Differentially Private Heavy Hitter Detection using Federated Analytics

Karan Chadha [1 *] **Junye Chen** [2] **John Duchi** [1 2] **Vitaly Feldman** [2] **Hanieh Hashemi** [2] **Omid Javidbakht** [2] **Audra McMillan** [2] **Kunal Talwar** [2]

Abstract

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 aggregate 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 most frequent words.

1. 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 & 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 high dimensional space 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, 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). However, our focus is aggregate DP guarantee, and we do not put an upper bound in the local epsilon. A potential architecture for running iterative algorithms in this model of privacy is outlined in (McMillan et al., 2022).

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 (McMillan et al., 2022; Cormode et al., 2012; Qardaji et al., 2012; Song et al., 2013; Bagdasaryan et al., 2021; Kim et al., 2021). 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

* Research was conducted when author was an intern at Apple. Authorship is in alphabetical order. [1]Stanford University [2]Apple Inc. Correspondence to: Karan Chadha <knchadha@stanford.edu>, Hanieh Hahsmei <h_hashemi@apple.come>, Audra McMillan <audra_mcmillan@apple.com>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

of finding the heavy hitters in each round of the algorithm varies in prior work, although is generally based on a DP frequency estimation subroutine. One should consider system constraints (communication, computation, number of participating devices, etc.) and the privacy model when choosing a frequency estimation subroutine. In this work, we will focus on using one-hot encoding with binary randomized response (inspired by RAPPOR (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 work that treats 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. Our threshold selection algorithm allows us to control the false positive rate.

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 iteration, 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). Note that the specific data selection mechanism does not impact privacy guarantees.

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.

2. 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) methods and combination of multi-party-computing and DP techniques (Böhler & 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 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 & 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, this method is able to report the frequency of heavy hitters without using additional privacy budget. To achieve this 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.

While `TrieHH` and `TrieHH` $++$ satisfy the desired central differential privacy, none of them provide any local differential privacy guarantee. In the Appendix G and H, we evaluate the effect of different number of iterations for both model. Using the optimum number of iterations for both method, we compare the empirical utility of these models with `OptPrefixTree` in section 4.

3. Algorithm

In this section, we describe a high level outline of our proposed algorithm `OptPrefixTree`. There are a number of parameter and subroutine choices in this 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 this interactive algorithm is well-suited to the federated setting since it does not require every user to be present at every iteration.

Our algorithm is designed to satisfy device-level (protects against a user changing all of the data points associated to them) differential privacy (DP) in the aggregate model *and* the local model of differential privacy. We use standard composition theorems (Dwork et al., 2010; Kairouz et al., 2017; Abadi et al., 2016; Mironov, 2017), and bound the aggregate DP epsilon parameter by the numerical bound on privacy amplification by shuffling. Since the aggregate privacy guarantee is our primary privacy guarantee, we do not directly optimize our local privacy parameter. For more details on the privacy model, see Appendix B.

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, or computational constraints for the server-side algorithm, as in (Feldman & Talwar, 2021; Feldman et al., 2022).

Notation. Let $\mathbb{P}_i$ be the empirical distribution of user i's data and $\mathbb{P} := \frac{1}{N}\sum_{i\in[N]}\mathbb{P}_i$ be the global empirical distribution. Each data point $d \in \{0,1\}^r$ is of length r.

3.1. Private Heavy Hitters Algorithm

At every iteration t, the server has a list of live prefixes $\mathcal{P}_{\text{prefixlist}_t}$ of length $l_{\text{pref,t}}$ and a segment length l_t and devices (privately) report back the length $l_{\text{pref,t}} + l_t (\leq r)$ prefix of a data point that is not in the deny list $\mathcal{P}_{\text{denylist}}$ and whose $l_{\text{pref,t}}$ length prefix belongs in $\mathcal{P}_{\text{prefixlist}_t}$. 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 l_{t+1} for the next round. Algorithm 2 in Appendix C gives the pseudo-code of the algorithm. We will use T to denote the number of iterations and ε_l to be the local DP parameter for a single iteration. At the end of T rounds, our algorithm outputs a set of heavy hitters that includes the prefixes found in the last iteration ($\mathcal{P}_{\text{prefixlist}_{T+1}}$) and (if used) the contents of $\mathcal{P}_{\text{denylist}}$. We use $(\varepsilon_{\text{agg}}, \delta)$-DP to refer to the privacy parameters of our algorithm in the aggregate model.

3.2. Device Algorithm

The device first uses the data selection mechanism *selectData* to choose a data point from their on-device dataset that is not in the deny list, and whose $l_{\text{pref,t}}$-length prefix is in $\mathcal{P}_{\text{prefixlist}}$. Then we pass the $l_{\text{pref,t}} + l_t$-length prefix of the chosen data point to a ε_l-local DP algorithm and send the privatized output to the aggregation protocol (Algorithm 3).

The Local Randomizer In our experiments we use one-hot encoding with asymmetric binary randomized response (denoted `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 & Talwar, 2021) (Appendix D) or Proj-RAPPOR (Feldman et al., 2022) for better communication-computation trade-offs. 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.

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 $\mathcal{P}_{\text{denylist}}$. 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 $\mathcal{P}_{\text{denylist}}$. To maintain local differential privacy, we introduce a special data element $\perp$ that devices (privately) report if they have no eligible data points to select.

The empirical results of Kim et al. (2021) imply that selecting more than one data point per device improved performance in the central DP setting. Both of the selection mechanisms above naturally extend to mechanisms that select more than one data point. In order to focus on the impact of weighted vs. unweighted sampling, we focus on selecting a single data point per device. We leave an exploration of the optimal number of data points per device per iteration in the aggregate DP setting to future work.

3.3. Server Algorithm

The server receives the aggregated privatized responses and computes the prefix list and segmentation length for the next iteration (Pseudo-code in Algorithm 4). These should be chosen to both keep the prefixes of any heavy hitters, and maintain any constraints on the data domain size for the next iteration. In most of the prior works, the segment length and threshold (described below) for keeping a prefix are treated as hyperparameters that need to be tuned. Tuning hyperparameters is notoriously hard in the federated setting. Our adaptive algorithms chooses these parameters in response to user data, without using additional privacy budget.

Adaptive segmentation Our adaptive segmentation algorithm first computes the list of live prefixes for the next iteration. Given the aggregated privatized results the server can compute an estimate $\widetilde{f}(d)$ of the number of devices who sent the data point d in the last iteration, for every

data element $d \in \mathcal{P}_{\text{prefixlist}_t} \times A^{l_t}$. The prefix list selection algorithm aims to keep as many of the elements such that $\widetilde{f}(d) > 0$ as possible, while minimizing the number of "false positives" in the prefix list (data elements which do not match the selected data point for *any* device). When using `OHE+2RR`, each estimate $\widetilde{f}(d)$ 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 propose using an adaptive algorithm to choose 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 is chosen to be as small as possible while ensuring that the fraction of reported elements that are false positives (have count 0) does exceed a specified False Positive Ratio threshold denoted by FPR. Furthermore, when some datapoints have smaller length encodings than others, we remove these data points from the prefix list for the next round as soon as they are "complete". They are then added to the set of heavy hitters in the final output (details in Appendix C.1).

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 Appendix E, 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.

4. 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 1092 words and an average of 379 unique words. For investigations on the single data point per device setting, each device i samples a single data point from $\mathbb{P}_i$. This data point remains fixed during the multiple iterations of the algorithm. Unless stated otherwise, we use a Huffman encoding to translate the words into binary strings. One token is reserved for unknown characters and we have an end character encoding at the end of each bit stream. We set the system constraint $P = 10^7$ and $r = 60$ in all experiments. In all experiments we set FPR= 0.5.

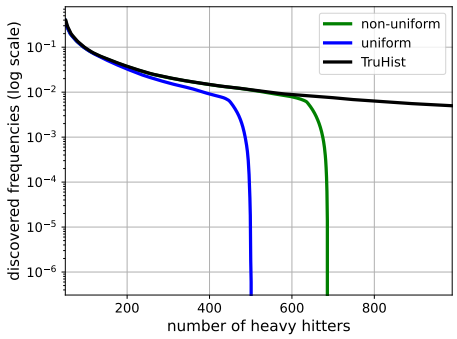


Figure 1. Discovered frequencies for different segmentation schemes in `OptPrefixTree`, $\varepsilon_{agg} = 1$, $\delta = 10^{-6}$ and $T = 4$.

