Learning with Privacy at Scale

Differential Privacy Team, Apple

Abstract

Understanding how people use their devices often helps in improving the user experience. However, accessing the data that provides such insights — for example, what users type on their keyboards and the websites they visit — can compromise user privacy. We design a system architecture that enables learning at scale by leveraging local differential privacy, combined with existing privacy best practices. We develop efficient and scalable local differentially private algorithms and provide rigorous analyses to demonstrate the tradeoffs among utility, privacy, server computation, and device bandwidth. Understanding the balance among these factors leads us to a successful practical deployment using local differential privacy. This deployment scales to hundreds of millions of users across a variety of use cases, such as identifying popular emojis, popular health data types, and media playback preferences in Safari.

1 Introduction

Gaining insight into the overall user population is crucial to improving the user experience. For example, what new words are trending? Which health categories are most popular with users? Which websites cause excessive energy drain? The data needed to derive such insights is personal and sensitive, and must be kept private. In addition to privacy concerns, practical deployments of learning systems using this data must also consider resource overhead, computation costs, and communication costs. In this paper, we describe a system architecture that combines differential privacy and privacy best practices to learn from a user population addressing both privacy and practical deployment concerns.

Differential privacy [6] provides a mathematically rigorous definition of privacy and is one of the strongest guarantees of privacy available. Within the differential privacy framework, there are two settings: *central* and *local*. Here, we choose not to collect raw data on the server which is required for central differential privacy; hence, we adopt local differential privacy, which is a superior form of privacy [8]. Local differential privacy has the advantage that the data is randomized before being sent from the device, so the server never sees or receives raw data.

Our system is designed to be opt-in and transparent. No data is recorded or transmitted before the user explicitly chooses to report usage information. Data is privatized on the user's device using event-level differential privacy [7] in the local model where an event might be, for example, a user typing an emoji. Additionally, we restrict the number of transmitted privatized events per use case. The transmission to the server occurs over an encrypted channel once per day, with no device identifiers. The records arrive on a restricted-access server where IP identifiers are immediately discarded, and any association between multiple records is also discarded. At this point, we cannot distinguish, for example, if an emoji record and a Safari web domain record came from the same user, or if two emoji records came from the same user. The records are processed to compute relevant statistics. These aggregate statistics are then shared internally with the relevant teams at Apple.

We focus on the problem of estimating frequencies of elements — for example, emojis and web domains — in the system architecture described above. In estimating frequencies of elements, we consider two subproblems. In the first, we compute the histogram from a *known* dictionary of elements. In the second, the dictionary is *unknown* and we want to obtain a list of the most frequent elements in a dataset.

Our main contributions are as follows:

1. We design a system that enables learning at scale by leveraging local differential privacy, combined with existing privacy best practices.
2. We design novel local differentially private algorithms for frequency estimation of a dataset in both the known and unknown dictionary settings.
3. We prove analytic expressions for the tradeoffs among various factors, including privacy, utility, server computation overhead, and device bandwidth. These expressions are crucial for making design decisions in a real-world deployment.
4. We deploy our algorithms on hundreds of millions of devices and present results in a variety of use cases.

This work provides new insights into the aggregate behavior of users, while preserving privacy, which can be used to improve the user experience.

2 Related Work

Differential privacy [6] began in the cryptography and theoretical computer science communities over a decade ago and has become the privacy benchmark across a variety of fields [8]. Most research on differential privacy considers the *central* setting, in which a trusted entity stores all the raw data and then releases aggregate statistics in a differentially private way. This paper focuses on the stronger privacy type — *local* differential privacy.

Differential privacy in the local setting is addressed in a number of papers, some of which obtain tight bounds for statistical estimators [5], address Probably-Approximately-Correct (PAC) learning problems [13], and study frequency estimation [3, 9, 10]. Bassily and Smith [3] presented a local differentially private algorithm for obtaining the set of the most frequent items in a dataset (the *heavy hitters*) and a frequency oracle of counts from a large domain $\mathcal{D}$ that achieves theoretically tight error bounds with low computation and communication cost between the server and client. Prior to [3], a series of works [16, 12] attempted to reduce the time and space complexity while preserving the optimal error, but suffered running time with a polynomial dependence on the domain size $|\mathcal{D}|$. The utility analysis in these prior works sets the parameters in a way that achieves the asymptotically optimal utility in the worst case, but if the parameters are not set in the prescribed way then utility is not guaranteed to be optimal. Thus, if the server computation or device bandwidth can be increased, doing so does not necessarily lead to better utility.

There have been a few practical deployments of differentially private systems. Unlike our system, most of these employ central differential privacy, including the PSI (Ψ) system [11], work being done at the Census Bureau [14, 2], Airavat [19], PINQ [15], and GUPT [17]. In the local setting, RAPPOR deploys differential privacy on the Chrome web browser [9, 10]. In this paper, we describe the end-to-end details of one of the first large scale deployments of differential privacy. Additionally, we design algorithms and prove expressions for how to balance the tradeoffs among privacy, accuracy, transmission cost, and computation cost. These analytic expressions are a major contribution of our paper and allow us to trade off these parameters in each of our use cases. Without such expressions, it is difficult to evaluate the impact on accuracy if, for example, transmission cost is reduced, without running costly iterations. In contrast to RAPPOR, our utility theorems give a principled way to choose these algorithmic parameters.

Further, to keep transmission costs to an absolute minimum, our Hadamard Count Mean Sketch algorithm can obtain accurate counts when each user sends only a single privatized bit. We describe these algorithms in more detail in Section 4.

3 Background and System Overview

Consider a data universe $\mathcal{D}$ which has p elements and is indexed by $[p] = \{1, \cdots, p\}$. For record $i \in [n]$ where n is the total number of records, denote its data entry as $d^{(i)} \in \mathcal{D}$. Vectors $\boldsymbol{v} = (v_1, \cdots, v_m)$ are denoted in bold face. We focus on estimating frequencies of data elements in a local differentially private way. More formally, let $f : \mathcal{D} \to \mathbb{R}$ be an oracle that gives the frequency of an element in $\mathcal{D}$ from a given dataset $\{d^{(1)}, \cdots, d^{(n)}\}$. We design a local differentially private algorithm $\mathcal{A}$ that outputs an oracle $\tilde{f} : \mathcal{D} \to \mathbb{R}$ which provides an unbiased estimate of $f(d)$, that is $\forall d \in \mathcal{D}, \mathbb{E}\left[\tilde{f}(d)\right] = f(d)$, and has small variance. We next cover the definition of local differential privacy.

3.1 Differential Privacy

We first define a local randomizer which we will use in the definition of local differential privacy.

Definition 3.1 (Local Randomizer [8, 13]). *Let $\mathcal{A} : \mathcal{D} \to \mathcal{Y}$ be a randomized algorithm mapping a data entry in $\mathcal{D}$ to $\mathcal{Y}$. The algorithm $\mathcal{A}$ is an ϵ-local randomizer if for all data entries $d, d' \in \mathcal{D}$ and all outputs $y \in \mathcal{Y}$, we have*

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

The privacy parameter ϵ captures the *privacy loss* consumed by the output of the algorithm; if $\epsilon = 0$ then we are ensuring perfect privacy in which the output is independent of its input, while $\epsilon = \infty$ gives no privacy guarantee.

In general, differential privacy is defined for algorithms with input databases of size larger than 1. In the local model of differential privacy, algorithms may only access the data via the output of a local randomizer so that no raw data is stored on a server. Formally:

Definition 3.2 (Local Differential Privacy [8]). *Let $\mathcal{A} : \mathcal{D}^n \to \mathcal{Z}$ 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 each $\mathcal{A}_i : \mathcal{D} \to \mathcal{Y}$ is an ϵ-local randomizer 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.*

3.2 System Architecture

Our system architecture consists of device- and server-side data processing. On the device, the *privatization* stage ensures raw data is made differentially private. The restricted-access server does data processing that can be further divided into the *ingestion* and *aggregation* stages. We explain each stage in detail below.

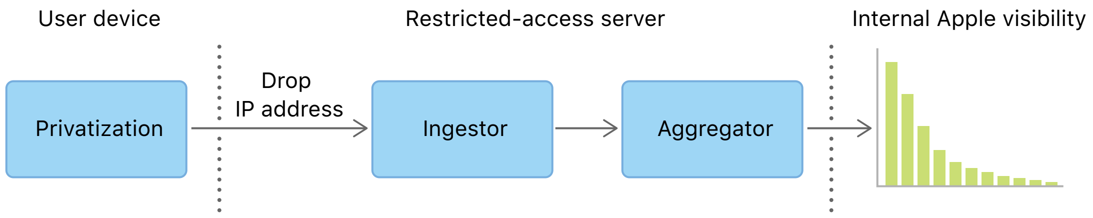


Figure 1: System Overview

3.2.1 Privatization

Users have the option, in System Preferences on macOS or in Settings on iOS, to share privatized records for analytics. For users who do not opt in, the system remains inactive. For users who do opt in, we define a per-event privacy parameter, ϵ. Additionally, we set a limit on the number of privatized records that can be transmitted daily for each use case. Our choice of ϵ is based on the privacy characteristics of the underlying dataset for each use case. These values are consistent with the parameters proposed in the differential privacy research community, such as [10, 18]. Moreover, the algorithms we present in Section 4 provide further deniability due to hash collisions. We provide additional privacy by removing user identifiers and IP addresses at the server where the records are separated by use case so that there is no association between multiple records as explained in Section 3.2.2 below.

Whenever an event is generated on device (for example, when a user types an emoji), the data is immediately privatized via ϵ-local differential privacy and temporarily stored on-device using data protection [1], rather than being immediately transmitted to the server. After a delay based on device conditions, the system randomly samples from the differentially private records subject to the above limit and sends the sampled records to the server. These records do not include device identifiers or timestamps of when events were generated. The communication between a device and the server is encrypted using TLS. See Figure 2 for an overview.

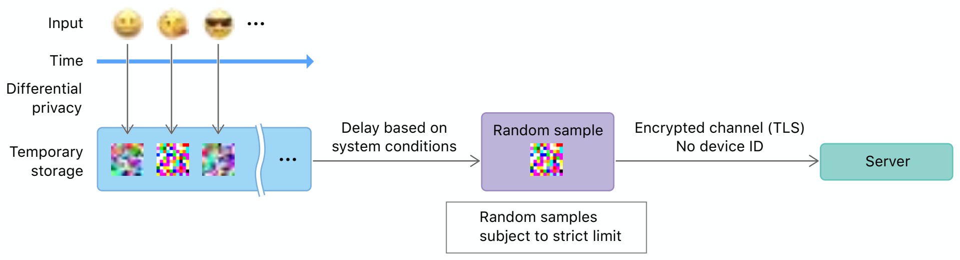
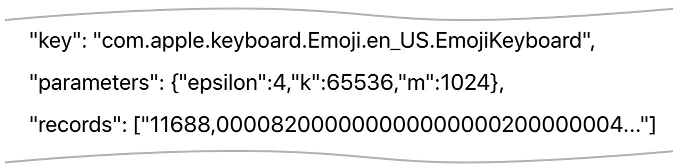


Figure 2: Privatization Stage

On iOS, the reports are visible in Settings > Privacy > Analytics > Analytics Data, in entries that begin with “DifferentialPrivacy." On macOS, these records are visible in Console, in System Reports. Figure 3 is a sample record of our algorithm for the Popular Emojis use case. The record lists algorithmic parameters, which are discussed in Section 4.1, and the privatized data entry. The privatized data entry is represented as a hexadecimal string. Note that the privatized data is elided here for presentation; the full size in this example is 128 bytes.



```
"key": "com.apple.keyboard.Emoji.en_US.EmojiKeyboard",
"parameters": {"epsilon":4,"k":65536,"m":1024},
"records": ["11688,000082000000000000000200000004..."]
```

Figure 3: Sample Report with Privatized Record

3.2.2 Ingestion and Aggregation

The privatized records are first stripped of their IP addresses prior to entering the ingestor. The ingestor then collects the data from all users and processes them in a batch. The batch process removes metadata, such as the timestamps of privatized records received, and separates these records based on their use case. The ingestor also randomly permutes the ordering of privatized records within each use case before forwarding the output to the next stage.

