# Persistent Data Sketching

Zhewei Wei*
School of Information, Renmin University of China
Key Laboratory of Data Engineering and Knowledge Engineering, MOE
zhewei@ruc.edu.cn

Ge Luo†
The Hong Kong University of Science and Technology
luoge@cse.ust.hk

Ke Yi
The Hong Kong University of Science and Technology
yike@cse.ust.hk

Xiaoyong Du‡
School of Information, Renmin University of China
and Key Laboratory of Data Engineering and Knowledge Engineering, MOE
duyong@ruc.edu.cn

Ji-Rong Wen
School of Information, Renmin University of China
and Key Laboratory of Data Engineering and Knowledge Engineering, MOE
jrwen@ruc.edu.cn

## ABSTRACT

A *persistent data structure*, also known as a *multiversion data structure* in the database literature, is a data structure that preserves all its previous versions as it is updated over time. Every update (inserting, deleting, or changing a data record) to the data structure creates a new version, while all the versions are kept in the data structure so that any previous version can still be queried.

Persistent data structures aim at recording all versions accurately, which results in a space requirement that is at least linear to the number of updates. In many of today's big data applications, in particular for high-speed streaming data, the volume and velocity of the data are so high that we cannot afford to store everything. Therefore, streaming algorithms have received a lot of attention in the research community, which use only sublinear space by sacrificing slightly on accuracy.

All streaming algorithms work by maintaining a small data structure in memory, which is usually called a *sketch*, *summary*, or *synopsis*. The sketch is updated upon the arrival of every element in the stream, thus is *ephemeral*, meaning that it can only answer queries about the current status of the stream. In this paper, we aim at designing persistent sketches, thereby giving streaming algorithms the ability to answer queries about the stream at any prior time.

*This work was partially supported by the National Key Basic Research Program (973 Program) of China under grant No. 2014CB340403 and the Fundamental Research Funds for the Central Universities, the Research Funds of Renmin University of China.

†Ge Luo and Ke Yi are supported by HKRGC under grants GRF-621413 and GRF-16211614, and by a Microsoft grant MRA14EG05.

‡Partially supported by 973 Program of China (Project No. 2012CB316205)


*SIGMOD'15,* May 31–June 4, 2015, Melbourne, Victoria, Australia.

http://dx.doi.org/10.1145/2723372.2749443.

## 1. INTRODUCTION

History is a lens on the future. As our ability to collect, store, and process data increases at an unprecedented rate, so is our desire to ask questions about our data at any prior time instance, instead of only on its current status. The study on managing historical data has a long history itself. After some earlier not-so-successful attempts, a breakthrough was made in 1989 in a landmark paper by Driscoll, Sarnak, Sleator, and Tarjan [15], in which general techniques for building *persistent data structures* were introduced. These techniques allow us to store all versions of a data structure, one after each update (insertion or deletion of a data record), into a single data structure whose size is linear in the total amount of differences between every two consecutive versions, so that each version can still be efficiently queried. After more years of theoretical and experimental work, persistent data structures have become a mature technique that is now being incorporated into commercial multiversion database systems such as Microsoft Immortal DB [18]. Search engines, such as Google, also have adopted similar ideas, but tailored to their specific needs and environment, to support queries about the web at any prior time instance or interval.

Most past research on persistent data structures has focused on supporting historical queries accurately. This means that we have to store all the updates, and a persistent data structure essentially attains this inherent limit. However, due to the high volume and high velocity of data in the "big data" era, even this linear space cost, which used to be good enough, is now often considered prohibitive. The area of streaming algorithms is exactly devoted to solving this problem. A streaming algorithm works by storing a small data structure, often called a *sketch*, *summary*, or *synopsis*, in memory, and updates it with every element in the stream. Since the sketch size is only sublinear in the amount of data it has processed, exact query results are not possible, but the error can be bounded by some small $\varepsilon$, which can be decided by the user in advance. The sketch size is then inversely related to $\varepsilon$, yielding a tradeoff between space and accuracy.

However, all the sketches used by streaming algorithms are *ephemeral*, i.e., every update creates a new version while past versions are forgot. In this paper, we aim at making the sketches persistent, thus giving streaming algorithms the ability to answer queries at any prior time in history, still with a small $\varepsilon$ error. More generally, we present sketching techniques that support *historical window queries*, which are asked on all elements in the stream that arrived in a given time interval $(s, t]$. It is clear that a traditional historical query supported by persistent data structures is a special case where $s = 0$. Our techniques are fairly general, and applicable to any sketch that consists of a set of counters. In particular, we show how to make the Count-Min Sketch [11] and the AMS/Count Sketch [2, 9] persistent, which are two of the most widely used sketches. They support some of the most fundamental queries in data analytics, such as point query, heavy hitters, and join size estimation. We show how the persistent versions of the Count-Min Sketch and the AMS/Count Sketch can be used to support these queries within any given time window $(s, t]$, with provably error guarantees. Meanwhile, the size of the persistent sketch remains sublinear, which is an essential property of streaming algorithms.

We envision many applications where persistent sketches could be useful. First, in any application that currently deploys persistent data structures, replacing them with persistent sketches will result in significant space savings, as long as a small error can be tolerated. At the same time, as the sketch is much smaller in size than a full data structure, it is possible to keep the sketch completely in main memory. This will greatly improve update and query efficiency, leading to better system scalability. Secondly, in scientific research where one experiment may run for days or even months while generating high-speed streams of numerical readings, it will be very useful to use persistent sketches to keep track of the progress over time. Since numerical readings in scientific experiments usually already have errors due to equipment inaccuracy and rounding, it is often acceptable to have some small and bounded error in the sketches in return for reduced space/time costs. Third, even for nonnumerical data streams such as network traffic (e.g., streams of IP packets), streaming algorithms have been widely deployed to monitor some statistical properties of the stream (e.g., the most frequent IP addresses). Using persistent sketches in these algorithms will enable statistical tracking of the entire history, rather than just the current status. Eventually, this line of work may lead to multiversion data stream systems, just like how persistent data structures have made multiversion database systems possible.

## 1.1 Related Work

This work lies in the intersection between persistent data structures and streaming algorithms, but these two areas have so far evolved separately with little interaction. Below we briefly review each area and point out where the gap is.

### *Persistent data structures.*

Early approaches for managing historical data include: (1) store all the updates in a "changelog", and "replay" the history when a historical query is issued; and (2) create and keep a version of the data structure after every update. These approaches will result in either a long query time or a huge storage cost. In a landmark paper by Driscoll, Sarnak, Sleator, and Tarjan [15], the concept of persistent data structures was formally introduced, together with a suite of techniques for making an ordinary (i.e., ephemeral) data structure persistent. The persistence (or multiversion) model used in the database community is actually called *partial persistence* in [15], in which queries can be asked on any previous version, but updates can only be applied to the current version. There are two other types of persistence, namely *full persistence* and *confluent persistence*, which allow more complicated history models. They are more relevant to version control in software development, so we will not discuss them further in this paper.

Driscoll et al. [15] have introduced general techniques for turning any pointer-based ephemeral data structure into a persistent one with optimal overhead. Since then, there has been considerable development of persistent data structures, both in theory and practice. In particular, their techniques have been extended to the Multiversion B-tree [5, 7, 31], Time-Split B-tree [19], which has enabled the application of persistence techniques in database systems, and eventually led to the development of multiversion database systems and techniques such as Microsoft Immortal DB [18, 20], SNAP [29], Ganymed [25], Skippy [28] and LIVE[27].

However, the existing persistence techniques are not readily applicable to the persistent sketch problem. First, they only work on a pointer-based data structure (i.e., lists, binary trees, graphs), but the majority of sketches are not pointer-based. More seriously, these techniques aim at preserving every change to the data structure, thus directly applying them to a sketch would result in a persistent sketch whose size is at least linear in the stream length, which completely violates the first requirement of streaming algorithms. On the other hand, since a sketch is already an approximate data structure, it does not make sense to preserve all the versions accurately. Therefore, the key question in making a sketch persistent is to decide when and how to preserve the changes as long as the error guarantee is maintained, which has never been an issue for persistent data structures.

### *Streaming algorithms.*

Streaming algorithms have been an area of active research in the past 20 years, with both rich theories and many practical data stream systems developed. Several streaming models have been considered in the literature [22]: In the *cash register model*, each record in the stream inserts an element, and queries are asked on the set of all the elements in the stream so far. In the *turnstile model*, the stream consists of both insertions and deletions, where a deletion removes an element previously inserted. These models only look at the current status of the stream.

Another streaming model that has a bit of "historical" flavor is the *sliding window* model [3, 6, 13], where we are only interested in the last $k$ elements in the stream (a *count-based window*), or all elements that have arrived in the past (say) 24 hours (a *time-based window*). However, as more elements arrive in the stream and the window slides forward, past windows are lost. Note that this is actually a special case of historical window query where $t = now$ and $s = t - w$ for some fixed window length $w$.

Finally, the closest work to ours is that of Tao et al. [30], who designed a quantile summary for historical data. However, it is still far from a real persistent sketch because it does not work on streaming data. All the updates must be given in advance. Once the summary is built, it supports historical queries but no more new changes can be made. Moreover, their quantile summary is pointer-based so it is easy to use the techniques from [15], but the majority of the sketches in the streaming literature are not.

## 1.2 Preliminaries

We adopt the standard streaming model [4, 22] (a.k.a. the *cash register model*), defined as follows. Let $S = (e_1, e_2, \ldots, e_m)$ be a data stream, where the $e_j$'s are elements from the universe $[n] = \{1, \ldots n\}$. The *frequency vector* of $S$ is $\mathbf{f} = (f_1, f_2, \ldots, f_n)$, where frequency $f_i = \sum_{j:e_j=i} 1$ is the number of times that ele-

ment $i$ appears. We use a discrete time model and assume that $e_t$ arrives at time $t$, for $t = 1, \ldots, m$. If nothing arrives at time $t$, then $e_t = null$. In the more complex situation where the tuples in the stream have multiple attributes, we can view each attribute as a stream, and build sketches on each attribute stream. Since we use the standard streaming model, our sketches can be plugged into any architecture that adopts the standard streaming model. Please see the survey paper [4] for more discussion. For a distributed streaming system such as S4 [23] and D-Streams [32], our technique can be deployed to each single node.

Another commonly used streaming model is the *turnstile model*, which generalizes the standard streaming model by allowing deletions. In this model, the data stream $S = ((e_1, c_1), (e_2, c_2), \ldots, (e_m, c_m))$, where $e_j$'s are the elements and $c_i$'s denote the frequency change of element $e_j$, which can be $1, 0$, or $-1$. The frequency of item $i$ is $f_i = \sum_{j:e_j=i} c_j$. Note that by restricting $c_i$ to be 1 or 0, the turnstile model becomes the standard streaming model. The turnstile model allows deletions and may result in negative frequency, so we will focus on the standard streaming model in this paper. We only point out that all our persistent sketches work for the turnstile model as well, and present the corresponding space and query time bounds. These bounds may be of theoretical interest, as the Count-Min Sketch and AMS Sketch work for the turnstile model as well.

The Count-Min Sketch and the AMS Sketch are two widely used sketches, and we briefly describe them here. Let $h_j : [n] \to [w], j = 1, \ldots, d$ be a set of $d$ pairwise-independent hash functions. The Count-Min Sketch maintains a $d \times w$ array of counters. Let $C[j][k]$ be the counter in the $j$-th row and $k$-th column. To process a new element $i$, the Count-Min Sketch performs $C[j][h_j(i)] \leftarrow C[j][h_j(i)] + 1$ for $j = 1, \ldots, d$. The Count-Min Sketch supports two fundamental queries: point query and heavy hitters. A point query asks for how many times an item $i$ appears in the stream, and is estimated as $\text{median}_{j\in[d]}\{C[j][h_j(i)]\}$ by the sketch. Point queries are a building block for more complex queries, such as wavelet [16], range query [11]. Heavy hitters, also known as frequent items, are the items with frequencies above a given threshold. This problem has been extensively studied in the database literature [12, 21], but existing data structures are not persistent.

The (fast) AMS Sketch is similar to the Count-Min Sketch. It maintains a $d \times w$ array of counters and uses $d$ hash functions $h_j : [n] \to [w]$ to map the elements in the universe to a counter in each row. The only difference is that, upon an update $i$, the AMS Sketch does $C[j][h_j(i)] \leftarrow C[j][h_j(i)] + \xi_j(i)$, where $\xi_j : [n] \to \{-1, +1\}$ is an 4-wise independent hash function, for $j \in [d]$. A important class of query supported by the AMS Sketch is the join size query, which asks for the join size of two streams. The join size query is useful for estimating query result sizes and measuring the skewness of the data [1, 26]. Meanwhile, join size query is the building block of more complex queries, such as binary joins, wavelets, and $l_p$ differencing [14].

## 1.3 Problem Formulation

We now formally define the problem we are trying to address in this paper. Given a time interval $(s, t] = \{s + 1, s + 2, \ldots, t\}$, define $\mathbf{f}_{s,t}$ to be the frequency vector of all the elements having arrived within $(s, t]$, i.e., $\mathbf{f}_{s,t} = (f_1(s, t), f_2(s, t), \ldots, f_n(s, t))$, where $f_i(s, t) = \sum_{j:e_j=i, s<j\le t} 1$ is the number of appearance of element $i$ in time window $(s, t]$, for $i = 1, \ldots, n$. For historical queries, we have $s = 0$, and we simplify the notation as $\mathbf{f}_{0,t} = \mathbf{f}_t$. With a slight abuse of notation, we also use the frequency vectors $\mathbf{f}$, $\mathbf{f}_t$ and $\mathbf{f}_{s,t}$ to denote that part of the stream that defines these frequency vectors, respectively.

In this paper, we focus on four types of queries: 1) Historical window point query: Given an index $i$ and a time range $(s, t]$, return $f_i(s, t)$. 2) Historical window heavy hitters query: Given an threshold $\phi$, and a time range $(s, t]$, return elements with frequency $f_i(s, t) \ge \phi\|\mathbf{f}_{s,t}\|_1$. Here $\|\mathbf{f}_{s,t}\|_1 = \sum_{i=1}^n |f_i(s, t)|$ is the $L_1$ norm of $\mathbf{f}_{s,t}$. 3) Historical window join size: For two streams $\mathbf{f}$ and $\mathbf{g}$, given a time range $(s, t]$, return $I(\mathbf{f}_{s,t}, \mathbf{g}_{s,t}) = \sum_{i=1}^n f_i(s, t)g_i(s, t)$. 4) Historical window self-join size: Given a time range $(s, t]$, return $\|\mathbf{f}_{s,t}\|_2^2 = \sum_{i=1}^n f_i(s, t)^2$.