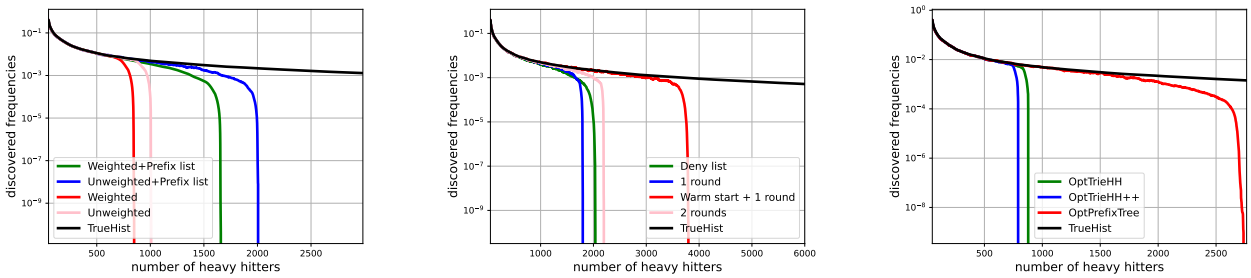
Evaluation metric: Let $H = (x_1, x_2, \ldots, x_{|H|})$ denote the set of heavy hitters output by an algorithm ordered by the empirical global frequency distribution $\mathbb{P}$. For our evaluations here, we report the frequencies (according to $\mathbb{P}$) of heavy hitters output by the algorithm. 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 about the plots are in Appendix F.

4.1. Adaptive Segmentation in Single Data Point Setting

We first focus on the simpler single data point per device setting and explore the effect of adaptive segmentation on the performance of `OptPrefixTree`. We demonstrate the benefit of adaptively choosing the segment length, as opposed to using fixed-length uniform segment lengths. Fig. 1 compares the discovered normalized frequencies of having uniform segments of sizes [15, 15, 15, 15] vs setting the segment length adaptively which leads to segment lengths of [23, 14, 11, 12]. Our adaptive scheme is able to discover 40% more heavy hitters (more plots in Fig. 15). We use adaptive thresholding in both algorithms, with $FPR = 0.5$. However, The empirical FPR is 0.35 and 0.41 for adaptive and uniform segment algorithms, respectively.

4.2. Data Selection for Multiple Data Points per Device

Effect of data selection. First, we evaluate the effect of weighted vs unweighted data selection schemes. 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. Fig. 2a shows the experimental results using the different data selection schemes. Unweighted sampling outperforms weighted sampling in both with and without prefix list experiments. As expected, conditioning on the prefix list has a significant impact. For the rest of this



(a) Discovered frequencies for different data selections in `OptPrefixTree`

(b) Discovered frequencies of adding `deny list` to `OptPrefixTree`

(c) Discovered frequencies comparison with prior works

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

paper, we use unweighted sampling conditioned on the prefix list for on-device data selection. For these experiments we set the $FPR = 0.5$. But, The empirical FPR is 0.30, 0.35, 0.35, and 0.38 for weighted selection + prefix list, unweighted selection + prefix list, weighted, and unweighted selection respectively. More evaluations in Fig. 16.

Effect of adding a deny list. We explore two ways of obtaining $\mathcal{P}_{\text{denylist}}$. Firstly, when the deny list comes from an auxiliary public data source (which is not protected with DP), which we refer to as a `warm start`. The second is when we run the algorithm twice, with the heavy hitters for the first round forming the deny list of the second round (referred to as `2 rounds`). In the 2 rounds case, we need to account for the privacy budget of both rounds. In Fig. 2b we compare the utility of different configurations. The warm start is initiated with the top 2000 popular words from Twitter Sentiment140 dataset (Go et al., 2009). We also present the utility of the warm start (`deny list` line) on its own, showing the difference between the distribution of two datasets. We also include the standard algorithm run for a single round, without a deny list (denoted as `1 round`). We further show the benefit of using deny list in one round of algorithm execution. Finally, we first execute one round of algorithm without a warm start and form a deny list out of discovered prefixes, which is then used in a second round. Fig. 2b shows that adding a deny list significantly increases performance. Adding warm start leads to $2.1\times$ more heavy hitters discovered. The 2 round algorithm increases the number of discovered heavy hitters discovered by $1.22\times$. For Twitter Sentiment140 dataset only 0.006 of the data points in this dataset do not belong to Reddit dataset. For the rest of configurations, we set the $FPR = 0.5$. The empirical FPR is 0.45, 0.37 and 0.39 for `1 round`, `1 round + warm start` and `2 rounds`, respectively.

Comparing with prior works. We compare with the optimized version of `TrieHH` (`OptTrieHH`) and `TrieHH++`(`OptTrieHH++`) in the setting where each device has multiple data points. We use our adaptive segmentation to determine the segment length at each round. For all three of the algorithms we use the unweighted data selection, which performed the best in our previous explorations. To have a fair comparison, the same binary encoding is used for all of the models (5 bits per character, not Huffman encoding). In `OptTrieHH` and `OptTrieHH++`, for $\varepsilon_{agg} = 1$ and $\delta = 10^{-6}$, we set the threshold for the number of reports that needs to be received for a word to be a part of the prefix list (denoted by θ in their paper) to 10. Accordingly, we set the sampling rate based for `TrieHH` based on Corollary 1 in (Zhu et al., 2020) and for `TrieHH++` based on Lemma 3 in (Cormode & Bharadwaj, 2022). In all of the methods, 12 iterations (1 character per iteration) shows the best performance for this encoding. Our analysis in Fig. 2c indicates `OptPrefixTree` is able to discover 3.2 times more heavy hitters for the same number of iterations and same dimension constraint. One explanation for this performance difference is that `OptTrieHH` and `OptTrieHH++` use sampling and thresholding to achieve the aggregated privacy guarantee, without adding any local differential privacy noise. When we set the threshold 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 G and Appendix H we explore the effect of different numbers of iterations in the utility of `TrieHH` and `TrieHH++` for both single and multiple data points setting.

5. 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 enhancing the utility of `OptPrefixTree`. Moreover, our method is able to provide both local and aggregated DP.

References

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

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

Apple. Apple differential privacy technical review, 2017. URL https://www.apple.com/privacy/docs/Differential_Privacy_Overview.pdf.

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

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

Böhler, J. and Kerschbaum, F. Secure multi-party computation of differentially private heavy hitters. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, pp. 2361–2377, 2021.

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

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

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

Cheu, A., Smith, A., Ullman, J., Zeber, D., and Zhilyaev, M. Distributed differential privacy via shuffling. In Ishai, Y. and Rijmen, V. (eds.), *Advances in Cryptology – EUROCRYPT 2019*, pp. 375–403, Cham, 2019. Springer International Publishing. ISBN 978-3-030-17653-2.

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

Cormode, G., Procopiuc, C. M., Srivastava, D., Shen, E., and Yu, T. Differentially private spatial decompositions. In Kementsietsidis, A. and Salles, M. A. V. (eds.), *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, pp. 20–31. IEEE Computer Society, 2012. doi: 10.1109/ICDE.2012.16. URL https://doi.org/10.1109/ICDE.2012.16.

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

Dwork, C., McSherry, F., Nissim, K., and Smith, A. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*, 2006.

Dwork, C., Rothblum, G. N., and Vadhan, S. P. Boosting and differential privacy. In *FOCS*, 2010.

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

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

Erlingsson, U., Feldman, V., Mironov, I., Raghunathan, A., Talwar, K., and Thakurta, A. 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, pp. 2468–2479, USA, 2019. Society for Industrial and Applied Mathematics.

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

Feldman, V., Nelson, J., Nguyen, H., and Talwar, K. Private frequency estimation via projective geometry. In *International Conference on Machine Learning*, pp. 6418–6433. PMLR, 2022.

Feldman, V., McMillan, A., and Talwar, K. Stronger privacy amplification by shuffling for rényi and approximate differential privacy. In *Proceedings of the 2023 Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 4966–4981. SIAM, 2023.

Ghazi, B., Golowich, N., Kumar, R., Pagh, R., and Velingker, A. 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*, pp. 463–488. Springer, 2021.

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

Jain, P., Raskhodnikova, S., Sivakumar, S., and Smith, A. D. The price of differential privacy under continual observation. *ArXiv*, abs/2112.00828, 2021.

Joseph, M., Mao, J., Neel, S., and Roth, A. The role of interactivity in local differential privacy. pp. 94–105, 11 2019. doi: 10.1109/FOCS.2019.00015.

Kairouz, P., Oh, S., and Viswanath, P. The composition theorem for differential privacy. *IEEE Transactions on Information Theory*, 63(6):4037–4049, 2017. doi: 10.1109/TIT.2017.2685505.

