# Differentially Private Clustering in Data Streams

**Alessandro Epasto** [1] **Tamalika Mukherjee** [2] **Peilin Zhong** [1]

## Abstract

Clustering problems (such as k-means and k-median) are fundamental unsupervised machine learning primitives. Recently, these problems have been subject to large interest in the privacy literature. All prior work on private clustering, however, has been devoted to the *offline* case where the entire dataset is known in advance. In this work, we focus on the more challenging private data stream setting where the aim is to design memory-efficient algorithms that process a large stream *incrementally* as points arrive in a private way. Our main contribution is to provide the first differentially private algorithms for $k$-means and $k$-median clustering in data streams. In particular, our algorithms are the first to guarantee differential privacy both in the continual release and in the one-shot setting while achieving space sublinear in the stream size. We complement our theoretical results with an empirical analysis of our algorithms on real data.

## 1. Introduction

Clustering is an essential primitive in unsupervised machine learning, and its geometric formulations, such as $k$-means and $k$-median, have been studied extensively, e.g., (Arya et al., 2001; Charikar et al., 2002; Har-Peled & Mazumdar, 2004; Chen, 2006; 2008; Awasthi et al., 2010; Ostrovsky et al., 2012; Li & Svensson, 2016; Ahmadian et al., 2020). In this paper, we focus on the study of clustering in the streaming model under the constraint of data privacy.

Differential privacy (DP) (Dwork et al., 2016) has become the de facto standard for preserving data privacy due to its compelling privacy guarantees and mathematically rigorous definition. There is a rich DP literature for clustering in the polynomial-time setting, e.g., (Nissim et al., 2007; Feldman et al., 2009; 2017; Gupta et al., 2010; Balcan et al., 2017; Huang & Liu, 2018; Nissim & Stemmer, 2018; Stemmer & Kaplan, 2018; Ghazi et al., 2020; Cohen et al., 2021) where the focus has been to improve the approximation ratios and achieve efficient algorithms in high-dimensional Euclidean space. More recent works have studied this problem in other models of computation, such as sublinear-time (Blocki et al., 2021) and massively parallel computing (MPC) (Cohen-Addad et al., 2022a;b). However, the study of DP clustering in the streaming model remains vastly unexplored.

### 1.1. Our Results

In this paper we address the problem of clustering in the streaming model in which the input $x_1, \ldots, x_T \in \mathbb{R}^d$ arrives in a stream. We the give the first pure DP $k$-means and $k$-median clustering algorithms that use space sublinear in the size in $T$ for (1) *continual release setting:* where the algorithm must output a clustering at every timestamp $t \in [T]$, and (2) *one-shot setting:* where the algorithm must output a clustering at the end of the stream. As is standard in DP clustering literature, we assume $\Lambda$ is an upper bound on the diameter of the space of input points.

In the following two theorems we assume we are given a non-DP algorithm in the offline setting that can compute a $\rho$-approximation to $k$-means (or $k$-median)—many such algorithms exists with constant approximation (e.g. (Ahmadian et al., 2020)).

**Theorem 1.1.** *There exists an $\varepsilon$-DP algorithm for $k$-means (or $k$-median) in the continual release model such that for every timestamp $t \in [T]$ it outputs a clustering with $\Theta(\rho)d^{O(1)}$-multiplicative error and $\tilde{O}(\frac{k\rho\Lambda^2}{\varepsilon} \cdot (d\log T)^{O(1)})$-additive error in $O(k\log^2(\Lambda)\log(k)poly(\log(T\Lambda k)))$ space.*[1]

**Theorem 1.2.** *There exists an $\varepsilon$-DP algorithm for $k$-means (or $k$-median) in the one-shot model such that it outputs a clustering with $\Theta(\rho)d^{O(1)}$-multiplicative error and $\tilde{O}(\frac{k\rho\Lambda^2}{\varepsilon} \cdot (d\log T)^{O(1)})$-additive error in*

*Equal contribution [1]Google Research NYC, USA [2]Purdue University, USA. Work partially done while a Student Researcher at Google. Correspondence to: Alessandro Epasto <aepasto@google.com>, Tamalika Mukherjee <tmukherj@purdue.edu>, Peilin Zhong <peilinz@google.com>.

*Presented at the $2^{nd}$ Annual Workshop on Topology, Algebra, and Geometry in Machine Learning (TAG-ML) at the $40^{th}$ International Conference on Machine Learning, Honolulu, Hawaii, USA. 2023. Copyright 2023 by the author(s).*

[1]We use the $\tilde{O}(\cdot)$ notation to neglect poly-logarithmic factors in $(\Lambda, k, T)$.

*$O(k\log^2(\Lambda)\log(k)poly(\log(T\Lambda k)))$ space at the end of the stream of length $T$.*

We observe that in both settings the memory used is $\tilde{O}(k)$ (ignoring poly-logarithmic factors in $(\Lambda, k, T)$) thus matching the space requirements of non-DP streaming algorithms (Charikar et al., 2003).

## 1.2. Technical Overview

Our techniques apply to both $k$-means and $k$-median clustering, but we assume we are dealing with $k$-means for simplicity. Before discussing our algorithm in more detail, we first outline the challenges to designing a DP $k$-means clustering algorithm in the continual release setting.

**Natural space-efficient approaches fail.** A natural first approach towards this problem in the one-shot setting and one that was employed in the sublinear-time model (Blocki et al., 2021) is to maintain a random sample using the same space as our proposed algorithm, i.e., $\tilde{O}(k)$ and apply a state-of-the-art DP clustering algorithm on this sample at the end of the stream. One can easily show that this algorithm preserves DP and is as space efficient as our method. However, the accuracy of the resulting clustering achieved will be considerably worse than our proposed algorithm. In the worst case, this approach can lead to an additive error (ignoring $k, \Lambda, d$ dependencies) of $O(\sqrt{T})$ (see Cohen-Addad et al. (2022b) for a detailed exposition). In contrast, our approach leads to an additive error of $O(\text{poly}(\log T))$. We demonstrate this experimentally by showing that our proposed algorithm outperforms the random sampling algorithm we use as a baseline.

**Our Approach.** For every timestamp $t \in [T]$, our algorithm for both continual release and one-shot settings can be split into two main steps — (1) Compute a weighted DP coreset $\mathcal{F}$ in an online fashion that satisfies a bicriteria approximation to $k$-means (see Theorem 1.3). (2) Compute a non-DP $k$-means $\rho$-approximation algorithm on $\mathcal{F}$ in a postprocessing step.

**Theorem 1.3** (Bicriteria approximation)**.** *There exists an $\varepsilon$-DP algorithm that for every timestamp $t \in [T]$, computes a weighted set of $O(k\log(k)\log^2(\Lambda)\log T)$ centers with $d^{O(1)}$-multiplicative error to the best $k$-means (or $k$-median) clustering and $\tilde{O}(\frac{k\rho\Lambda^2}{\varepsilon}\cdot(d\log T)^{O(1)})$-additive error in $O(k\log^2\Lambda\log(k)poly(\log(T\Lambda k)))$ space.*

**Quadtrees and Heavy Hitters.** A quadtree creates a nested series of grids that partitions $\mathbb{R}^d$ and can be used to embed input points into a Hierarchically Separated Tree (HST) metric, which often makes computing $k$-means cost simpler. We use this embedding to map every input point to the center of a grid (or cell) at every quadtree level. For a fixed level, our goal is to approximately choose the $O(k)$ cells that have the most points, i.e., we want to find the "heaviest" cells in a DP fashion and store them as candidate centers in set $\mathcal{F}$. We achieve this by hashing the cells into $O(k)$ substreams and running a continual release black-box DP heavy hitter algorithm on each hash substream. Since with large enough probability, the heavy cells will not collide, this achieves our goal. Note that since we need to do this over logarithmically many levels of the quadtree, we will end up with a bicriteria approximation.

We stress that we need to run a *continual release* black-box DP heavy hitter algorithm for both our continual release and one-shot setting clustering algorithms. This is because we need to assign $x_t$ to a candidate center in $\mathcal{F}$ (obtained from computing the heavy-hitters) in an online fashion at every timestep $t \in [T]$ in both settings. The main difference in our algorithms for these two settings is that in the continual release setting we release the resulting weighted coreset consisting of candidate centers and their noisy weights at every timestep $t \in [T]$, while in the one-shot setting we release the weighted coreset at the end of the stream. Thus, we keep track of the noisy weights in the continual release setting via multiple instantiations of the binary mechanism (Dwork et al., 2010; Chan et al., 2011) while we can add Laplace noise to release the noisy weights at the end of the stream in the one-shot setting.