## 1.4 Our Results

We observe that most known sketches in the turnstile model are *linear sketches* of the following form. One or more hash functions are used to map the elements in the universe $[n]$ to an array of counters, and the value of each counter is a linear combination of the frequencies of the elements mapped to that counter. Upon the arrival of each update in the stream for a particular element, all counters that the element is mapped to are then updated accordingly. We present two general persistence techniques that, practically speaking, can be applied to any linear sketch of this form. However, in order to provide provable error guarantees, each technique is more suitable for a particular class of sketches. Specifically, we obtain the following results.

### *PLA-based persistent sketch.*

We first present a persistence technique that uses *piecewise-linear functions* to approximate each counter of the sketch over time. This technique introduces an error that depends on the $L_1$-norm of the frequency vector, so it is more suitable for sketches that also provide such an error guarantee. In particular, we apply this technique to the Count-Min Sketch [11], and show that a persistent Count-Min Sketch can be used to answer historical window point queries and heavy hitters queries with error $\varepsilon\|\mathbf{f}_{s,t}\|_1 + \Delta$, with an arbitrarily high constant probability. The error consists of a relative term $\varepsilon\|\mathbf{f}_{s,t}\|_1$, which inherits from the Count-Min Sketch itself, as well as an additive term $\Delta$, which is due to persistence. Note that, however, an additive term is unavoidable because the window can be arbitrarily small and it may contain each single element in the stream. If we did not have an additive error term, the persistent sketch would have to store all the data accurately. On the other hand, if we only consider historical queries (i.e., $s = 0$), then we can avoid this term (see later).

The size of a persistent Count-Min Sketch depends on the data distribution. It can be large on adversarial inputs, but our experimental results show that on many real-world and non-adversarial synthetic data sets, its size is actually very small. We back up this empirical observation with a theoretical justification: Under a reasonable random streaming model (random turnstile model and random stream model defined in Section 3), the persistent Count-Min Sketch has size $O(\frac{1}{\varepsilon} + \frac{m}{\Delta^2})$ in expectation, which is a factor of $\Delta$ smaller than a baseline solution.

The persistent Count-Min Sketch cannot handle join size queries (or rather, cannot provide a meaningful error bound), which are addressed by our second persistence technique.

### *Sampling based persistent sketch.*

Next, we present a sampling based persistence technique that works well with sketches with error guarantees in terms of the $L_2$-

norm. We show how this can be applied to the AMS Sketch [2][1] to answer historical window self-join size queries with error $\varepsilon\|\mathbf{f}_{s,t}\|_2^2 + \frac{\Delta^2}{\varepsilon}$, and join size queries between two streams $\mathbf{f}$ and $\mathbf{g}$ with error

$$\mathcal{E} = \varepsilon\sqrt{\left(\|\mathbf{f}_{s,t}\|_2^2 + (\frac{\Delta_\mathbf{f}}{\varepsilon})^2\right)\left(\|\mathbf{g}_{s,t}\|_2^2 + (\frac{\Delta_\mathbf{g}}{\varepsilon})^2\right)}.$$

Similarly, these errors all have an additive term, for the same reason as argued above. For join size queries between two streams $\mathbf{f}$ and $\mathbf{g}$, we allow them to use different additive error parameters $\Delta_\mathbf{f}$ and $\Delta_\mathbf{g}$. The size of a persistent AMS Sketch is $O(\frac{1}{\varepsilon^2} + \frac{m}{\Delta})$ in expectation regardless of the input, in both the turnstile model and the standard streaming model.

*Historical queries.*

Finally, we show how these persistent sketches can be specialized to solve historical queries for which $s$ is fixed at 0. In this case, we can avoid the additive error term, and provide an error bound identical to that of the ephemeral sketch at time $t$. More precisely, the persistent Count-Min Sketch answers historical point queries and heavy hitters queries with error $\varepsilon\|\mathbf{f}_t\|_1$, and the persistent AMS Sketch answers historical self-join size queries with error $\varepsilon\|\mathbf{f}_t\|_2^2$ and join size queries with error $\varepsilon\|\mathbf{f}_t\|_2\|\mathbf{g}_t\|_2$.

Under this setting, the size of the persistent Count-Min Sketch is $O(\frac{1}{\varepsilon^2})$ under the random stream model and $O\left(\sum_{t=1}^m \frac{1}{\varepsilon^2\|\mathbf{f}_t\|_1^2}\right)$ under the random turnstile model. The size of the persistent AMS Sketch becomes $O(\frac{1}{\varepsilon^2} + \frac{\sqrt{m}}{\varepsilon})$ in the standard streaming model, and $O\left(\frac{1}{\varepsilon^2} + \sum_{t=1}^m \frac{1}{\varepsilon\|\mathbf{f}_t\|_2}\right)$ in the turnstile model, regardless of the input.

Finally, another appealing feature of our persistent sketches is that they are very easy to implement, with minimal modifications to the original, ephemeral sketches. However, the analyses, including both error analysis and space complexity, are nontrivial and quite involved for a general audience. Thus, we only present the results and some high-level ideas of the proofs in the main text, while deferring all details to the appendix.

## 1.5 An Illustrating Example

Consider the following example: a website receives an enormous number of visits every day, and one is interested in the most frequently requested URLs, and more importantly, how the frequencies of these URLs change over time. One could build a persistent data structure that supports top-k queries for any arbitrary past time window. However, as the update speed is very fast and data size is huge, such a persistent data structure must reside on disk or distributed system, which is costly. In this case, we can build a persistent sketch which supports historical window heavy hitters queries and hence top-k queries [9]. Since the size of a persistent sketch is sub-linear, it can fit in main memory. This will greatly improve update and query efficiency, leading to better system scalability.

We used the 1998 World Cup web site access log[2], which consists of all requests made to the 1998 World Cup web site between April 30 and July 26, 1998. Each request is a tuple that consists of 8 attributes: 1) timestamp: time of the request; 2) clientID, an anonymized integer of the IP address of the request ; 3) objectID, an anonymized integer of the requested URL; 4) size: number of bytes in the response; 5) method: method contained in the client's request; 6) status: response status code; 7) type: type of file requested 8) server: indicates which server handled the request.

[1]The version we are using is the improved version known as the "fast AMS Sketch", which is similar to the Count Sketch [9].

[2]http://ita.ee.lbl.gov/traces/WorldCup/WorldCup.html

| URL | actual count | estimation |
|---|---|---|
| / | 1138896 | 1138970 |
| /images/space.gif | 1117634 | 1120050 |
| /images/dot.gif | 880322 | 880765 |
| /images/hm_nbg.jpg | 818126 | 818586 |
| /images/home_intro.anim.gif | 799697 | 800323 |

**Table 1: Top-5 most frequently requested URLs and their estimated frequencies at the end of the stream .**

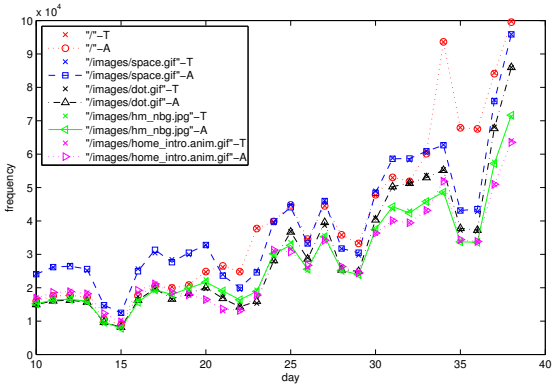



**Figure 1: A illustration of the frequency change of top-5 most requested URLs over the days. Here legends with"T" denote the true frequencies, and legends with"A" denote the approximated frequencies.**

Suppose we are interested in the most frequently requested URLs, and how their frequencies change over time. Then we can view the objectID attribute as a stream of elements, each arriving at the respective timestamp. If we build an ephemeral Count-Min Sketch on this objectID stream, we could only get the top-$k$ URLs at the end of the stream (Table 1). However, if we had built a persistent Count-Min Sketch, we would be able to query the sketch and see how the frequencies of these URLs change over time (Figure 1). Note that all this information is extracted from the persistent sketch only, *without* accessing the raw data stream, which may have been dumped to slow secondary storage or even discarded.

# 2. A BASELINE SOLUTION

A baseline solution can be designed using properties of *linear sketches*. Recall that each counter in a linear sketch is a linear function of the frequency vector $\mathbf{f}$. Let $C_t$ be the set of counters at time $t$. If we record $C_t$ for every $t$, then given a historical window $(s, t]$, we can compute $C_{s,t} = C_t - C_s$. By linearity, this is exactly the set of counters for $\mathbf{f}_{s,t}$, i.e., the partial stream within that window, which can be used to answer various queries on $\mathbf{f}_{s,t}$.

The above approach would result in a persistent sketch with linear space. To achieve sub-linear space, we keep track of each counter in the sketch over time, but record a value only when it has deviated from the last recorded value by more than $\Delta$. To answer a historical window query with time range $(s, t]$, for each counter we find the recorded value with timestamp closest to $s$ (resp. $t$) to form $\hat{C}_s$ (resp. $\hat{C}_t$) as an approximation of $C_s$ (resp. $C_t$), and uses $\hat{C}_{s,t} = \hat{C}_t - \hat{C}_s$ as an approximation of $C_{s,t}$ to answer queries on $\mathbf{f}_{s,t}$. Note that each counter in $\hat{C}_{s,t}$ might deviate from that in $C_{s,t}$ by up to $2\Delta$; but by scaling $\Delta$ down by a factor of 2, this is not an issue.

However, the baseline solution suffers from two drawbacks: (1) The space complexity is still high. Given an additive error parameter $\Delta$, the space usage is proportional to $\frac{m}{\Delta}$, where $m$ is the length of the stream. (2) This approach can only handle queries that depend on one (or a small number of) counters in the sketch, which is the case for point and heavy hitters queries. For join size and self-join queries, the answer depends non-linearly on *all* the counters in the sketch. Thus the total error would be amplified significantly. We present our two persistence techniques in the next two sections, which will exactly address these drawbacks.

# 3. PLA-BASED PERSISTENT SKETCH

In this section, we present our first persistence technique, which is based on the idea of using piecewise-linear approximations (PLA) to keep track of the counters over time. For concreteness, we show how it is applied to the Count-Min Sketch, resulting in a persistent Count-Min Sketch that answers historical window point and heavy hitter queries with error depending on the $L_1$-norm.

## 3.1 Piecewise Linear Approximation

We first briefly review the PLA technique. Here, the aim is to approximate a function $v(\cdot)$ with a piecewise-linear function $f(\cdot)$ with bounded error having a small number of segments. Note that the segments in $f$ may not be connected. In our setting, $v(\cdot)$ will be the value of a counter over time, and we want $|f(t) - v(t)| \leq \Delta$ for all $t$ for some error parameter $\Delta$.

There is a greedy algorithm that can construct such an $f$ having the minimum number of segments [24]. The algorithm is very simple and we describe it here for completeness. We view each counter value at time $t$ as a point $(t, v(t))$ in the plane. Let $P$ be the set of points we have processed so far. The algorithm maintains the invariant that all points in $P$ can be approximated with a single line within error $\Delta$. Let $p$ be the next point. We check if $P \cup \{p\}$ can still be approximated with a single line. If yes, we add $p$ to $P$ and continue onto the next point; otherwise, we output a line segment that approximates $P$, set $P \leftarrow \{p\}$, and continue. The optimality of the algorithm is trivial; O'Rourke [24] in addition showed how this algorithm can be implemented in $O(1)$ space and amortized $O(1)$ time per point.

## 3.2 Persistent Count-Min Sketch

Our basic idea is to keep track of each counter in the Count-Min Sketch using a PLA. Recall that the Count-Min Sketch maintains a $d \times w$ array of counters. Let $C[j][k]$ be the counter in the $j$-th row and $k$-th column. Let $h_j : [n] \to [w], j = 1, \ldots, d$ be a set of $d$ pairwise-independent hash functions. To make the Count-Min Sketch persistent, we run an instance of the above PLA algorithm for each counter $C[j][k]$, which generates a piecewise-linear function that approximates this counter over time with additive error $\Delta$. We store this function together with $C[j][k]$, and denote it as $\mathsf{PLA}[j][k]$. Below, we provide detailed procedures for updating and querying this persistent Count-Min Sketch.

**Update.** Consider an update $i$ arriving at time $t$, meaning that the frequency $f_i$ is updated as $f_i \leftarrow f_i + 1$. For $j = 1, \ldots, d$, we first update the counter in the ephemeral sketch as

$$C[j][h_j(i)] \leftarrow C[j][h_j(i)] + 1.$$

Then, we feed the point $(t, C[j][h_j(i)])$ to the PLA algorithm running for $C[j][h_j(i)]$. If this algorithm outputs a segment, we append it to $\mathsf{PLA}[j][h_j(i)]$.

**Historical window point query.** Recall that in a historical window point query, we are given an element $i$ and a time interval $(s, t]$, and the goal is to estimate $f_i(s, t)$, the frequency of element $i$ in time range $(s, t]$. For each $j = 1, \ldots, d$, we retrieve an approximation of $C[j][h_j(i)]$ at time $s$, denoted as $\hat{C}^s[j][h_j(i)]$, from $\mathsf{PLA}[j][h_j(i)]$. This is done by a binary search on $\mathsf{PLA}[j][h_j(i)]$. Similarly we retrieve $\hat{C}^t[j][h_j(i)]$. Finally we return

$$\hat{f}_i(s, t) = \text{median}_{j \in [d]} \{\hat{C}^t[j][h_j(i)] - \hat{C}^s[j][h_j(i)]\}$$

as the estimator for $f_i(s, t)$.

Note that here we do not take the minimum as in the original Count-Min Sketch, since $\hat{C}^t[j][h_j(i)] - \hat{C}^s[j][h_j(i)]$ may be either an overestimate or an underestimate of $f_i(s, t)$.

**Historical window heavy hitters queries.** Recall that in a historical window heavy hitters query, we are given a time range $(s, t]$ and a parameter $\phi$, and the goal is to return all the elements $i$ with frequency $f_i(s, t) \geq \phi \|\mathbf{f}\|_1$. To be able to efficiently identify the heavy hitters, we use the dyadic range sum technique in [12] to decompose the universe $[n]$. More precisely, a dyadic range at level $\ell$ is an interval of the form $[j2^\ell + 1, (j + 1)2^\ell)$, for $\ell = 0, \ldots, \log n$ and $j = 0, \ldots, \frac{n}{2^\ell} - 1$. For a particular level $\ell$, we divide the universe into $\frac{n}{2^\ell}$ dyadic ranges, each of size $2^\ell$, and use a persistent Count-Min Sketch to track total frequency of elements in every dyadic range. Thus we maintain $\log n + 1$ persistent Count-Min Sketches in total. When an update $(i, c)$ arrives, in each level, we find the dyadic interval that contains $i$, and update the corresponding counters.