The aggregator takes the private records from the ingestor and for each use case generates a differentially private histogram according to the algorithms described in Section 4. The data from multiple use cases is never combined. In these histograms, only domain elements whose counts are above a prescribed threshold T are included. These histograms are then shared internally with relevant teams at Apple.

4 Algorithms

Of relevance to our algorithms is the *count sketch* algorithm [4] which finds frequently reported data elements along with accurate counts from a stream. We also use a sketch matrix data structure to compute counts for a collection of privatized domain elements. However, to ensure differential privacy, our algorithms deviate significantly. We next present our algorithms in depth.

4.1 Private Count Mean Sketch

We develop a local differentially private algorithm that outputs a histogram of counts over domain $\mathcal{D}$ for a dataset consisting of n records. At a high level, Count Mean Sketch (CMS) is composed of a client-side algorithm and a server-side algorithm. The client-side algorithm ensures that the data that leaves the user's device is ϵ-differentially private. Further, the client-side algorithm ensures the transmission is of reasonable size m by mapping the domain element d with one of k hash functions. The data structure used on the server to aggregate the privatized data is a sketch matrix M of dimension $k \times m$. To compute an estimate for domain element $d \in \mathcal{D}$, the server-side algorithm then averages the counts corresponding to each of the k hash functions in M.

The full protocol CMS is presented in Algorithm 1, which outputs a histogram of counts over a given dictionary $\hat{\mathcal{D}} \subseteq \mathcal{D}$ of domain elements. It first selects independently and uniformly a set of k three-wise independent hash functions uniformly at random, denoted as $\mathcal{H} = \{h_j : \mathcal{D} \to [m] : j \in [k]\}$. This set of hash functions is then shared between the client-side and server-side algorithms. We conclude the section with the utility analysis as a function of the parameters ϵ, m, and k.

Algorithm 1 Count Mean Sketch CMS

```
Require: $d^{(1)}, \cdots, d^{(n)} \in \mathcal{D}^n$; $\epsilon$, $k$, $m$, dictionary $\hat{\mathcal{D}} \subseteq \mathcal{D}$.
1. From a set of three-wise independent hashes mapping $\mathcal{D}$ to $[m]$, select a set $\mathcal{H} = \{h_1, \cdots, h_k\}$ of $k$ uniformly at random.
2. for $i \in [n]$ do
3.     $(\tilde{\boldsymbol{v}}^{(i)}, j^{(i)}) \leftarrow \mathcal{A}_{\mathsf{client-CMS}}(d^{(i)}; \epsilon, \mathcal{H})$
4. Construct the sketch $\mathrm{M} \leftarrow \texttt{Sketch-CMS}\left((\tilde{\boldsymbol{v}}^{(1)}, j^{(1)}), \cdots, (\tilde{\boldsymbol{v}}^{(n)}, j^{(n)}); \epsilon, k, m\right)$
5. for $d \in \hat{\mathcal{D}}$ do
6.     $\tilde{f}(d) \leftarrow \mathcal{A}_{\mathsf{server}}(d; \mathrm{M}, \epsilon, \mathcal{H})$
7. return Histogram $\left(\tilde{f}(d) : d \in \hat{\mathcal{D}}\right)$.
```

4.1.1 Client-Side Algorithm

Given a privacy parameter $\epsilon > 0$ and data entry $d \in \mathcal{D}$, the *client-side algorithm* $\mathcal{A}_{\mathsf{client-CMS}} : \mathcal{D} \to \{-1, 1\}^m$ selects an index j uniformly at random from $[k]$, which corresponds to hash function $h_j \in \mathcal{H}$ where $h_j : \mathcal{D} \to [m]$. Next, it sets the encoding vector $\boldsymbol{v} \in \{-1, 1\}^m$ to be 1 in position $h_j(d)$ and -1 everywhere else. Finally, each bit of $\boldsymbol{v}$ is flipped with probability $\frac{1}{1+e^{\epsilon/2}}$ and with the remaining probability, the bit remains unchanged. This privatized vector $\tilde{\boldsymbol{v}}$ along with the index j is what is sent to the server.

Algorithm 2 Client-Side $\mathcal{A}_{\mathsf{client-CMS}}$

```
Require: Data element: $d \in \mathcal{D}$; $\epsilon$, $\mathcal{H}$.
1. Sample $j$ uniformly at random from $[k]$.
2. Initialize a vector $\boldsymbol{v} \leftarrow -\mathbf{1} \in \mathbb{R}^m$.
3. Set $v_{h_j(d)} \leftarrow 1$.
4. Sample $\boldsymbol{b} \in \{-1, +1\}^m$, where each $b_\ell$ is i.i.d. where $\Pr[b_\ell = +1] = \frac{e^{\epsilon/2}}{e^{\epsilon/2}+1}$.
5. $\tilde{\boldsymbol{v}} \leftarrow (v_1 b_1, \cdots, v_m b_m)$.
6. return $\tilde{\boldsymbol{v}}$, index $j$.
```

The privacy analysis follows from showing that $\mathcal{A}_{\mathsf{client-CMS}}$ is an ϵ-local randomizer, since the server merely implements a post-processing function on the privatized data entries from the clients. Hence, ensuring that $\mathcal{A}_{\mathsf{client-CMS}}$ is an ϵ-local randomizer results in CMS being ϵ-local differentially private.

Theorem 4.1 (Privacy guarantee). *Algorithm $\mathcal{A}_{\mathsf{client-CMS}}$ is an ϵ-local randomizer for any $\mathcal{H}$. Further, CMS is ϵ-local differentially private.*

Proof. We show for an arbitrary output $\tilde{\boldsymbol{v}} \in \{-1, 1\}^m$, the probability of observing that the output is similar whether the user reported data d or d'. Let J be the random variable that selects uniformly from $[k]$ and B_ℓ be the random variable for the biased bit b_ℓ in $\mathcal{A}_{\mathsf{client-CMS}}$. Given a $j \in [k]$, we write the corresponding one-hot vector $\boldsymbol{v}$ and $\boldsymbol{v}'$, respectively for data entry d and d'.

$$
\begin{aligned}
\frac{\Pr[\mathcal{A}_{\mathsf{client-CMS}}(d; \epsilon, \mathcal{H}) = (\tilde{\boldsymbol{v}}, j)]}{\Pr[\mathcal{A}_{\mathsf{client-CMS}}(d'; \epsilon, \mathcal{H}) = (\tilde{\boldsymbol{v}}, j)]} &= \frac{\Pr[J = j] \prod_{\ell=1}^m \Pr[v_\ell B_\ell = \tilde{v}_\ell \mid J = j]}{\Pr[J = j] \prod_{\ell'=1}^m \Pr\left[v'_{\ell'} B_{\ell'} = \tilde{v}_{\ell'} \mid J = j\right]} \\
&= \prod_{\ell=1}^m \frac{\Pr[v_\ell B_\ell = \tilde{v}_\ell \mid J = j]}{\Pr\left[v'_\ell B_\ell = \tilde{v}_\ell \mid J = j\right]} \qquad (1)
\end{aligned}
$$

Note that the two vectors $\boldsymbol{v}, \boldsymbol{v}' \in \{-1,1\}^m$ can only differ in at most two locations, corresponding to $v_{h_j(d)} = 1$ and $v_\ell = -1$ for all $\ell \neq h_j(d)$, while $v_{h_j(d')} = 1$ and $v_\ell = -1$ for all $\ell \neq h_j(d')$. We can then simplify the ratio in (1) because all terms $v_\ell, v'_\ell = -1$ for $\ell \notin \{h_j(d), h_j(d')\}$. We first assume the case where $h_j(d) = h_j(d')$,

$$\prod_{\ell=1}^{m} \frac{\Pr\left[v_\ell B_\ell = \tilde{v}_\ell \mid J = j\right]}{\Pr\left[v'_\ell B_\ell = \tilde{v}_\ell \mid J = j\right]} = \frac{\Pr\left[v_{h_j(d)} B_{h_j(d)} = \tilde{v}_{h_j(d)} \mid J = j\right]}{\Pr\left[v'_{h_j(d)} B_{h_j(d)} = \tilde{v}_{h_j(d)} \mid J = j\right]}.$$

In this case we have $v_{h_j(d)} = v'_{h_j(d)} = 1$, so that the ratio is one.

We next assume that $h_j(d) \neq h_j(d')$,

$$\prod_{\ell=1}^{m} \frac{\Pr\left[v_\ell B_\ell = \tilde{v}_\ell \mid J = j\right]}{\Pr\left[v'_\ell B_\ell = \tilde{v}_\ell \mid J = j\right]} = \frac{\Pr\left[v_{h_j(d)} B_{h_j(d)} = \tilde{v}_{h_j(d)} \mid J = j\right]}{\Pr\left[v'_{h_j(d)} B_{h_j(d)} = \tilde{v}_{h_j(d)} \mid J = j\right]} \frac{\Pr\left[v_{h_j(d')} B_{h_j(d')} = \tilde{v}_{h_j(d')} \mid J = j\right]}{\Pr\left[v'_{h_j(d')} B_{h_j(d')} = \tilde{v}_{h_j(d')} \mid J = j\right]}$$

Recall, that we have $\Pr\left[B_\ell = 1\right] = \frac{e^{\epsilon/2}}{1+e^{\epsilon/2}}$, and $\Pr\left[B_\ell = -1\right] = \frac{1}{1+e^{\epsilon/2}}$ for each $\ell \in [m]$. Hence, we can bound the total product

$$e^{-\epsilon} \leq \frac{\Pr\left[v_{h_j(d)} B_{h_j(d)} = \tilde{v}_{h_j(d)} \mid J = j\right]}{\Pr\left[v'_{h_j(d)} B_{h_j(d)} = \tilde{v}_{h_j(d)} \mid J = j\right]} \frac{\Pr\left[v_{h_j(d')} B_{h_j(d')} = \tilde{v}_{h_j(d')} \mid J = j\right]}{\Pr\left[v'_{h_j(d')} B_{h_j(d')} = \tilde{v}_{h_j(d')} \mid J = j\right]} \leq e^{\epsilon}.$$

In either case, we have our privacy guarantee:

$$-\epsilon \leq \ln\left(\frac{\Pr\left[\mathcal{A}_{\mathsf{client-CMS}}(d; \epsilon, \mathcal{H}) = (\tilde{\boldsymbol{v}}, j)\right]}{\Pr\left[\mathcal{A}_{\mathsf{client-CMS}}(d'; \epsilon, \mathcal{H}) = (\tilde{\boldsymbol{v}}, j)\right]}\right) \leq \epsilon.$$

Because `CMS` is a post-processing function of the outputs from $\mathcal{A}_{\mathsf{client-CMS}}$, we then have that `CMS` is ϵ-local differentially private. □

4.1.2 Server-Side Algorithm

The server-side algorithm $\mathcal{A}_{\mathsf{server}}$, given in Algorithm 4, has several steps before it outputs the counts for any data entry in $\mathcal{D}$, which we will go over in turn. First, it takes each record $\tilde{\boldsymbol{v}}^{(i)}$ and transforms it to $\tilde{\boldsymbol{x}}^{(i)} = k \cdot \left(\frac{c_\epsilon}{2}\tilde{\boldsymbol{v}}^{(i)} + \frac{1}{2}\mathbf{1}\right)$ where $c_\epsilon = \frac{e^{\epsilon/2}+1}{e^{\epsilon/2}-1}$. The second step constructs the *sketch matrix* $\mathrm{M} \in \mathbb{R}^{k \times m}$, in which the rows are indexed by hash functions, and each row j is the sum of the donations of users who selected the hash function indexed by j. Let $D = \{(\tilde{\boldsymbol{v}}^{(1)}, j^{(1)}), \cdots, (\tilde{\boldsymbol{v}}^{(n)}, j^{(n)})\}$ be the dataset collected from devices. For each data record $(\tilde{\boldsymbol{x}}^{(i)}, j^{(i)})$, we add $\tilde{\boldsymbol{x}}^{(i)}$ to row $j^{(i)}$ of M. We give the formal procedure for computing the sketch matrix M in Algorithm 3.