## 2. Preliminaries

An event $E$ occurs *with high probability* if for any $c \geq 1$, there is an appropriate choice of constants for which $E$ occurs with probability at least $1 - O(1/k^c)$ where $k$ is the $k$-clustering input parameter.

**Norms and heavy hitters.** Let $p \geq 1$, the $\ell_p$-norm of a vector $\mathbf{x} = (x_1, \dots, x_t)$ is defined as $\|\mathbf{x}\|_p = (\sum_{i=1}^t |x_i|^p)^{1/p}$. Given a multiset $\mathcal{S}$, denote the frequency of an item $x$ appearing in $\mathcal{S}$ as $f(x)$. We say that an item $x$ is an $\alpha$-heavy hitter ($\alpha$-HH for short) if $f(x) \geq \alpha\|\mathcal{S}\|_1$.

**Differential Privacy.** Streams $\mathcal{S} = (x_1, \dots, x_T)$ and $\mathcal{S}' = (x'_1, \dots, x'_T)$ are *neighboring* if there exists at most one timestamp $t^* \in [T]$ for which $x_{t^*} \neq x'_{t^*}$ and $x_t = x'_t$ for all $t \neq t^*$.

**Definition 2.1** (Differential privacy Dwork et al. (2016))**.** A randomized algorithm $\mathcal{A}$ is $\varepsilon$-DP if for every pair of neighboring datasets $D \sim D'$, and for all sets $\mathcal{S}$ of possible outputs, we have that $\Pr[\mathcal{A}(D) \in \mathcal{S}] \leq e^\varepsilon \Pr[\mathcal{A}(D') \in \mathcal{S}]$

**Theorem 2.2** (Binary Mechanism BM Chan et al. (2011); Dwork et al. (2010))**.** *Let $\varepsilon \geq 0, \gamma \in (0, 0.5)$, there is an $\varepsilon$-DP algorithm for the sum of the stream in the continual release model. With probability $1-\gamma$, the additive error of the output for every timestamp $t \in [T]$ is always at most $O(\frac{1}{\varepsilon}\log^{2.5}(T)\log(\frac{1}{\gamma}))$ and uses $O(\log T)$ space.*

See (Dwork & Roth, 2014) for more foundational concepts in differential privacy.

**Clustering.** For points $x, y \in \mathbb{R}^d$, we let $d(x,y) = \|x - y\|_2$ be the Euclidean distance between $x$ and $y$. Given a set $\mathcal{C}$, we define $d(x,\mathcal{C}) := \min_{c\in\mathcal{C}} d(x,c)$.

For a set of centers $C$, we define the cost of clustering for the set $\mathcal{S}$ wrt $C$ as

$$cost(C,\mathcal{S}) = \sum_{x\in\mathcal{S}} d^z(x,C)$$

where $z = 1$ for $k$-median, and $z = 2$ for $k$-means.

Our goal in DP clustering is to produce a set of $k$ centers $C_{\mathcal{S}}$ for input stream $\mathcal{S}$ such that (1) $C_{\mathcal{S}}$ is $\varepsilon$-DP wrt $\mathcal{S}$, and (2) $cost(C_{\mathcal{S}},\mathcal{S}) \le \alpha \cdot cost(C_{\mathcal{S}}^{opt},\mathcal{S}) + \beta$.

## 3. Bicriteria Approximation in Continual Release Setting

We describe our algorithm in more detail here. We focus on the $k$-means problem in the sequel, however we stress that our techniques easily extend to the $k$-median problem and the algorithm and analysis are nearly identical.

**Algorithm.** Our main algorithm is given by Algorithm 1 which initializes $\log\Lambda$ parallel instances of randomly shifted quadtrees. At every timestep $t \in [T]$ the input point $x_t$ is assigned to a cell in the $\log\Lambda$ many levels of every quadtree. For a fixed quadtree, the subroutine DPFindCenters (see Algorithm 2) is called on every level. The subroutine DPFindCenters returns a candidate set of centers $\hat{\mathcal{F}}_t$ which is first added to the current set of candidate set of centers $\mathcal{F}$, and $x_t$ is then assigned to the nearest center in $\mathcal{F}$. Finally, the DP counts of all centers in $\mathcal{F}$ are updated via the Binary Mechanism.

The DPFindCenters subroutine (see Algorithm 2) finds the approximate heaviest $O(k)$ cells in a fixed level of a fixed quadtree. It achieves this by first hashing all the cells in that level to $w := O(k)$ many substreams (or buckets) $\mathcal{B}_j$ for all $j \in [w]$ and then runs a continual release $\alpha$-heavy hitter algorithm DP-HH on each bucket.[2] We use the $\ell_1$-heavy hitter algorithm from (Epasto et al., 2023) as DP-HH — it returns a set $H$ of $\alpha$-heavy hitters and their approximate counts $\hat{f}(\mathbf{c})$ for all $\mathbf{c} \in H$. Since we are storing all the cells marked as heavy hitters as candidate centers over at most $T$ timesteps, we need to ensure that we do not store too many false positives, i.e., cells whose counts are much smaller than $\alpha\|\mathcal{B}_j\|_1$. To address this challenge, we have an additional pruning step that eliminates any cell $\mathbf{c}$ whose approximate count is less than $\Theta(\alpha)\hat{T}_{\mathcal{B}_{h(\mathbf{c})}}$ where $\hat{T}_{\mathcal{B}_{h(\mathbf{c})}}$ denotes the DP size of the hash stream $\mathcal{B}_{h(\mathbf{c})}$ at timestep $t \in [T]$. We keep track of $\hat{T}_{\mathcal{B}_{h(\mathbf{c})}}$ via another instance of the Binary Mechanism. Finally, only the cells that pass this pruning step are added as candidate centers to the set $\hat{\mathcal{F}}_t$.

[2]Notice that in the pseudo code Algorithm 2, $\perp$ represents an empty update that does not affect the counters of elements of the stream and is ignored. This is needed for technical reasons to ensure DP by avoiding the value of the hash affecting the number of events in the sub-streams.

**Algorithm 1** DP Clustering Algorithm in Continual Release Setting

```
Require: Privacy parameter $\varepsilon$, Threshold parameter for heavy hitters $\alpha$, Time bound $T$, Binary Mechanism BM, Continual Release DP-HH algorithm, Stream $\mathcal{S}$ of points $x_1,\dots,x_T \in \mathbb{R}^d$
Ensure: Set of DP centers $\mathcal{F}$ and their noisy weights DP-Coreset at every timestep $t$
1: $\varepsilon' := \frac{\varepsilon}{\log^2\Lambda\log k}$
2: Initialize hashmap DPCoreset to empty{used to store centers and noisy weights}
3: Initialize parallel quadtrees $Q_1,\dots,Q_{\log(\Lambda)}$ as follows: Initialize each quadtree $Q_q$ as $\mathcal{S}_1^{(q)},\dots,\mathcal{S}_{\log(\Lambda)}^{(q)}$ parallel streams or levels with the bottom stream/level having grid size $\Theta(1)$
4: Initialize $\mathcal{F} := \emptyset$
5: for $t = 1$ to $T$ do
6:   for each $\mathcal{S}_\ell^{(q)}$ where $0 \le \ell \le \log(\Lambda)$ and $1 \le q \le \log(\Lambda)$ do
7:     Add $x_t$ to $\mathcal{S}_\ell^{(q)}$
8:     $\hat{\mathcal{F}}_t$ = DPFindCenters($\varepsilon', \mathcal{S}_\ell^{(q)}$)
9:     $\mathcal{F} = \mathcal{F} \cup \hat{\mathcal{F}}_t$
10:    {add new centers to hashmap DPCoreset and initialize their DP weights}
11:    for $\mathbf{c} \in \hat{\mathcal{F}}_t - \mathcal{F}$ do
12:      Add $\mathbf{c}$ as a key to DPCoreset
13:      Initialize an instance of $\mathsf{BM}_{\mathbf{c}}(T,\varepsilon',0)$ for DPCoreset($\mathbf{c}$)
14:    end for
15:    if $\mathcal{F} \ne \emptyset$ then
16:      Let $\mathbf{c}^* := \mathrm{argmin}_{\mathbf{c}\in\mathcal{F}}\, d(x_t,\mathbf{c})$ {assign $x_t$ to the nearest center; if $\mathcal{F} = \emptyset$ then discard $x_t$}
17:      DPCoreset($\mathbf{c}^*$) = $\mathsf{BM}_{\mathbf{c}^*}(T,\varepsilon,1)$
18:    end if
19:    for $\mathbf{c} \ne \mathbf{c}^*$ do
20:      DPCoreset($\mathbf{c}$) = $\mathsf{BM}_{\mathbf{c}}(T,\varepsilon,0)$
21:    end for
22:  end for
23:  Output $\mathcal{F}$, DPCoreset
24: end for
```