Kasiviswanathan, S. P., Lee, H. K., Nissim, K., Raskhodnikova, S., and Smith, A. What can we learn privately? volume 40, pp. 793–826, 2011. doi: 10.1137/090756090. URL https://doi.org/10.1137/090756090.

Kim, K., Gopi, S., Kulkarni, J., and Yekhanin, S. Differentially private n-gram extraction. *ArXiv*, abs/2108.02831, 2021.

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

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

Qardaji, W., Yang, W., and Li, N. Differentially private grids for geospatial data. *Proceedings - International Conference on Data Engineering*, 09 2012. doi: 10.1109/ICDE.2013.6544872.

Song, S., Chaudhuri, K., and Sarwate, A. D. Stochastic gradient descent with differentially private updates. In *2013 IEEE Global Conference on Signal and Information Processing*, pp. 245–248, 2013. doi: 10.1109/GlobalSIP.2013.6736861.

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

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

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

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
r	Fixed total length of all the words
l_t	Segment length in iteration t
$\mathcal{H}_t$	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 1. Notations

B. Differential Privacy

In this work, we will consider an algorithm that satisfies two levels of privacy protection appropriate for federated learning; differential privacy in the aggregate model and differential privacy in the local model. For more details on a potential system for achieving these guarantees please see (McMillan et al., 2022). In the federated setting where users may have more than one data point, there are two main choices for the granularity of the privacy guarantee: device-level DP and event (or data point)-level DP. We will focus on the stronger of these two guarantees, device-level DP, which protects against a user changing all of the data points associated to them. We will introduce these two types of privacy guarantees in this section. Throughout the remainder of this section, when we refer to a user's data point, we are referring to their set of data points.

B.1. Local Differential Privacy

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

Definition B.1 (Local Randomizer (Dwork & 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

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

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

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

Definition B.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.

B.2. Aggregate Differential Privacy

In the aggregate model of differential privacy, we assume the existence of an aggregation protocol that sums the local reports before they are released to the analyst. The analyst still interacts with the clients in a federated manner to perform the algorithm, but the aggregation protocol guarantees that the analyst does not receive anything about the local reports *except their sum*. The aggregate model of DP is a derivative of the more general and more common than aggregate DP shuffle model of differential privacy introduced in (Erlingsson et al., 2019; Cheu et al., 2019).

Definition B.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 $\mathcal{A}(d^{(1)}, \cdots, d^{(n)}) = \phi(\texttt{Aggregator}(f(d^{(1)}), \cdots, f(d^{(n)}))$ where $f : \mathcal{D} \to \mathcal{Z}$ 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,

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

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

When $\delta > 0$, we call this approximate DP. When each user uses a local randomizer (i.e. the functions f in Definition B.3 are local randomizers), the privacy guarantee in the aggregation model can be bounded by a quantity that is a function of both ε_0, the privacy guarantee in the local model, and n, the number of users that participate in the aggregation protocol (Erlingsson et al., 2019; Cheu et al., 2019). As the number of users increases, the privacy guarantee on the output of the aggregation protocol gets stronger; essentially each user gets "lost in the crowd". In this work, we will bound aggregate DP guarantee by the numerical bound on privacy amplification by shuffling due to (Feldman et al., 2023), who provide bounds for both approximate DP and a related privacy notion called Rényi DP.

A multi-round algorithm $\mathcal{A}$ is (ε, δ)-DP in the aggregate model if it is the composition of single round algorithms which are DP in the aggregate model, and the total privacy loss of $\mathcal{A}$ is (ε, δ)-DP. One can formulate a version of Definition B.3 specifically for multi-round algorithms, for a more in-depth discussion see (Jain et al., 2021). There are a number of standard theorems for analysing the privacy guarantee of composing multiple differentially private algorithms (Dwork et al., 2006; 2010). When the number of iterations is small, the advanced composition theorem (Dwork et al., 2010; Kairouz et al., 2017) provides a tight analysis. When the number of iterations is large, a tighter analysis is obtained by computing the composition bound in terms of Rényi differential privacy (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, the number of iterations, and the desired aggregate privacy guarantee, we can use binary search to approximate the smallest per iteration local epsilon that will achieve the given aggregate privacy guarantee. This algorithm is given in Algorithm 1 where RenyiShuffleAnalysis computes the Rényi privacy guarantee for amplification by shuffling, Composition uses the composition theorem for Rényi DP, Conversion converts the Rényi DP guarantees to approximate DP guarantees and BinarySearch makes the decision on whether to increase or decrease $\varepsilon_l{}'$. Since the aggregate privacy guarantee is our primary privacy guarantee, we do not put an upper bound on our local epsilon. Table 2 in Appendix B demonstrates how the local epsilon increases with the number of iterations, and the desired aggregate privacy guarantee. For details, see Appendix B. Table 2 shows different values of ε_l and ε_{agg} depending on the number of iterations for $N = 1.6 \times 10^6$ devices (number of users in the Reddit data set used for our experiments).

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
$\varepsilon_l{}' \leftarrow$ Initialization
while $|\epsilon_{agg} - \epsilon'_{agg}| \leq E$ **do**
  $\epsilon'_r \leftarrow \text{RenyiShuffleAnalysis}(\epsilon'_l, \delta, T, N, \alpha)$
  $\epsilon'_{composition} \leftarrow \text{Composition}(\epsilon'_r, T)$ // $\epsilon'_{composition} = T \times \epsilon'_r$
  $\epsilon'_{agg} \leftarrow \text{Conversion}(\epsilon'_{composition}, \delta, \alpha)$ // *Proposition 12 of (Canonne et al., 2020)*
  $\epsilon'_l \leftarrow \text{BinarySearch}(\epsilon_{agg}, \epsilon'_{agg}, \varepsilon_l{}')$
end while
$\epsilon_l \leftarrow \epsilon'_l$
return ϵ_l

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

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

C. Further Algorithm Details and Pseudo-code

We give the pseudo-code of our proposed algorithm `OptPrefixTree`. Pseudo-code for the overall outline is given in Algorithm 2. The algorithm allows to include both a deny list, and a "discovered list" $\mathcal{P}_{\text{discoveredlist}}$, which includes heavy hitters that are removed from the prefix list in earlier iterations because the encoding the data element was smaller than r. Algorithm 3 gives more details for the device-side algorithm. Recall that our local randomizer A_{priv} is one-hot encoding with asymmetric binary randomized response, see Appendix A of (McMillan et al., 2022) for more details on this choice of local randomizer. The prefix length l_{pref} is the length of the prefixes in the prefix list. More details on the server-side algorithm can be found in the next subsection.

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*
  $\mathcal{D}_t \leftarrow \mathcal{P}_{\text{prefixlist}_t} \times \{0,1\}^{l_t}$ // *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 $\mathcal{P}_{\text{prefixlist}_{T+1}} \cup \mathcal{P}_{\text{denylist}} \cup \mathcal{P}_{\text{discoveredlist}}$

C.1. Server-side algorithm

The general outline of the server-side algorithm is given in Algorithm 4. In this section we provide the details of the post-processing method on the server which determines the prefix list and next iteration segment length. We pass this noise

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

parameter, the frequency estimates and some extra parameters to `PruneHH` (Algorithm 5), which chooses heavy hitters based on some parameters set by the data analyst. In the simplest setting, one may simply define the heavy hitters as those with a frequency estimate greater than a fixed constant times the noise σ.

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 ration and η: Extra parameters for `PruneHH`
Output: $\mathcal{P}_{\text{prefixlist}}$: Heavy hitters list
$\widetilde{f}(\cdot) \leftarrow A_{\text{privagg}}(V_t, \mathcal{D}_t; \varepsilon_l)$ *// 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*
$\mathcal{P}_{\text{prefixlist}_{t+1}} \leftarrow$ `PruneHH`$(\mathcal{D}, \widetilde{f}(\mathcal{D}), C, FPR, \sigma, \eta)$
$\mathcal{P}_{\text{prefixlist}_{t+1}}, \mathcal{P}_{\text{discoveredlist}} \leftarrow$ `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 $\mathcal{P}_{\text{prefixlist}_{t+1}}, l_{t+1}, \mathcal{P}_{\text{discoveredlist}}$

Algorithm 5 describes the post-processing of the frequency estimator to choose the prefix list of the next round. First, E denotes the expected percentage of false positives. Based on the initialized confidence level (C), a threshold multiplier (τ) is computed based on the Gaussian approximation, where z_C denotes the C^{th} gaussian quantile, i.e. $z_C = \Phi^{-1}(C)$, where $\Phi()$ denotes the gaussian CDF. Then if the frequency of a discovered bin is above $\tau\sigma$, this means that with confidence C the discovered bin is a true positive because its count is above the noise level with probability C (line 4). In the while loop (line 5 to 9), we make sure that the ratio of the expected false positives to all the bins we keep in the prefix list does not go above a certain threshold. $len(\mathcal{D}) \times E$ represents the expected number of false positive bins. As explained before an end symbol is used to mark the end of stream. After extracting the prefix list, server checks if any of the prefixes already reached the end symbol. If a prefix reaches its end symbol it can be excluded from the prefix list sent to the devices in the next iteration. We append these finished prefixes to $\mathcal{P}_{\text{discoveredlist}}$ and form the prefix list of the next iteration ($\mathcal{P}_{\text{prefixlist}}$) by removing them (line 11). Finally, the next iteration segment length is defined based on the dimension constraint and size of the prefix list to ensure the total domain size does not go above dimension limit (line 12).