In order to find all heavy hitters, we ask point queries on the $\log n + 1$ levels recursively. More precisely, suppose at level $\ell$ we have identified at most $\frac{1}{\phi}$ heavy dyadic ranges (a dyadic range is *heavy* if the estimated total frequency of all elements in that range is more than $\phi \|\mathbf{f}_{s,t}\|_1$). These ranges are split into $\frac{2}{\phi}$ dyadic ranges in level $\ell + 1$. Then at level $\ell + 1$, we ask point queries on these $\frac{2}{\phi}$ heavy dyadic ranges to identify again at most $\frac{1}{\phi}$ heavy dyadic ranges. We repeat this process until we find all heavy hitters at level 0, which are returned as the estimated heavy hitters.

## 3.3 Analysis

In this section, we give the error guarantees of the persistent Count-Min Sketch, as well as the analysis of its space and time complexity. We set the parameters as $w = O(1/\varepsilon)$, $d = O(\log \frac{1}{\delta})$ where $\delta$ is the failure probability, and PLA error $O(\Delta)$.

**Error guarantees.** For historical window point queries, the persistent Count-Min Sketch can provide the following error bound:

THEOREM 3.1. *Given a time interval $(s, t]$ and an element $i$, the persistent Count-Min Sketch provides an estimate $\hat{f}_i(s, t)$ for $f_i(s, t)$ such that*

$$\Pr\left[\left|\hat{f}_i(s, t) - f_i(s, t)\right| \leq \varepsilon \|\mathbf{f}_{s,t}\|_1 + \Delta\right] \geq 1 - \delta.$$

Theorem 3.1 follows from the linearity of the Count-Min Sketch, its own $\varepsilon$-guarantee, and the fact that the PLA introduces an additional additive $\Delta$ error. The detailed proof can be found in the appendix.

Similarly, we have the following error guarantee for historical window heavy hitters queries:

THEOREM 3.2. *Given a time interval $(s, t]$, for any element $i$ with $f_i(s, t) \geq (\phi + \varepsilon)\|\mathbf{f}_{s,t}\|_1 + \Delta$, the persistent Count-Min Sketch returns it with probability as least $1 - \delta$; for any element $i$ with $f_i(s, t) < \phi \|\mathbf{f}_{s,t}\|_1$, the persistent Count-Min Sketch returns it with probability at most $\delta$.*

**Space complexity.** For space complexity analysis, we only focus on point queries; if heavy hitters queries are to be supported, the

space usage simply grows by a $\log n$ factor due to the dyadic range sum technique.

In the worst case, every $\Delta$ updates might trigger the PLA algorithm to generate a segment, resulting in a total space of $O((\frac{m}{\Delta} + \frac{1}{\varepsilon})\log\frac{1}{\delta})$ (the $\frac{1}{\varepsilon}$ part is for the ephemeral sketch). This is as bad as the baseline solution. However, our empirical results in Section 6 show that the persistent Count-Min Sketch achieves much better space usage in practice. To provide some theoretical justification behind this phenomenon, we analyze its space complexity under the following *random stream model*, and prove a much better space bound under this model.

DEFINITION 3.1 (RANDOM STREAM MODEL). *Let $\mathcal{P}$ be an unknown but fixed distribution over the set of all possible elements $[n] = \{1, 2, \ldots, n\}$. Each update in the stream is drawn independently and uniformly at random from $[n]$ according to $\mathcal{P}$.*

This model is also known as the *random order model* [17]. As argued in [17], there is often a lot of randomness in real-world data streams. Although they may not be as fully random as required in the model above, the model is actually a better approximation of reality than assuming worst-case, adversarial inputs. Note that we do not have any assumption on $\mathcal{P}$ (such as uniform or Gaussian); it can be any any distribution (unknown to the algorithm) so that the our results will apply to a variety of scenarios.

Note that the random stream model can also be generalized to *random turnstile model*, in which each update is drawn independently and uniformly at random from $U = \{(i, c) \mid i \in [n], c \in \{-1, 0, +1\}\}$ following some unknown but fixed distribution $\mathcal{P}$. To generalize our technique, we will prove the following space bound for the persistent Count-Min Sketch under the random turnstile model:

THEOREM 3.3. *In the random turnstile model, the persistent Count-Min Sketch uses $O((\frac{m}{\Delta^2} + \frac{1}{\varepsilon})\log\frac{1}{\delta})$ space in expectation.*

Note that Theorem 3.3 directly implies the same space for the random stream model. This bound is roughly a $\Delta$-factor improvement from the baseline solution, which can be significant, since $\Delta$ is an additive error, and is usually set to be on the same order as $\varepsilon\|\mathbf{f}_{s,t}\|_1$. The key insight in the proof of this bound is that the value of each counter over time can be viewed as a random walk that vibrates around a line, which is being constructed by the PLA algorithm. A single line segment of the PLA fails to approximate the counter only when the random walk "escapes" from a region of width $2\Delta$ centered at the line. Then using some random walk theory, we are able to show that it takes a quadratic number of steps to escape from this region. The rigorous proof, however, is quite involved and is provided in the appendix.

**Query time.** The analysis on query time is straightforward. For point queries, since we retrieve $O(d)$ estimates from the PLA, and each involves a binary search, the total query cost is $O(d\log m) = O(\log\frac{1}{\delta}\log m)$. By the standard technique of fractional cascading [10], this can be improved to $O(\log\frac{1}{\delta} + \log m)$.

For heavy hitters queries, observe that we make $O(\frac{1}{\phi})$ point queries to $O(\log n)$ persistent Count-Min Sketches. Thus the query cost is $O(\frac{1}{\phi}\log m\log n\log\frac{\log n}{\phi\delta})$, which can again be improved to $O(\log m + \frac{1}{\phi}\log n\log\frac{\log n}{\phi\delta})$ by fractional cascading. We omit the details as the focus of the paper is on the error-space tradeoff.

# 4. SAMPLING BASED PERSISTENT SKETCH

In this section, we present our second persistence technique based on random sampling. When applied with the AMS Sketch, this yields a persistent sketch answers historical window point queries and join size queries with provable error bounds.

## 4.1 Persistent AMS Sketch

In order to apply the sampling technique, we need each counter to be monotonically increasing (for reasons that will become clear later). Thus, we decompose each counter $C[j][k]$ into two components $C[j][k][1]$ and $C[j][k][0]$, such that both components monotonically increase. Specifically, to process an update $i$, if $\xi_j(i) = 1$, we increment $C[j][h_j(j)][1]$ by 1; if $\xi_j(i) = -1$, we increment $C[j][h_j(j)][0]$ by 1. It is clear that the correct value as in the original AMS Sketch, $C[j][k]$, is equal to $C[j][k][1] - C[j][k][0]$.

To make the AMS Sketch persistent, for each counter $C[j][k][b]$, we associate it with $L[j][k][b]$, a list that we call a *history list*. This list stores the values of the counter sampled over time, together with a timestamp indicating when each value is sampled.

**Update.** Consider an update $i$ arriving at time $t$, meaning that frequency $f_i$ is updated as $f_i \leftarrow f_i + 1$. For each $j = 1, \ldots, d$, we first update $C[j][h_j(i)][b]$ as described above where $b = 1$ or 0 depending on $\xi_j(i)$. Then we update the history list $L[j][h_j(i)][b]$ as follows. With probability $p = 1/\Delta$, we sample the new value of the counter and append the pair $(C[j][h_j(i)][b], t)$ at the end of $L[j][h_j(i)][b]$.

**Historical window point query.** The primary functionality of the AMS Sketch is to support join size estimation. Nevertheless, it also supports point queries as noted in [9], with an error depending on the $L_2$-norm of the frequency vector. As point queries are simpler to describe, below we first show how our persistent AMS Sketch supports historical window point queries, and then move on to join size estimation.

Consider a historical window point query for element $i$ in a time range $(s, t]$, which asks for an estimate of $f_i(s, t)$. For each $j = 1, \ldots, d$ and each $b = 0, 1$, we do a binary search in $L[j][h_j(i)][b]$ to find the *predecessor* of $t$, i.e., the record with the largest timestamp before $t$ (including $t$). Let $\bar{C}^t[j][h_j(i)][b]$ be the counter value of the predecessor of $t$ in $L[j][h_j(i)][b]$. Intuitively, $\bar{C}^t[j][h_j(i)][b]$ is a good approximation of the counter value at time $t$, but we may have missed some increments since $\bar{C}^t[j][h_j(i)][b]$ was sampled. We thus compensate our estimate as

$$\hat{C}^t[j][h_j(i)][b] = \begin{cases} \bar{C}^t[j][h_j(i)][b] + \Delta - 1, \\ \qquad \text{if } t \text{ has a predecessor in } L[j][h_j(i)][b]; \\ 0, \quad \text{otherwise.} \end{cases} \tag{1}$$

Then we compute

$$\hat{D}^t[j][h_j(i)] = \xi_j(i) \cdot \left(\hat{C}^t[j][h_j(i)][1] - \hat{C}^t[j][h_j(i)][0]\right),$$

which is an estimate of $f_i$ at time $t$. We compute $\hat{D}^s[j][h_j(i)]$, an estimate of $f_i$ at time $s$ in the same manner. We then compute $\hat{D}^{s,t}[j][h_j(i)] = \hat{D}^t[j][h_j(i)] - \hat{D}^s[j][h_j(i)]$ as an estimate of $f_i(s, t)$. Finally, across all rows $j$, we return the median as the final estimate $\hat{f}_i(s, t) = \text{median}_{i\in[d]}\hat{D}^{s,t}[j][h_j(i)]$.

**Historical window join size estimation.** To support join size estimation between two streams $\mathbf{f}$ and $\mathbf{g}$, the two persistent AMS Sketches on the two streams, denoted as $C_\mathbf{f}$ and $C_\mathbf{g}$, need to use the same set of hash functions $h_j$ and $\xi_j$, $j = 1, \ldots, d$. As they are just pairwise and 4-wise independent hash functions, each can be described in $O(1)$ space, and can be shared between the two streams with $O(1)$ communication.

Recall that for a given a time interval $(s, t]$, the join size between the two streams is $I(\mathbf{f}_{s,t}, \mathbf{g}_{s,t}) = \sum_i f_i(s, t)g_i(s, t)$. For each $j = 1, \ldots, d$, we first compute $\hat{D}^{s,t}_\mathbf{f}[j][k]$ and $\hat{D}^{s,t}_\mathbf{g}[j][k]$ in as we did for point queries above, the only difference being that for point

queries, we only care about a single $k = h_j(i)$, but now we need to compute $\hat{D}^{s,t}_{\mathbf{f}}[j][k]$ and $\hat{D}^{s,t}_{\mathbf{g}}[j][k]$ for all $k \in [w]$.

After we have computed $\hat{D}^{s,t}_{\mathbf{f}}[j][k]$ and $\hat{D}^{s,t}_{\mathbf{g}}[j][k]$ for all $j$ and $k$ from the two persistent sketches, we can estimate the join size between $\mathbf{f}_{s,t}$ and $\mathbf{g}_{s,t}$ as in the original AMS Sketch, i.e.,

$$\hat{I}(\mathbf{f}_{s,t}, \mathbf{g}_{s,t}) = \text{median}_{j\in[d]} \left\{ \sum_{k=1}^{w} \hat{D}^{s,t}_{\mathbf{f}}[j][k] \cdot \hat{D}^{s,t}_{\mathbf{g}}[j][k] \right\}.$$

**Historical window self-join size estimation.** A self join query on stream $\mathbf{f}$ can be processed as a join between $\mathbf{f}$ and itself. One technical issue, however, is that our analysis on join size estimation requires the history lists from two streams to be independent. This is not true if we use one set of history lists for the same stream. We can solve this issue by storing two independent history lists for each counter. This modification will increase the size of the persistent sketch by a factor of 2.

## 4.2 Analysis

In this section, we first explain on an intuitive level why the sampling based persistence technique can provide provable error guarantees for join size estimation, while the baseline method and the PLA based technique cannot. Then we formalize our results and give rigorous proofs (in the appendix). To ensure the error guarantee, we set the parameters as $w = O(1/\varepsilon)$, and $d = O(\log \frac{1}{\delta})$ where $\delta$ is the failure probability.

The fundamental difference between a point query and join size estimation is that the former depends only one the value of one counter per row, while the latter is a *holistic* query that depends on all counters. With the baseline method and the PLA based technique, the bias of estimating a counter is as large as $\Omega(\Delta)$. As the algorithm is deterministic, there is no way to correct the bias. For self-join size estimation, we need to compute the *sum of squares* of all counters, so the bias from each counter will be significantly amplified (by a factor of the $L_1$ norm in the worst case). On the other hand, we show that the sampling technique can provide *unbiased* estimators for the counters, which is the key behind a much better error bound in the final estimate for join and self-join queries.

**Error guarantees.** For historical window point queries, the persistent AMS Sketch can provide the following error bound:

THEOREM 4.1. *Given a time interval $(s,t]$ and an index $i$, the persistent AMS Sketch provides an estimate $\hat{f}_i(s,t)$ for $f_i(s,t)$ such that*

$$\Pr\left[\left|\hat{f}_i(s,t) - f_i(s,t)\right| \le \varepsilon\|\mathbf{f}_{s,t}\|_2 + \Delta\right] \ge 1 - \delta.$$

The $\|\mathbf{f}_{s,t}\|_2$ term inherits from the original AMS Sketch (or rather, the Count Sketch analysis [9]), while the $\Delta$ term is the error introduced by the sampling based persistence technique.

For join size queries involving two streams $\mathbf{f}$ and $\mathbf{g}$, we allow them to use different error parameters $\Delta_{\mathbf{f}}$ and $\Delta_{\mathbf{g}}$ for the persistence part, but for the ephemeral sketch, the parameters (i.e., $w$ and $d$) have to be the same, together with the same set of hash functions. With this setup, we have the following error bound for historical window join size queries:

THEOREM 4.2. *Given a historical window join size query with a time interval $(s,t]$, the persistent AMS Sketches provides an estimate $\hat{I}(\mathbf{f}_{s,t}, \mathbf{g}_{s,t})$ for $I(\mathbf{f}_{s,t}, \mathbf{g}_{s,t})$ such that*

$$\Pr\left[\left|\hat{I}(\mathbf{f}_{s,t}, \mathbf{g}_{s,t}) - I(\mathbf{f}_{s,t}, \mathbf{g}_{s,t})\right| \le \mathcal{E}\right] \ge 1 - \delta,$$

*where* $\mathcal{E} = \varepsilon\sqrt{\left(\|\mathbf{f}_{s,t}\|_2^2 + (\frac{\Delta_{\mathbf{f}}}{\varepsilon})^2\right)\left(\|\mathbf{g}_{s,t}\|_2^2 + (\frac{\Delta_{\mathbf{g}}}{\varepsilon})^2\right)}$.

Self-join size estimation is a special case, and the above theorem holds by setting $\|\mathbf{f}_{s,t}\|_2 = \|\mathbf{g}_{s,t}\|_2$ and $\Delta_{\mathbf{f}} = \Delta_{\mathbf{g}}$.