**Theorem 3.1.** *Let $\mathcal{S} := \{x_1,\dots,x_T\}$ be the stream of input points in Euclidean space. For $t = 1,\dots,T$, let $\mathcal{F}_t$ be the set of centers until time step $t$. Let $cost(\mathcal{F},\mathcal{S}) := \sum_{t=1}^{T} cost(\mathcal{F}_t)$ where $cost(\mathcal{F}_t) := \min_{f\in\mathcal{F}_t} dist^2(x_t,f)$.*

**Algorithm 2** DPFindCenters

```
Require: Privacy parameter $\varepsilon'$, Stream $\mathcal{S}_\ell$ with $2^\ell$ cells (representing the $\ell$-th level of quadtree instantiation), Binary Mechanism BM
Ensure: Set of candidate centers $\hat{\mathcal{F}}_t$ at every timestep $t \in [T]$
1:  Initialize $\hat{\mathcal{F}}_t = \emptyset$
2:  Let $w = O(k)$
3:  Initialize $\hat{T}_{\mathcal{B}_1}, \dots, \hat{T}_{\mathcal{B}_w}$ {DP Count for the size of hash bucket}
4:  for $p = 1, \dots, L$, where $L := O(\log k)$ run in parallel do
5:      Initialize hash function $h : [2^\ell] \to [w]$ s.t. $\forall \mathbf{c}, \forall j \in [w], \Pr[h(\mathbf{c}) = j] = \frac{1}{w}$
6:      Initialize empty hash streams $\mathcal{B}_1, \dots, \mathcal{B}_w$
7:      for each cell $\mathbf{c}$ at level $\ell$ do
8:          Append $\mathbf{c}$ to $\mathcal{B}_{h(\mathbf{c})}$ and append $\perp$ to the end of every stream $\mathcal{B}_j$ such that $j \neq h(\mathbf{c})$.
9:          $\hat{T}_{\mathcal{B}_{h(\mathbf{c})}} = \mathsf{BM}_{h(\mathbf{c})}(T, \varepsilon', 1)$
10:         for $j \neq h(\mathbf{c})$ do
11:             $\hat{T}_{\mathcal{B}_j} = \mathsf{BM}_j(T, \varepsilon', 0)$
12:         end for
13:     end for
14:     for $j \in [w]$ do
15:         $\hat{f}, H \leftarrow$ DP-HH$(\varepsilon', \mathcal{B}_j)$ {$\varepsilon' := \frac{\varepsilon}{\log^2 \Lambda \log k}$}
16:         for $\mathbf{c} \in H$ do
17:             if $\hat{f}(\mathbf{c}) \geq \frac{\alpha}{1000} \cdot \hat{T}_{\mathcal{B}_{h(\mathbf{c})}}$ then
18:                 Append $\mathbf{c}$ to $\hat{\mathcal{F}}_t$ as a center
19:             end if
20:         end for
21:     end for
22: end for
23: Return $\hat{\mathcal{F}}_t$
```

*There exists an algorithm $\mathcal{A}$ (see Algorithm 1) that outputs a set of centers $\mathcal{F}$ and their corresponding weights* DPCoreset *at every timestep $t \in [T]$ such that*

1. *(Privacy) $\mathcal{A}$ is $3\varepsilon$-DP.*
2. *(Accuracy) With high probability,*

$$cost(\mathcal{F}, \mathcal{S}) \leq O(d^3) cost(C_{\mathcal{S}}^{opt}, \mathcal{S}) + \tilde{O}\left(\frac{d^2 \Lambda^2 k}{\varepsilon} \log^C (T \cdot k \cdot \Lambda)\right)$$

   *where $cost(C_{\mathcal{S}}^{opt}, \mathcal{S})$ is the optimal $k$-means cost for $\mathcal{S}$.*
3. *(Space) $\mathcal{A}$ uses $O(k \log^2(\Lambda) \log(k) poly(\log(T\Lambda k)))$ space.*
4. *(Size of $\mathcal{F}$) $\mathcal{F}$ has at most $O(k \log(k) \log^2(\Lambda) \log T)$ centers.*

**Privacy.** Since we are outputting the center point of the cells marked as heavy hitters and their respective noisy counts, we only need to show that DP is maintained wrt these centers and noisy counts of centers and hash substreams. An input point is assigned to a specific cell for a specific level of the quadtree, and cells at the same level are disjoint. Since there are $\log \Lambda$ levels per quadtree, a point is a member of $\log \Lambda$ cells. Since there are $\log \Lambda \log k$ parallel processes, a single point participates in $\log^2 \Lambda \log k$ total calls to DP-HH. Note that we do not account for the $O(k)$ buckets that the cells are hashed into, as DP-HH is called on disjoint inputs for each bucket. Thus calling each DP-HH instance with a privacy budget of $\frac{\varepsilon}{\log^2 \Lambda \log k}$ preserves $\varepsilon$-DP. We use the Binary Mechanism to keep track of the size of each hash substream $\mathcal{B}_j \; \forall j \in [w]$. Since the input cells (and corresponding points within cells) are disjoint in each substream due to hashing, this preserves $\frac{\varepsilon}{\log^2 \Lambda \log k}$-DP which over $\log^2 \Lambda \log k$ parallel processes preserves $\varepsilon$-DP. Finally, we release the number of points per center via the Binary Mechanism where each point can only contribute to a single cell count which preserves $\varepsilon$-DP. Therefore by composition, we get $3\varepsilon$-DP for the entire algorithm.

**Accuracy.** We first state some geometric properties regarding the cells within the quadtree construction.

**Proposition 3.2.** *(Cohen-Addad et al., 2022b) Let $\mathcal{B}$ be an $\ell_\infty$ ball of radius $r$ contained in $[-\Lambda, \Lambda]^d$ (it forms a $d$-dimensional cube with each side length $2r$). Then for a randomly shifted quadtree and any level $\ell$ with grid size at least $r' \geq 2r$, $\mathcal{B}$ is split by the grid in each dimension $j \in [d]$ independently with probability $\frac{2r}{r'}$.*

Let $C_{\mathcal{S}}^{opt} = \{c_1, \dots, c_k\}$ be the optimal set of $k$ centers for the input set $\mathcal{S} = \{x_1, \dots, x_T\}$. For any radius, define $n_r$ as the number of points $x \in \mathcal{S}$ such that $d(x, C_{\mathcal{S}}^{opt}) \geq r$. Note that the opt cost of $k$-means (and $k$-median) is given by $\sum_{p \in \mathbb{Z}} 2^{2p} \cdot n_{2^p}$ and $\sum_{p \in \mathbb{Z}} 2^p \cdot n_{2^p}$ (up to an $O(1)$-approximation).

Fix some radius $r = 2^p$ where $p \in \mathbb{Z}$ and consider a randomly shifted grid of size $20rd$. The following lemma characterizes cells containing $\cup_{i=1}^k \mathcal{B}(c_i, r)$ with respect to the grid size.

**Lemma 3.3.** *(Cohen-Addad et al., 2022b) $\cup_{i=1}^k \mathcal{B}(c_i, r)$ is contained in at most $4k$ cells of grid length $20rd$ by the corresponding level of the quadtree with probability at least 1/2.*

Let $G_\ell$ where $0 \leq \ell \leq \log(\Lambda)$ be the set of $4k$ good cells of length $20rd$ (equivalently $\ell_2$-radius of $10rd^{3/2}$) at level $\ell$. Let the number of points in $\mathcal{S}$ uncovered by $G_\ell$ be $n_{G_\ell}$. Observe that by Lemma 3.3, since $G_\ell$ contains $\cup_{i=1}^k \mathcal{B}(c_i, r)$ with probability at least 1/2, we have that $n_{G_\ell} \leq n_r$. It follows that

$$\sum_{\ell=0}^{\log(\Lambda)} (\text{grid length at level } \ell)^2 \cdot n_{G_\ell}$$
$$\leq O(d^3) \sum_{p\in\mathbb{Z}\,:\, r=2^p\leq\Lambda} r^2 \cdot n_r \leq O(d^3) \cdot cost(C_{\mathcal{S}}^{opt}, \mathcal{S}) \tag{1}$$