Given the sketch matrix, our last step is to estimate the count for entry $d \in \mathcal{D}$, which we do by debiasing the counts and averaging over the corresponding hash entries in M. This procedure is presented in Algorithm 4.

We analyze the utility of our combined client-server algorithm by calculating its mean and variance. We defer the proof to Appendix A.

Algorithm 3 Compute Sketch Matrix `Sketch-CMS`

```
Require: Dataset $D = \{(\tilde{\boldsymbol{v}}^{(1)}, j^{(1)}), \cdots, (\tilde{\boldsymbol{v}}^{(n)}, j^{(n)})\}$; privacy parameter $\epsilon$, dimensions $k$ and $m$.
1. Set $c_\epsilon \leftarrow \frac{e^{\epsilon/2}+1}{e^{\epsilon/2}-1}$.
2. For each $i \in [n]$ set $\tilde{\boldsymbol{x}}^{(i)} \leftarrow k \cdot \left(\frac{c_\epsilon}{2} \cdot \tilde{\boldsymbol{v}}^{(i)} + \frac{1}{2} \cdot \mathbf{1}\right)$.
3. Initialize $\mathrm{M} \in \{0\}^{k \times m}$.
4. for $i \in [n]$ do
5.     for $\ell \in [m]$ do
6.         $\mathrm{M}_{j^{(i)},\ell} \leftarrow \mathrm{M}_{j^{(i)},\ell} + \tilde{x}_\ell^{(i)}$.
7. return Sketch matrix M.
```

Algorithm 4 Server-Side $\mathcal{A}_{\mathsf{server}}$

```
Require: Data element $d \in \mathcal{D}$; sketch $\mathrm{M} \in \mathbb{R}^{k \times m}$, $\epsilon$, hashes $\mathcal{H}$.
1. Construct $\tilde{f} : \mathcal{D} \to \mathbb{R}$ where
       $$\tilde{f}(d) = \left(\frac{m}{m-1}\right)\left(\frac{1}{k}\sum_{\ell=1}^{k} \mathrm{M}_{\ell,h_\ell(d)} - \frac{n}{m}\right). \quad (2)$$
2. return $\tilde{f}(d)$.
```

Theorem 4.2 (Utility guarantee). *Let $\epsilon > 0$, $m, k > 1$, and $d \in \hat{\mathcal{D}}$ be an arbitrary element in the dictionary. Let $\tilde{f}(d)$ be the d-th coordinate of the output of* `CMS`$((d^{(1)}, \cdots, d^{(n)}); \epsilon, k, m, \hat{\mathcal{D}})$. *If $f(d)$ is the true frequency of d, then,*

$$\mathbb{E}\left[\tilde{f}(d)\right] = f(d)$$

$$\mathbf{Var}\left[\tilde{f}(d)\right] \leq \left(\frac{m}{m-1}\right)^2 \times \left(\frac{e^{\epsilon/2}}{\left(e^{\epsilon/2}-1\right)^2} + \frac{1}{m} + \frac{\sum_{d' \in \mathcal{D}} f(d')^2}{n \cdot k \cdot m}\right) \times n.$$

4.2 Private Hadamard Count Mean Sketch

From the utility guarantee of `CMS`, if the ℓ_2-norm of the true data distribution is large, then the only way to reduce the term $\frac{\sum_{d' \in \mathcal{D}} f(d')^2}{k \cdot m}$ in the variance is to increase the product $k \cdot m$. Unfortunately, for a particular server runtime budget, we would need to increase the device bandwidth cost to the user to ensure accurate counts. Rather than have the user pay for more accurate counts, we develop a variant of the Count Mean Sketch algorithm — Hadamard Count Mean Sketch (`HCMS`) — which still forms a $k \times m$ sketch matrix on the server, but now has the client-side algorithm only send a single bit to the server. It turns out that even when a single bit is transmitted to the server, we can still achieve small variance comparable to Theorem 4.2.

Intuitively, we want to be able to spread information from a sparse vector, which is 1 in a single location across multiple dimensions, so that when we sample a coordinate from this dense set we still have a sufficiently strong signal about the original sparse vector. The Hadamard basis transform is particularly useful for this purpose. The Hadamard basis is defined recursively in (3) for powers of two, ℓ and the base case:

$H_1 = [1]$

$$H_\ell = \begin{bmatrix} H_{\ell/2} & H_{\ell/2} \\ H_{\ell/2} & -H_{\ell/2} \end{bmatrix}. \tag{3}$$

Note that the columns of H_ℓ are orthogonal and $H_\ell H_\ell = \ell \cdot I_\ell$. There are standard recursive algorithms to compute the Hadamard transform of any vector. The running time of these algorithms is $O(\ell \log \ell)$.

Algorithm 5 Hadamard Count Mean Sketch `HCMS`

```
Require: $d^{(1)}, \cdots, d^{(n)} \in \mathcal{D}^n$; $\epsilon$, $k$, $m$, dictionary $\hat{\mathcal{D}} \subseteq \mathcal{D}$.
1. From a set of three-wise independent hashes mapping $\mathcal{D}$ to $[m]$, select a set $\mathcal{H} = \{h_1, \cdots, h_k\}$ of size
   $k$ uniformly at random.
2. for $i \in [n]$ do
3.     $(\tilde{w}^{(i)}, j^{(i)}, \ell^{(i)}) \leftarrow \mathcal{A}_{\mathsf{client-HCMS}}(d^{(i)}; \epsilon, \mathcal{H})$.
4. Construct the sketch $\mathrm{M} \leftarrow$ Sketch-HCMS $\left((\tilde{w}^{(1)}, j^{(1)}, \ell^{(1)}), \cdots, (\tilde{w}^{(n)}, j^{(n)}, \ell^{(n)}); \epsilon, k, m\right)$.
5. for $d \in \hat{\mathcal{D}}$ do
6.     $\tilde{f}(d) \leftarrow \mathcal{A}_{\mathsf{server}}(d; \mathrm{M}, \epsilon, \mathcal{H})$.
7. return Histogram $\left(\tilde{f}(d) : d \in \hat{\mathcal{D}}\right)$.
```

We show that the Hadamard transform helps control the high variance in the estimator mentioned above. For each $\boldsymbol{v} \in \{-1, 1\}^m$ the client-side algorithm generates $\boldsymbol{w} = H_m \boldsymbol{v}$, and then samples $\ell \overset{iid}{\sim} [m]$ so that it only privatizes the bit w_ℓ. The server-side algorithm remains the same as in `CMS`, except the sketch matrix needs to be modified. We then denote the complete algorithm as `HCMS`, which is given in Algorithm 5, and gives the histogram of counts over a dictionary of domain elements $\hat{\mathcal{D}} \subseteq \mathcal{D}$.

4.2.1 Client-Side Algorithm

The client-side algorithm from Algorithm 2 remains largely the same, but we include the index ℓ that is selected along with the index of the hash function selected and the single bit of the transformed vector $\boldsymbol{w}$.

Algorithm 6 Client-Side Algorithm $\mathcal{A}_{\mathsf{client-HCMS}}$

```
Require: Data element: $d \in \mathcal{D}$; privacy parameter $\epsilon$, $\mathcal{H}$.
1. Sample $j$ uniformly at random from $[k]$.
2. Initialize a vector $\boldsymbol{v} \leftarrow \{0\}^m$.
3. Set $v_{h_j(d)} \leftarrow 1$.
4. Transform $\boldsymbol{w} \leftarrow H_m \boldsymbol{v}$.
5. Sample $\ell$ uniformly at random from $[m]$.
6. Sample $b \in \{-1, +1\}$, which is $+1$ with probability $\frac{e^\epsilon}{e^\epsilon + 1}$.
7. Set $\tilde{w} \leftarrow b w_\ell$.
8. return $\tilde{w}$, index $j$, index $\ell$.
```

The privacy analysis is similar to Theorem 4.1.

Theorem 4.3 (Privacy guarantee). *Algorithm $\mathcal{A}_{\mathsf{client-HCMS}}$ is an ϵ-local randomizer. Further,* `HCMS` *is ϵ-local differentially private.*

Proof. The analysis here is a slight variant of the proof in Theorem 4.1, due to the fact that $\mathcal{A}_{\textsf{client-HCMS}}$ is sending a single bit. We show for an arbitrary output $\tilde{w} \in \{-1, 1\}$, the probability of observing the output is similar whether the user reported data d or d'. Let J be the random variable that selects uniformly from $[k]$, L be the random variable that selects uniformly from $[m]$, and B be the random variable for the biased bit b in $\mathcal{A}_{\textsf{client-HCMS}}$. Given output indices $j \in [k]$ and $\ell \in [m]$, we write the corresponding one-hot vector $\boldsymbol{v}$ and $\boldsymbol{v}'$, respectively for data entry d and d'. We will write the ℓth component of $H_m\boldsymbol{v}$ as $(H_m\boldsymbol{v})_\ell$

$$\frac{\Pr\left[\mathcal{A}_{\textsf{client-HCMS}}(d;\epsilon,\mathcal{H}) = (\tilde{w}, j, \ell)\right]}{\Pr\left[\mathcal{A}_{\textsf{client-HCMS}}(d';\epsilon,\mathcal{H}) = (\tilde{w}, j, \ell)\right]} = \frac{\Pr\left[B\left(H_m\boldsymbol{v}\right)_\ell = \tilde{w} | J = j, L = \ell\right]\Pr\left[J = j\right]\Pr\left[L = \ell\right]}{\Pr\left[B\left(H_m\boldsymbol{v}'\right)_\ell = \tilde{w} | J = j, L = \ell\right]\Pr\left[J = j\right]\Pr\left[L = \ell\right]}$$
$$= \frac{\Pr\left[B\left(H_m\boldsymbol{v}\right)_\ell = \tilde{w} | J = j\right]}{\Pr\left[B\left(H_m\boldsymbol{v}'\right)_\ell = \tilde{w} | J = j\right]}$$

Note that $H_m\boldsymbol{v}, H_m\boldsymbol{v}' \in \{-1, 1\}^m$, $\Pr[B = 1] = \frac{e^\epsilon}{1+e^\epsilon}$, and $\Pr[B = -1] = \frac{1}{e^\epsilon+1}$. Hence the ratio can be no more than e^ϵ and no less than $e^{-\epsilon}$. We then have the our privacy guarantee,

$$-\epsilon \leq \ln\left(\frac{\Pr\left[\mathcal{A}_{\textsf{client-HCMS}}(d;\epsilon,\mathcal{H}) = (\tilde{\boldsymbol{v}}, j)\right]}{\Pr\left[\mathcal{A}_{\textsf{client-HCMS}}(d';\epsilon,\mathcal{H}) = (\tilde{\boldsymbol{v}}, j)\right]}\right) \leq \epsilon.$$

Because `HCMS` is a post-processing function of the outputs from $\mathcal{A}_{\textsf{client-HCMS}}$, we then have that `HCMS` is ϵ-local differentially private. □

4.2.2 Server-Side Algorithm

If the client uses $\mathcal{A}_{\textsf{client-HCMS}}$, we will need to adjust the way we update the sketch matrix from Algorithm 3. However, given the modified sketch matrix M^{H}, the server-side algorithm $\mathcal{A}_{\textsf{server}}$ given in Algorithm 4 remains the same for computing the counts.