**Space and query time.** The space complexity analysis is straightforward for the persistent AMS Sketch. The ephemeral sketch has $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$ counters. Each update goes to $O(\log\frac{1}{\delta})$ counters, and in expectation, a fraction of $1/\Delta$ of the updates are sampled. So the total space used is $O((\frac{m}{\Delta} + \frac{1}{\varepsilon^2})\log\frac{1}{\delta})$. This is the same as the baseline method, but remember that the baseline method cannot handle join size queries.

For point queries, since we retrieve $O(d)$ estimates from $O(d)$ history lists, and each involves a binary search, the total query cost is $O(d\log m) = O(\log\frac{1}{\delta}\log m)$. By fractional cascading, this can be improved to $O(\log\frac{1}{\delta} + \log m)$.

For join size queries, we retrieve $O(wd)$ estimates from $O(wd)$ history lists, and each involves a binary search. The total query cost is $O(wd\log m) = O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta}\log m)$ which again can be improved to $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta} + \log m)$ using fractional cascading.

# 5. SKETCHES FOR HISTORICAL QUERIES

Our persistent sketches for general historical windows all have an additive error term that does not exist for the respective ephemeral sketch. But we argued earlier that this is necessary because we allow arbitrary historical windows. Nevertheless, if the starting time of the window is fixed at $s = 0$, the persistent sketches described above can be easily modified so as to eliminate the additive error, therefore yielding the same error guarantee as in the ephemeral sketch.

## 5.1 Persistent Count-Min Sketch for Historical Queries

Recall that a historical query is a historical window query with time interval $(0,t]$. We now show how to modify the persistent Count-Min Sketch to answer historical point queries and heavy hitter queries with additive error $\varepsilon\|\mathbf{f}_t\|_1$. The idea is to adaptively change the PLA error $\Delta$ in a way such that it is always within a constant factor of $\varepsilon\|\mathbf{f}_t\|$, so that the error bound translates to $\varepsilon\|\mathbf{f}_t\|_1 + \Delta = O(\varepsilon\|\mathbf{f}_t\|_1)$.

More precisely, we divide the stream into *epochs*, such that $\|\mathbf{f}_t\|_1$ fluctuates within a constant factor (say, 2) in an epoch. Note that $\|\mathbf{f}_t\|_1$ is simply the total number of insertions minus the total number of deletions so far, so we can track it easily using a single counter. Whenever $\|\mathbf{f}_t\|_1$ doubles or halves, we declare end for the current epoch and starts a new one. Let $[t_i, t_{i+1})$ be the time period for the $i$-th epoch. In the $i$-th epoch, we set $\Delta = \varepsilon\|\mathbf{f}_{t_i}\|_1$ as the PLA error and construct a persistent Count-Min Sketch until time $t_{i+1}$.

To answer a historical query with time instance $t$, we simply find the epoch that contains $t$, and handle the query with the persistent sketch in that epoch.

**Error guarantees.** As before, we set $w = O(1/\varepsilon)$ and $d = O(\log\frac{1}{\delta})$. For historical point and heavy hitters queries, the modified persistent Count-Min Sketch provides the following error bounds:

THEOREM 5.1. *Given a time instance $t$ and an element $i$, the modified persistent Count-Min Sketch provides $\hat{f}_i(t)$ as an estimate for $f_i(t)$ such that*

$$\Pr\left[\left|\hat{f}_i(t) - f_i(t)\right| \le \varepsilon\|\mathbf{f}_t\|_1\right] \ge 1 - \delta.$$

THEOREM 5.2. *Given a time instance $t$, for any element $i$ with $f_i(t) \ge (\phi + \varepsilon)\|\mathbf{f}_t\|_1$, the modified persistent Count-Min a sketch returns it with probability as least $1 - \delta$; for any element $i$ with $f_i(t) < \phi\|\mathbf{f}_t\|_1$, the persistent Count-Min Sketch returns it with probability at most $\delta$.*

Theorem 5.1 and Theorem 5.2 follow directly by setting $s = 0$ and $\Delta = \Theta(\varepsilon\|\mathbf{f}_t\|_1)$ in Theorem 3.1 and Theorem 3.2, respectively.

**Space complexity.** For space complexity analysis, we only focus on point queries; the bound for heavy hitters queries grows by a $\log n$ factor. We are able to prove the following result under the random stream model:

THEOREM 5.3. *In the random stream model, the modified persistent Count-Min Sketch uses $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$ space in expectation.*

In fact, we will prove that in the random turnstile model, the modified persistent sketch uses $O((\sum_{t=1}^{m}\frac{1}{\varepsilon^2\|\mathbf{f}_t\|_1^2}+\frac{1}{\varepsilon})\log\frac{1}{\delta})$ space in expectation. In the standard streaming model, $\|\mathbf{f}_t\|_1 = t$, so the size of the sketch becomes $O(\frac{1}{\varepsilon^2}\log\frac{1}{\delta})$. A more rigorous proof is given in the appendix.

**Query time.** Comparing to the query time analysis in Section 3, the only additional cost is to search for the epoch that contains the time instance $t$, which takes $\log m$ time. Therefore the query time remains the same as before (asymptotically).

## 5.2 Persistent AMS Sketch for Historical Queries

Next, we show how to modify the persistent AMS Sketch to support historical queries with error only related to the $L_2$ norm.

The basic idea is the same as in the Count-Min Sketch, that is, we will adaptively change $\Delta$ so that it is always within a constant factor of $\varepsilon\|\mathbf{f}_t\|_2$. However, unlike the $L_1$-norm, the $L_2$-norm cannot be tracked with a single counter. We will use an additional AMS Sketch of width $w = O(1)$ and depth $d = O(\log\frac{m}{\delta})$, which can approximate $\|\mathbf{f}_t\|_2$ within a constant factor with probability at least $1-\delta$, *for all* time instances. This way we always have a constant approximation of $\|\mathbf{f}_t\|_2$. Then we can divide the stream into epochs, such that the $L_2$ norm of the frequency vector fluctuates within a constant factor in each epoch. Let $[t_i, t_{i+1})$ denote the $i$-th epoch. Within the $i$-th epoch, we set $\Delta = \varepsilon\|\mathbf{f}_i\|_2$ and construct the sampling-based persistent AMS Sketch until time $t_{i+1}$.

To answer a historical query with time $t$, we first find the epoch that contains $t$. Let $[t_{i-1}, t_i)$ be that epoch. Then we simply invoke the same algorithm as in Section 4. One small technical issue is that in Section 4, if we cannot find any predecessor for time $t$ in a history list, we simply set the estimate as 0 (as in Equation (1)). Here we have multiple epochs, and a counter in an epoch may not start from 0, thus we need to record the starting value of each counter in each epoch. When we cannot find a predecessor in a counter's history list, we set the estimate to its starting value in this epoch.

**Error guarantees.** We set $w = O(1/\varepsilon^2)$ and $d = O(\log\frac{1}{\delta})$. For historical point and join size queries, the modified persistent AMS Sketch can provide the following error bounds:

THEOREM 5.4. *Given a time instance $t$ and an element $i$, the modified persistent AMS Sketch provides an estimate $\hat{f}_i(t)$ for $f_i(t)$ such that*

$$\Pr\left[\left|\hat{f}_i(t) - f_i(t)\right| \le \varepsilon\|\mathbf{f}_t\|_2\right] \ge 1-\delta.$$

THEOREM 5.5. *Given a time instance $t$, the modified persistent AMS Sketches provide an estimate $\hat{I}(\mathbf{f}_t, \mathbf{g}_t)$ for $I(\mathbf{f}_t, \mathbf{g}_t)$ such that*

$$\Pr\left[\left|\hat{I}(\mathbf{f}_t, \mathbf{g}_t) - I(\mathbf{f}_t, \mathbf{g}_t)\right| \le \varepsilon\|\mathbf{f}_t\|_2\|\mathbf{g}_t\|_2\right] \ge 1-\delta.$$

These two theorems follow by setting $s = 0$ and $\Delta_{\mathbf{f}} = \Theta(\varepsilon\|\mathbf{f}_t\|_2)$, $\Delta_{\mathbf{g}} = \Theta(\varepsilon\|\mathbf{g}_t\|_2)$ in Theorem 4.1 and Theorem 4.2, respectively.

**Space complexity** We have the following result on the space complexity of the modified persistent AMS Sketch.

THEOREM 5.6. *In the standard streaming model, the modified persistent AMS Sketch uses expected space $O((\frac{\sqrt{m}}{\varepsilon}+\frac{1}{\varepsilon^2})\log\frac{1}{\delta})$.*

We will prove the space usage is $O((\sum_{t=1}^{m}\frac{1}{\varepsilon\|\mathbf{f}_t\|_2}+\frac{1}{\varepsilon^2})\log\frac{1}{\delta})$ in the turnstile model. In the standard streaming model, we have $\|\mathbf{f}_t\|_2 = \sqrt{t}$, so the size of the sketch becomes $O((\frac{\sqrt{m}}{\varepsilon}+\frac{1}{\varepsilon^2})\log\frac{1}{\delta})$. As with the modified Count-Min Sketch, there is no change in the query time for the modified persistent AMS Sketch.

# 6. EXPERIMENTS

## 6.1 Experimental setup

We conducted an experimental study on synthetic and real-world data sets to evaluate the goodness of the proposed persistent sketches. They are evaluated primarily on two measures: the space used by the sketch, and the accuracy of the query results.

**Data sets.** For synthetic data, we generated a data set called Zipf_3, which consists of 1 million items randomly drawn from a universe $[2^{24}]$ with Zipf distribution of coefficient 3. This is a highly skewed data set. For real-world data, we used the 1998 World Cup web site access log. We selected 7,000,000 requests from Day 46, and insert each request as an update according to the timestamp attribute. We are interested in two attributes: 1) objectID, an anonymized integer of the requested URL, and 2) clientID, an anonymized integer of the IP address of the requests. Each timestamp, clientID, and objectID can fit into a 32-bit integer. We denote the set of (timestamp, clientID) tuples as ClientID, and the set of the (timestamp, objectID) tuples as ObjectID. We note that ClientID is a very uniform data set, with maximum frequency being 14645, while ObjectID is more skewed, with most frequencies concentrating on around 500 items. The two data sets exhibit different characteristics in terms of join size and heavy hitters.

**Competitors and parameters.** In the following, we refer to the two persistent sketches by their persistence technique, i.e., we use the name PLA for the persistent Count-Min Sketch , and Sample for the persistent AMS Sketch. We compare PLA and Sample to the two baseline solutions: PWC_CountMin and PWC_AMS. Recall that in these two sketched, we record the value of a counter when it has deviated from the last recorded value by more than $\Delta$, so it is essentially using a piecewise-constant function to approximate each counter, hence the names.

We set the width and depth of all ephemeral sketches to be $w = 20000$ and $d = 7$, regardless of the type of the sketches. Note that this corresponds to $\varepsilon = 0.000005$ for the persistent Count-Min Sketch, and $\varepsilon = 0.01$ for the persistent AMS Sketch. The hash functions used by the sketches are pairwise and 4-wise independent functions constructed with the polynomial universal hashing method by Carter and Wegman [8]. All experiments are conducted on a machine with 16G RAM and a 3.00GHz processor.

## 6.2 Space usage

We measure the *extra space* needed to make an ephemeral sketch persistent, that is, the space used by the sketch excluding the size of the ephemeral sketch. As each sketch uses different approaches to store information, we will measure the actual space in terms of the number of machine words used. For Sample, PWC_CountMin and PWC_AMS, we need an integer to store the recorded value, and another integer to store the timestamp, so the space usage is the number of recorded values multiplied by 2. On the other hand, each segment in PLA needs 3 parameters: the slope $a$, the offset $b$ and the starting timestamp $t$. Therefore the space usage for PLA is the number of segments multiplied by 3.

The tradeoffs between the space usage and the error parameter $\Delta$ are shown in Figure 3. Note that we use log scale on both axes. Our first observation from the results is that the space used by PLA is indeed smaller than the worst-case space bound $O(dm/\Delta)$. In

Zipf_3 where each update is drawn from a Zipf distribution independently, the space usage of PLA is up to 500 times smaller than the worst-case bound. This concurs with the inverse quadratic relation between space and $\Delta$ in Theorem 3.3. We also notice that the space usage for Sample is always $O(dm/\Delta)$ in all three data sets. This is because on average we exactly sample one out of every $\Delta$ tuples, regardless of the data distribution.

Finally, we observe that while PWC_CountMin and PWC_AMS use almost the same space as the theoretical bound on Zipf_3 and ObjectID, they do exhibit an accelerated space drop in ClientID after $\Delta \geq 300$, and almost match the space usage of PLA. This phenomenon can be explained as follow. By the construction of PWC_CountMin and PWC_AMS, they do not record any value (or generate a PLA segment) for counters less than $\Delta$. So on a "long tail" data distribution, the additional space usage for most counters will be 0 once $\Delta$ exceeds some threshold. Note that this does not make PWC_CountMin and PWC_AMS superior to Sample since (1) the space drop will hurt the quality of the approximation, which we shall see later; and (2) the space drop makes it very hard to get desired space by controlling $\Delta$, unless we know the data distribution beforehand. On the other hand, in Sample, we can precisely control the space usage by adjusting the error parameter $\Delta$.

## 6.3 Query accuracy

To evaluate the qualities of the approximation provided by the persistent sketches, we fix a historical window $(s, t]$ with $s = 0.2 * m$ and $t = 0.6 * m$, and measure the actual errors for various historical window queries with time interval $(s, t]$.

**Point Query.** We measure the absolute error between the estimators and the actual frequencies. More precisely, if $f_i$ is the actual frequency and $\hat{f}_i$ is the estimator, then the absolute error of the approximation is equal to $\left|\hat{f}_i - f_i\right|$. We used the top 1000 most frequent elements in the time interval $(s, t]$ from each data sets, and queried about them with each sketch over $(s, t]$. We then took the average absolute error of the 1000 point queries. We constructed the persistent sketches for various values of $\Delta$, and measured the average error for each $\Delta$. Note that we do not evaluate point queries for Sample, since answering queries only involve 1 counter per row is not the strong point for Sample, as we suggested in Section 4.

Figure 4 shows how the average measured errors varies according to the error parameter $\Delta$, and Figure 5 shows the tradeoffs between the measured error and the actual space usage, for each sketch on the 3 data sets. From Figure 5a and Figure 5c we observe that PLA provides a better accuracy-space tradeoffs on Zipf_3 and ObjectID. This is as expected, since PLA uses smaller space on these two data sets, and provides the same error guarantee as the baseline solution. Moreover, Figure 4a and Figure 4c suggest that on these two data sets, even for a fixed $\Delta$, the measured error of PLA is smaller than that of PWC_CountMin and PWC_AMS. We conjecture that this is due to the fact that these two data sets exhibit more randomness than ClientID, which may reduce bias for a linear approximation.