Observe that we can define a one-one mapping between the level $\ell$ and the radius $r$, i.e., the radius $r$ (ranging from 1 to $\Lambda$) maps to the grid length of a cell which is at most $\Lambda/2^\ell$ (level $\ell$ ranges from $\log(\Lambda)$ to 0). Since the grid length of a cell in $G_\ell$ at level $\ell$ is $20rd$ which maps to $20d\frac{\Lambda}{2^\ell}$, we can replace the leftmost term in the expression above as follows

$$O(d^2) \sum_{\ell=0}^{\log(\Lambda)} (\Lambda/2^\ell)^2 n_{G_\ell} \leq O(d^3) \sum_{p\in\mathbb{Z}\,:\, r=2^p\leq\Lambda} r^2 \cdot n_r$$
$$\leq O(d^3) \cdot cost(C_{\mathcal{S}}^{opt}, \mathcal{S}) \tag{2}$$

Recall that we define $\mathcal{F}_t$ as the set of centers until time step $t$. For a fixed level $\ell$, let the set of cells the algorithm DP-HH marks as heavy at timestep $t$ at level $\ell$ as $H_{\ell,t}$. Note that although there is an extra pruning step in DPFindCenters after the cells are marked heavy by DP-HH, we do not account for this here — as if a cell is an $\alpha$-HH and marked heavy by DP-HH, and it survives the pruning step, it will still be an $\alpha$-HH. Then,

$$cost(\mathcal{F}_t) \leq O(d^2) \sum_{\ell=0}^{\log(\Lambda)} (\Lambda/2^\ell)^2 \cdot \mathbb{1}[x_t \text{ uncovered by } H_{\ell,t}]$$

Observe that

$$cost(\mathcal{F})$$
$$= \sum_{t=1}^{T} cost(\mathcal{F}_t)$$
$$\leq O(d^2) \sum_{\ell=0}^{\log(\Lambda)} (\Lambda/2^\ell)^2 \cdot \sum_{t=1}^{T} \mathbb{1}[x_t \text{ uncovered by } H_{\ell,t}] \tag{3}$$

**Lemma 3.4.** *For a fixed level $\ell$, with probability at least $1-\frac{12}{k}$,*

$$\sum_{t=1}^{T} \mathbb{1}[x_t \textit{ uncovered by } H_{\ell,t}] \leq (1+\alpha) n_{G_\ell}$$
$$+ \frac{4k\log^2\Lambda\log k}{\varepsilon\eta} poly\left(\log\left(\frac{T\cdot 2^\ell}{\alpha\xi\eta}\right)\right) \tag{4}$$

*Proof.* Observe that

$$\sum_{t=1}^{T} \mathbb{1}[x_t \text{ uncovered by } H_{\ell,t}]$$
$$= \sum_{t=1}^{T} (\mathbb{1}[(x_t \text{ uncovered by } H_{\ell,t}) \wedge (x_t \text{ uncovered by } G_\ell)]$$
$$+ \mathbb{1}[(x_t \text{ uncovered by } H_{\ell,t}) \wedge (x_t \text{ covered by } G_\ell)])$$
$$= \sum_{t=1}^{T} \mathbb{1}[(x_t \text{ uncovered by } H_{\ell,t}) \wedge (x_t \text{ uncovered by } G_\ell)]$$
$$+ \sum_{t=1}^{T} \mathbb{1}[(x_t \text{ uncovered by } H_{\ell,t}) \wedge (x_t \text{ covered by } G_\ell)]$$

The first sum in the above expression can be upper bounded by $n_{G_\ell}$, thus it remains to bound the second sum. In order to bound the second sum, we will need some properties of good cells that are hashed to buckets in DPFindCenters. In the sequel, we denote $N_{\ell,\mathbf{c}}$ as the number of points in the cell $\mathbf{c}$ at level $\ell$. For simplicity, we consider the number of hash buckets $w := 40k$. We first show that for any good cell $\mathbf{c}$, it is unlikely that the bucket it is hashed to contains another good cell $\mathbf{c}' \neq \mathbf{c}$.

*Claim* 1. Let $\mathbf{c} \in G_\ell$, then with probability at least $1/2$, for any $\mathbf{c}' \in G_\ell$ such that $\mathbf{c}' \neq \mathbf{c}$, we have that $h(\mathbf{c}') \neq h(\mathbf{c})$.

In the next claim we give a bound on the size of the hash bucket in terms of the size of a good cell that is hashed to it and $n_r$.

*Claim* 2. For each $\mathbf{c} \in G_\ell$, suppose the hash bucket $\mathcal{B}_j$ where $j \in [w]$, contains only one good cell which is $\mathbf{c}$. Let $N_{\ell,\mathbf{c}} := y$. Then with probability at least 1/2, $|\mathcal{B}_j| \leq 2(y + \frac{n_{G_\ell}}{40k})$.

Note that since the hashing procedure is run $O(\log k)$ times in parallel, we can boost the success probabilities in the above claims to be sufficiently high, e.g., $1 - 1/k^2$.

Observe that for a fixed hash bucket $\mathcal{B}_j$, any cell $\mathbf{c}$ such that $N_{\ell,\mathbf{c}} \geq \alpha \cdot 2(y + \frac{n_{G_\ell}}{40k})$ qualifies as an $\alpha$-heavy hitter since $N_{\ell,\mathbf{c}} \geq \alpha \cdot 2(y + \frac{n_{G_\ell}}{40k}) \geq \alpha|\mathcal{B}_j|$ (by Claim 2). In particular, for good cell $\mathbf{c}_y$ such that $N_{\ell,\mathbf{c}_y} = y$, if $\mathbf{c}_y$ is an $\alpha$-HH then $y \geq \alpha \cdot 2(y + \frac{n_{G_\ell}}{40k})$ or $y \geq \frac{\alpha n_{G_\ell}}{20k}$. We formalize this intuition in the claim below where we use the accuracy guarantees of DP-HH given by Theorem A.1 to characterize the good cells that are reported as $\alpha$-HHs.

*Claim* 3. Let $\mathbf{c} \in G_\ell$. If $N_{\ell,\mathbf{c}} \geq \frac{\alpha n_{G_\ell}}{20k}$, and $N_{\ell,\mathbf{c}} \geq \frac{\log^2\Lambda\log k}{\varepsilon\eta}\text{poly}(\log(\frac{T\cdot k\cdot 2^\ell}{\alpha\eta}))$, then with probability at least $1-\frac{12}{k}$, $\mathbf{c}$ is reported as an $\alpha$-heavy hitter by DP-HH.

Finally, we give an upper bound for the number of points that are covered by good cells but for which DP-HH fails to report as heavy.

*Claim* 4. With probability $1-\frac{12}{k}$,

$$\sum_{t=1}^{T} \mathbb{1}[(x_t \text{ uncovered by } H_{\ell,t}) \wedge (x_t \text{ covered by } G_\ell)]$$

$$\leq \alpha n_{G_\ell} + \frac{4k\log^2\Lambda\log k}{\varepsilon\eta}\text{poly}\left(\log\left(\frac{T\cdot k\cdot 2^\ell}{\alpha\eta}\right)\right)$$

Thus by combining Claim 4 with our observation about the first sum being upper bounded by $n_{G_\ell}$ in the decomposition of $\sum_{t=1}^{T}\mathbb{1}[x_t \text{ uncovered by } H_{\ell,t}]$, we obtain our desired statement in Lemma 3.4. □

Not that we have shown Lemma 3.4 is true with probability at least $1-\frac{12}{k}$, for a fixed level. Since we have $\log(\Lambda)$ many levels in a specific quadtree, and $\log(\Lambda)$ many quadtree instances in parallel — we can boost our probability of success to be sufficiently high. It remains to bound the total $k$-means cost for the set of centers $\mathcal{F}$ output by our algorithm. Combining Equation (1), Equation (2) and Equation (3) along with Lemma 3.4, we obtain the following.