Algorithm 7 Compute Sketch Matrix `Sketch-HCMS`

```
Require: Dataset $D = \{(\tilde{w}^{(1)}, j^{(1)}, \ell^{(1)}), \cdots, (\tilde{w}^{(n)}, j^{(n)}, \ell^{(n)})\}$, $\epsilon$, dimensions $k$ and $m$.
1. Set $c_\epsilon \leftarrow \frac{e^\epsilon+1}{e^\epsilon-1}$.
2. For each $i \in [n]$ set $\tilde{x}^{(i)} \leftarrow k \cdot c_\epsilon \cdot \tilde{w}^{(i)}$.
3. Initialize $\mathrm{M}^{\mathrm{H}} \in \{0\}^{k \times m}$.
4. for $i \in [n]$ do
5.     $\mathrm{M}^{\mathrm{H}}_{j^{(i)},\ell^{(i)}} \leftarrow \mathrm{M}^{\mathrm{H}}_{j^{(i)},\ell^{(i)}} + \tilde{x}^{(i)}$.
6. Transform the rows of sketch back: $\mathrm{M}^{\mathrm{H}} \leftarrow \mathrm{M}^{\mathrm{H}} H_m^{\intercal}$.
7. return Sketch matrix $\mathrm{M}^{\mathrm{H}}$.
```

We can obtain a similar utility theorem as Theorem 4.2 when we use $\mathcal{A}_{\textsf{client-HCMS}}$ and the corresponding sketch matrix procedure `Sketch-HCMS`. The analysis is similar to the Count Mean Sketch version and is deferred to Appendix B.

Theorem 4.4 (Utility guarantee). *Let $\epsilon > 0$ and $d \in \hat{\mathcal{D}}$ be an arbitrary element in the dictionary. Let $\tilde{f}(d)$ be the d-th coordinate of the output of `HCMS`$((d_1, \cdots, d_n); \epsilon, k, m, \hat{\mathcal{D}})$. If $f(d)$ is the true frequency of d, then,*

$$\mathbb{E}\left[\tilde{f}(d)\right] = f(d)$$

$$\mathbf{Var}\left[\tilde{f}(d)\right] \leq \left(\frac{m}{m-1}\right)^2 \times \left(\left(\frac{e^\epsilon + 1}{e^\epsilon - 1}\right)^2 + \frac{\sum_{d' \in \mathcal{D}} f(d')^2}{n \cdot k \cdot m}\right) \times n.$$

Comparing this result to the utility guarantee from `CMS` in Theorem 4.2, we see similar terms. However, the term that depends on the privacy parameter ϵ has increased. The main benefit of `HCMS` is that the device bandwidth no longer scales with m.

4.3 Private Sequence Fragment Puzzle

The previous algorithms crucially assume there is some known dictionary $\hat{\mathcal{D}} \subseteq \mathcal{D}$ of domain elements over which the server can enumerate to find counts for all data entries $d \in \hat{\mathcal{D}}$. However, that assumption does not always hold. This is the case when we want to find new words that the user types and that are not part of any Apple-deployed dictionary. Once we have a sketch matrix M, one approach is to enumerate over all possible elements in $\mathcal{D}$ and see what counts are significantly larger than zero. However, in the case of 10-letter English words, for example, this approach would require the server to loop over 26^{10} elements, which is not feasible. Further, we would expect to find many false positives — words that appear to have a significant count, but are never typed by any user.

We consider strings of length up to 10 that are typed by the user where the string does not appear in an on-device dictionary. We keep the length at 10 by padding shorter strings with spaces and truncating longer strings. This approach can be easily generalized to arbitrary-length strings, but we present the above setting for brevity. Our goal is to send information from the client to the server in a differentially private way, and then, on the server, construct the most popular new words.

We call the combined client-side and server-side algorithm the *Sequence Fragment Puzzle (`SFP`) algorithm*, and it is shown in Algorithm 8.

Algorithm 8 Sequence Fragment Puzzle `SFP`

```
Require: Dataset $d^{(1)}, \cdots, d^{(n)} \in \mathcal{D}^n$; $(\epsilon, \epsilon')$, $(k, k')$,$(m, m')$, hash function $h : \mathcal{D} \to [256]$, threshold $\mathcal{T}$.
1. From a set of three-wise independent hashes mapping a subsequence of $\mathcal{D}$ to $[m]$, select a set $\mathcal{H} = \{h_1, \cdots, h_k\}$ of $k$ uniformly at random.
2. From a set of three-wise independent hashes mapping a subsequence of $\mathcal{D}$ to $[m']$, select a set $\mathcal{H}' = \{h_1, \cdots, h_{k'}\}$ of $k'$ uniformly at random.
3. for $i \in [n]$ do
4.     $(\alpha^{(i)}, \beta^{(i)}, \ell^{(i)}) \leftarrow \mathcal{A}_{\mathsf{client-SFP}}(d^{(i)}; (\epsilon, \epsilon'), (\mathcal{H}, \mathcal{H}'), h)$.
5. return $\mathcal{A}_{\mathsf{server-SFP}}\left(\{(\alpha^{(1)}, \beta^{(1)}, \ell^{(1)}), \cdots, (\alpha^{(n)}, \beta^{(n)}, \ell^{(n)})\}, (\epsilon, \epsilon'); (\mathcal{H}, \mathcal{H}')\right), \mathcal{T}$.
```

4.3.1 Client-Side Algorithm

For ease of notation, we present the following setting where $\mathcal{D}$ is the set of all 10-character strings, but we can easily generalize to other domains. We will write a string as $\boldsymbol{s} \in \mathcal{D}$, and $\boldsymbol{s}[i : j]$ will denote the characters in $\boldsymbol{s}$ starting at index i to j. Our client-side algorithm works as follows: We first use a small range (256-bit output) hash function h to hash the string $\boldsymbol{s}$, which we call the *sequence*. We then sample an odd index $\ell \in \{1, 3, 5, 7, 9\}$ uniformly at random and construct a substring $\boldsymbol{s}[\ell : \ell + 1]$. Note that we can use arbitrary length substrings, but we use substrings of length 2 for easier exposition. With the above in place, we then construct the *fragment* $\boldsymbol{r} = h(\boldsymbol{s}) \| \boldsymbol{s}[\ell : \ell + 1]$ where the notation $a \| b$ denotes a concatenated with

b. The client-side algorithm then sends the index ℓ, the output $\mathcal{A}_{\mathsf{client-CMS}}(\boldsymbol{r}; \epsilon', \mathcal{H}')$ from Algorithm 2 with the fragment, and the output $\mathcal{A}_{\mathsf{client-CMS}}(\boldsymbol{s}; \epsilon, \mathcal{H})$ with the full sequence.

Algorithm 9 Client-Side $\mathcal{A}_{\mathsf{client-SFP}}$

```
Require: String: $\boldsymbol{s} \in \mathcal{D}$; privacy parameters: $(\epsilon, \epsilon')$, hash functions $(\mathcal{H}, \mathcal{H}')$, and $h$ with output size $256$.
1. Sample $\ell$ uniformly at random from $\{1, 3, 5, 7, 9\}$.
2. Set $\boldsymbol{r} \leftarrow h(\boldsymbol{s}) \| \boldsymbol{s}[\ell : \ell + 1]$.
3. return $(\mathcal{A}_{\mathsf{client-CMS}}(\boldsymbol{r}, \epsilon', \mathcal{H}'), \mathcal{A}_{\mathsf{client-CMS}}(\boldsymbol{s}, \epsilon, \mathcal{H}), \ell)$.
```

Because we are sending two records from the same data entry, we need to allocate a privacy budget of ϵ' for the fragment record $\boldsymbol{r}$ and ϵ budget for the sequence record $\boldsymbol{s}$. Due to composition properties of differential privacy [8], this will ensure that the total privacy parameter is no more than $\epsilon + \epsilon'$, so that $\mathcal{A}_{\mathsf{client-SFP}}$ is an $(\epsilon + \epsilon')$-local randomizer. The privacy analysis follows immediately from Theorem 4.1 and uses composition of the two outputs. Thus, `SFP` is $(\epsilon + \epsilon')$-local differentially private.

Theorem 4.5 (Privacy guarantee). *Algorithm $\mathcal{A}_{\mathsf{client-SFP}}$ is an $(\epsilon + \epsilon')$-local randomizer. Further,* `SFP` *is $(\epsilon + \epsilon')$-local differentially private.*

4.3.2 Server-Side Algorithm

We will denote the output of $\mathcal{A}_{\mathsf{client-CMS}}(\boldsymbol{r}; \epsilon', \mathcal{H}')$ as α and the output of $\mathcal{A}_{\mathsf{client-CMS}}(\boldsymbol{s}; \epsilon, \mathcal{H})$ as β. Further, we will write the full dataset as $\{(\ell^{(1)}, \alpha^{(1)}, \beta^{(1)}), \cdots, (\ell^{(n)}, \alpha^{(n)}, \beta^{(n)})\}$ given to the server-side algorithm. The server uses $\mathcal{A}_{\mathsf{server}}$ with input $(\beta^{(1)}, \cdots, \beta^{(n)})$ to create a frequency oracle $\tilde{f}$ for the sequence. For each $\ell \in \{1, 3, 5, 7, 9\}$, the server-side algorithm creates a frequency oracle $\tilde{f}_\ell$ for fragments at that location. Next, it computes the heavy hitters in each fragment location and leverages the *puzzle piece*, or the small hash h, to create the Cartesian product across fragments. This must be done carefully, as a naive Cartesian product would increase the dictionary size exponentially. We anchor each Cartesian product based on the output of the small hash h and take the union of the sequences thus constructed to create a dictionary. Finally, we iterate through this dictionary to query the sequence frequencies using the frequency oracle for the full sequence.