From Figure 5b, we do not see a major difference between PLA and PWC_CountMin or PWC_AMS, with PLA and PWC_AMS performing slightly better. The main reason is that the sketch sizes of PWC_CountMin and PWC_AMS are also much smaller than $O(dm/\Delta)$ on ClientID, which offsets the space advantage of PLA. Meanwhile, Figure 4b suggests there is no significant difference PLA and PWC_CountMin or PWC_AMS in terms of actual error, as all the curves rise rapidly to 500. This is due to the fact that ClientID is a very uniform data set, so the frequencies are hard to approximate for any method.

**Heavy Hitters.** Recall that a historical window heavy hitters query with time interval $(s, t]$ should return elements with frequency $\geq \phi\|\mathbf{f}_{s,t}\|$. We set $\phi = 0.0002$, and issue a historical window heavy hitters query with $(s, t]$ on PLA and PWC_CountMin. The quality of heavy hitters query is evaluated by *precision* and *recall*, where precision is the fraction of heavy hitters returned by the sketch that are actual heavy hitters, and recall is the fraction of actual heavy hitters that are returned by the sketch.

Figure 6 shows how the space usage varies according to the error parameter $\Delta$. These tradeoffs concur with Figure 3 as they are simply scaled by a factor $\log n$ according to the construction in Section 3. Figure 7 shows how the measured error varies according to the error parameter $\Delta$, and Figure 8 shows the tradeoffs between the measured error and space usage. Note that we use log scale on the x-axes for both figures. We first observe that both PLA and PWC_CountMin achieve better precision than recall, which reveal that they tend to return a subset of real heavy hitters. Figure 7 shows that for a fixed $\Delta$, PWC_CountMin provides a slightly better precision, while PLA provides a significantly better recall. For data sets Zipf_3 and ObjectID, Figure 8a and 8c show that the recall of PWC_CountMin quickly becomes unusable as the sketch size drops to $10^4$, while PLA retains both a high recall and a high precision for smaller sketch size. Finally, Figure 8b suggests that there is no clear winner between PLA and PWC_CountMin on ClientID, which concurs with our observation in the point query experiments.

**Self-Join Size Estimation.** The quality of join size approximations has been evaluated by issuing a historical window self-join size query with the same time interval $(s, t]$ as above. We measured the relative error between the estimators and the actual self-join size. More precisely, let $I_{s,t}(\mathbf{f}, \mathbf{f})$ denote the actual self-join size, and $\hat{I}_{s,t}(\mathbf{f}, \mathbf{f})$ denote the estimator, then the relative error is equal to $\frac{|\hat{I}_{s,t}(\mathbf{f},\mathbf{f}) - I_{s,t}(\mathbf{f},\mathbf{f})|}{I_{s,t}(\mathbf{f},\mathbf{f})}$.We constructed the persistent sketches for various values of $\Delta$, and issued self-join size queries with $(s, t]$ and computed the relative error. We repeated the process 10 times and took the average relative error.

Figure 9 shows how the measured error varies according to the error parameter $\Delta$, and Figure 10 shows the tradeoffs between the measured error and the space usage, for self-join size queries on the 3 data sets. Note that we use log scale on the y-axes for Figure 9 and log scale on both axes for Figure 10. As expected, Sample gives better accuracy in general. However, the performance gap varies on different data sets. Figure 9c shows that on ObjectID, both Sample and PWC_AMS perform well for large sketch size ($\geq$ 50000). However, once the sketch size drops down to 1000 to 5000, the errors of Sample are 5 to 10 times smaller than that of PWC_CountMin or PWC_AMS on average. This translates to a gap of around 5 to 10 on the error-space tradeoff in Figure 10. For data set ClientID, Figure 10b shows that Sample provides significantly better error-space tradeoffs. For reasonable error range (0.0001 to 0.1), the space usage of Sample is 10 to 100 times smaller than that of PWC_AMS or PWC_CountMin. Figure 9b also suggests that the errors of PWC_CountMin and PWC_AMS grow rapidly to close to 1, for $\Delta \leq 1000$. This makes it difficult to choose $\Delta$ in practice. On the other hand, Sample provides a much less skewed tradeoffs between error and $\Delta$, which enable us to achieve desired error-space tradeoff by manipulating $\Delta$.

Figure 9a and Figure 10a indicate that on the synthetic data set Zipf_3, the advantage of Sample over the baseline solutions becomes less clear. The accuracy of Sample is 2-5 times better than that of PWC_AMS or PWC_CountMin on average for a fixed sketch size. This can be explained theoretically by observing that each update from the synthetic data set is independently generated,

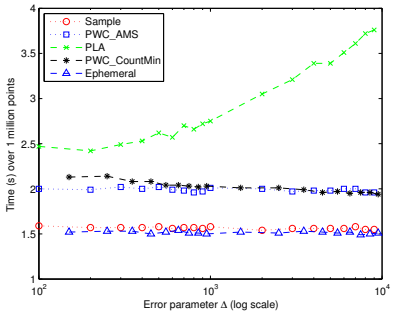



**Figure 2: The processing time over 1 million updates**

and thus the estimators provided by the baseline sketches can be viewed as a random sample without compensation.

## 6.4 Update time

Finally, Figure 2 shows the processing time over 1 million updates. We see that Sample is the fastest, followed up by PWC_CountMin and PWC_AMS. PLA has the highest update cost, which scales roughly with $\log \Delta$. However, all running times are within a small constant factor of the running time for ephemeral sketch, meaning that we do not pay a large time overhead in making the sketches persistent.

# 7. CONCLUSION

In this paper, we present sketching techniques that support historical window queries and historical queries, which give the Count-Min Sketch and the AMS Sketch the ability to do statistical tracking of the entire history, rather than just the current status. We have shown that our persistent sketches are theoretically sound by proving various space and error bounds. We also have evaluated the persistent sketches on synthetic and real-world data, showing that they provide satisfying performance in practice.

# 8. REFERENCES


[1] N. Alon, P. B. Gibbons, Y. Matias, and M. Szegedy. Tracking join and self-join sizes in limited storage. In *Proc. ACM Symposium on Principles of Database Systems*, pages 10–20, 1999.

[2] N. Alon, Y. Matias, and M. Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and System Sciences*, 58(1):137–147, 1999.

[3] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *Proc. ACM Symposium on Principles of Database Systems*, pages 286–296, 2004.

[4] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *Proc. ACM Symposium on Principles of Database Systems*, 2002.

[5] B. Becker, S. Gschwind, T. Ohler, B. Seeger, and P. Widmayer. An asymptotically optimal multiversion B-tree. *The VLDB Journal*, 5(4):264–275, 1996.

[6] V. Braverman, R. Ostrovsky, and C. Zaniolo. Optimal sampling from sliding windows. In *Proc. ACM Symposium on Principles of Database Systems*, pages 147–156, 2009.

[7] G. S. Brodal, S. Sioutas, K. Tsakalidis, and K. Tsichlas. Fully persistent B-trees. In *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2012.

[8] J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *Proc. ACM Symposium on Theory of Computing*, pages 106–112, 1977.

[9] M. Charikar, K. Chen, and M. Farach-Colton. Finding frequent items in data streams. In *Proc. International Colloquium on Automata, Languages, and Programming*, pages 693–703. Springer, 2002.

[10] B. Chazelle and L. Guibas. Fractional cascading: I. A data structuring technique. *Algorithmica*, 1(1), 1986.

[11] G. Cormode and S. Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 55(1):58–75, 2005.

[12] G. Cormode and S. Muthukrishnan. What's hot and what's not: tracking most frequent items dynamically. *ACM Transactions on Database Systems*, 30(1):249–278, 2005.

[13] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows. *SIAM Journal on Computing*, 31(6):1794–1813, 2002.

[14] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 61–72, 2002.

[15] J. R. Driscoll, N. Sarnak, D. D. Sleator, and R. Tarjan. Making data structures persistent. *Journal of Computer and System Sciences*, 38:86–124, 1989.

[16] A. C. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. Surfing wavelets on streams: One-pass summaries for approximate aggregate queries. In *Proc. International Conference on Very Large Data Bases*, volume 1, pages 79–88, 2001.

[17] S. Guha and A. McGregor. Stream order and order statistics: Quantile estimation in random-order streams. *SIAM Journal on Computing*, 38(5), 2009.

[18] D. Lomet, R. Barga, M. F. Mokbel, G. Shegalov, R. Wang, and Y. Zhu. Immortal DB: transaction time support for SQL server. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 939–941, 2005.

[19] D. Lomet and B. Salzberg. Access methods for multiversion data. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 315–324, 1989.

[20] D. B. Lomet and F. Li. Improving transaction-time dbms performance and functionality. In *Proc. IEEE International Conference on Data Engineering*, pages 581–591, 2009.

[21] A. Metwally, D. Agrawal, and A. E. Abbadi. An integrated efficient solution for computing frequent and top-k elements in data streams. *ACM Transactions on Database Systems*, 31(3):1095–1133, 2006.

[22] S. Muthukrishnan. *Data streams: algorithms and applications*. Foundations and trends in theoretical computer science. Now Publishers, 2005.

[23] L. Neumeyer, B. Robbins, A. Nair, and A. Kesari. S4: Distributed stream computing platform. In *Pro. IEEE International Conference on Data Mining Workshops*, pages 170–177, 2010.

[24] J. O'Rourke. An on-line algorithm for fitting straight lines between data ranges. *Communications of the ACM*, 24(9):574–578, 1981.

[25] C. Plattner, A. Wapf, and G. Alonso. Searching in time. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 754–756, 2006.

[26] F. Rusu and A. Dobra. Statistical analysis of sketch estimators. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 187–198, 2007.

[27] A. D. Sarma, M. Theobald, and J. Widom. Live: a lineage-supported versioned dbms. In *Scientific and Statistical Database Management*, pages 416–433. Springer, 2010.

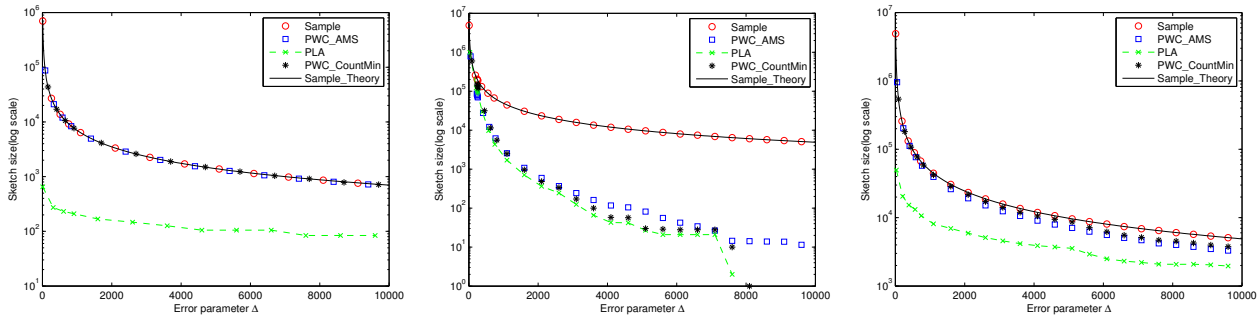
[28] R. Shaull, L. Shrira, and H. Xu. Skippy: a new snapshot indexing method for time travel in the storage manager. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 637–648, 2008.

[29] L. Shrira and H. Xu. Snap: Efficient snapshots for back-in-time execution. In *Proc. IEEE International Conference on Data Engineering*, pages 434–445, 2005.

[30] Y. Tao, K. Yi, C. Sheng, J. Pei, and F. Li. Logging every footstep: Quantile summaries for the entire history. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 639–650, 2010.

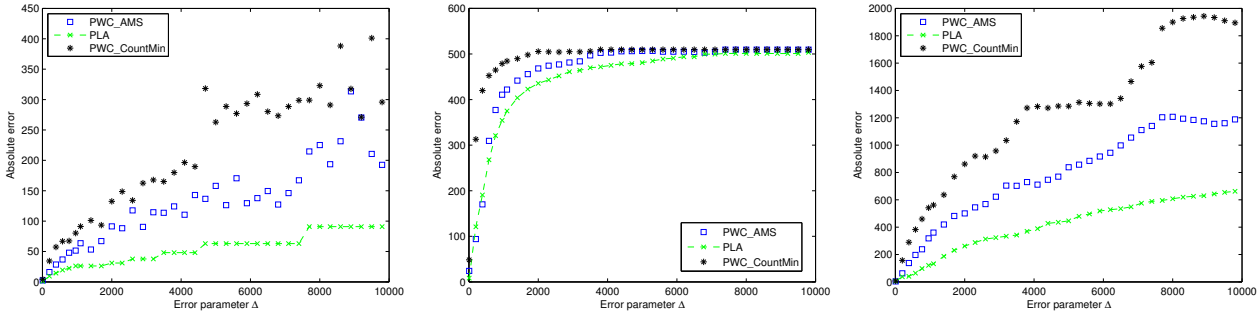
[31] P. J. Varman and R. M. Verma. An efficient multiversion access structure. *IEEE Transactions on Knowledge and Data Engineering*, 9(3):391–409, 1997.

[32] M. Zaharia, T. Das, H. Li, T. Hunter, S. Shenker, and I. Stoica. Discretized streams: Fault-tolerant streaming computation at scale. In *Proc.ACM Symposium on Operating Systems Principles*, pages 423–438, 2013.