$$
\begin{aligned}
&cost(\mathcal{F},\mathcal{S})\\
&=\sum_{t=1}^{T} cost(\mathcal{F}_t,\mathcal{S})\\
&\leq O(d^2)\sum_{\ell=0}^{\log(\Lambda)}(\Lambda/2^\ell)^2\cdot\sum_{t=1}^{T}\mathbb{1}[x_t \text{ uncovered by } H_{\ell,t}]\\
&\leq O(d^2)\sum_{\ell=0}^{\log(\Lambda)}(\Lambda/2^\ell)^2\cdot((1+\alpha)n_{G_\ell}\\
&+\frac{4k\log^2\Lambda\log k}{\varepsilon\eta}\text{poly}(\log(\frac{T\cdot k\cdot 2^\ell}{\alpha\eta})))\\
&=O(d^2)\sum_{\ell=0}^{\log(\Lambda)}(\Lambda/2^\ell)^2\cdot(1+\alpha)n_{G_\ell}\\
&+O(d^2)\sum_{\ell=0}^{\log(\Lambda)}(\Lambda/2^\ell)^2\cdot\frac{4k\log^2\Lambda\log k}{\varepsilon\eta}\text{poly}(\log(\frac{T\cdot k\cdot 2^\ell}{\alpha\eta}))\\
&\leq O(d^3)(1+\alpha)\sum_{p\in\mathbb{Z}\,:\,r=2^p\leq\Lambda} r^2\cdot n_r\\
&+O(d^2)\sum_{\ell=0}^{\log(\Lambda)}(\Lambda/2^\ell)^2\cdot\frac{4k\log^2\Lambda\log k}{\varepsilon\eta}\text{poly}(\log(\frac{T\cdot k\cdot 2^\ell}{\alpha\eta}))\\
&\leq O(d^3)(1+\alpha)cost(C_{\mathcal{S}}^{opt},\mathcal{S})\\
&+O(d^2)\sum_{\ell=0}^{\log(\Lambda)}(\Lambda/2^\ell)^2\cdot\frac{4k\log^2\Lambda\log k}{\varepsilon\eta}\text{poly}(\log(\frac{T\cdot k\cdot 2^\ell}{\alpha\eta}))\\
&=O(d^3)(1+\alpha)cost(C_{\mathcal{S}}^{opt},\mathcal{S})\\
&+O\left(d^2\Lambda^2\frac{k\log^2\Lambda\log k}{\varepsilon\eta}\text{poly}(\log(\frac{T\cdot k\cdot\Lambda}{\alpha\eta}))\right)
\end{aligned}
$$

Finally, we can set $\alpha$ (threshold for HHs) and $\eta$ (approximation factor for frequency of a cell marked as heavy from Theorem A.1) to some constants. The accuracy claim in Theorem 3.1 follows.

**Space.** We analyze the total space usage for DP-HH in Algorithm 2 as this dominates space usage for the entire algorithm. From Theorem A.1, one instance of DP-HH uses $\text{poly}(\log(T\Lambda k))$. Since we run DP-HH on $O(k)$ many hash substreams, and $\log^2\Lambda\log k$ parallel processes, the total space is $O(k\log^2(\Lambda)\log(k)\text{poly}(\log(T\Lambda k)))$.

*Claim* 5 (Upper bound on size of $\mathcal{F}$). For all $j\in[w]$, suppose $|\mathcal{B}_j| = \Omega(\frac{\log^2(\Lambda)\log^2(k)}{\varepsilon}\log^{2.5}T)$ then with high probability, the total number of historical heavy hitters at the end of the stream is $O(k\log(k)\log^2(\Lambda)\log T)$.

*Proof.* The algorithm runs independent instances of the DP-HH algorithm for each bucket of each level in each instantiation of the quadtree, thus it is sufficient to first show that for a fixed quadtree $Q$, a fixed level $\ell$, and a fixed bucket $\mathcal{B}_j$ where $j\in[w]$, the total number of historical heavy hitters is at most $O(\frac{(1+\eta)}{\alpha}\log T)$.

Let the timestamps of points that end up in $\mathcal{B}_j$ be $t_i = 2^i$, where $0\leq i\leq\log(T)$. Let the state of the hash bucket at time step $t$ be $\mathcal{B}_j^{(t)}$. We set the failure probability in Theorem 2.2 as $\gamma := \frac{1}{k^2}$. From Theorem 2.2 we know that with probability $1-\frac{1}{k^2}$, the DP count of the hash bucket $\hat{T}_j$ at timestep $t$ has additive error $O(\frac{\log^2(\Lambda)\log^2(k)}{\varepsilon}\log^{2.5}(T))$. Thus for a fixed timestamp $t$, if $|\mathcal{B}_j^{(t)}| = \Omega(\frac{\log^2(\Lambda)\log^2(k)}{\varepsilon}\log^{2.5}(T))$, then we can see that a cell $\mathbf{c}$ is added to $\mathcal{F}_t$ only if $\hat{f}(\mathbf{c}) > \frac{2\alpha}{1000}|\mathcal{B}_j^{(t)}|$. Recall from Condition 1 of Theorem A.1 that $\hat{N}_{\ell,\mathbf{c}}\in(1\pm\eta)N_{\ell,\mathbf{c}}$. Thus if $\mathbf{c}\in\mathcal{F}_t$ and $|\mathcal{B}_j^{(t)}| = \Omega(\frac{\log^2(\Lambda)\log^2(k)}{\varepsilon}\log^{2.5}(T))$ then it must be the case that with probability $1-\frac{1}{k^2}$, $N_{\ell,\mathbf{c}}\geq\frac{2\alpha}{1000(1+\eta)}\|\mathcal{B}_j^{(i)}\|_1\geq\frac{2\alpha}{1000(1+\eta)}t_{i-1}$.

Now, suppose for a contradiction, that the number of heavy hitters between $t_{i-1}$ and $t_i$ is at least $\frac{1000(1+\eta)}{\alpha}$. Then for each such cell $\mathbf{c}$, we have that $N_{\ell,\mathbf{c}}\geq\frac{2\alpha}{1000(1+\eta)}t_{i-2}$. Since there are at least $\frac{2000(1+\eta)}{2\alpha}$ such cells, this implies that the total number of points between $t_{i-1}$ and $t_i$ is $\geq\frac{2\alpha}{1000(1+\eta)}t_{i-2}\frac{2000(1+\eta)}{2\alpha} = 2^i = t_i$, which is a contradiction. Thus there must be at most $\frac{1000(1+\eta)}{\alpha}$ cells marked as heavy hitters between consecutive intervals, and since there are $\log(T)$ such intervals, we have that the total number of historical $\ell_1$ heavy hitters for a fixed bucket is $O(\frac{(1+\eta)}{\alpha}\log T)$.

Boosting the success probability over $O(k)$ buckets, $O(\log(k))$ parallel processes, $\log(\Lambda)$ such levels, and $\log(\Lambda)$ parallel processes of the quadtree instantiation, accounting for the additional number of historical HHs, and taking $\alpha$ and $\eta$ as constants, we obtain the claim as stated. □

## 4. From Bicriteria Approximation to $k$-Clustering

Suppose we have a non-DP $k$-means algorithm $\mathcal{A}'$ that gives a $\rho$-approximation. We run $\mathcal{A}'(\mathsf{DPCoreset})$ where DPCoreset is the output of Algorithm 1. Note that by postprocessing, this computation preserves privacy.

For simplicity we denote the centers and their corresponding noisy weights in DPCoreset as a tuple $(\mathcal{F}, \hat{w})$. Let $C_{\mathcal{F},\hat{w}}$ denote the $k$-clustering obtained as output from $\mathcal{A}'((\mathcal{F}, \hat{w}))$. We need to show that $cost(C_{\mathcal{F},\hat{w}}, \mathcal{S})$ is reasonably bounded by the optimal cost of clustering of $\mathcal{S}$ denoted as $cost(C^{opt}_{\mathcal{S}}, \mathcal{S})$.

**Theorem 4.1.** *Let $\mathcal{S} = \{x_1, \ldots, x_T\}$. Suppose $C^{opt}_{\mathcal{S}}$ is the optimal set of centers for $\mathcal{S}$. Then*

$$cost(C_{\mathcal{F},\hat{w}}, \mathcal{S}) \leq (2\rho + 1)O(d^3)cost(C^{opt}_{\mathcal{S}}, \mathcal{S}) + \tilde{O}(\frac{k\rho\Lambda^2}{\varepsilon} \cdot (d \log T)^{O(1)})$$

*Proof.* Let $(\mathcal{F}, w)$ denote the set of centers $\mathcal{F}$ and their non-DP weights and let the optimal cost of clustering for $(\mathcal{F}, w)$ be $C^{opt}_{\mathcal{F},w}$. The proof proceeds as follows,

$$cost(C_{\mathcal{F},\hat{w}}, \mathcal{S}) < cost(C_{\mathcal{F},\hat{w}}, (\mathcal{F}, \hat{w})) + cost(\mathcal{F}, \mathcal{S}) \tag{5}$$

$$cost(C_{\mathcal{F},\hat{w}}, (\mathcal{F}, \hat{w})) < cost(C_{\mathcal{F}.w}, (\mathcal{F}, w)) + O(k\Lambda^2 + \frac{|\mathcal{F}|\Lambda^2}{\varepsilon} \log^{2.5}(|T|) \log(k)) \tag{6}$$

$$cost(C_{\mathcal{F}.w}, (\mathcal{F}, w)) < \rho \cdot cost(C^{opt}_{\mathcal{F},w}, (\mathcal{F}, w)) \tag{7}$$

$$cost(C^{opt}_{\mathcal{F},w}, (\mathcal{F}, w)) < 2cost(\mathcal{F}, \mathcal{S}) + |\mathcal{F}|\Lambda^2 \tag{8}$$