Concatenating the small hash value with the two characters in the fragment $\boldsymbol{r}$ correlates the fragment with the full word. If we did not include this small hash value, there might be fragments that have very large counts in each position but are not part of any word, thus leading to false positives.

5 Balancing Utility, Privacy, Computation, and Communication Overhead

We want to ensure that our system architecture and algorithms scale to the hundreds of millions of people using iOS and macOS devices. To that end, we provide a way to tradeoff utility with privacy budget, device bandwidth, and server computation cost. Further, we describe a systematic approach to deploying local differentially private algorithms that achieve good utility and privacy. To be transparent, we make the privatized data donated to Apple's server available on users' devices.

State-of-the-art, local differentially private algorithms solve the known and unknown dictionary problems in a theoretical context with asymptotically minimal device bandwidth and server computation cost [3] by relying on the Johnson-Lindenstrauss (JL) transform. However, the effect on utility is unclear if the device bandwidth or the server computation is varied from the prescribed values.

Algorithm 10 Server-Side $\mathcal{A}_{\mathsf{server-SFP}}$

```
Require: Client submissions $(\alpha^{(i)}, \beta^{(i)}, \ell^{(i)})$ for each $i$; privacy parameters: $(\epsilon, \epsilon')$, hashes $(\mathcal{H}, \mathcal{H}'), \mathcal{T}$.
1.  Create sketch $\mathrm{M} \leftarrow$ Sketch-CMS $(\beta^{(1)}, \cdots, \beta^{(n)}; \epsilon, k, m)$.
2.  Create frequency oracle $\tilde{f}(\cdot)$ using $\mathcal{A}_{\mathsf{server}}(\cdot; \mathrm{M}, \epsilon, \mathcal{H})$.
3.  for $\ell \in \{1, 3, 5, 7, 9\}$ do
4.      Denote $S_\ell \subseteq [n]$ as the set of records that have index $\ell$ selected in the output of $\mathcal{A}_{\mathsf{client-SFP}}$.
5.      Create sketch $\mathrm{M}_\ell \leftarrow$ Sketch-CMS $((\alpha^{(i)} : i \in S_\ell); \epsilon', k', m')$.
6.      Construct frequency oracle $\tilde{f}_\ell(\cdot)$ for every fragment location $\ell$ using $\mathcal{A}_{\mathsf{server}}(\cdot; \mathrm{M}_\ell, \epsilon', \mathcal{H}')$.
7.  Dictionary $D \leftarrow \emptyset$.
8.  for $\ell \in \{1, 3, 5, 7, 9\}$ do
9.      Initialize $Q_\ell \leftarrow \emptyset$.
10.     Update $Q_\ell$ with $\mathcal{T}$ tuples $\{w \| \boldsymbol{s}\}$ with the largest counts $\tilde{f}_\ell(w \| \boldsymbol{s})$ for $\boldsymbol{s} \in \Omega^2, w \in [256]$ for alphabet $\Omega$.
11. for $w \in [256]$ do
12.     Form the Cartesian product of terms in $Q(w) = \{q_1 \| \cdots \| q_9 : w \| q_\ell \in Q_\ell$ for $\ell \in \{1, 3, 5, 7, 9\}\}$.
13.     for $\boldsymbol{s} \in Q(w)$ do
14.         $D \leftarrow D \cup \{\boldsymbol{s}\}$.
15. return Dictionary $D$ and frequencies $\tilde{f}(d)$ for $d \in D$.
```

One of the main benefits of our algorithms CMS and HCMS is that Theorems 4.2 and 4.4 give us the hypersurface in which we can see the tradeoff on the error of our counts when, for example, we halve the device bandwidth. In the case of HCMS, devices can still submit a single bit to the server, and we can reduce the variance by increasing the parameter m, which materially impacts only the server computation cost.

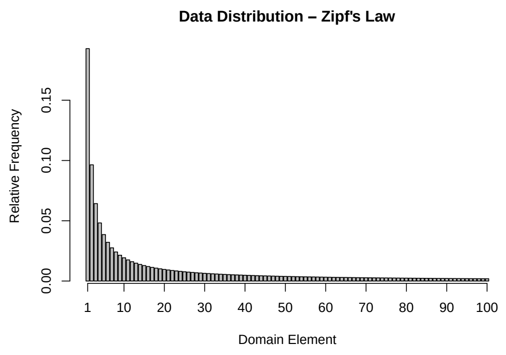
Our implementation obtains high utility counts from aggregated privatized reports. Further, we can see the tradeoffs among utility, privacy, server computation, and device bandwidth. Let $m_{\texttt{CMS}}$ and $k_{\texttt{CMS}}$ be the chosen parameters for m and k in CMS, respectively, and similarly $m_{\texttt{HCMS}}$ and $k_{\texttt{HCMS}}$ for HCMS. We denote the parameter for device bandwidth (in bits) to be M_i for our algorithms $i \in \{\texttt{CMS}, \texttt{HCMS}\}$, where

$$M_{\texttt{CMS}} = \log_2(k_{\texttt{CMS}}) + m_{\texttt{CMS}} \qquad M_{\texttt{HCMS}} = \log_2(k_{\texttt{HCMS}}) + \log_2(m_{\texttt{HCMS}}) + 1.$$

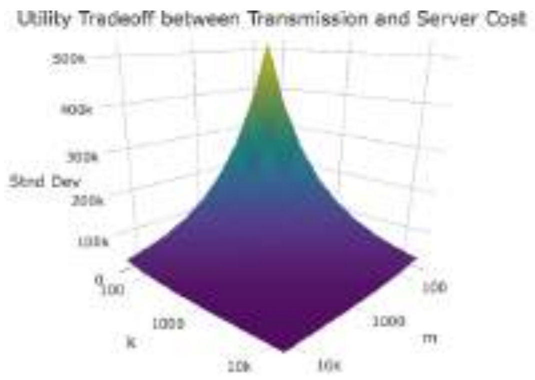
Similarly, we write the server runtime as T_i for our algorithms $i \in \{\texttt{CMS}, \texttt{HCMS}\}$, which includes the computation time for the sketch-matrix data structure. Recall that n is the number of records collected and $p = |\hat{\mathcal{D}}|$ is the size of the dictionary.

$$T_{\texttt{CMS}} = \mathcal{O}(\underbrace{n m_{\texttt{CMS}}}_{\text{sketch-matrix}} + p k_{\texttt{CMS}}) \qquad T_{\texttt{HCMS}} = \mathcal{O}(\underbrace{n + k_{\texttt{HCMS}} m_{\texttt{HCMS}} \log_2(m_{\texttt{HCMS}})}_{\text{sketch-matrix}} + p k_{\texttt{HCMS}}).$$

Given the utility guarantees in Theorems 4.2 and 4.4, we can now analyze how each parameter contributes to the underlying variance of the estimated counts. The parameter m determines the device bandwidth cost to the user, while the parameter k directly affects the server runtime. We can then plot the standard deviation of the counts as a function of ϵ, device bandwidth cost, and server runtime to see the tradeoffs among the various factors. Figure 4b gives the standard deviation as a function of m and k for a particular count along with a fixed $\epsilon = 2$, n = 100,000,000, and data distribution given in Figure 4a.



(a) Data distribution that follows Zipf's law with parameter 1.2



(b) Fixing $\epsilon = 2$ and data distribution with $n =$ 100,000,000

Figure 4: Tradeoffs Among Utility, Device Bandwidth (m), and Server Computation (k) for Fixed $\epsilon = 2$

6 Results

We deploy our algorithms across hundreds of millions of devices. We present results for the following use cases: New Words, Popular Emojis, Video Playback Preferences (Auto-play Intent) in Safari, High Energy and Memory Usage in Safari, and Popular HealthKit Usage.

6.1 Discovering New Words

We want to learn words that are not present in the lexicons included on the device in order to improve auto-correction. To discover new words, we deploy the Sequence Fragment Puzzle (`SFP`) algorithm from Section 4.3 with the parameters $(m, m') = (1024, 1024)$, $(k, k') = (2048, 2048)$, and $(\epsilon, \epsilon') = (2, 6)$.

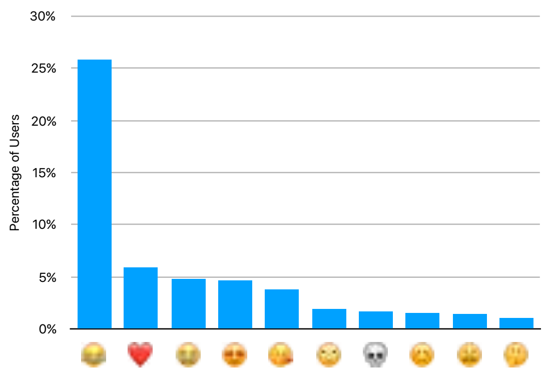
The algorithm produces results spanning several languages, including English, French, and Spanish. The learned words for the English keyboard, for example, can be divided into multiple categories: abbreviations like *wyd*, *wbu*, and *idc*; popular expressions like *bruh*, *hun*, *bae*, and *tryna*, seasonal or trending words like *Mayweather*, *McGregor*, *Despacito*, *Moana*, and *Leia*; and foreign words like *dia*, *queso*, *aqui*, and *jai*. Using the data, we are constantly updating our on-device lexicons to improve the keyboard experience.

Another category of words discovered are known words without the trailing *e* (*lov* or *th*) or *w* (*kno*). If users accidentally type the left-most prediction cell, which contains the literal string typed thus far, a space will be added to their current word instead of the character they intended to type. This is a key insight that we were able to learn due to our local differentially private algorithm.

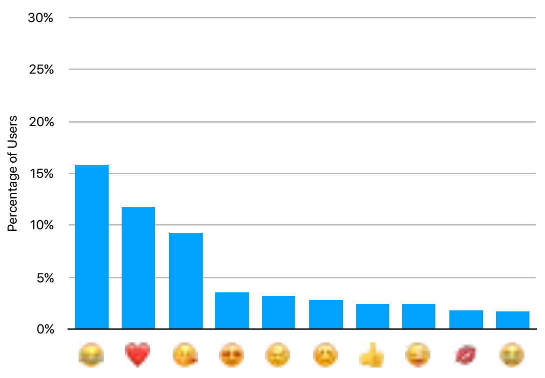
6.2 Discovering Popular Emojis

Given the popularity of emojis across our user base, we want to determine which specific emojis are most used by our customers, and the relative distribution of these characters. To that end, we deploy our algorithms on different locales to understand the distribution of emojis used across keyboard locales. For this use case, we set the parameters for `CMS` to be $m = 1024$, k = 65,536, and $\epsilon = 4$ with $p = 2600$ emojis.

With the data, we see many differences across keyboard locales. In Figure 5, we observe snapshots from two locales: English and French. Using this data, we can improve our predictive emoji QuickType across different locales.



(a) English



(b) French

Figure 5: Emojis in Different Keyboard Locales

6.3 Safari Auto-play Intent

Safari in macOS High Sierra provides users the ability to browse the web with fewer distractions, particularly in the form of relief from websites that auto-play with sound.

We are able to infer default auto-play policies algorithmically, based on crowdsourced engagement metrics. Safari determines a user may want auto-play with sound if the user plays a media element that was originally prevented from auto-playing, or when a media element that was allowed to auto-play is allowed to continue. We only collect privatized domains, not full URLs.