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

Input: $\mathcal{D}, \widetilde{f}[\mathcal{D}]$: query set and estimated frequencies after aggregation, C: Initialization of confidence level, FPR: ratio of expected false positives to the total number of bins, σ: aggregated noise standard deviation, η: step size
$E \leftarrow 1 - C$
$\tau \leftarrow z_C$
$\mathcal{P}_{\text{prefixlist}}{}' = \{q \in \mathcal{D} \mid \widetilde{f}[q] > \tau \times \sigma\}$
while $|\mathcal{P}_{\text{prefixlist}}{}'| < E \times len(\mathcal{D}) \times \frac{1}{FPR}$ **do**
  $E \leftarrow \eta * E$
  $C \leftarrow 1 - E$
  $\tau \leftarrow z_C$
  $\mathcal{P}_{\text{prefixlist}}{}' = \{q \in \mathcal{D} \mid \widetilde{f}[q] > \tau \times \sigma\}$
end while
$\mathcal{P}_{\text{prefixlist}}, \mathcal{P}_{\text{discoveredlist}} \leftarrow RemoveEnds(\mathcal{P}_{\text{prefixlist}}{}')$
$l \leftarrow \lfloor \log\left(\frac{P}{|\mathcal{P}_{\text{prefixlist}}|}\right) \rfloor$
return $\mathcal{P}_{\text{prefixlist}}, l$

D. 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 $e^{\varepsilon_l} + 1$, and provide guidelines for practitioners on how to choose one amongst the two frequency estimation algorithms based on the computation costs. Both the local randomizer (A_{priv}) and the frequency estimation (A_{privagg}) algorithms are parameterized by two constants α_0 and α_1 which can be chosen suitably to satisfy deletion ε_l - DP ($\alpha_0 = 1 - \alpha_1 = \frac{1}{e^{\varepsilon_l}+1}$) and replacement ε_l - DP ($\alpha_0 = \frac{1}{e^{\varepsilon_l}+1}, \alpha_1 = \frac{1}{2}$), 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 & Talwar, 2021)). We let F_1 denote the $\alpha_0 q$ smallest elements of the field and let bool(z) denote the indicator of the event $z \in F_1$. We can describe an affine function v over $\mathbb{F}_q^d$ using $d+1$ coefficients $v_0, v_1, \ldots, v_d$ and for any element $w \in \mathbb{F}_q^d$,we define the function as $v(w) = v_0 + \sum_{i \in [d]} v_i w_i$. Let $\mathcal{V}$ denote the family of functions defined by all $d+1$ tuples in $\mathbb{F}_q$. For each element $w \in \mathbb{F}_q^d$, and bit $b \in 0, 1$, we define the set of functions

$$\mathcal{V}_{w,b} \coloneqq \{v \in \mathcal{V} | \text{bool}(v(w)) = b\}. \tag{1}$$

Algorithm 6 describes the device side compressor using PI-RAPPOR. It takes ε_l, the local DP parameter, q, a prime power, d, the field dimension, Enc : $\mathbb{S} \to \mathbb{F}_q^d$, a mapping from data domain to field $\mathbb{F}_q^d$, s the data point to be privatized as inputs. We first convert the datapoint to be privatized and transmitted to an element in $\mathbb{F}_q^d$ using the mapping Enc. Then we sample $b \sim \text{Ber}(\alpha_1)$ and finally sample v uniformly from $\mathcal{V}_{w,b}$. This can be done by choosing $v_1, \ldots, v_d$ uniformly and randomly from $\mathbb{F}_q$ and choosing v_0 from the set $\{v_0 | v(w) = b\}$. Since this set consists of at most two contiguous ranges of integers, this sampling can be done in $O(\log q)$ time. The communication cost of this algorithm is $\lceil (d+1) \log_2 q \rceil$ bits.

Algorithm 6 PI-RAPPOR Local Randomizer

Input: ε_l: Local DP parameter, d: data point
Params: q: prime power, d: field dimension, Enc : $U \to \mathbb{F}_q^d$: mapping from data domain to field $\mathbb{F}_q^d$
Output: v: compressed and privatized version of d
Set $w = \text{Enc}(d)$
Sample $b \sim \text{Ber}(\alpha_1)$
Sample v uniformly from $V_{w,b}$ defined in (1)
Return v

Let $V = (v^1, v^2, \ldots, v^n)$ denote the collection of privatized responses collected from all the devices. For any element

$w \in \mathbb{F}_q^d$, we can find its estimated frequency using the following equation:

$$\tilde{f}[w] = \sum_{i \in [n]} \frac{\text{bool}(v^i(w)) - \alpha_0}{\alpha_1 - \alpha_0}. \tag{2}$$

Note that for user i, if v^i was generated using the field element w, we have $\text{bool}(v^i(w)) \sim \text{Ber}(\alpha_1)$ and if v^i was generated using any other field element, $\text{bool}(v^i(w)) \sim \text{Ber}(\alpha_0)$. Thus,

$$\tilde{f}[w] = \frac{\text{Bin}(f[w], \alpha_1) + \text{Bin}(n - f[w], \alpha_0) - n\alpha_0}{\alpha_1 - \alpha_0}, \tag{3}$$

where $f[w]$ is the true frequency of w. As shown in Lemma 4.2 of (Feldman & Talwar, 2021), $\tilde{f}$ is an unbiased estimator of f with the minimum possible variance for locally private estimators.

Algorithms 7 and 8 describe two possible implementations of the decompressor to calculate (2). Both of them take ε_l, the local DP parameter, Q, a set of elements of $\mathbb{F}_q^d$ of which we would like to know frequency estimates, and V, the collection of outputs of all users as inputs. In Algorithm 7, we compute the dot product for all outputs and all query vectors giving a computational complexity of the order $n|Q|$. In Algorithm 8 however, we try and reduce the number of dot products computed by first keeping a counter of values in V, and computing the dot products only for the unique values in the counter. Thus, the complexity of Algorithm 8 is of order $\sim |\text{Set}(V)||Q| + n$, where $|\text{Set}(V)|$ is the number of unique elements in V. Note that $|\text{Set}(V)| < \min\{n, q^{d+1}\}$ and Algorithm 8 runs meaningfully faster than Algorithm 7 only when $q^{d+1} < n$.

Algorithm 7 PI-RAPPOR Frequency Oracle - I

Input: ε_l: Local DP parameter, V: compressed user outputs, Q: query set
Output: $\tilde{f}[w]\ \forall\ w \in Q$: Private estimates of frequencies of elements in query set
Set $\tilde{f} = 0 \in \mathbb{R}^{|Q|}$
for $i \in [n]$ **do**
  for $w \in Q$ **do**
    $\tilde{f}[w] += \text{bool}(v_i(w))$
  end for
end for
$\tilde{f} = \frac{\tilde{f} - \alpha_0 n}{\alpha_1 - \alpha_0}$
Return $\tilde{f}$

Algorithm 8 PI-RAPPOR Frequency Oracle - II

Input: ε_l: Local DP parameter, V: compressed user outputs, Q: query set
Output: $\tilde{f}[w]\ \forall\ w \in Q$: Private estimates of frequencies of elements in query set
Set $\tilde{f} = 0 \in \mathbb{R}^{|Q|}$
for $v \in V$ **do**
  $n_v = 0$
end for
for $v \in V$ **do**
  $n_v += 1$
end for
for $v \in [V]$ **do**
  for $w \in Q$ **do**
    $\tilde{f}[w] += n_v * \text{bool}(v_i(w))$
  end for
end for
$\tilde{f} = \frac{\tilde{f} - \alpha_0 n}{\alpha_1 - \alpha_0}$
Return $\tilde{f}$

Speedups in the decompressor for prefix based algorithms: When the query set is a contiguous set of elements, we can choose the prime power q so that bool$(v(w))$ evaluates to 0 for most values of w and precisely calculate the values of w for which it evaluates to 1. This can potentially save up computation by a factor of roughly $\frac{1}{\alpha_0}$. We choose q to be the smallest prime power bigger than $K = \max\{e^{\varepsilon_l} + 1, 2^l\}$, where l denotes the segment length. Also, let $c = \lfloor \frac{q}{e^{\varepsilon_l}+1} \rfloor$ and denote $\{0, 1, \ldots, c\}$ by $[c]$. We describe the encoding for the two algorithms below:

1. **Noninteractive algorithm**: (no `prefix list` filtering on device side) Each datapoint (assume it to be a bitstring) of length r is divided into T segments of length l each. Each segment corresponds to a dimension in the field $\mathbb{F}_q$ and hence in iteration j, the dimension used by the compressor is j, and the mapping Enc can be defined by taking each chunk of size l and converting the bitstring to an integer. In each iteration, since we search over extensions of a given set of prefixes (denoted by $\mathcal{P}_{\text{prefixlist}}$), the query set is of the form $\mathcal{P}_{\text{prefixlist}} \times \mathbb{F}_q$.

