\texttt{TrieHH}++$ and we refer to it as $\texttt{OptTrieHH}++$.

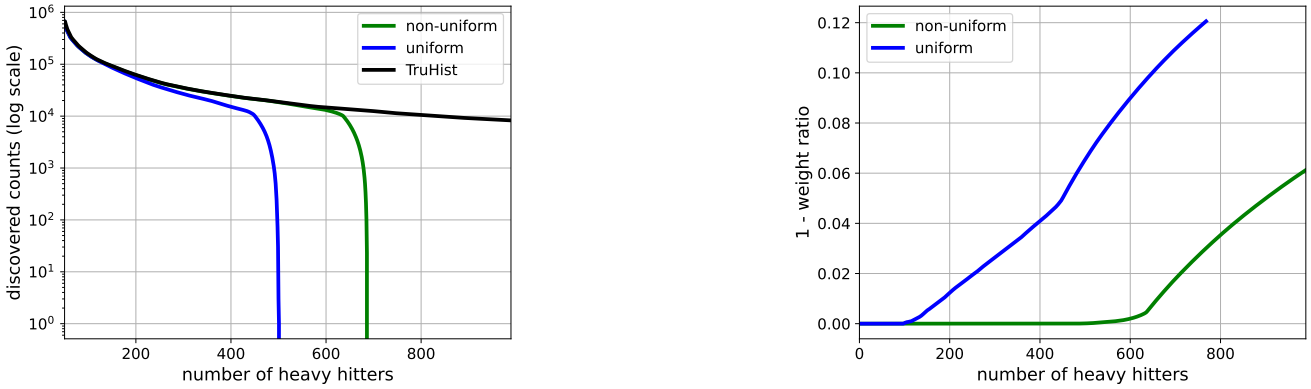
APPENDIX G
COMPARISON OF $\texttt{OptTrieHH}$, $\texttt{OptTrieHH}++$, AND `OptPrefixTree`

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



(a) Marginal discovered frequencies (b) Marginal discovered counts (c) Total WR loss

Fig. 15: Utility comparison between `OptTrieHH` and `OptPrefixTree` for single data point setting($\epsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)



(a) Marginal discovered counts comparison (b) Total WR loss

Fig. 16: Effect of different segmentation on `OptPrefixTree` for single data point setting ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)

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 15a 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 15b shows the same plot but in the y-axis we have the marginal counts of discovered bins. Also, Figure 15c shows the total utility loss.

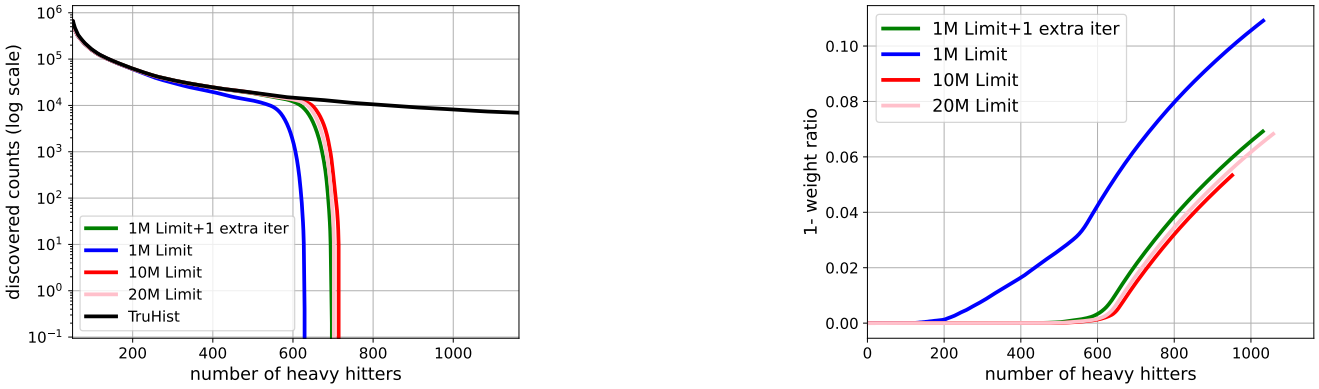
APPENDIX H
ADDITIONAL EXPERIMENTAL RESULTS

a) Effect of Uniform vs Non-uniform Segmentation in Single Data Point Setting: As explained in VI using our adaptive segmentation algorithm helps discovering more heavy hitters. In Figures 16a and 16b we show the discovered counts and total utility loss comparison of this uniform and non-uniform segmentation.

b) Effect of Dimension Limitation in Single Data Point Setting: In Figures 17a and 17b 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.

c) Effect of Data Selection in Multiple Data Points Setting: There are different ways to measure the frequency of data points. In Section IV, we discuss how averaging the distribution of words overall devices can be used to define the global frequency of words (weighted metric). Figure 18b shows the total utility loss when using this distribution.