Note that Equation 5 follows from triangle inequality. Equation 6 follows from Claim 6. Equation 7 follows from $\mathcal{A}'$ giving a $\rho$-approximation. And finally, Equation 8 follows from Claim 7. Also, recall that we have an upper bound for $cost(\mathcal{F}, \mathcal{S})$ in terms of $cost(C^{opt}_{\mathcal{S}}, \mathcal{S})$ from Theorem 3.1. Using the upper bound on $|\mathcal{F}|$ from Claim 5 and simplifying the additive error terms, the theorem statement follows. □

## 5. Bicriteria Approximation in One-Shot Setting

The algorithm in the one-shot setting is the same as in Algorithm 1, except we do not need to release the noisy weights of centers in DPCoreset via the Binary Mechanism at every timestep $t \in [T]$. Instead, we can output the DPCoreset at the end of the stream by adding noise via the Laplace Mechanism. Crucially, our one-shot algorithm still needs to use the continual release algorithm for finding heavy-hitters denoted as DP-HH as a subroutine. Thus, the guarantees of this algorithm are the same as Theorem 3.1.

## 6. Experiments

In this section we provide an empirical evaluation of our algorithms. First, we describe the methodology of our empirical evaluation and then we present the results. To foster the reproducibility of our work, all datasets used are *publicly-available*. Moreover, we will make our code available *open-source* before the camera-ready. We stress that no dataset contains private user data.

In all our experiments we focus on the one-shot setting (as opposed to the continual release one). All private algorithms evaluated are one-shot and we report the cost of all algorithms (private and non-private) at the end of the stream. We repeat all experiments with randomized algorithms 10 times and report mean and deviation statistics.

**Datasets.** In our experimental analysis we used real-world datasets obtained from the public UCI Repository (Dheeru & Karra Taniskidou, 2017) that have been used in various experiments on $k$-clustering. We used **skin** (Bhatt & Dhall, 2010), $n = 245057, d = 4$, **shuttle**, $n = 58000, d = 10$, and **rangequeries** (Savva et al., 2018), $n = 200000, d = 6$. We stream all points in the natural order (as they are stored in the dataset). In all datasets, we pre-process the points to be in the $\ell_\infty$ bounding box of $[0, 2]^d$.

**Metrics.** We report as metrics the $k$-means cost of the clustering and the maximum space requirement of our algorithms over the stream. In order to assess the memory requirement in an implementation-independent way, we use the number of points stored–this is consistent with prior work Borassi et al. (2020).

**Modifications to our algorithm.** In our experiments we implemented a simplified version of our one-shot algorithm, where we made several modifications to make it more space efficient. These changes are similarly to that in past streaming clustering papers (Lattanzi & Vassilvitskii, 2017; Borassi et al., 2020) and have the additional advantage of reducing the noise required for DP by limiting the splitting of the privacy budget. We stress that these changes do not affect the privacy guarantees. (1) We do not repeat $O(\log(\Lambda))$ independent times the grid creation, but instead use a single instance. The grid is fixed to have 5 levels only (instead of $\log(\Lambda)$). (2) We do not split the stream using the hash function in $w$ buckets (but use a single heavy hitter instance for each layer).

**One-shot DP heavy hitter subroutine.** As DP-HH subroutine we implemented a version of the simpler DP Misra-Gries-based algorithm of Chan et al. (2012) instead of the

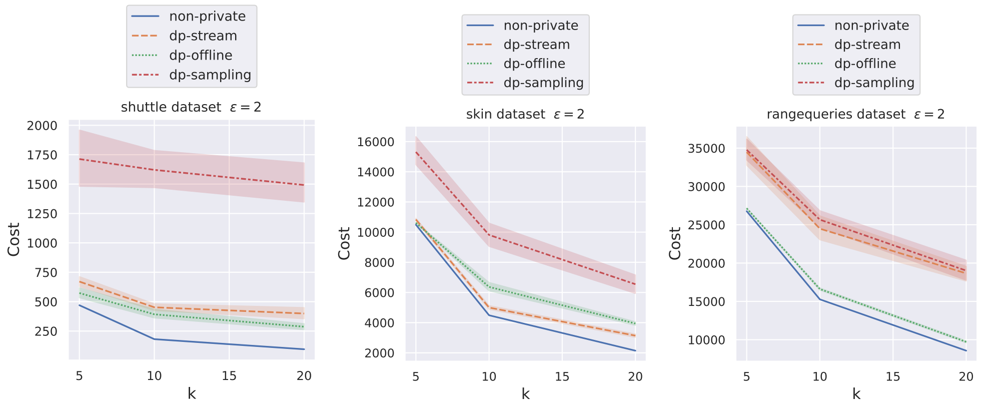

*Figure 1.* k-means cost vs $k$ for different datasets at $\varepsilon = 2$, $\delta = 0.001$ for various algorithms. Shades represent the 95% C.I..

| $\varepsilon$ | Sample | cost | total memory |
|---|---|---|---|
| 2.0 | .5% | 4999.5 | .63% |
| 5.0 | .5% | 5118.4 | .62% |
| 2.0 | 1% | 5090.0 | 1.1% |
| 5.0 | 1% | 4912.5 | 1.1% |

*Table 1.* Cost and total memory for different $\varepsilon$ and sample rates for the skin dataset, $k = 10$ and $\delta = .001$.

algorithm in Epasto et al. (2023). In our experiments we set the $\alpha = 1/32$ for the $\alpha$-HH parameter.

Finally, as we operate in the one-shot setting, we made a further simplification to the method to assign weights to the centers selected as heavy hitters. We keep a sample of *sampling rate* fraction of the stream in memory, for *sampling rate* $\in [0.5\%, 1\%]$, and assign, at the end of the stream, all points in the sample to the nearest heavy hitters. Then, we use a DP count of the assigned point as weight of the heavy hitter. We report the results of this modified version of our algorithm as **dp-stream**.

**Baselines.** We consider as well the following baselines.
**non-private**: We use the standard non-private sklearn solver for k-means. This is also the solver used in all private algorithms as sub-routine when needed. This has access to the whole dataset and has no privacy so we consider it the gold standard for cost.

**dp-offline**: We consider an open-source DP k-means algorithm part of a standard DP library[3] that works in the offline setting (i.e. accessing the whole dataset). This algorithm establishes a good lower bound for DP cost while neglecting space efficiency in streaming.

[3]Release announcement https://ai.googleblog.com/2021/10/practical-differentially-private.html

**dp-sampling**: Finally, since there are no known DP streaming algorithms, we consider a DP baseline which simulates the same space efficiency of our algorithm. After running our algorithm and measuring the number of points stored by it, we collect a random sample of points (with the same size of the ones stored by our algorithm) and use it as input of the **dp-offline** algorithm. Notice that we account for the added privacy given by the sampling procedure allowing higher budget to the **dp-offline** sub-routine as shown in (Blocki et al., 2021). This algorithm provides a fair baseline result which has the same space and privacy constraints of our algorithm.

**Parameters.** We vary the number of centers, $k$, from 5 to 20. We fix $\delta = 0.001$ and vary $\varepsilon$ (for the private algorithms).

## 6.1. Results

We report the results of running the various algorithms in Figure 1 where we show the $k$-means cost on the 3 datasets for $\varepsilon = 2$ (for the private algorithms) and for different $k$'s. Notice that for all $k$'s and datasets that the **non-private** baseline has the lowest cost — this is expected, as it does not pay the price of DP. A higher cost is reported for the (non-streaming) **dp-offline** baseline which, despite being private, has access to the whole dataset in memory.

Interestingly, we observe that our algorithm **dp-stream**, despite storing only a fraction of the input, has a cost that is only slightly higher (and for one dataset, even lower) than the **dp-offline** baseline. Moreover, in all cases our algorithm outperforms (or is comparable to) the DP same-space baseline **dp-sampling**.

**Effect of the parameters on cost and memory** We now report the mean cost and total memory storage (as a fraction of the input) for running our algorithm on the skin dataset with $k = 10$ and different $\varepsilon$ and *sampling rates*. The results are reported in Table 1. Notice that $\varepsilon$ and the sampling rate do not significantly affect the cost while the total memory is (as expected) close to the sample rate. This shows that the additional cost for storing the heavy hitters is quite small compared to the sample.