2. **Interactive Algorithm**: (with `prefix list` filtering on device side) Each datapoint (assume it to be a bitstring) of length r is divided into T segments of length l each. In each iteration, the set of prefixes can be assigned to the first $\lceil \log_q |\mathcal{P}_{\text{prefixlist}}| \rceil$ dimensions and the last dimension can be the chunk we search over. The mapping Enc can be defined using the encoding for the prefixes which has already been communicated (since we can be interactive) and the bitstring for the new segment can be converted to an integer less than q. Thus, the query set for each iteration will also be of the form $\mathcal{P}_{\text{prefixlist}} \times \mathbb{F}_q$.

Given a query set of the form $\mathcal{P}_{\text{prefixlist}} \times \mathbb{F}_q$, we can improve the complexity of Algorithm 7 to $\sim n * c * |\mathcal{P}_{\text{prefixlist}}|$ and that of Algorithm 8 to $\sim |\text{Set}(V)|c|\mathcal{P}_{\text{prefixlist}}| + n$. Note that this is a roughly $e^{\varepsilon_l} + 1$-fold improvement in both the cases. To do this, instead of calculating the product $v^i(w) = v_0^i + \sum_{j\in[d]} v_j^i w_j$ for all $w \in Q$, 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 bool$(v^i(w))$ will evaluate to 1 and correspondingly increase the estimated frequency count for $w = (h, w_d)$. The inverse operation can be computed using a lookup table that can be calculated ahead of time. Thus, instead of searching over all elements in the query set of size $|Q|$, we loop over all prefixes and possible cutoffs, giving a size of $c|\mathcal{P}_{\text{prefixlist}}|$, giving an effective improvement of roughly $e^{\varepsilon_l} + 1$ (or $\frac{1}{\alpha_0}$) in the runtime.

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

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

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

We note that this assumption is satisfied by `OHE+2RR`, as well as other common frequency estimation algorithms such as PI-Rappor. For the purpose of this analysis, we will use a different metric to the metric that will be the main focus in our experimental results. We will measure the performance of `OptPrefixTree` using a metric we define in Definition E.2, which is standard in the differentially private heavy hitters literature (Bassily et al., 2017).

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

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

We now prove the utility of `OptPrefixTree` when we have one iteration with $l = r$ in Proposition E.3.

Proposition E.3. *Let the local randomizer (A_{priv}) and frequency estimator (A_{privagg}) pair satisfy Assumption E.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.*

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

Now, we prove that this algorithm is λ-accurate (Definition E.2) with $\lambda = C\sqrt{\frac{ne^{\varepsilon_l}}{(e^{\varepsilon_l}-1)^2}\log(\frac{|A|^r}{\beta})}$. 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 $\lambda = C\sqrt{\frac{ne^{\varepsilon_l}}{(e^{\varepsilon_l}-1)^2}\log(\frac{|A|^r}{\beta})}$.

□

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 F.2 which indicates sending one character at a time improves the utility. However, we conjecture that reducing the number of iterations per round, and including multiple rounds with deny-lists will improve utility.

F. Adaptive Segmentation Exploration

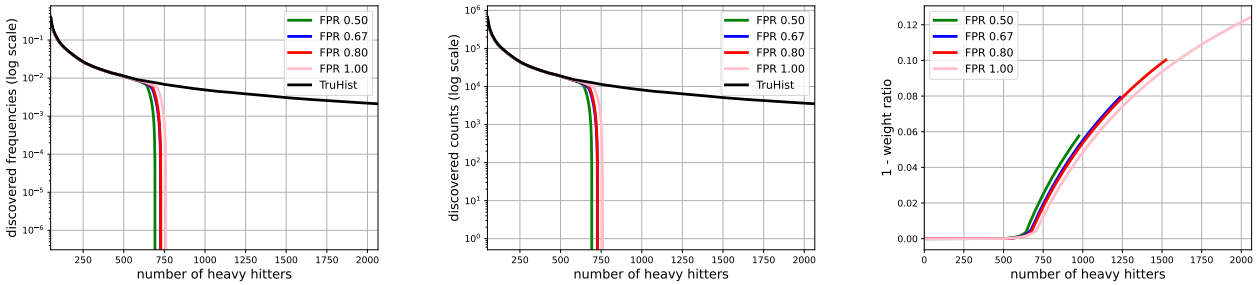
Here, in addition to showing discovered frequencies and discovered counts, we show another metric which we refer to as utility loss.

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) Marginal discovered frequencies (b) Marginal discovered counts (c) Total WR Loss

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

F.1. Adaptive Segmentation for Single Data Point

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

False Positives Ratio

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

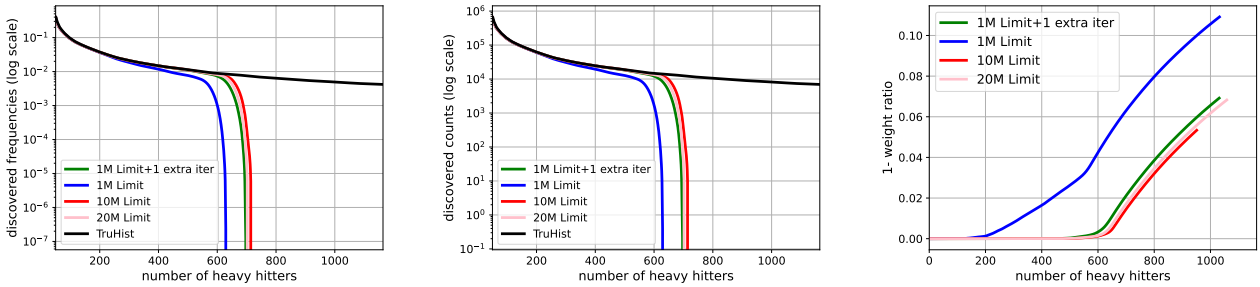
As observed increasing the FPR from 0.5 to 1 increases the number of heavy hitters detected slightly [660, 656, 678.0, 695] but increasing FPR also increases the number of discovered false positives significantly [357, 636, 828, 1400]. Based on Figure 3c, although lower FPR detects fewer true bins to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

Dimension Limitation

In production-scale systems the computation cost of decoding and aggregating the responses can be a significant bottleneck considering the high dimension of data. Thus, we tried to evaluate the effect of different dimension constraints on the performance of `OptPrefixTree`. In order to do so, we used different constraints of 20, 10, and 1 million for each device 's dimension in one iteration. This limitation in iteration t specifies $2^{l_{t+1}} \times |\mathcal{P}_{\text{prefixlist}_t}|$. First for all the configurations we set the number of iterations to 4. As illustrated in Figure 4c, 4b, and 4a changing the limit to 1 million degrades the utility of the algorithm significantly. 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 noise value we use for each iteration increases (check Table 2). However, with 5 rounds the algorithm is able to detect 15% more heavy hitters in comparison to the same dimension limit of 1 million when uses 4 iterations.

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 & Wang, 2019) for estimating the upper-bound of ε_c. The effect of different sub-sampling rates on the value of ε_l when $\varepsilon_c = 1$ and $N = 1.6 * (10)^6$ is shown in Table 3. As shown small sampling rate, increase the ε_l by adding to the randomness. However, as shown in the table, no sampling leads to higher ε_l in comparison to the moderate sampling rates ε_l. The reason is when



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

Figure 4. Effect of dimension limitation for single data point setting on `OptPrefixTree` utility ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$ 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 3. Sampling rate effect on the ε_l for $\varepsilon_{agg} = 1$ and $\delta = 10^{-6}$

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 5c, 5b, 5a 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.