For these reports, we use our CMS algorithm where $\mathcal{D}$ is all possible web domains and the dictionary is a set of p = 250,000 web domains. For this use case, we set the parameters for CMS to be $m = 1024$, k = 65,536, and $\epsilon = 8$. Note that we are mapping a large number of web domains (250,000) to a much smaller space (1024 bits), so there are expected to be many hash collisions, which provides further deniability to the original record. Using differential privacy, we are able to successfully identify an up-to-date set of websites for which most users prefer auto-play with sound worldwide, such as streaming services including massive open online courses (MOOC) websites and other popular video websites.

6.4 Identifying High Energy and Memory Usage in Safari

Some websites are exceedingly resource-intensive and we wish to identify these sites in order to ensure a better user experience. We consider two types of domains: those that cause high memory usage and those that cause excessive energy drain due to CPU usage. On iOS 11 and macOS High Sierra, Safari automatically can detect these exceptional domains, and report these using differential privacy.

Using our algorithms, we are able to determine which domains have high resource consumption. For this use case, we set the parameters for HCMS to be m = 32,768, k = 1024, and $\epsilon = 4$ with p = 250,000 web domains. Recall that the differentially private record is only 1 bit in HCMS. Our data shows that the top domains include video consumption websites, shopping websites, and news websites.

6.5 Understanding HealthKit Usage

We want to know popular health data types to drive future improvement in the Health app. However, health data types are sensitive; for example, if a user tracks blood glucose levels, it may imply that the user has diabetes. We deploy CMS to observe the most popular health data types with the following parameters: $m = 256$, k = 65,536, and $\epsilon = 2$.

Our findings indicate that sleep analysis, heart rate, active calories, reproductive health, and mindfulness are among the most popular health types recorded.

7 Conclusion

In this paper, we have presented a novel learning system architecture, which leverages local differential privacy and combines it with privacy best practices. To scale our system across millions of users and a variety of use cases, we have developed local differentially private algorithms – CMS, HCMS, and SFP – for both the known and unknown dictionary settings. We have provided analytic expressions for the tradeoffs among various factors, including privacy, utility, server computation overhead, and device bandwidth. Our utility theorems give a principled way to choose algorithmic parameters to minimize the transmission cost for the users without lowering accuracy. Without such expressions, it is difficult to evaluate the impact on accuracy if, for example, transmission cost is reduced without running costly iterations. Further, to keep transmission costs to an absolute minimum, our HCMS algorithm can obtain accurate counts when each user sends only a single privatized bit. We believe that this paper is one of the first to demonstrate the successful deployment of local differential privacy in a real-world setting across multiple use cases. We showed that we could find popular abbreviations and slang words typed, popular emojis, popular health data types while satisfying local differential privacy. Further, we can identify websites that are consuming too much energy and memory and websites where users want Auto-play. This information can and already has been used to improve features for the benefit of the user experience.

We hope that this paper will serve a role in bridging the gap between theory and practice of private systems. We also believe our work will continue to support research in a broad set of large-scale learning problems while preserving users' privacy.

References

[1] https://www.apple.com/business/docs/iOS_Security_Guide.pdf.

[2] J. M. Abowd. The Challenge of Scientific Reproducibility and Privacy Protection for Statistical Agencies. https://www2.census.gov/cac/sac/meetings/2016-09/2016-abowd.pdf, September 15, 2016. Census Scientific Advisory Committee.

[3] R. Bassily and A. Smith. Local, Private, Efficient Protocols for Succinct Histograms. In *STOC*, 2015.

[4] G. Cormode and S. Muthukrishnan. An improved data stream summary: The count-min sketch and its applications. *J. Algorithms*, 55(1):58–75, Apr. 2005.

[5] J. C. Duchi, M. Jordan, M. J. Wainwright, et al. Local Privacy and Statistical Minimax Rates. In *FOCS*. IEEE, 2013.

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

[7] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *Proceedings of the Forty-second ACM Symposium on Theory of Computing*, STOC '10, pages 715–724, New York, NY, USA, 2010. ACM.

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

[9] Ú. Erlingsson, V. Pihur, and A. Korolova. RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response. In *ACM CCS*, 2014.

[10] G. Fanti, V. Pihur, and Ú. Erlingsson. Building a RAPPOR with the Unknown: Privacy-Preserving Learning of Associations and Data Dictionaries. *PoPETS*, issue 3, 2016.

[11] M. Gaboardi, J. Honaker, G. King, K. Nissim, J. Ullman, S. Vadhan, and J. Murtagh. PSI (Ψ): a Private data Sharing Interface. In *Theory and Practice of Differential Privacy*, 2016.

[12] J. Hsu, S. Khanna, and A. Roth. Distributed Private Heavy Hitters. In *ICALP*. 2012.

[13] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith. What Can We Learn Privately? *SIAM Journal on Computing*, 40(3):793–826, 2011.

[14] A. Machanavajjhala, D. Kifer, J. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory Meets Practice on the Map. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, pages 277–286, Washington, DC, USA, 2008. IEEE Computer Society.

[15] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 19–30, New York, NY, USA, 2009. ACM.

[16] N. Mishra and M. Sandler. Privacy via Pseudorandom Sketches. In *PODS*, 2006.

[17] P. Mohan, A. Thakurta, E. Shi, D. Song, and D. E. Culler. GUPT: Privacy Preserving Data Analysis Made Easy. In *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 349–360, 2012.

[18] Z. Qin, Y. Yang, T. Yu, I. Khalil, X. Xiao, and K. Ren. Heavy hitter estimation over set-valued data with local differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, pages 192–203, New York, NY, USA, 2016. ACM.

[19] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and Privacy for MapReduce. In *Proceedings of the 7th USENIX Conference on Networked Systems Design and Implementation*, pages 20–20, 2010.

A Proof of Utility Guarantee for CMS

Here we will present the proof of Theorem 4.2. Recall that we write $f(d)$ to denote the true count for the number of people with entry d and $\tilde{f}(d)$ to denote output of `CMS` for data entry $d \in \hat{\mathcal{D}}$. We now set up some notation. Let $B \in \{-1, 1\}$ denote the random variable that is 1 with probability $\frac{e^{\epsilon/2}}{1+e^{\epsilon/2}}$ and $J \sim \text{Uniform}[k]$. We will write the encoding vector $\boldsymbol{v}^{(i)} \in \{-1, 1\}^m$ which is -1 everywhere except at position $h_J(d^{(i)})$ for record i. Note that $d^{(i)}$ is i's data entry. To avoid writing too many subscripts, we will denote the $h_j(d)$th index of $\boldsymbol{v}^{(i)}$ as $v^{(i)}[h_j(d)]$. Consider the $(j, h_j(d))$th entry of the sketch matrix for the ith data record, which can be written as,

$$Y_j^{(i)}(d) = k\left(\frac{c_\epsilon\, B\, v^{(i)}[h_j(d)] + 1}{2}\right) \mathbb{1}\{J = j\}, \tag{4}$$

where $c_\epsilon = \frac{e^{\epsilon/2}+1}{e^{\epsilon/2}-1}$. Note that $Y_j^{(i)}(d)$ is nonzero only when $J = j$.

Note that if i's data entry $d^{(i)} = d$ then $v^{(i)}[h_j(d)] = 1$. If $d^{(i)} \neq d$, then $v^{(i)}[h_j(d)] = 1$ with probability $1/m$ and -1 otherwise since the set $\mathcal{H}$ is chosen uniformly at random from a three-wise independent hash family. This connection will help us relate $v^{(i)}[h_j(d)]$ to the true count $f(d)$.

We first compute the moments of $Y_j^{(i)}(d)$ when $d^{(i)} = d$ or some other element.

Lemma A.1.

$$\mathbb{E}\left[Y_j^{(i)}(d)\right] = \mathbb{1}\{d^{(i)} = d\} + \frac{1}{m} \cdot \mathbb{1}\{d^{(i)} \neq d\}$$

Proof. We carry out the calculations for the case when $d^{(i)} = d$. Note that $\mathbb{E}[B] = \frac{1}{c_\epsilon}$.

$$\mathbb{E}\left[Y_j^{(i)}(d)\right] = k \cdot \mathbb{E}\left[\frac{c_\epsilon\, B\, v^{(i)}[h_j(d)] + 1}{2} | J = j\right] \Pr[J = j] = 1$$

Similarly, we can compute the other case where $d^{(i)} \neq d$,

$$\begin{aligned}
\mathbb{E}\left[Y_j^{(i)}(d)\right] &= k \cdot \mathbb{E}\left[\frac{c_\epsilon\, B\, v^{(i)}[h_j(d)] + 1}{2} | J = j\right] \Pr[J = j] \\
&= \left(1 - \frac{1}{m}\right) \cdot \mathbb{E}\left[\frac{c_\epsilon B(-1) + 1}{2}\right] + \frac{1}{m} \cdot \mathbb{E}\left[\frac{c_\epsilon B(1) + 1}{2}\right] \\
&= \left(1 - \frac{1}{m}\right) \cdot 0 + \frac{1}{m} \cdot 1 = \frac{1}{m}
\end{aligned}$$

□

Lemma A.2.

$$\mathbb{E}\left[\left(Y_j^{(i)}(d)\right)^2\right] = \frac{k}{4}\left(c_\epsilon^2 - 1\right) + k\mathbb{1}\{d^{(i)} = d\} + \frac{k}{m}\mathbb{1}\{d^{(i)} \neq d\}$$

Proof. We compute the second moment with $d^{(i)} = d$,

$$\begin{aligned}
k^2 \cdot \mathbb{E}&\left[\left(\frac{c_\epsilon\, B\, v^{(i)}[h_j(d)] + 1}{2}\right)^2 \mathbb{1}\{J = j\}\right] \\
&= k \cdot \mathbb{E}\left[\left(\frac{c_\epsilon B + 1}{2}\right)^2\right] \\
&= k \cdot \left(\left(\frac{c_\epsilon + 1}{2}\right)^2 \left(\frac{e^{\epsilon/2}}{1 + e^{\epsilon/2}}\right) + \left(\frac{-c_\epsilon + 1}{2}\right)^2 \left(\frac{1}{1 + e^{\epsilon/2}}\right)\right) \\
&= \frac{k}{4} \cdot \left(c_\epsilon^2 + 3\right) = \frac{k}{4} \cdot \left(c_\epsilon^2 - 1\right) + k
\end{aligned}$$

We now compute the term with $d^{(i)} \neq d$,

$$\begin{aligned}
k^2 \cdot \mathbb{E}&\left[\left(\frac{c_\epsilon\, B\, v^{(i)}[h_j(d)] + 1}{2}\right)^2 \mathbb{1}\{J = j\}\right] \\
&= \frac{k}{m}\mathbb{E}\left[\left(\frac{c_\epsilon B + 1}{2}\right)^2\right] + k \cdot \left(1 - \frac{1}{m}\right)\mathbb{E}\left[\left(\frac{c_\epsilon B - 1}{2}\right)^2\right] \\
&= \frac{k}{m}\left[\left(\frac{c_\epsilon + 1}{2}\right)^2 \left(\frac{e^{\epsilon/2}}{1 + e^{\epsilon/2}}\right) + \left(\frac{c_\epsilon - 1}{2}\right)^2 \left(\frac{1}{1 + e^{\epsilon/2}}\right)\right] \\
&\quad + k \cdot \left(1 - \frac{1}{m}\right)\left[\left(\frac{c_\epsilon - 1}{2}\right)^2 \left(\frac{e^{\epsilon/2}}{1 + e^{\epsilon/2}}\right) + \left(\frac{c_\epsilon + 1}{2}\right)^2 \left(\frac{1}{1 + e^{\epsilon/2}}\right)\right] \\
&= k \cdot \left(\frac{c_\epsilon^2 - 1}{4} + \frac{1}{m}\right)
\end{aligned}$$

□

We now compute the cross term between different indices $i_1, i_2 \in [n]$ with reports $d^{(i_1)}$ and $d^{(i_2)}$.