## References

Ahmadian, S., Norouzi-Fard, A., Svensson, O., and Ward, J. Better guarantees for k-means and euclidean k-median by primal-dual algorithms. *SIAM Journal on Computing*, 49, 2020.

Arya, V., Garg, N., Khandekar, R., Meyerson, A., Munagala, K., and Pandit, V. Local search heuristic for k-median and facility location problems. In *STOC*, 2001.

Awasthi, P., Blum, A., and Sheffet, O. Stability yields a ptas for k-median and k-means clustering. In *FOCS*, 2010.

Balcan, M., Dick, T., Liang, Y., Mou, W., and Zhang, H. Differentially private clustering in high-dimensional euclidean spaces. In *Proceedings of the 34th International Conference on Machine Learning, ICML 2017, Sydney, NSW, Australia, 6-11 August 2017*, volume 70 of *Proceedings of Machine Learning Research*, pp. 322–331. PMLR, 2017.

Bhatt, R. and Dhall, A. Skin segmentation dataset. *UCI Machine Learning Repository*, 2010.

Blocki, J., Grigorescu, E., and Mukherjee, T. Differentially-private sublinear-time clustering. In *2021 IEEE International Symposium on Information Theory (ISIT)*, pp. 332–337. IEEE, 2021.

Borassi, M., Epasto, A., Lattanzi, S., Vassilvitskii, S., and Zadimoghaddam, M. Sliding window algorithms for k-clustering problems. *Advances in Neural Information Processing Systems*, 33:8716–8727, 2020.

Chan, T. H., Shi, E., and Song, D. Private and continual release of statistics. *ACM Trans. Inf. Syst. Secur.*, 14(3):26:1–26:24, 2011. doi: 10.1145/2043621.2043626. URL https://doi.org/10.1145/2043621.2043626.

Chan, T. H. H., Li, M., Shi, E., and Xu, W. Differentially private continual monitoring of heavy hitters from distributed streams. In *Privacy Enhancing Technologies: 12th International Symposium, PETS 2012, Vigo, Spain, July 11-13, 2012. Proceedings 12*, pp. 140–159. Springer, 2012.

Charikar, M., Guha, S., Tardos, É., and Shmoys, D. B. A constant-factor approximation algorithm for the k-median problem. *Journal of Computer and System Sciences*, 65, 2002.

Charikar, M., O'Callaghan, L., and Panigrahy, R. Better streaming algorithms for clustering problems. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pp. 30–39, 2003.

Chen, K. On k-median clustering in high dimensions. In *SODA*, 2006.

Chen, K. A constant factor approximation algorithm for k-median clustering with outliers. In *SODA*, 2008.

Cohen, E., Kaplan, H., Mansour, Y., Stemmer, U., and Tsfadia, E. Differentially-private clustering of easy instances. In Meila, M. and Zhang, T. (eds.), *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pp. 2049–2059. PMLR, 18–24 Jul 2021.

Cohen-Addad, V., Epasto, A., Lattanzi, S., Mirrokni, V., Munoz Medina, A., Saulpic, D., Schwiegelshohn, C., and Vassilvitskii, S. Scalable differentially private clustering via hierarchically separated trees. In *KDD*, pp. 221–230, New York, NY, USA, 2022a. Association for Computing Machinery. ISBN 9781450393850.

Cohen-Addad, V., Epasto, A., Mirrokni, V., Narayanan, S., and Zhong, P. Near-optimal private and scalable $k$-clustering. In *Advances in Neural Information Processing Systems*, 2022b.

Dheeru, D. and Karra Taniskidou, E. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

Dwork, C. and Roth, A. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 9(3-4):211–407, 2014. doi: 10.1561/0400000042. URL https://doi.org/10.1561/0400000042.

Dwork, C., Naor, M., Pitassi, T., and Rothblum, G. N. Differential privacy under continual observation. In Schulman, L. J. (ed.), *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, pp. 715–724. ACM, 2010. doi: 10.1145/1806689.1806787. URL https://doi.org/10.1145/1806689.1806787.

Dwork, C., McSherry, F., Nissim, K., and Smith, A. D. Calibrating noise to sensitivity in private data analysis. *J. Priv. Confidentiality*, 7, 2016.

Epasto, A., Mao, J., Medina, A. M., Mirrokni, V., Vassilvitskii, S., and Zhong, P. Differentially private continual releases of streaming frequency moment estimations. *CoRR*, abs/2301.05605, 2023. doi: 10.48550/arXiv.2301.05605. URL https://doi.org/10.48550/arXiv.2301.05605.

Feldman, D., Fiat, A., Kaplan, H., and Nissim, K. Private coresets. In *Proceedings of the 41st Annual ACM Symposium on Theory of Computing, STOC 2009, Bethesda, MD, USA, May 31 - June 2, 2009*, pp. 361–370. ACM, 2009.

Feldman, D., Xiang, C., Zhu, R., and Rus, D. Coresets for differentially private k-means clustering and applications to privacy in mobile sensor networks. In *Proceedings of the 16th ACM/IEEE International Conference on Information Processing in Sensor Networks, IPSN 2017, Pittsburgh, PA, USA, April 18-21, 2017*, 2017.

Ghazi, B., Kumar, R., and Manurangsi, P. Differentially private clustering: Tight approximation ratios. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

Gupta, A., Ligett, K., McSherry, F., Roth, A., and Talwar, K. Differentially private combinatorial optimization. In *Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2010, Austin, Texas, USA, January 17-19, 2010*, pp. 1106–1125. SIAM, 2010.

Har-Peled, S. and Mazumdar, S. On coresets for k-means and k-median clustering. In *Proceedings of the thirty-sixth annual ACM symposium on Theory of computing*, pp. 291–300, 2004.

Huang, Z. and Liu, J. Optimal differentially private algorithms for k-means clustering. In den Bussche, J. V. and Arenas, M. (eds.), *Proceedings of the 37th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, Houston, TX, USA, June 10-15, 2018*, pp. 395–408. ACM, 2018.

Lattanzi, S. and Vassilvitskii, S. Consistent k-clustering. In *International Conference on Machine Learning*, pp. 1975–1984. PMLR, 2017.

Li, S. and Svensson, O. Approximating k-median via pseudo-approximation. *SIAM Journal on Computing*, 45, 2016.

Nissim, K. and Stemmer, U. Clustering algorithms for the centralized and local models. In *Algorithmic Learning Theory, ALT 2018, 7-9 April 2018, Lanzarote, Canary Islands, Spain*, volume 83 of *Proceedings of Machine Learning Research*, pp. 619–653. PMLR, 2018.

Nissim, K., Raskhodnikova, S., and Smith, A. D. Smooth sensitivity and sampling in private data analysis. In Johnson, D. S. and Feige, U. (eds.), *Proceedings of the 39th Annual ACM Symposium on Theory of Computing, San Diego, California, USA, June 11-13, 2007*, pp. 75–84. ACM, 2007.

Ostrovsky, R., Rabani, Y., Schulman, L. J., and Swamy, C. The effectiveness of lloyd-type methods for the k-means problem. *Journal of the ACM (JACM)*, 59, 2012.

Savva, F., Anagnostopoulos, C., and Triantafillou, P. Explaining aggregates for exploratory analytics. In *2018 IEEE International Conference on Big Data (Big Data)*, pp. 478–487. IEEE, 2018.

Stemmer, U. and Kaplan, H. Differentially private k-means with constant multiplicative error. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, December 3-8, 2018, Montréal, Canada*, 2018.