F.2. Multiple Data point Adaptive Segmentation

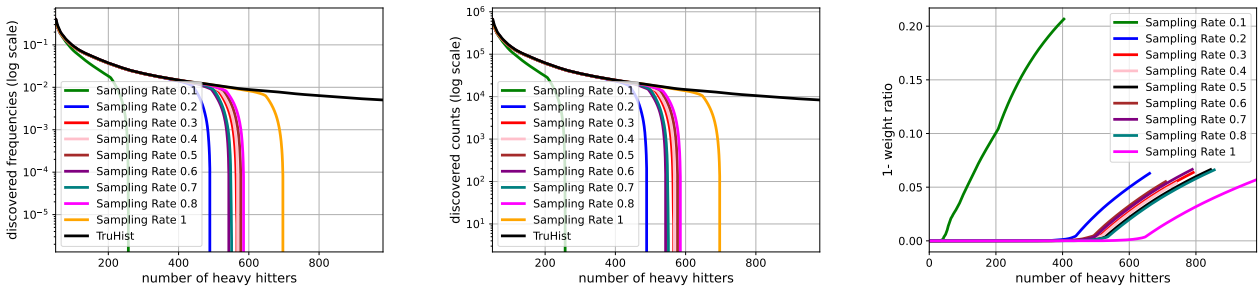
Segmentation Size

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

G. TrieHH

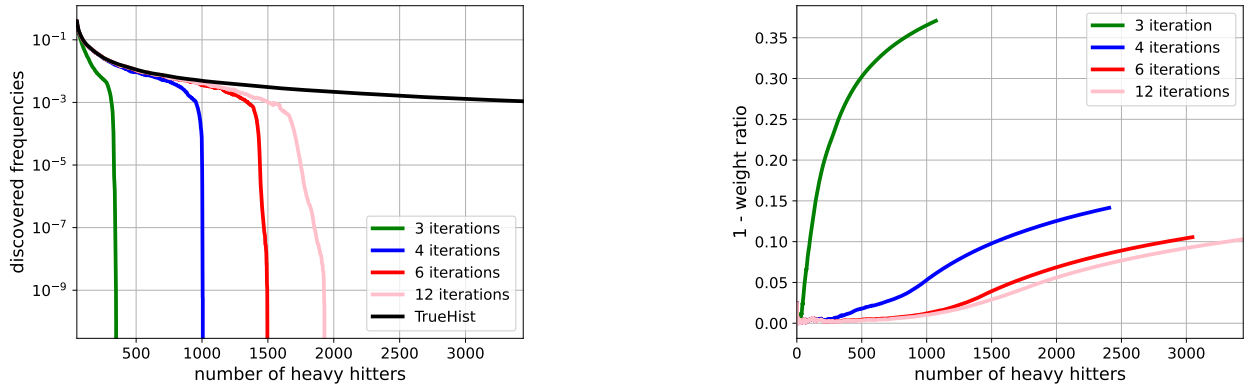
G.1. Single Data Point Setting for TrieHH

In this section we discuss the effect of number of iterations on the utility of a single data point setting for TrieHH (Zhu et al., 2020). In Figure 8a, we show the effect of different segmentation on the utility of the algorithm for a single data point per device setting. For these experiments we used $\varepsilon_{agg} = 1$ and sampling rates are set based on table 4. To evaluate the effect of



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

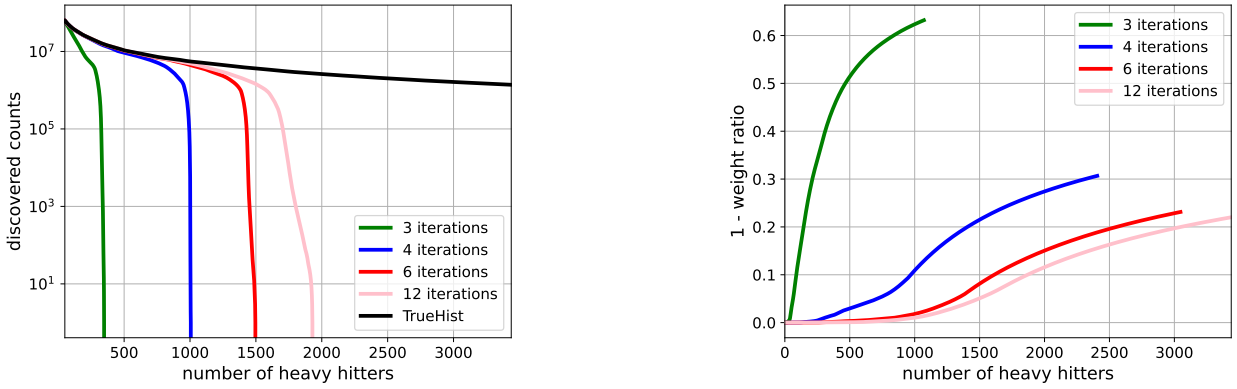
Figure 5. Effect of the different sampling rates for single data point setting on `OptPrefixTree` utility ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$ and $T = 4$)



(a) Marginal discovered frequencies (weighted metric)

(b) Total WR loss (weighted metric)

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



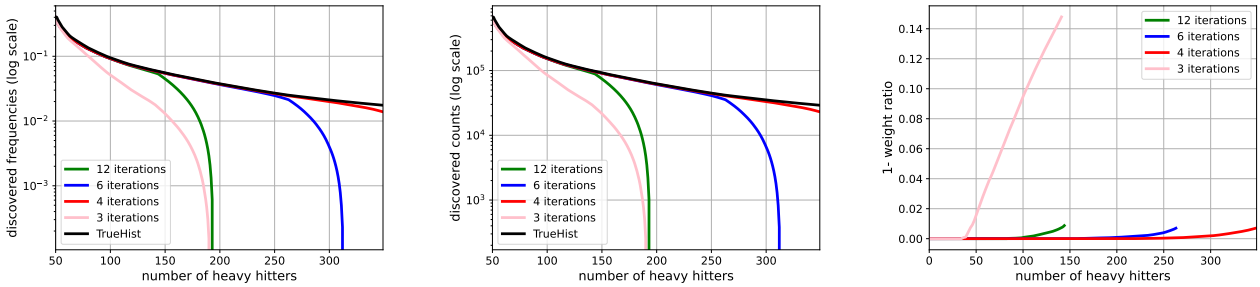
(a) Marginal discovered counts (unweighted metric)

(b) Total WR Loss (unweighted metric)

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

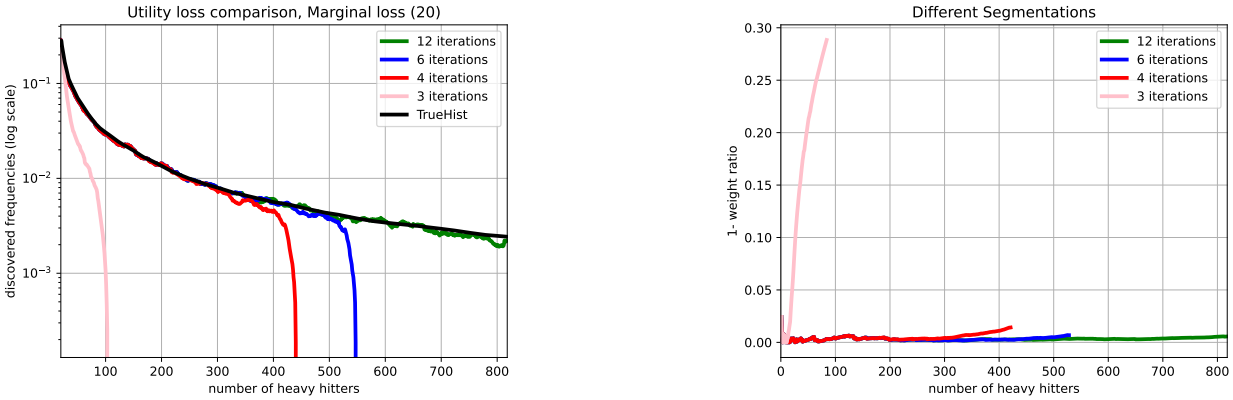
ε_{agg}	1				0.5				0.25			
T	12	6	4	3	12	6	4	3	12	6	4	3
Sampling Rate	0.0079	0.0153	0.0221	0.0283	0.0040	0.0079	0.0117	0.0153	0.0020	0.0040	0.0060	0.0079

Table 4. Number of rounds effect on the sampling rate of TrieHH



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

Figure 8. The effect of different number of iterations on TrieHH for single data point setting ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$)