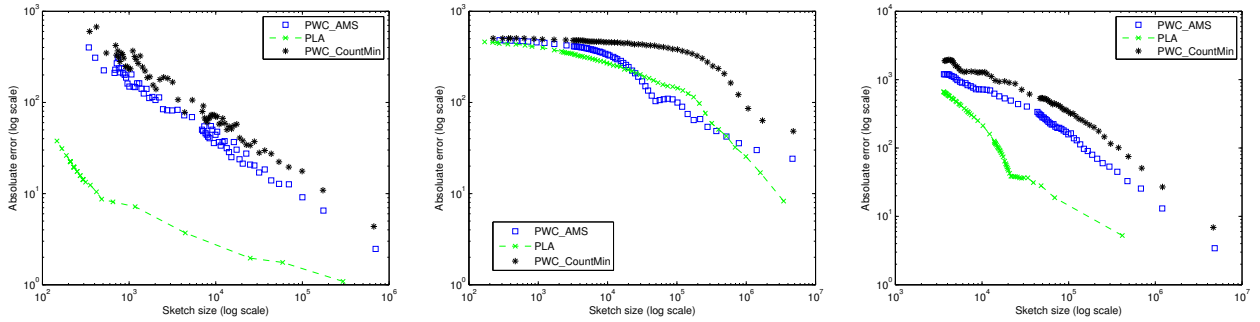

(a) Zipf_3 (b) ClientID (c) ObjectID

**Figure 3: Tradeoffs between sketch size and error parameter $\Delta$.**



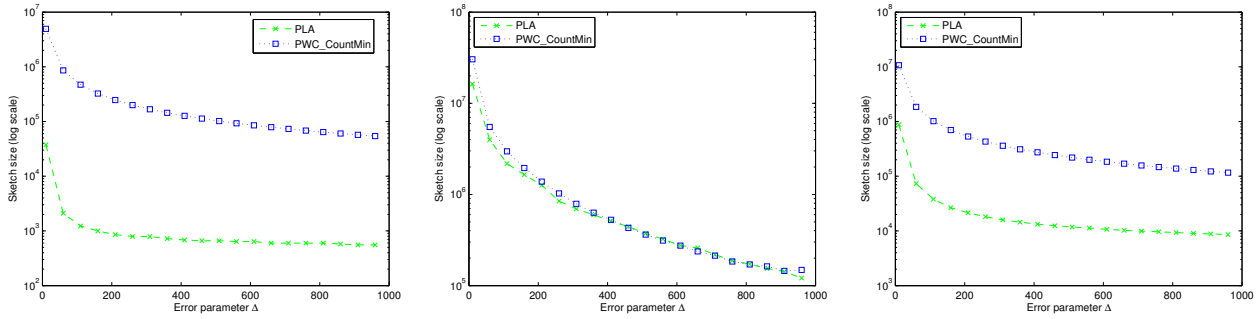

(a) Zipf_3 (b) ClientID (c) ObjectID

**Figure 4: Tradeoffs between actual error and $\Delta$ for point queries.**



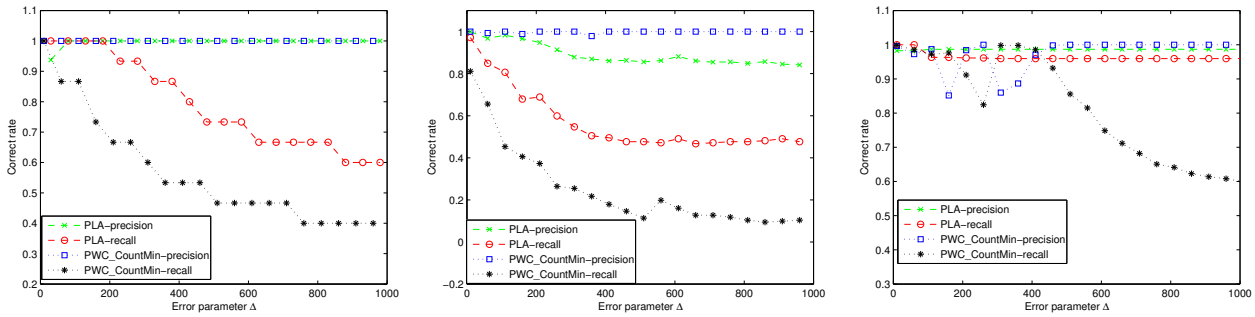

(a) Zipf_3 (b) ClientID (c) ObjectID

**Figure 5: Tradeoffs between actual error and actual sketch size for point queries.**



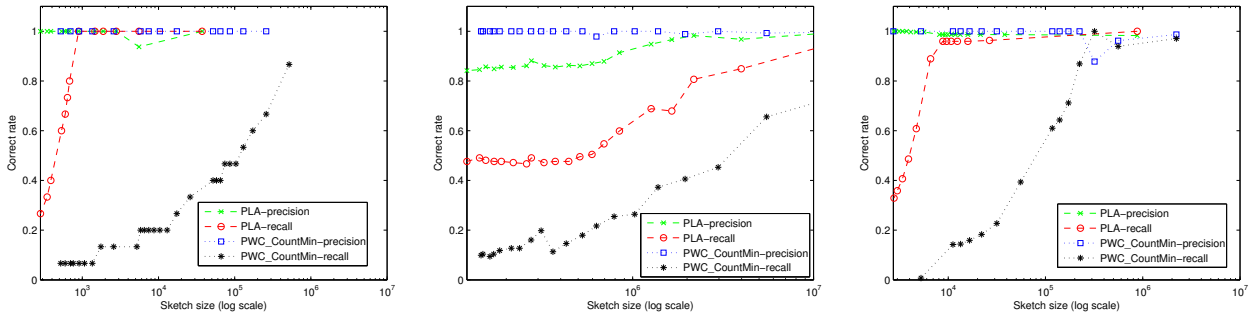

(a) Zipf_3 (b) ClientID (c) ObjectID

**Figure 6: Tradeoffs between sketch size and $\Delta$ for heavy hitters queries.**



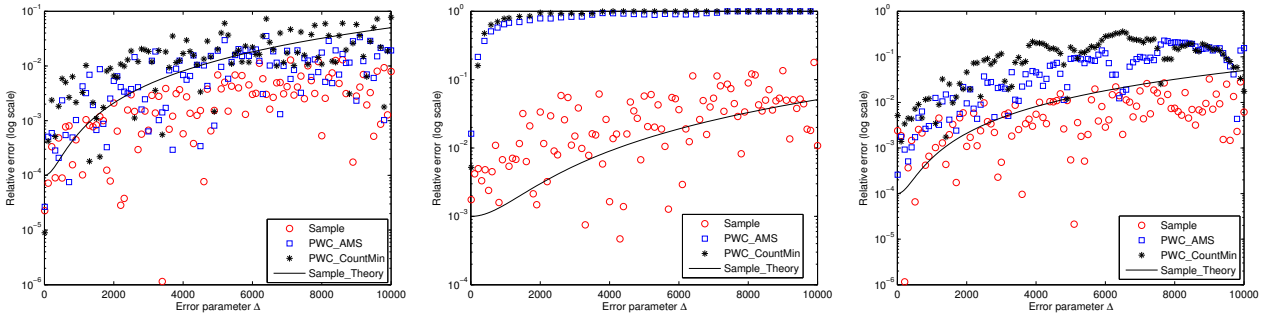

(a) Zipf_3 (b) ClientID (c) ObjectID

**Figure 7: Tradeoffs between the precision and $\Delta$ for heavy hitters queries.**



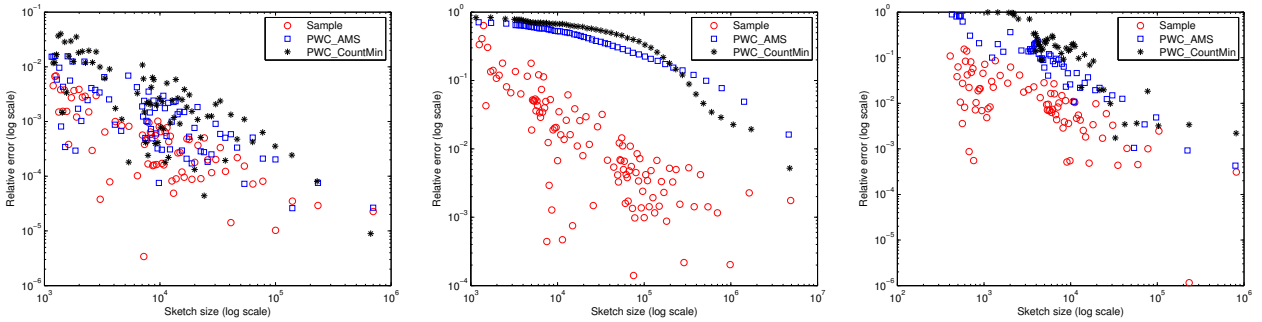

(a) Zipf_3 (b) ClientID (c) ObjectID

**Figure 8: Tradeoffs between precision and actual sketch size for heavy hitters queries.**




(a) Zipf_3 (b) ClientID (c) ObjectID

**Figure 9: Tradeoffs between actual error and $\Delta$ for self-join size queries.**




(a) Zipf_3 (b) ClientID (c) ObjectID

**Figure 10: Tradeoffs between actual error and actual space for self-join size queries.**

# APPENDIX

# A. DETAILED PROOFS

## A.1 Proof of Theorem 3.1

PROOF. Consider a single row $C$. For the ease of presentation, we ignore the row index $j$ and use $C[k]$ to denote the $k$-th counter. Let $h$ denote the hash function. We define $C^l[k]$ to be the value of counter $C[k]$ at time $l$, and correspondingly $D^{s,t}[k] = C^t[k] - C^s[k]$ for $k \in [w]$. Since Count-Min Sketch is a linear sketch, $D^{s,t} = \{D^{s,t}[k] \mid k \in [w]\}$ is a Count-Min Sketch on stream $\mathbf{f}_{s,t}$. We call $D^{s,t}$ the *ephemeral Count-Min Sketch* on stream $\mathbf{f}_{s,t}$. By the property of the Count-Min Sketch, $D^{s,t}$ provides $D^{s,t}[h(i)]$ as an estimate for $f_i(s,t)$ such that

$$\Pr\left[\left|D^{s,t}[h(i)] - f_i(s,t)\right| \le \varepsilon\|\mathbf{f}_{s,t}\|_1\right] \ge 1/2.$$

Let $\hat{D}^{s,t}[h(i)]$ denote the estimator provided by the PLA generator for $D^{s,t}[h(i)]$, it follows that $\|\hat{D}^{s,t}[h(i)] - D^{s,t}[h(i)]\| \le 2\Delta$. Shrinking $\Delta$ by a factor of 2 follows that

$$\Pr\left[\left|\hat{D}^{s,t}[h(i)] - f_i(s,t)\right| \le \varepsilon\|\mathbf{f}_{s,t}\|_1 + \Delta\right] \ge 1/2.$$

Standard Chernoff bound shows that by taking the median of $O(\log \frac{1}{\delta})$ independent estimates, we can boost the successful probability to $1-\delta$, and the Theorem follows. □

## A.2 Proof of Theorem 3.2

PROOF. We note that we need to perform $\frac{2\log n}{\Phi}$ point queries to identify all heavy hitters at level 0. It is sufficient to prove all these point queries succeed with probability $1-\delta$. Since we set the depth of each sketch to be $d = \Theta(\log \frac{\log n}{\Phi\delta})$, a single query succeeds with probability $1 - \frac{\Phi\delta}{2\log n}$. By union bound, all $\frac{\log n}{\Phi\delta}$ queries succeed with probability $1-\delta$, and the theorem follows. □

## A.3 Proof of Theorem 3.3

Consider a single row in a persistent Count-Min Sketch. If we can prove that the expected total number of segments used by the piece-wise linear functions in this row is $O(\frac{m}{\Delta^2})$, the Theorem will follow. Consider a fixed counter, since each update is drawn independently from a probability distribution, we can assume that at each time instance, with probability $p_1$, $p_2$ and $1-p_1-p_2$, each update increases by 1, decreases by 1, and remains unchanged, respectively. Since at each time instance at most one of the counter will be updated, so the summation of $p_1$ and $p_2$ over all counters in this row is less or equal to 1. Thus it is sufficient to prove the expected number of segments used for this counter is $O(\frac{(p_1+p_2)m}{\Delta^2})$.

We use $v(t)$ to denote the value of the counter at time $t$. Let $S$ denote the number of segments used by the PLA generator to cover all $m$ data points, and $H_i$ denote the number of data points covered by the $i$-th segment. Notice that $S$ and $H_i$'s are all random variables. Our goal is to prove $\mathrm{E}[S] \le \frac{2(p_1-p_2)m}{\Delta^2}$.

### *Random Walk and Escaping Time.*

Consider the following random walk process: starting from the origin, at each timestamp a player moves 1 step forward with probability $p_1$, and move 1 step backward with probability $p_2$, and stays at current position with probability $1-p_1-p_2$. It is easy to see that the player will vibrate along the line $y = (p_1-p_2)x$. Let $\tau$ denote the *escaping time*, that is, the first time the player escapes from the line $y = (p_1-p_2)x$ by deviation $\Delta$. It is easy to see that $\tau$ is the time before linear function $y = (p_1-p_2)x$ fails to serve as a valid linear approximation, and thus is a lower bound on the number of points covered by a segment. To prove Theorem 3.3, we need the following lemma on the expectation and variance of $\tau$.

LEMMA A.1. *For $\Delta > 10$, the escaping time $\tau$ satisfies $\mathrm{E}[\tau] = \Delta^2/\alpha$ and $\mathrm{Var}[\tau^2] \le 5\Delta^4/6\alpha^2$.*

Here $\alpha$ is defined as follow. Let $X_1, \ldots, X_m$ be $m$ independently and identically distributed random variables. Each $X_i$ takes value $1-(p_1-p_2)$ with probability $p_1$, $-1-(p_1-p_2)$ with probability $p_2$, and value $-(p_1-p_2)$ with probability $1-p_1-p_2$. Let $S_l = \sum_{i=1}^{k} X_i$, then $S_k$ is the deviation from the line $y = (p_1-p_2)x$ at time $k$. The random walk stops at time $\tau$ where $S_\tau = \Delta$ or $-\Delta$. We notice that $\mathrm{E}[X_i] = 0$. Let $\alpha = \mathrm{E}[X_i^2]$, $\beta = \mathrm{E}[X_i^3]$, $\gamma = \mathrm{E}[X_i^4]$ denote the second, third and fourth moments of $X_i$.

### *Martingales and Optional Stopping Theorem.*

To prove Lemma A.1, we will make use of the fact that the escaping time $\tau$ is a type of stopping time on certain martingales. Formally, we define four martingale sequences.

LEMMA A.2. *For $k \in [m]$, the following four sequences are martingales: $Z_1 = \{S_k\}$, $Z_2 = \{S_k^2 - \alpha k\}$, $Z_3 = \{S_k^3 - 3\alpha k S_k - \beta k\}$, $Z_4 = \{S_k^4 - 6\alpha k S_k^2 - 4\beta k S_k + 3\alpha^2 k^2 + (3\alpha^2 - \gamma)k\}$.*

PROOF. We call $Z_1$, $Z_2$, $Z_3$ and $Z_4$ the first, second, third and fourth martingale sequence of $S$, respectively. To prove they are indeed martingales, we verify using the definition of martingale. More precisely, we will prove $E[Z_j[k] \mid Z_j[k-1]] = Z_j[k-1]$ for any $k$ and $j \in [4]$.