## A. Appendix.

### A.1. Guarantees for DP-HH

**Theorem A.1** (DP-HH algorithm (Epasto et al., 2023)). *Let $\varepsilon > 0$, $\eta \in (0, 0.5)$, $0 < \alpha < 1$, $\xi \in (0, 0.5)$. There is an $\varepsilon$-DP algorithm in the streaming continual release model such that with probability at least $1 - \xi$, it always outputs a set $H \subseteq \mathcal{U}$ and a function $\hat{f} : H \to \mathbb{R}$ for every timestamp $t \in [T]$ such that*

1. *$\forall a \in H$, $\hat{f}(a) \in (1 \pm \eta) \cdot f_a$ where $f_a$ is the frequency of $a$ in the stream $\mathcal{S} = (a_1, a_2, \ldots, a_t)$*
2. *$\forall a \in \mathcal{U}$, if $f_a \geq \frac{1}{\varepsilon\eta} poly\big(\log\big(\frac{T \cdot |\mathcal{U}|}{\alpha\xi\eta}\big)\big)$ and $f_a^1 \geq \alpha \|\mathcal{S}\|_1$ then $a \in H$*
3. *The size of $H$ is at most $O((\log(T/\xi) + \log|\mathcal{U}|) \cdot (\frac{1+\eta}{1-\eta}) \cdot \frac{1}{\alpha})$*

*The algorithm uses $\frac{1}{\eta^2\alpha^3} poly\left(\log\left(\frac{T \cdot |\mathcal{U}|}{\xi\alpha}\right)\right)$ space.*

### A.2. Proofs for Bicriteria Approximation

**Proof of Claim 1.**

*Proof.* For any good cell $\mathbf{c}' \neq \mathbf{c}$, define

$$X_{\mathbf{c}'} = \begin{cases} 1 & \text{if } \mathbf{c}' \text{ collides with } \mathbf{c} \\ 0 & \text{otherwise} \end{cases}$$

Now for a fixed $\mathbf{c}' \neq \mathbf{c}$, the expected number of collisions with $\mathbf{c}$ is given by $\mathbb{E}_h[X_{\mathbf{c}'}] = \Pr[X_{\mathbf{c}'} = 1] = \Pr[h(\mathbf{c}) = h(\mathbf{c}')] = \frac{1}{w} = \frac{1}{40k}$. Thus the total number of collisions for $\mathbf{c}'$ with $\mathbf{c}$, in expectation, is given by $\mathbb{E}_h[\sum_{\mathbf{c}'} X_{\mathbf{c}'}] \leq 4k/w = 1/10 \leq 1/2$. Therefore, by Markov, $\Pr[\sum_{\mathbf{c}'} X_{\mathbf{c}'} > 1] \leq 1/2$. The claim follows. □

**Proof of Claim 2.**

*Proof.*

$$\mathop{\mathbb{E}}_h |\mathcal{B}_j| = \mathop{\mathbb{E}}_h[y + \sum_{\mathbf{c} \notin G_\ell} N_{\ell,\mathbf{c}}] \quad (9)$$

$$= y + \mathop{\mathbb{E}}_h[\sum_{\mathbf{c} \notin G_\ell} N_{\ell,\mathbf{c}}] \quad (10)$$

$$= y + \frac{n_{G_\ell}}{40k} \quad (11)$$

The claim follows by Markov inequality. □

**Proof of Claim 3.**

*Proof.* We set the failure probability of DP-HH (see Theorem A.1) to be $\xi := \frac{1}{k^2}$. For $\mathbf{c} \in G_\ell$, define $\mathcal{E}_\mathbf{c} := \mathcal{E}_1 \wedge \mathcal{E}_2 \wedge \mathcal{E}_3$ where $\mathcal{E}_1$ is the event that DP-HH algorithm is correct on all instances, $\mathcal{E}_2$ is the event that there are no collisions between $\mathbf{c}$ and other good cells, and $\mathcal{E}_3$ is the event that there exists a hash bucket that contains only $\mathbf{c}$, and if $N_{\ell,\mathbf{c}} := y$, then the size of that hash bucket is $\leq 2(y + \frac{n_{G_\ell}}{40k})$. We know that $\Pr[\mathcal{E}_1] \geq 1 - \xi = 1 - \frac{1}{k^2}$, by the accuracy guarantee of DP-HH algorithm, $\Pr[\mathcal{E}_2] \geq 1 - \frac{1}{k^2}$ by Claim 1 (and boosting the success probability), and $\Pr[\mathcal{E}_3] \geq 1 - \frac{1}{k^2}$, by Claim 2 (and boosting the success probability). Thus by a union bound, we have that for a fixed $\mathbf{c} \in G_\ell$, $\Pr[\mathcal{E}_\mathbf{c}] \geq 1 - \frac{3}{k^2}$. By taking a union bound over all $4k$ good cells, with probability at least $1 - \frac{12}{k}$, the claim holds. □

**Proof of Claim 4.**

*Proof.* Recall that $N_{\ell,\mathbf{c}}^{(t)}$ is the number of points in cell $\mathbf{c}$ at time step $t$. Because $x_t$ is covered in $G_\ell$, this means that we only need to care about the points covered by a good cell $\mathbf{c} \in G_\ell$ but $\mathbf{c} \notin H_{\ell,t}$. Using Claim 3, we know that if $N_{\ell,\mathbf{c}}^{(t)} \geq \frac{\alpha n_{G_\ell}}{20k}$, and $N_{\ell,\mathbf{c}}^{(t)} \geq \frac{\log^2 \Lambda \log k}{\varepsilon\eta}\text{poly}(\log(\frac{T\cdot k\cdot 2^\ell}{\alpha\eta}))$, then with probability at least $1 - \frac{12}{k}$, we have that $\mathbf{c} \in H_{\ell,t}$.

If $\mathbf{c} \in G_\ell$, and $\mathbf{c} \notin H_{\ell,t}$, this means that either (1) $N_{\ell,\mathbf{c}}^{(t)} < \frac{\log^2 \Lambda \log k}{\varepsilon\eta}\text{poly}(\log(\frac{T\cdot k\cdot 2^\ell}{\alpha\eta}))$, or, (2) $\mathbf{c}$ is not an $\alpha$-HH. For case (1), since there are $4k$ such good cells, the total number of uncovered points in such cells are $\frac{4k\log^2 \Lambda \log k}{\varepsilon\eta}\text{poly}(\log(\frac{T\cdot k\cdot 2^\ell}{\alpha\eta}))$. For case (2), this means that $N_{\ell,\mathbf{c}}^{(t)} < \frac{\alpha n_{G_\ell}}{20k}$. Again, since there are $4k$ such good cells, the total number of uncovered points in such cells are $< \frac{\alpha n_{G_\ell}}{20k} \cdot 4k < \alpha n_{G_\ell}$. □

### A.3. Additional Claims from Section 4

*Claim* 6.

$$cost(C_{\mathcal{F},\hat{w}}, (\mathcal{F}, \hat{w})) < cost(C_{\mathcal{F},w}, (\mathcal{F}, w)) + O(k\Lambda^2 + \frac{|\mathcal{F}|\Lambda^2}{\varepsilon}\log^{2.5}(T)\log(k))$$

*Proof.*

$$\begin{aligned}
&cost(C_{\mathcal{F},\hat{w}}, (\mathcal{F}, \hat{w})) \\
&= \sum_{f\in\mathcal{F}} \hat{w}(f) d^2(f, C_{\mathcal{F},\hat{w}}) \\
&= \sum_{f\in\mathcal{F}} (w(f) + O(\frac{1}{\varepsilon}\log^{2.5}(T)\log(k)) d^2(f, C_{\mathcal{F},\hat{w}}) \\
&= \sum_{f\in\mathcal{F}} w(f) d^2(f, C_{\mathcal{F},\hat{w}}) + O(\frac{1}{\varepsilon}\log^{2.5}(T)\log(k) \sum_{f\in\mathcal{F}} d^2(f, C_{\mathcal{F},\hat{w}}) \\
&\leq cost(C_{\mathcal{F},\hat{w}}, (\mathcal{F}, w)) + O(\frac{|\mathcal{F}|\Lambda^2}{\varepsilon}\log^{2.5}(T)\log(k)) \\
&\leq cost(C_{\mathcal{F},\hat{w}}, C_{\mathcal{F},w}) + cost(C_{\mathcal{F},w}, (\mathcal{F}, w)) + O(\frac{|\mathcal{F}|\Lambda^2}{\varepsilon}\log^{2.5}(T)\log(k)) \\
&\leq cost(C_{\mathcal{F},w}, (\mathcal{F}, w)) + O(k\Lambda^2) + O(\frac{|\mathcal{F}|\Lambda^2}{\varepsilon}\log^{2.5}(T)\log(k))
\end{aligned}$$

□

*Claim* 7.

$$cost(C^{opt}_{\mathcal{F},w}, (\mathcal{F}, w)) < 2cost(\mathcal{F}, \mathcal{S}) + |\mathcal{F}|\Lambda^2$$

*Proof.* By triangle inequality,

$$cost(C^{opt}_{\mathcal{F},w}, (\mathcal{F}, w)) < cost(\mathcal{F}, \mathcal{S}) + cost(C^{opt}_{\mathcal{F},w}, \mathcal{S})$$

Observe that $cost(C^{opt}_{\mathcal{F},w}, \mathcal{S}) < cost(C^{opt}_{F,w}, \mathcal{F}) + cost(\mathcal{F}, \mathcal{S})$, where

$$cost(C^{opt}_{\mathcal{F},w}, \mathcal{F}) = \sum_{f\in\mathcal{F}} d^2(f, C^{opt}_{\mathcal{F},w}) < |\mathcal{F}|\Lambda^2$$

□