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

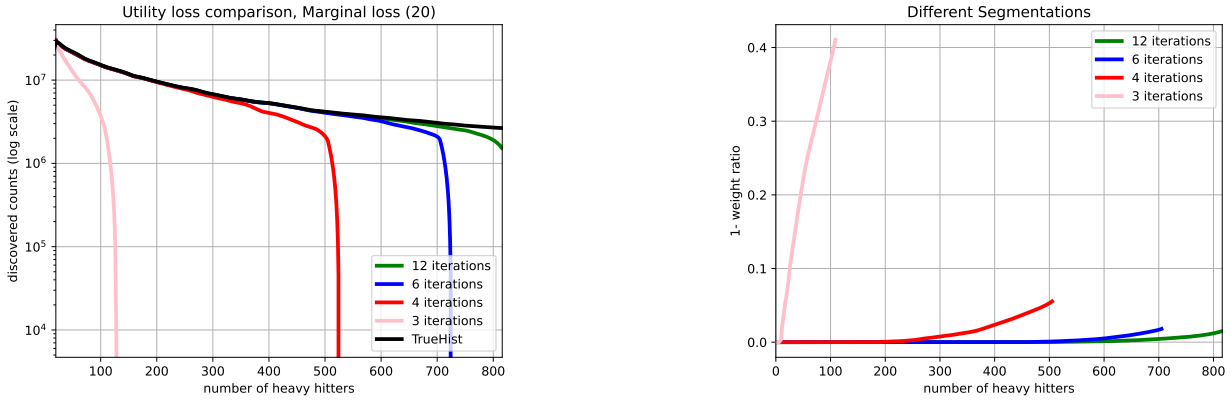
Figure 9. The effect of different number of iterations on TrieHH for multiple data points setting with weighted sampling($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$)

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.

G.2. Multiple Data Points Setting for TrieHH

We further analyze the multiple data points setting. To optimize the algorithm we took advantage of a prefix list for each iteration. Devices send their data only if they find a match with a prefix in the prefix list. We also use an end character symbol to indicate the end of string. If end character symbol is observed in a prefix at the end of an iteration, the corresponding prefix will be excluded from the prefix list. Therefore, users can send other unfinished prefixes. Also, for our evaluation, we used binary encoding which uses 5 bits to represent each character. The total number of heavy hitters detected by TrieHH algorithm, when the number of iterations are 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char), are $[816, 706, 506, 110]$ respectively. In this setting, 12 iterations shows the best utility. In Figure 9a and 9b we used weighted sampling described in the original paper.

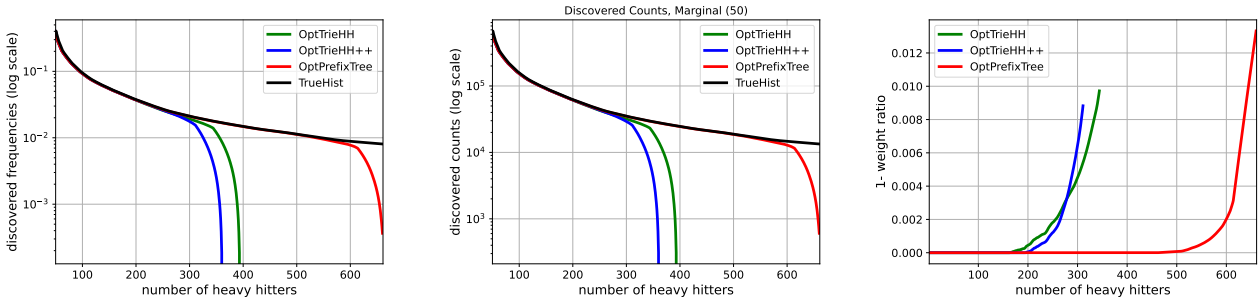
To further improve the utility of TrieHH, we used unweighted sampling in another set of experiments. This new sampling scheme leads to finding $[816, 529, 422, 85]$ heavy hitters when having 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char) iterations respectively. Figure 10a and 10b shows the loss and marginal discovered counts based on the unweighted sampling scheme.



(a) Marginal discovered counts

(b) Total WR loss

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



(a) Marginal discovered frequencies

(b) Marginal discovered counts

(c) Total WR loss

Figure 11. Utility comparison between `OptTrieHH` and `OptPrefixTree` for single data point setting($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)

As demonstrated in these figures, unweighted scheme is able to find more heavy hitters and cause less utility degradation. Also using both sampling schemes, 12 iterations shows the highest utility. Thus, for the comparisons with `OptPrefixTree` we use unweighted sampling, prefix-list and 12 iterations for `OptTrieHH`.

G.3. Comparison of `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 this section we showed the effect of different number of iterations on the sampling rate and utility of TrieHH. In this set of experiments we used 4 iterations that shows the highest utility for TrieHH based on the dimension limitations ($P = 10^7$). We refer to this optimized version of TrieHH as `OptTrieHH`.

To have a fair comparison, we use the same binary encoding for both `OptPrefixTree` and `OptTrieHH`. 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 11a 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` by finding $1.85X$ more heavy hitters. Figure 11b shows the same plot but in the y-axis we have the marginal counts of discovered bins. Also, Figure 11c shows the total utility loss.

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

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

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

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

H. TrieHH++

Based on Lemma 3 of (Cormode & 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 5 and 6.

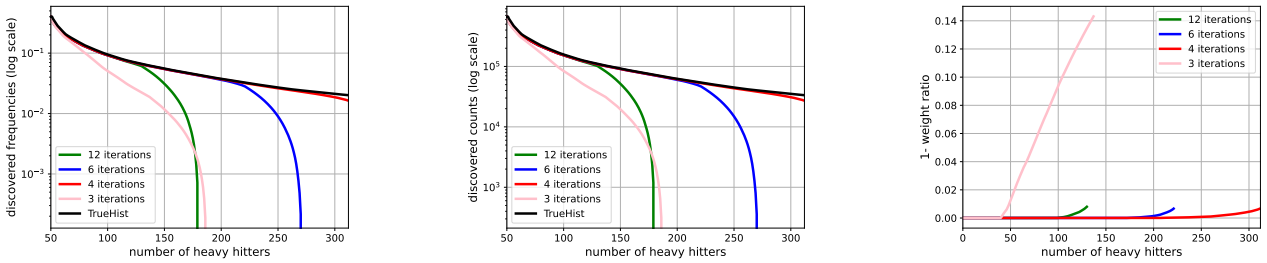
H.1. Single Data Point Setting for `TrieHH++`

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

H.2. Multiple Data Points Setting for `TrieHH++`

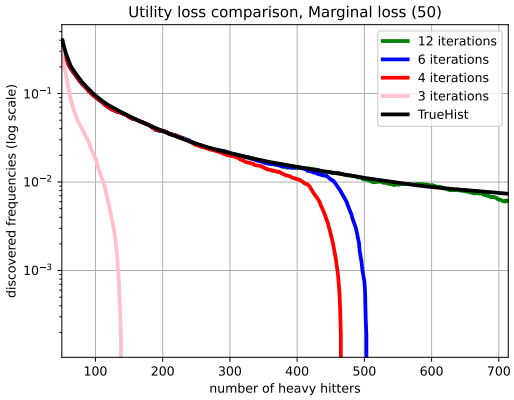
For our evaluation, we used the same binary encoding we described before. The total number of heavy hitters detected by `TrieHH++` algorithm, when the number of iterations are 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char), are $[714, 455, 417, 90]$ respectively. As shown in Figure 13, in this setting, 12 iterations shows the best utility.