Lemma A.3. *Let $i_1 \neq i_2$ be different indices. Let $j_1 \neq j_2$, then we have*

$$\mathbf{Cov}\left(Y_{j_1}^{(i_1)}(d), Y_{j_2}^{(i_2)}(d)\right) = 0$$

If $d^{(i_1)} = d$ or $d^{(i_2)} = d$ or $d^{(i_1)} \neq d^{(i_2)}$ then

$$\mathbf{Cov}\left(Y_j^{(i_1)}(d), Y_j^{(i_2)}(d)\right) = 0$$

Further, if $d^{(i_1)}, d^{(i_2)} = d'$ where $d' \neq d$ then

$$\mathbf{Cov}\left(Y_j^{(i_1)}(d), Y_j^{(i_2)}(d)\right) = \frac{1}{m} - \frac{1}{m^2}.$$

Proof. We will denote with a prime an independent copy of a random variable that is unprimed, for example B and B'. We first prove the lemma for terms that have zero covariance. Let $j_1 \neq j_2$. Note that the set $\mathcal{H}$ is chosen independently and uniformly from a three-wise independent hash family ($h_{j_1} \not\equiv h_{j_2}$). We then have,

$$\mathbb{E}\left[Y_{j_1}^{(i_1)}(d) Y_{j_2}^{(i_2)}(d)\right] = \mathbb{E}\left[Y_{j_1}^{(i_1)}(d)\right] \mathbb{E}\left[Y_{j_2}^{(i_2)}(d)\right].$$

Hence, $\mathbf{Cov}\left(Y_{j_1}^{(i_1)}(d), Y_{j_2}^{(i_2)}(d)\right) = 0$

We next fix $j_1 = j_2 = j$.

$$\begin{aligned}
&\mathbb{E}\left[Y_j^{(i_1)}(d) Y_j^{(i_2)}(d)\right] \\
&\quad = \mathbb{E}\left[\left(\frac{c_\epsilon\ B\ v^{(i_1)}[h_j(d)] + 1}{2}\right)\left(\frac{c_\epsilon\ B'\ v^{(i_2)}[h_j(d)] + 1}{2}\right)\right] \\
&\quad = \frac{1}{4} \cdot \left\{\mathbb{E}\left[c_\epsilon^2 B B' v^{(i_1)}[h_j(d)] v^{(i_2)}[h_j(d)]\right] + \mathbb{E}\left[c_\epsilon B v^{(i_1)}[h_j(d)]\right] + \mathbb{E}\left[c_\epsilon B' v^{(i_2)}[h_j(d)]\right] + 1\right\} \\
&\quad = \frac{1}{4}\left\{\mathbb{E}\left[v^{(i_1)}[h_j(d)]\ v^{(i_2)}[h_j(d)]\right] + \mathbb{E}\left[v^{(i_1)}[h_j(d)]\right] + \mathbb{E}\left[v^{(i_2)}[h_j(d)]\right] + 1\right\} \\
&\quad = \frac{1}{4}\left\{\mathbb{E}\left[\left(2 \cdot \mathbb{1}\{h_j(d) = h_j(d^{(i_1)})\} - 1\right)\left(2 \cdot \mathbb{1}\{h_j(d) = h_j(d^{(i_2)})\} - 1\right)\right]\right\} \\
&\qquad + \frac{1}{4}\left\{\mathbb{E}\left[\left(2 \cdot \mathbb{1}\{h_j(d) = h_j(d^{(i_1)})\} - 1\right)\right] + \mathbb{E}\left[\left(2 \cdot \mathbb{1}\{h_j(d) = h_j(d^{(i_2)})\} - 1\right)\right] + 1\right\} \\
&\quad = \Pr\left[h_j(d) = h_j(d^{(i_1)}) = h_j(d^{(i_2)})\right].
\end{aligned}$$

We then obtain the following expression for the covariance using Lemma A.1,

$$\begin{aligned}
\mathbf{Cov}\left(Y_j^{(i_1)}(d), Y_j^{(i_2)}(d)\right) &= \Pr\left[h_j(d) = h_j(d^{(i_1)}) = h_j(d^{(i_2)}\right] \\
&- \left(\mathbb{1}\{d = d^{(i_1)}\} + \frac{1}{m}\mathbb{1}\{d \neq d^{(i_1)}\}\right) \cdot \left(\mathbb{1}\{d = d^{(i_2)}\} + \frac{1}{m}\mathbb{1}\{d \neq d^{(i_2)}\}\right)
\end{aligned}$$

We now consider our various cases. Let $d = d^{(i_1)}$, we then have

$$\begin{aligned}
\mathbf{Cov}\left(Y_j^{(i_1)}(d), Y_j^{(i_2)}(d)\right) &= \Pr\left[h_j(d^{(i_2)}) = h_j(d)\right] - \left(\mathbb{1}\{d = d^{(i_2)}\} + \frac{1}{m}\mathbb{1}\{d \neq d^{(i_2)}\}\right) \\
&= 0
\end{aligned}$$

The proof is the same for $d = d^{(i_2)}$.

We next consider the case where $d^{(i_1)} \neq d^{(i_2)}$ and neither is equal to d. We use the fact that the hash functions are three-wise independent, so that

$$\Pr\left[h_j(d^{(i_2)}) = h_j(d) = h_j(d^{(i_1)})\right] = \frac{1}{m^2}$$

Hence, we have $\mathbf{Cov}\left(Y_j^{(i_1)}(d), Y_j^{(i_2)}(d)\right) = \frac{1}{m^2} - \frac{1}{m^2}$ = 0.

Lastly, we consider the case where $d^{(i_1)} = d^{(i_2)}$ but they do not equal d. In this case we have,

$$\mathbf{Cov}\left(Y_j^{(i_1)}(d), Y_j^{(i_2)}(d)\right) = \Pr\left[h_j(d^{(i_1)}) = h_j(d)\right] - \frac{1}{m^2} = \frac{1}{m} - \frac{1}{m^2}$$

□

We are now ready to prove our result.

Proof of Theorem 4.2. We can apply the above lemmas to get the desired result. We first compute the expectation of the output from $\mathcal{A}_{\mathsf{server}}$ for a particular data record $d \in \mathcal{D}$, which is a function of the following variable whose terms are given in (4)

$$Y = \frac{1}{k} \sum_{j=1}^{k} \sum_{i=1}^{n} Y_j^{(i)}(d).$$

We can write the output of $\mathcal{A}_{\mathsf{server}}$ now as a function of Y

$$\tilde{f}(d) = \frac{m}{m-1} \left(Y - \frac{n}{m} \right).$$

We then compute the expectation of Y, where we have already computed the terms in Lemma A.1. Recall that we write $f(d)$ to be the true count for data record $d \in \mathcal{D}$.

$$\mathbb{E} \left[\frac{1}{k} \sum_{i=1}^{n} \sum_{j=1}^{k} Y_j^{(i)}(d) \right] = f(d) \left(1 - \frac{1}{m} \right) + \frac{n}{m}$$

We then can compute the variance

$$\begin{aligned}
\mathbf{Var} \left[\frac{1}{k} \sum_{i=1}^{n} \sum_{j=1}^{k} Y_j^{(i)}(d) \right] &= \sum_{i=1}^{n} \mathbf{Var} \left[\frac{1}{k} \sum_{j=1}^{k} Y_j^{(i)}(d) \right] + \frac{1}{k^2} \sum_{i_1 \neq i_2} \mathbf{Cov} \left(\sum_{j=1}^{k} Y_j^{(i_1)}(d), \sum_{j'=1}^{k} Y_{j'}^{(i_2)}(d) \right) \\
&= \frac{1}{k^2} \sum_{i=1}^{n} \left(\sum_{j=1}^{k} \mathbf{Var} \left[Y_j^{(i)}(d) \right] + \sum_{j \neq j'} \mathbf{Cov} \left(Y_j^{(i)}(d), Y_{j'}^{(i)}(d) \right) \right) + \frac{1}{k^2} \sum_{i_1 \neq i_2} \sum_{j=1}^{k} \mathbf{Cov} \left(Y_j^{(i_1)}(d), Y_j^{(i_2)}(d) \right) \\
&= \frac{n(c_\epsilon^2 - 1)}{4} + \frac{(n - f(d))}{m} \left(1 - \frac{1}{m} \right) + \frac{1}{k^2} \sum_{i_1 \neq i_2} \sum_{j=1}^{k} \mathbf{Cov} \left(Y_j^{(i_1)}(d), Y_j^{(i_2)}(d) \right) \\
&= \frac{n(c_\epsilon^2 - 1)}{4} + \frac{(n - f(d))}{m} \left(1 - \frac{1}{m} \right) + \left(\frac{1}{km} - \frac{1}{km^2} \right) \sum_{d^* \neq d} \sum_{i_1 : d^{(i_1)} = d^*} \sum_{i_2 \neq i_1} \mathbb{1}\{ d^{(i_2)} = d^* \} \\
&= \frac{n(c_\epsilon^2 - 1)}{4} + \frac{(n - f(d))}{m} \left(1 - \frac{1}{m} \right) + \left(\frac{1}{km} - \frac{1}{km^2} \right) \sum_{d^* \neq d} \sum_{i : d^{(i)} = d^*} (f(d^*) - 1) \\
&= \frac{n(c_\epsilon^2 - 1)}{4} + \frac{(n - f(d))}{m} \left(1 - \frac{1}{m} - \frac{1}{k} + \frac{1}{km} \right) + \left(\frac{1}{km} - \frac{1}{km^2} \right) \left(\sum_{d^* \neq d} f(d^*)^2 \right) \\
&\leq n \left(\frac{c_\epsilon^2 - 1}{4} + \frac{1}{m} + \frac{1}{nkm} \left(\sum_{d^* \in \mathcal{D}} f(d^*)^2 \right) \right)
\end{aligned}$$

□

B Utility Proof for HCMS

We present the analysis for Theorem 4.4. We largely follow the same analysis as in Theorem 4.2.

We begin by setting up some notation. We will write the (ℓ, j)th entry of the matrix H_m as $H_m[\ell, j] \in \{-1, 1\}$. We will use the same notation as in the previous analysis, so that $B \in \{-1, 1\}$ is 1 with probability $\frac{e^\epsilon}{1+e^\epsilon}$, $J \sim$ Uniform$[k]$ and we add the random variable for $L \sim$ Uniform$[m]$. Here we consider the jth row and $h_j(d)$th column in the sketch matrix M^{H} for only the ith data record, which can be written as

$$Z_j^{(i)}(d) = c_\epsilon k \; H_m[h_j(d), L] \; B \; H_m[L, h_j(d^{(i)})] \mathbb{1}\{J = j\}, \tag{5}$$

where $c_\epsilon = \frac{e^\epsilon+1}{e^\epsilon-1}$. Note that $Z_j^{(i)}(d)$ is zero when $J \neq j$.