For the first martingale list, we have $\mathrm{E}[Z_1[k] \mid Z_1[k-1]] = \mathrm{E}[S_k \mid S_{k-1}] = Z_1[k-1]$. For the second martingale list, we verify $\mathrm{E}[Z_2[k] \mid Z_2[k-1]] = Z_2[k-1]$ by

$$\mathrm{E}[S_k^2 - \alpha k \mid S_{k-1}] = S_{k-1}^2 + \alpha - \alpha k = S_{k-1}^2 - \alpha(k-1).$$

For the third martingale list, we verify $\mathrm{E}[Z_3[k] \mid Z_3[k-1]] = Z_3[k-1]$ as follow

$$\begin{aligned}\mathrm{E}[S_k^3 - 3\alpha k S_k - \beta k \mid S_{k-1}] &= S_{k-1}^3 + 3\alpha S_{k-1} + \beta - 3\alpha k S_{k-1} - \beta k \\ &= S_{k-1}^3 - 3\alpha(k-1)S_{k-1} - \beta(k-1).\end{aligned}$$

Finally, we verify $\mathrm{E}[Z_4[k] \mid Z_4[k-1]] = Z_2[k-1]$ as follow:

$$\begin{aligned}&\mathrm{E}[S_k^4 - 6\alpha k S_k^2 - 4\beta k S_k + 3\alpha^2 k^2 + (3\alpha^2 - \gamma)k \mid S_{k-1}] \\ &= S_{k-1}^4 - 6\alpha(k-1)S_{k-1}^2 - 4\beta(k-1)S_{k-1} + 3\alpha^2(k-1)^2 \\ &\quad + (3\alpha^2 - \gamma)(k-1).\end{aligned}$$

Thus the lemma follows. □

We also need the well-known *optional stopping theorem*:

LEMMA A.3 (OPTIONAL STOPPING THEOREM). *Let $Z[k]$ be a martingale and $T$ be a stopping time with respect to a filter $(\mathcal{F}_k)$. Then $\mathrm{E}[Z[T]] = \mathrm{E}[Z[0]]$ provided: 1. $E[T] < \infty$; 2. $E[|Z[k] - Z[k-1]| \mid \mathcal{F}_k)] \le c$ for some $c$.*

It is easy to see that all four martingale sequences in Lemma A.2 have bounded stopping time expectations. We also note that the difference $|Z_i[k] - Z_i[k-1]|$ is bounded by $O(\Delta^4)$ for $i \in [4]$, so the optional stopping theorem applies to all four martingales.

PROOF OF LEMMA A.1. For the expectation of $\tau$, we first notice that $S_\tau$ takes value from $\{\Delta, -\Delta\}$. Suppose $\Pr[S_\tau = \Delta] = p^*$, and $\Pr[S_\tau = -\Delta] = 1-p^*$. Applying optional stopping theorem to the first martingale list $Z_1 = \{S_k\}$, we have $\mathrm{E}[S_\tau] =$

$\mathrm{E}[S_0] = 0$, which implies $p^* = 1/2$. Applying the optional stopping theorem to the second martingale list $Z_2 = \{S_k^2 - \alpha k\}$ follows that $\mathrm{E}[S_\tau^2 - \alpha\tau] = \mathrm{E}[S_0^2 - 0] = 0$. This implies

$$\mathrm{E}[\alpha\tau] = \mathrm{E}[S_\tau^2] = p^*\Delta^2 + (1-p^*)\Delta^2 = \Delta^2,$$

and thus we have $\mathrm{E}[\tau] = \Delta^2/\alpha$. This gives the expectation of $\tau$.

To bound the variance of $\tau$, we first compute $\mathrm{E}[\tau S_\tau]$ as an immediate step. Apply optional stopping theorem to the third martingale list $Z_3 = \{S_k^3 - 3\alpha k S_k - \beta k\}$, and we have $\mathrm{E}[S_\tau^3 - 3\alpha k S_\tau - \beta k] = \mathrm{E}[S_0^3 - 0 - 0] = 0$. It follows that

$$\mathrm{E}[3\alpha\tau S_\tau] = \mathrm{E}[S_\tau^3 - \beta\tau] = \mathrm{E}[S_\tau^3] + \beta\mathrm{E}[\tau] = \frac{\beta\Delta^2}{\alpha},$$

and thus $\mathrm{E}[\tau S_\tau] = \beta\Delta^2/3\alpha^2$. Finally, we consider the fourth matingale sequence $Z_4 = \{S_k^4 - 6\alpha k S_k^2 - 4\beta k S_k + 3\alpha^2 k^2 + (3\alpha^2 - \gamma)k\}$. By optional stopping theorem, we have

$$\mathrm{E}[S_\tau^4 - 6\alpha\tau S_\tau^2 - 4\beta\tau S_\tau + 3\alpha^2\tau^2 + (3\alpha^2 - \gamma)\tau] = 0. \quad (2)$$

Plugging $S_\tau^4 = \Delta^4$, $S_\tau^2 = \Delta^2$, $\mathrm{E}[\tau] = \Delta^2/\alpha$ and $\mathrm{E}[\tau S_\tau] = \beta\Delta^2/3\alpha^2$ in(2), it follows that

$$\begin{aligned}\mathrm{E}[3\alpha^2\tau^2] &= -\Delta^4 + 6\alpha\frac{\Delta^4}{\alpha} + \frac{4\beta^2\Delta^2}{3\alpha^2} - \frac{(3\alpha^2-\gamma)\Delta^2}{\alpha} \\ &= 5\Delta^4 + \left(\frac{4\beta^2}{3\alpha^2} - \frac{(3\alpha^2-\gamma)}{\alpha}\right)\Delta^2.\end{aligned}$$

Notice that since $X_i$ takes value in $[-2, 2]$, we have $\mathrm{E}[X_i^4] \le 2\left|\mathrm{E}[X_i^3]\right| \le 4\mathrm{E}[X_i^2]$, therefore $\gamma \le 2|\beta| \le 4\alpha$, so

$$\mathrm{E}[3\alpha^2\tau^2] \le 5\Delta^4 + (\frac{16}{3} + 4)\Delta^2 \le 5\Delta^4 + \frac{1}{6}\Delta^4 = \frac{11}{3}\Delta^2,$$

Here we use $\Delta^2 \ge 56$. This implies $\mathrm{E}[\tau^2] \le 11\Delta^4/6\alpha^2$, and thus

$$\mathrm{Var}[\tau] = \mathrm{E}[\tau^2] - \mathrm{E}[\tau]^2 = \frac{11\Delta^4}{6\alpha^2} - \frac{\Delta^4}{\alpha^2} = \frac{5\Delta^4}{6\alpha^2}.$$

This completes the proof of Lemma A.1. □

### *Bernoulli Trials and Negative Binomial distribution.*

Withe the help of Lemma A.1, we are able to prove Theorem 3.3.

PROOF OF THEOREM 3.3. By Lemma A.1, we can bound the probability of $\tau$ being too small using Chebyshev's inequality:

$$\Pr\left[\tau \le \frac{\Delta^2}{12\alpha}\right] = \Pr\left[\tau - \frac{\Delta^2}{\alpha} \le -\frac{11\Delta^2}{12\alpha}\right] \le 120/121.$$

It follows $\Pr[\tau \ge \Delta^2/12\alpha] \ge 1/121$. Combining with $\alpha = \mathrm{E}[X_i^2] \le p_1 + p_2$, we have

$$\Pr[\tau \ge \frac{\Delta^2}{12(p_1+p_2)}] \ge \frac{1}{121}, \quad (3)$$

for any $1 \le i \le S$.

We notice that the $H_i$'s, the number of points covered by each segment, are identically and independently distributed (i.i.d.). Equation (3) implies that each $H_i$ covers at least $\frac{\Delta^2}{12(p_1+p_2)}$ points with some constant probability $c = 1/121$. We work with the worst case where $H_i$ covers exactly $\frac{\Delta^2}{12(p_1+p_2)}$ points with probability $d$, and 0 points with probability $1-c$. Note that this modification will not reduce the expected number of segments $\mathrm{E}[S]$. We consider a sequence of independent Bernoulli trials, in which the $i$-th trial succeeds if $H_i$ covers exactly $\frac{\Delta^2}{12(p_1+p_2)}$ points , and fails if $H_i$ covers 0 points. Observe that if we have $\frac{12(p_1+p_2)m}{\Delta^2}$ successful trials, all $m$ points will be covered. Therefore, $S$ is the number of trials until we see $\frac{12(p_1+p_2)m}{\Delta^2}$ successful trials. This immediate implies that the number of fail trials, denoted $S - \frac{12(p_1+p_2)m}{\Delta^2}$, follows the *negative binomial distribution* with successful trial number $\frac{12(p_1+p_2)m}{\Delta^2}$ and probability $c$, and thus has expectation equals to $\frac{12c(p_1+p_2)m}{(1-c)\Delta^2}$. This proves that $\mathrm{E}[S] = O(\frac{(p_1+p_2)m}{\Delta^2})$, and Theorem 3.3 follows. □

## A.4 Proof of Theorem 4.1

Consider one row of counters. To simplify notation, we omit the row index $j$ and use $C[k][b]$ to denote the counters in this row, and use $h$ to denote the hash function for that row. For any $k \in [w]$, let $C^l[k][b]$ be the value of counter $C[k][b]$ at time $l$. We further define $D^l[k] = C^l[k][1] - C^l[k][0]$. By the construction of the persistent AMS Sketch, $D^l[k]$ is the value of the corresponding counter in the ephemeral sketch at time $l$. By the linearity of the AMS Sketch, $D^{s,t}[k] = D^t[k] - D^s[k]$ is the value of the counter if we built an AMS Sketch on $\mathbf{f}_{s,t}$.

### *Unbiasedness of the Estimator.*

We first show that $\hat{D}^{s,t}[k]$ is an unbiased estimator of $D^{s,t}[k]$ with a bounded variance. Define $x_k = \hat{D}^{s,t}[k] - D^{s,t}[k]$ to be the deviation between our estimate from the persistent sketch from the counter value in the AMS Sketch built on $\mathbf{f}_{s,t}$.

LEMMA A.4. *For any $k \in [w]$, $E[x_k] = 0$ and $\mathrm{E}[x_k^2] \le 24\Delta^2$.*

For any time $r$, define $X^r[k] = \hat{C}^r[k][1] - C^r[k][1]$ and $Y^r[k] = \hat{C}^r[k][0] - C^r[k][0]$. Recall $x_i = \hat{D}^{s,t}[k] - D^{s,t}[k]$ is the deviation between the final estimator and ephemeral estimator. It follows that

$$\begin{aligned}\hat{D}^t[k] &= \hat{C}^t[k][1] - \hat{C}^t[k][0] = C^t[k][1] + X^t[k] - (C^t[k][0] + Y^t[k]) \\ &= D^t[k] + X^t[k] - Y^t[k].\end{aligned}$$

Similarly , we have $\hat{D}^s[k] = D^s[k] + X^s[k] - Y^s[k]$ for $s$. Thus

$$\begin{aligned}x_i &= \hat{D}^{s,t}[k] - D^{s,t}[k] = \left(\hat{D}^t[k] - \hat{D}^s[k]\right) - \left(D^t[k] - D^s[k]\right) \\ &= \left(\hat{D}^t[k] - D^t[k]\right) - \left(\hat{D}^s[k] - D^s[k]\right) \\ &= X^t[k] - Y^t[k] - (X^s[k] - Y^s[k]). \quad (4)\end{aligned}$$

Set $p = 1/\Delta$, we have the following lemma that bounds the expectation and variance of $X^r[k]$ and $Y^r[k]$:

LEMMA A.5. *For any time instances $r, q \in [m]$ and $i \in [4w]$, $X^r[k]$ and $Y^q[k]$ are independent, and satisfy (1) $E[X^r[k]] = E[Y^q[k]] = 0$; (2) $\mathrm{E}[X^r[k]^2] \le 1/p^2$ and $\mathrm{E}[Y^q[k]^2] \le 1/p^2$.*

PROOF. Since $\bar{C}^r[k][1]$ and $\bar{C}^r[k][0]$ are independently sampled, it is easy to verify that $X^r[k]$ and $Y^r[k]$ are independent.

By the definitions of $X^r[k]$ and $\hat{C}^r[k][1]$, we have

$$X^r[k] = \begin{cases}\bar{C}^r[k][1] - C^r[k][1] - 1 + \frac{1}{p}, & 0 < \bar{C}^r[k][1] \le C^r[k][1]; \\ \bar{C}^r[k][1] - C^r[k][1], & \bar{C}^r[k][1] = 0.\end{cases} \quad (5)$$

Recall that $\bar{C}^r[k][1]$ is the last sampled predecessor of $C^r[k][1]$, and here we set $\bar{C}^r[k][1] = 0$ is there is no such predecessor $L[j][h_j(i)][1]$. If we review the sample process backwards from $C^r[k][1]$ to $\bar{C}^r[k][1]$, then $\bar{C}^r[k][1]$ is the first successful trial as we keep flipping a coin with success probability $p$, with $C^r[k][1]$ being the upper bound of the number of trials. It follows that

$$\Pr[\bar{C}^r[k] = C^r[k][1] - l] = \begin{cases}(1-p)^l p, & \text{for } 0 \le l < C^r[k][1]; \\ (1-p)^{C^r[k][1]}, & \text{for } l = C^r[k][1].\end{cases} \quad (6)$$

Combing (5) and (6), we have

$$\Pr\left[X^r[k] = \frac{1}{p} - l - 1\right] = \begin{cases}(1-p)^l p, & 0 \le l < C^r[k][1];\\ (1-p)^{C^r[k][1]}, & l = C^r[k][1] + 1 + \frac{1}{p}.\end{cases}$$

We can compute the expectation of $X^r[k][1]$ as follows:

$$\begin{aligned}\mathrm{E}[X^r[k]] &= \sum_{l=0}^{C^r[k][1]-1} (-l - 1 + \frac{1}{p})(1-p)^l p - C^r[k][1](1-p)^{C^r[k][1]}\\ &= C^r[k][1](1-p)^{C^r[k][1]} - C^r[k][1](1-p)^{C^r[k][1]} = 0.\end{aligned}$$

We can bound the expectation of $X^r[k]^2$ as follow:

$$\begin{aligned}\mathrm{E}[X^r[k]^2] &= \sum_{l=1}^{C^r[k][1]} (-l + \frac{1}{p})^2(1-p)^{l-1} p + C^r[k][1]^2(1-p)^{C^r[k][1]}\\ &= \frac{(1-p)(1-(1-p)^{C^r[k][1]})}{p^2} \le \frac{1}{p^2}.\end{aligned}$$

The same analysis holds for $Y^q[k]$, and thus the lemma follows. □

Now we are ready to prove Lemma A.4.