To further improve the utility of `TrieHH++`, we used unweighted sampling in another set of experiments. This sampling scheme leads to finding $[704, 610, 484, 102]$ heavy hitters when having 12 (1 char), 6 (2 char), 4 (3 char), 3 (4 char) iterations

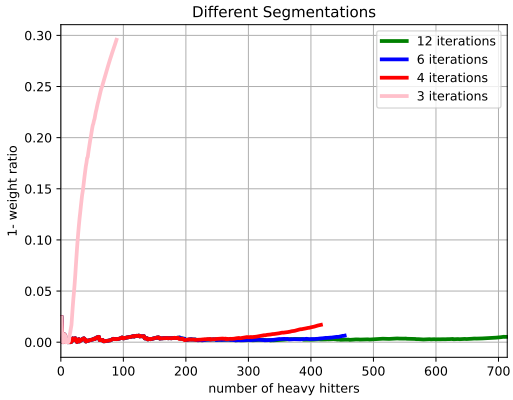


(a) Marginal discovered frequencies (b) Marginal discovered counts (c) Total WR loss

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

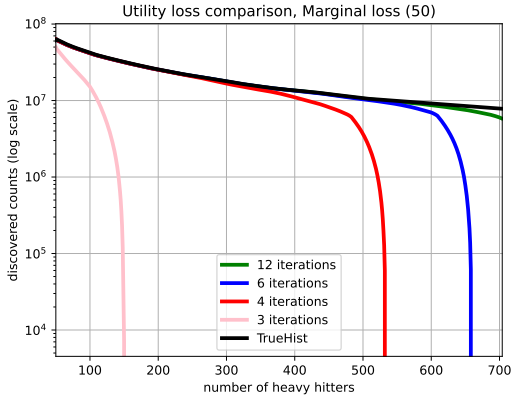


(a) Marginal discovered frequencies

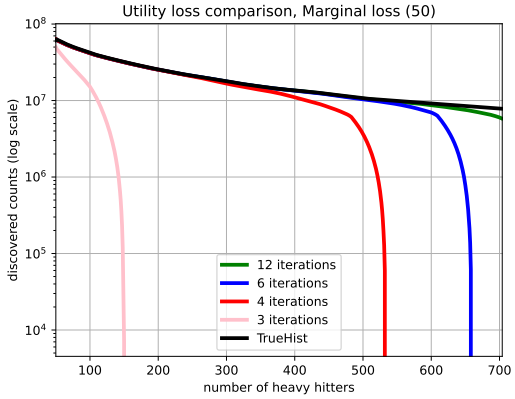


(b) Total WR loss

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



(a) Marginal discovered counts



(b) Total WR loss

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

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 `TrieHH++` and we refer to it as `OptTrieHH++`.

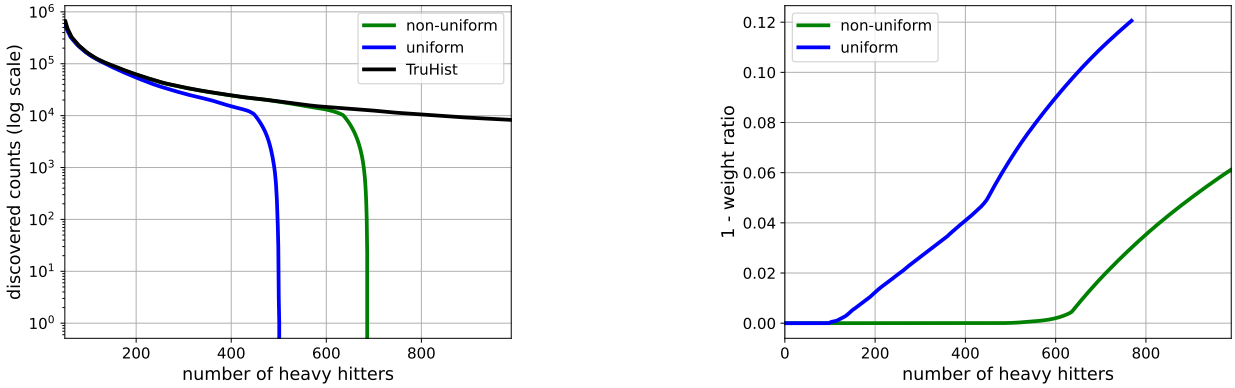
I. Additional Experimental Results

Effect of Uniform vs Non-uniform Segmentation in Single Data Point Setting

As explained in 4 using our adaptive segmentation algorithm helps discovering more heavy hitters. In Figures 15a and 15b we show the discovered counts and total utility loss comparison of this uniform and non-uniform segmentation.

Effect of Data Selection in Multiple Data Points Setting

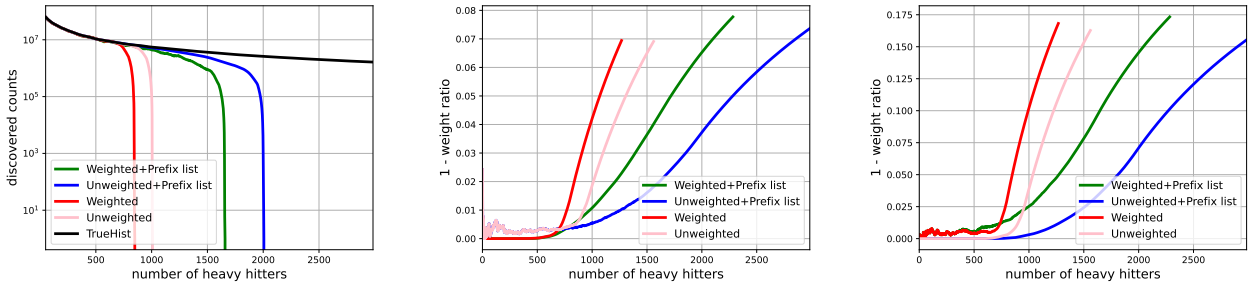
There are different ways to measure the frequency of data points. In Section 3, we discuss how averaging the distribution of words overall devices can be used to define the global frequency of words (weighted metric). Figure 16b shows the total utility loss when using this distribution.



(a) Marginal discovered counts comparison

(b) Total WR loss

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



(a) Marginal discovered counts

(b) Total WR loss (weighted metric)

(c) Total WR loss (unweighted metric)

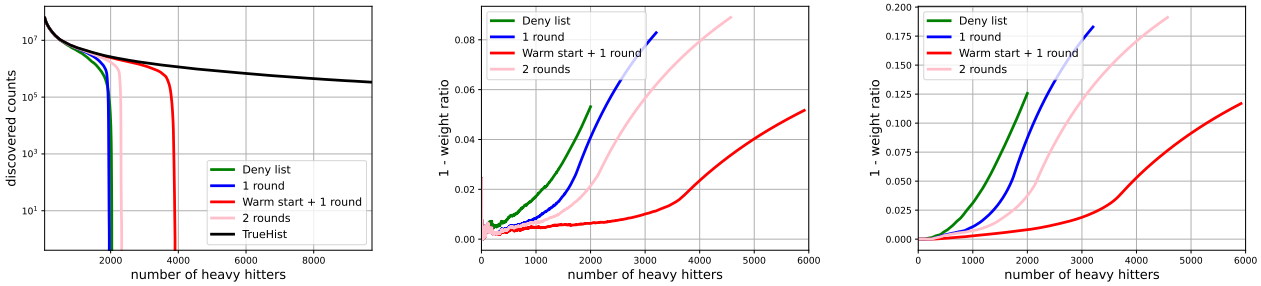
Figure 16. Comparing different data selection schemes in `OptPrefixTree` for multiple data points ($\varepsilon_{agg} = 1$, $\delta = 10^{-6}$, $T = 4$)

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

Effect of adding a `deny list` in Multiple Data Points Setting

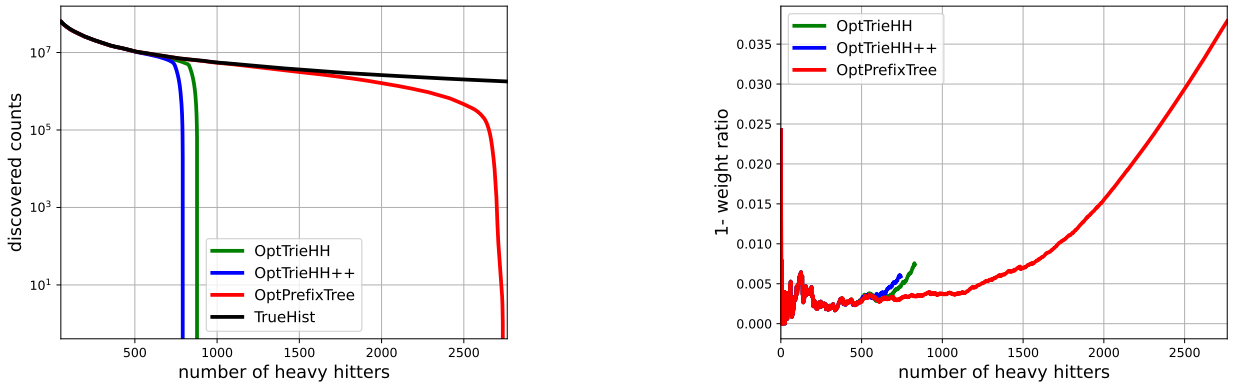
Figure 17b shows the total utility loss when using the frequency of the words in devices for extracting the global distribution. In Figure 17a and 17c we use the number of devices with the word to demonstrate the true counts of the discovered words and total utility loss.

Comparison with previous works As illustrated in section 4 `OptPrefixTree` outperforms `OptTrieHH` under the same constraints. In Figure 18a and 18b, we show a comparison of `OptPrefixTree` and `OptTrieHH` for the multiple data points setting.



(a) Marginal discovered counts (b) Total WR loss (weighted metric) (c) Total WR loss (unweighted metric)

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



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

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