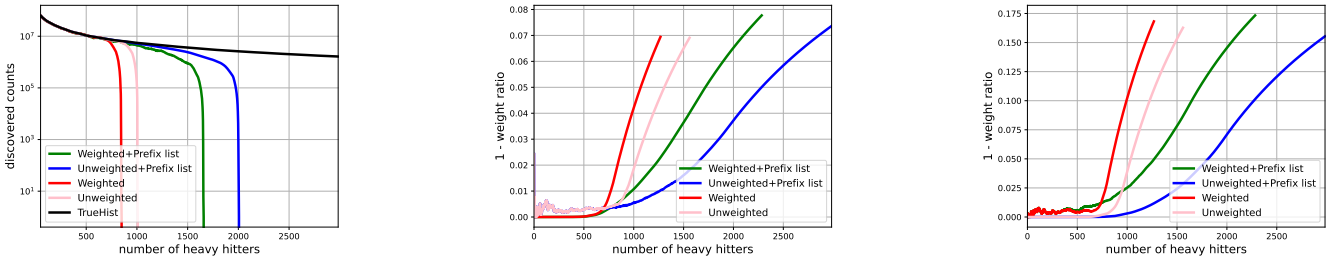
One other way to measure the frequency is the percentage of devices who has the word in their support (unweighted metric). Figure 18a shows the global number of discovered bins based on how many devices have the word. Figure 18c demonstrates



(a) Marginal discovered counts

(b) Total WR Loss

Fig. 17: Effect of dimension limitation for single data point setting on `OptPrefixTree` utility ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$ and $T = 4$)



(a) Marginal discovered counts

(b) Total WR loss (weighted metric)

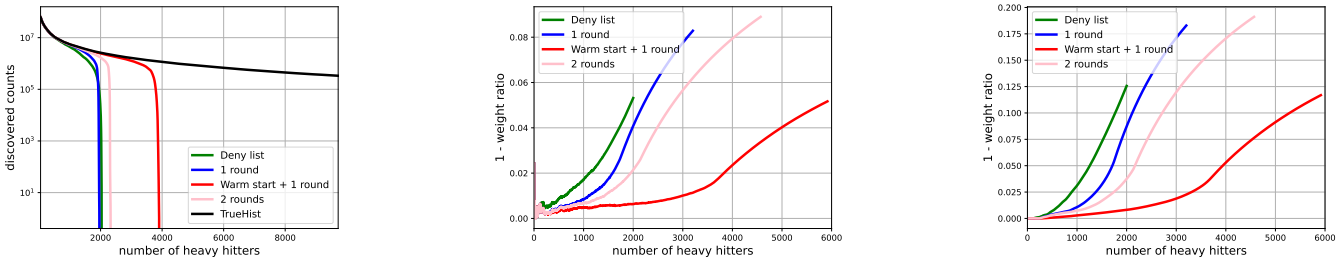
(c) Total WR loss (unweighted metric)

Fig. 18: Comparing different data selection schemes in `OptPrefixTree` for multiple data points ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)

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.

d) Effect of adding a `deny list` in Multiple Data Points Setting: Figure 19b shows the total utility loss when using the frequency of the words in devices for extracting the global distribution. In Figure 19a and 19c we use the number of devices with the word to demonstrate the true counts of the discovered words and total utility loss.

e) Comparison with previous works: As illustrated in section VI `OptPrefixTree` outperforms `OptTrieHH` under the same constraints. In Figure 20a and 20b, we show a comparison of `OptPrefixTree` and `OptTrieHH` for the multiple data points setting.

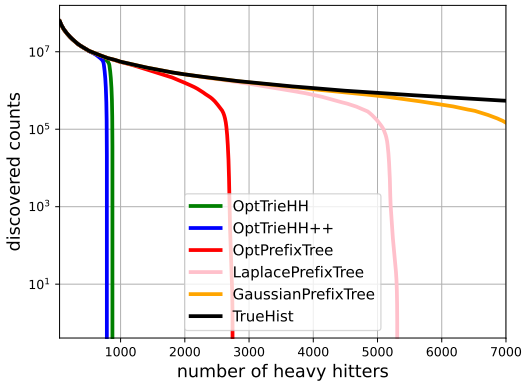


(a) Marginal discovered counts

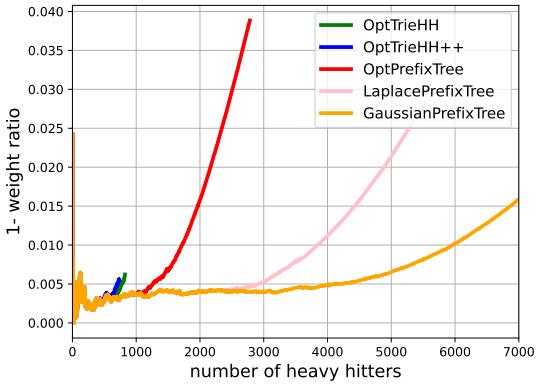
(b) Total WR loss (weighted metric)

(c) Total WR loss (unweighted metric)

Fig. 19: Effect of adding deny list to `OptPrefixTree` for multiple data points setting ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)



(a) Marginal discovered counts



(b) Total WR loss

Fig. 20: Comparing `OptPrefixTree`, `OptTrieHH`, `OptTrieHH++` for multiple data points setting ($\epsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)