We first compute the expectation.

Lemma B.1.
$$\mathbb{E}\left[Z_j^{(i)}(d)\right] = \mathbb{1}\{d^{(i)} = d\} + \frac{1}{m} \cdot \mathbb{1}\{d^{(i)} \neq d\}.$$

Proof. We first prove the case when $d^{(i)} = d$. We will use the property of the Hadamard matrix that its columns are orthogonal.

$$\begin{aligned}
\mathbb{E}\left[Z_j^{(i)}(d)\right] &= \frac{1}{k}\mathbb{E}\left[Z_j^{(i)}(d) | J = j\right] + \left(1 - \frac{1}{k}\right)\mathbb{E}\left[Z_j^{(i)}(d) | J \neq j\right] \\
&= c_\epsilon \; \mathbb{E}\left[H_m[h_j(d), L] \; B \; H_m[L, h_j(d^{(i)})]\right] \\
&= c_\epsilon \mathbb{E}\left[B\right] \mathbb{E}\left[H_m[h_j(d), L]^2\right] \\
&= 1
\end{aligned}$$

We next turn to the case when $d^{(i)} \neq d$.

$$\begin{aligned}
\mathbb{E}\left[Z_j^{(i)}(d)\right] &= \frac{1}{k}\mathbb{E}\left[Z_j^{(i)}(d) | J = j\right] + \left(1 - \frac{1}{k}\right)\mathbb{E}\left[Z_j^{(i)}(d) | J \neq j\right] \\
&= c_\epsilon \mathbb{E}\left[B \; H_m[h_j(d), L] \; H_m[L, h_j(d^{(i)})]\right] \\
&= c_\epsilon \mathbb{E}\left[B\right] \mathbb{E}\left[H_m[h_j(d), L] \; H_m[L, h_j(d^{(i)})]\right] \\
&= \mathbb{E}\left[H_m[h_j(d), L] \; H_m[L, h_j(d^{(i)})]\right] \\
&= \mathbb{E}\left[\mathbb{1}\{h_j(d) = h_j(d^{(i)})\}\right] \\
&= \frac{1}{m}
\end{aligned}$$

□

We now prove the second moment of $Z_j^{(i)}(d)$.

Lemma B.2.
$$\mathbb{E}\left[\left(Z_j^{(i)}(d)\right)^2\right] = c_\epsilon^2 k$$

Proof.

$$\mathbb{E}\left[\left(Z_j^{(i)}(d)\right)^2\right] = c_\epsilon^2 k^2 \mathbb{E}\left[\mathbb{1}\{J = j\}\right] = c_\epsilon^2 k$$

□

We now prove the covariance term across indices $i_1 \neq i_2$. As in the analysis of Theorem 4.2, the only term with nonnegative covariance is the term when $d_{i_1}, d_{i_2} = d'$ where $d' \neq d$, and $j_{i_1} = j_{i_2}$.

Lemma B.3. *Let $i_1 \neq i_2$ be different indices. Let $j_1 \neq j_2$, then we have*

$$\mathbf{Cov}\left(Z_{j_1}^{(i_1)}(d), Z_{j_2}^{(i_2)}(d)\right) = 0.$$

If $d^{(i_1)} \neq d^{(i_2)}$ or $d^{(i_1)} = d^{(i_2)} = d$ then

$$\mathbf{Cov}\left(Z_j^{(i_1)}(d), Z_j^{(i_2)}(d)\right) = 0.$$

If $d^{(i_1)}, d^{(i_2)} = d'$ where $d' \neq d$ then

$$\mathbf{Cov}\left(Z_j^{(i_1)}(d), Z_j^{(i_2)}(d)\right) = \frac{1}{m} - \frac{1}{m^2}$$

Proof. We will denote with a prime an independent copy of a random variable that is unprimed, for example B and B'. We first prove the lemma for terms that have zero covariance. Let $j_1 \neq j_2$. Note that the set $\mathcal{H}$ is chosen uniformly from a three-wise independent hash family, so that the choice of h_{j_1} is independent of the choice of h_{j_2}. We then have,

$$\mathbb{E}\left[Z_{j_1}^{(i_1)}(d) Z_{j_2}^{(i_2)}(d)\right] = \mathbb{E}\left[Z_{j_1}^{(i_1)}(d)\right] \mathbb{E}\left[Z_{j_2}^{(i_2)}(d)\right].$$

Hence, $\mathbf{Cov}\left(Z_{j_1}^{(i_1)}(d), Z_{j_2}^{(i_2)}(d)\right) = 0$

We next fix $j_1 = j_2 = j$.

$$\begin{aligned}
\mathbb{E}\left[Z_j^{(i_1)}(d) Z_j^{(i_2)}(d)\right] &= c_\epsilon^2 \mathbb{E}\left[BB' H_m[h_j(d), L] H_m[L, h_j(d^{(i_1)})]\ H_m[h_j(d), L'] H_m[L', h_j(d^{(i_2)})]\right] \\
&= \mathbb{E}\left[\frac{1}{m^2} \sum_{\ell=1}^{m} H_m[h_j(d), \ell] H_m[\ell, h_j(d^{(i_1)})] \sum_{\ell'=1}^{m} H_m[h_j(d), \ell'] H_m[\ell', h_j(d^{(i_2)})]\right] \\
&= \mathbb{E}\left[\mathbb{1}\{h_j(d) = h_j(d^{(i_1)})\} \mathbb{1}\{h_j(d) = h_j(d^{(i_2)})\}\right] \\
&= \Pr\left[h_j(d) = h_j(d^{(i_1)}) = h_j(d^{(i_2)})\right]
\end{aligned}$$

We then have three cases we need to consider, if $d^{(i_1)} = d^{(i_2)} = d$, then this probability is 1, and hence the covariance is zero. If $d^{(i_1)} \neq d^{(i_2)}$ and both are not d, then by $\mathcal{H}$ being a three-wise independent hash family we have

$$\begin{aligned}
\Pr\left[h_j(d) = h_j(d^{(i_1)}) = h_j(d^{(i_2)})\right] &= \sum_{a=1}^{m} \Pr\left[h_j(d^{(i_1)}) = a \ \&\ h_j(d^{(i_2)}) = a\right] \Pr\left[h_j(d) = a\right] \\
&= \frac{1}{m^2}
\end{aligned}$$

Hence, in the case that $d^{(i_1)} \neq d^{(i_2)}$ and both are not d, we have $\mathbf{Cov}\left(Z_j^{(i_1)}(d), Z_j^{(i_2)}(d)\right) = \frac{1}{m^2} - \frac{1}{m^2} = 0$. Without loss of generality, let $d^{(i_1)} \neq d^{(i_2)}$ but $d^{(i_1)} = d$. We then have,

$$\Pr\left[h_j(d) = h_j(d^{(i_1)}) = h_j(d^{(i_2)})\right] = \Pr\left[h_j(d) = h_j(d^{(i_2)})\right] = \frac{1}{m}$$

Hence, in the case that $d^{(i_1)} \neq d^{(i_2)}$ and one of them is equal to d, we have $\mathbf{Cov}\left(Z_j^{(i_1)}(d), Z_j^{(i_2)}(d)\right) = \frac{1}{m} - \frac{1}{m} = 0$

Lastly, we compute the cross term when $d^{(i_1)}, d^{(i_2)} = d'$ but $d' \neq d$. In this case we have

$$\Pr\left[h_j(d) = h_j(d^{(i_1)}) = h_j(d^{(i_2)})\right] = \Pr\left[h_j(d) = h_j(d^{(i_1)})\right] = \frac{1}{m}$$

Hence, we can write out the covariance term,

$$\mathbf{Cov}\left(Z_j^{(i_1)}(d), Z_j^{(i_2)}(d)\right) = \frac{1}{m} - \frac{1}{m^2}$$

□

We are now ready to prove the utility result for `HCMS`.

Proof of Theorem 4.4. We first compute the expectation of the output from $\mathcal{A}_{\mathsf{server-HCMS}}$ for a particular data record $d \in \mathcal{D}$, which is a function of the following variable whose terms are given in (5)

$$Z = \frac{1}{k}\sum_{j=1}^{k}\sum_{i=1}^{n} Z_j^{(i)}(d)$$

We can write the output of $\mathcal{A}_{\mathsf{server-HCMS}}$ now as a function of Z

$$\tilde{f}(d) = \frac{m}{m-1}\left(Z - \frac{n}{m}\right)$$

We then compute the expectation of the summation, where we have already computed the terms in Lemma B.1. Recall that we write $f(d)$ to be the true count for data record $d \in \mathcal{D}$.

$$\mathbb{E}\left[Z\right] = \frac{1}{k}\sum_{j=1}^{k}\sum_{i=1}^{n}\mathbb{E}\left[Z_j^{(i)}(d)\right] = f(d) + (n - f(d))\frac{1}{m}$$

Thus, the count that is output by $\mathcal{A}_{\mathsf{server-HCMS}}$ debiases the variable Z. We now compute the variance of Z, for which we have already computed each term in Lemmas B.2 and B.3

$$
\begin{aligned}
\mathbf{Var}\left[Z\right] &= \sum_{i=1}^{n} \mathbf{Var}\left[\frac{1}{k}\sum_{j=1}^{k} Z_j^{(i)}(d)\right] + \frac{1}{k^2}\sum_{i_1 \neq i_2} \mathbf{Cov}\left(\sum_{j=1}^{k} Z_j^{(i_1)}(d), \sum_{j'=1}^{k} Z_{j'}^{(i_2)}(d)\right) \\
&= \frac{1}{k^2}\sum_{i=1}^{n}\left(\sum_{j=1}^{k}\mathbf{Var}\left[Z_j^{(i)}(d)\right] + \sum_{j\neq j'}\mathbf{Cov}\left(Z_j^{(i)}(d), Z_{j'}^{(i)}(d)\right)\right) + \frac{1}{k^2}\sum_{j=1}^{k}\sum_{i_1\neq i_2}\mathbf{Cov}\left(Z_j^{(i_1)}(d), Z_j^{(i_2)}(d)\right) \\
&= nc_\epsilon^2 - f(d) - \frac{(n-f(d))}{m^2} + \frac{1}{k^2}\sum_{j=1}^{k}\sum_{i_1\neq i_2}\mathbf{Cov}\left(Z_j^{(i_1)}(d), Z_j^{(i_2)}(d)\right) \\
&= nc_\epsilon^2 - f(d) - \frac{(n-f(d))}{m^2} + \frac{1}{k}\left(\frac{1}{m}-\frac{1}{m^2}\right)\left(\sum_{d^*\neq d} f(d^*)^2 - (n-f(d))\right) \\
&= nc_\epsilon^2 + f(d)\left(\frac{1}{m^2}+\frac{1}{km}-\frac{1}{km^2}-1\right) - \frac{n}{m}\left(\frac{1}{m}+\frac{1}{k}\left(1-\frac{1}{m}\right)\right) + \frac{1}{k}\left(\frac{1}{m}-\frac{1}{m^2}\right)\sum_{d^*\neq d} f(d^*)^2 \\
&\leq n\left(c_\epsilon^2 + \frac{1}{knm}\sum_{d^*\in\mathcal{D}} f(d)^2\right)
\end{aligned}
$$

□