PROOF OF LEMMA A.4. By equation (4), we have

$$\begin{aligned}\mathrm{E}[x_k] &= \mathrm{E}\left[X^t[k] - Y^t[k] - (X^s[k] - Y^s[k])\right]\\ &= \mathrm{E}[X^t[k]] - \mathrm{E}[Y^t[k]] - (\mathrm{E}[X^s[k]] - \mathrm{E}[Y^s[k]]) = 0.\end{aligned}$$

For the second moment $x_k^2$, we have

$$\begin{aligned}\mathrm{E}[x_k^2] &= \mathrm{E}\left[\left(X^t[k] - Y^t[k] - (X^s[k] - Y^s[k])\right)^2\right]\\ &= \mathrm{E}[X^t[k]^2] + \mathrm{E}[Y^t[k]^2] + \mathrm{E}[X^s[k]^2] + \mathrm{E}[Y^s[k]^2]\\ &\quad - 2\mathrm{E}[X^t[k]X^s[k]] - 2\mathrm{E}[Y^t[k]Y^s[k]]. \qquad (7)\end{aligned}$$

The second equation follows by the fact that any cross term of form $X^t[k]Y^t[k]$, $X^t[k]Y^s[k]$, $X^s[k]Y^t[k]$, and $X^s[k]Y^s[k]$ are products of two independent and unbiased random variables, and thus have 0 expectation. By Lemma A.4, we can upper bound the expectation of each square term with $\frac{1}{p^2}$. We also notice that the expectation of a cross term $-2\mathrm{E}[X^t[k]X^s[k]]$ satisfies

$$-2\mathrm{E}[X^t[k]X^s[k]] \le \left(\mathrm{E}[X^t[k]^2] + \mathrm{E}[X^s[k]^2]\right) \le \frac{2}{p^2},$$

Similarly, we have $-2\mathrm{E}[Y^t[k]Y^s[k]] \le \frac{2}{p^2}$. Plugging into (7) follows that $\mathrm{E}[x_k^2] \le \frac{4}{p^2} + \frac{2}{p^2} + \frac{2}{p^2} = \frac{8}{p^2}$. □

### *Proof of Theorem 4.1.*

PROOF. The following lemma follows from the key properties of the AMS Sketch.

LEMMA A.6 ([9]). *For any $j \in [d]$,*

1. $\mathrm{E}[\xi(i) \cdot D^{s,t}[h(i)]] = f_i(s,t)$ *and* $\mathrm{Var}[\xi(i) \cdot D^{s,t}[h(i)]] \le \varepsilon^2 \|\mathbf{f}_{s,t}\|_2^2$;
2. $\mathrm{E}\left[\sum_{k=1}^w D_{\mathbf{f}}^{s,t}[k]^2\right] = \|\mathbf{f}_{s,t}\|_2^2$ *and* $\mathrm{E}[\sum_{k=1}^w D_{\mathbf{f}}^{s,t}[k] D_{\mathbf{g}}^{s,t}[k]] = I(\mathbf{f}_{s,t}, \mathbf{g}_{s,t})$;
3. $\mathrm{Var}\left[\sum_{k=1}^w D_{\mathbf{f}}^{s,t}[k] D_{\mathbf{g}}^{s,t}[k]\right] \le \frac{1}{w}\|\mathbf{f}_{s,t}\|_2 \|\mathbf{g}_{s,t}\|_2$.

The space bound follows by the fact that we sample each of the $m$ updates with probability $p = \Theta(1/\Delta)$ in a row, and thus the expected number of samples is $O(\frac{m}{\Delta})$.

Now we turn to the error bound. For a single row $C$ with hash function $h$, recall that the persistent AMS Sketch uses $\xi(i) \cdot \hat{D}^{s,t}[h(i)]$ as an estimator for $f_i(s,t)$. Thus

$$\mathrm{E}[\xi(i)\hat{D}^{s,t}[h(i)]] = \mathrm{E}\left[\xi(i) D^{s,t}[h(i)] + \xi(i) x_{h(i)}\right] = f_i(s,t).$$

The last equation follows from $\mathrm{E}[D^{s,t}[h(i)]] = f_i(s,t)$ by Lemma A.6, and $\mathrm{E}[x_{h(i)}] = 0$ by Lemma A.4. To utilize Chebyshev's inequality, we consider the variance of $\xi(i) \cdot \hat{D}^{s,t}[h(i)]$:

$$\mathrm{Var}[\xi(i) \cdot \hat{D}^{s,t}[h(i)]] = \mathrm{Var}\left[D^{s,t}[h(i)] + x_{h(i)}\right].$$

Since $D^{s,t}[h(i)]$ and $x_{h(i)}$ are independent, we have

$$\mathrm{Var}[\xi \cdot \hat{D}^{s,t}[h(i)]] = \mathrm{Var}\left[D^{s,t}[h(i)]\right] + \mathrm{Var}\left[x_{h(i)}\right],$$

We know that $\mathrm{Var}[D^{s,t}[h(i)]] \le \varepsilon^2 \|\mathbf{f}_{s,t}\|_2^2$ by Lemma A.6, and $\mathrm{Var}[x_{h(i)}] \le \mathrm{E}[x_{h(i)}^2] \le 8\Delta^2$ by Lemma A.4, it follows that

$$\mathrm{Var}[\xi \cdot \hat{D}^{s,t}[h(i)]] \le \varepsilon^2 \|\mathbf{f}_{s,t}\|_2^2 + 8\Delta^2 \le 8 \cdot (\varepsilon \|\mathbf{f}_{s,t}\|_2 + \Delta)^2.$$

By Chebyshev's inequality, we have

$$\Pr[\left|D^{s,t}[h(i)] - f^{s,t}(i)\right| \ge 4(\varepsilon \|\mathbf{f}_{s,t}\|_2 + \Delta)] \le \frac{1}{2}.$$

Since the final estimator $\hat{f}_i(s,t)$ is the median of $d = O(\log \frac{1}{\delta})$ independent copies of $D^{s,t}[h(i)]$, standard chernoff bound implies

$$\Pr\left[\left|\hat{f}_i(s,t) - f_i(s,t)\right| \ge 4(\varepsilon \|\mathbf{f}_{s,t}\|_2 + \Delta)\right] \ge 1 - \delta.$$

Scaling down $\varepsilon$ and $\Delta$ by a constant factor, Theorem 4.1 follows. □

## A.5 Proof of Theorem 4.2

With the help of Lemma A.6 and Lemma A.4, we compute the expectations and variances of our estimators, and then use Chebyshev's inequality to bound the error probability.

Consider a single row from each sketch. Let $C_{\mathbf{f}}$ and $C_{\mathbf{g}}$ denote the rows, respectively, and $h$ denote the common hash function used by the two rows. We assume the two rows use sample probabilities $p_{\mathbf{f}} = 3/\Delta_{\mathbf{f}}$ and $p_{\mathbf{g}} = 3/\Delta_{\mathbf{g}}$, respectively. Given a window join size query with time range $[s,t]$, recall that we extract $\hat{D}_{\mathbf{f}}^{s,t}[k] = D_{\mathbf{f}}^{s,t}[k] + x_k$ from $C_{\mathbf{f}}$ and $\hat{D}_{\mathbf{g}}^{s,t}[k] = D_{\mathbf{g}}^{s,t}[k] + y_k$ from $C_{\mathbf{g}}$ for all $k \in [w]$, and use

$$\sum_{k=1}^w \hat{D}_{\mathbf{f}}^{s,t}[k]\hat{D}_{\mathbf{g}}^{s,t}[k] = \sum_{k=1}^w (D_{\mathbf{f}}^{s,t}[k] + x_k)(D_{\mathbf{g}}^{s,t}[k] + y_k).$$

to estimate $I(\mathbf{f}_t, \mathbf{g}_t)$. Lemma A.4 gives a probabilistic bound on $x_k$ and $y_k$. We also note $x_k$ and $y_k$ are independent. To simplify the representation, we define *singular cross term* to be a term with one of the following forms: (1) $x_k \cdot r(x_{i_1}, y_{i_2}, D_{\mathbf{f}}^{s,t}[i_3], D_{\mathbf{g}}^{s,t}[i_4])$, for $i \ne i_1$; (2) $y_k \cdot r(x_{i_1}, y_{i_2}, D_{\mathbf{f}}^{s,t}[i_3], D_{\mathbf{g}}^{s,t}[i_4])$, for $i \ne i_2$. Here $r$ denotes an arbitrary function. We claim that all singular cross terms are unbiased.

LEMMA A.7. *The expectation of a singular cross term is* 0.

PROOF. Without loss of generality, assume the singular term $T = x_k r(x_{i_1}, y_{i_2}, D_{\mathbf{f}}^{s,t}[i_3], D_{\mathbf{g}}^{s,t}[i_4])$ where $i \ne i_1$. Observe that $x_k$ is unbiased and independent from $x_{i_1}$, $y_{i_2}$, $D_{\mathbf{f}}^{s,t}[i_3]$ and $D_{\mathbf{g}}^{s,t}[i_4]$, so we have

$$\mathrm{E}[T] = \mathrm{E}[x_k]\,\mathrm{E}\left[r(x_{i_1}, y_{i_2}, D_{\mathbf{f}}^{s,t}[i_3], D_{\mathbf{g}}^{s,t}[i_4])\right] = 0,$$

and the lemma follows. □

PROOF OF THEOREM 4.2. Now we can compute the expectation of the estimation.

$$\mathrm{E}\left[\sum_{k=1}^{w}\hat{D}_{\mathbf{f}}^{s,t}[k]\hat{D}_{\mathbf{g}}^{s,t}[k]\right]=\mathrm{E}\left[\sum_{k=1}^{w}D_{\mathbf{f}}^{s,t}[k]D_{\mathbf{g}}^{s,t}[k]\right]+\sum_{k=1}^{w}\mathrm{E}\left[x_kD_{\mathbf{g}}^{s,t}[k]\right]+\sum_{k=1}^{w}E\left[y_kD_{\mathbf{f}}^{s,t}[i]\right]+\sum_{k=1}^{w}\mathrm{E}\left[x_ky_k\right].$$

By Lemma A.6, the first term $\mathrm{E}\left[\sum_{k=1}^{w}D_{\mathbf{f}}^{s,t}[k]D_{\mathbf{g}}^{s,t}[k]\right]=I(\mathbf{f}_{s,t},\mathbf{g}_{s,t})$. We also observe that the rest terms are summations of singular cross terms, and thus have expectations 0. It follows that

$$E[\hat{I}(\mathbf{f}_{s,t},\mathbf{g}_{s,t})]=\mathrm{E}\left[\sum_{k=1}^{w}D_{\mathbf{f}}^{s,t}[k]D_{\mathbf{g}}^{s,t}[k]\right]=I(\mathbf{f}_{s,t},\mathbf{g}_{s,t}).$$

This proves the unbiasedness of our estimator.

We now bound the variance of our estimation.

$$\mathrm{Var}\left[\hat{I}(\mathbf{f}_{s,t},\mathbf{g}_{s,t})\right]=\mathrm{E}\left[\hat{I}(\mathbf{f}_{s,t},\mathbf{g}_{s,t})^2\right]-\mathrm{E}\left[\hat{I}(\mathbf{f}_{s,t},\mathbf{g}_{s,t})\right]^2. \tag{8}$$

Note that $\mathrm{E}[(\sum_{k=1}^{w}\hat{D}_{\mathbf{f}}^{s,t}[k]\hat{D}_{\mathbf{g}}^{s,t}[k])^2]$ can be expanded as

$$\mathrm{E}\left[\hat{I}(\mathbf{f}_{s,t},\mathbf{g}_{s,t})^2\right]=\mathrm{E}\left[\left(\sum_{k=1}^{w}(D_{\mathbf{f}}^{s,t}[k]+x_k)(D_{\mathbf{g}}^{s,t}[k]+y_k)\right)^2\right]$$
$$=\mathrm{E}\left[\left(\sum_{k=1}^{w}\left(D_{\mathbf{f}}^{s,t}[k]D_{\mathbf{g}}^{s,t}[k]+x_kD_{\mathbf{g}}^{s,t}[k]+y_kD_{\mathbf{f}}^{s,t}[k]+x_ky_k\right)\right)^2\right].$$

Note that the right hand of above equation can be expanded as sum of square terms and cross terms. We observe that any cross term that contains $x_k$ or $y_k$, for $k\in[w]$, is a singular cross terms. It follows that

$$\mathrm{E}\left[\hat{I}(\mathbf{f}_{s,t},\mathbf{g}_{s,t})^2\right]=\mathrm{E}\left[\left(\sum_{k=1}^{w}D_{\mathbf{f}}^{s,t}[k]D_{\mathbf{g}}^{s,t}[k]\right)^2\right]+E\left[\sum_{k=1}^{w}x_k^2D_{\mathbf{g}}^{s,t}[k]^2\right]+E\left[\sum_{k=1}^{w}y_k^2D_{\mathbf{f}}^{s,t}[k]^2\right]+E\left[\sum_{k=1}^{w}x_k^2y_k^2\right]. \tag{9}$$

Using the fact that $\mathrm{E}[x_k^2]\le 8/p_{\mathbf{f}}^2\le\Delta_{\mathbf{f}}^2$ from Lemma A.4, and $\mathrm{E}[\sum_{k=1}^{w}D_{\mathbf{g}}^{s,t}[k]^2]=\|\mathbf{g}_{s,t}\|_2^2$ from Lemma A.6, and the fact that $x_k^2$ and $D_{\mathbf{g}}^{s,t}[k]^2$ are independent, we can bound the second term in (9) as follow:

$$\mathrm{E}\left[\sum_{k=1}^{w}x_k^2D_{\mathbf{g}}^{s,t}[k]^2\right]\le\Delta_{\mathbf{f}}^2\mathrm{E}\left[\sum_{k=1}^{w}D_{\mathbf{g}}^{s,t}[k]^2\right]\le\Delta_{\mathbf{f}}^2\|\mathbf{g}_{s,t}\|_2^2. \tag{10}$$

Similarly, we have the following inequality for the third term in (9):

$$\mathrm{E}\left[\sum_{k=1}^{w}y_k^2D_{\mathbf{f}}^{s,t}[k]^2\right]\le\Delta_{\mathbf{g}}^2\|\mathbf{f}_{s,t}\|_2^2. \tag{11}$$

Using $\mathrm{E}[x_k^2]\le 8/p_{\mathbf{f}}^2\le\Delta_{\mathbf{f}}^2$ and Using $\mathrm{E}[y_k^2]\le 8/p_{\mathbf{g}}^2\le\Delta_{\mathbf{g}}^2$ from Lemma A.4 and the fact that $x_k^2$ and $y_k^2$ are independent, we have

$$E\left[\sum_{k=1}^{w}x_k^2y_k^2\right]=\sum_{k=1}^{w}E\left[x_k^2y_k^2\right]\le\sum_{k=1}^{w}\Delta_{\mathbf{f}}^2\cdot\Delta_{\mathbf{g}}^2=w\Delta_{\mathbf{f}}^2\Delta_{\mathbf{g}}^2. \tag{12}$$

Finally, by Lemma A.6, it follows that

$$\mathrm{E}\left[(\sum_{k=1}^{w}D_{\mathbf{f}}^{s,t}[k]D_{\mathbf{g}}^{s,t}[k])^2\right]-\mathrm{E}\left[\sum_{k=1}^{w}D_{\mathbf{f}}^{s,t}[k]D_{\mathbf{g}}^{s,t}[k]^2\right]=\mathrm{Var}[\sum_{k=1}^{w}D_{\mathbf{f}}^{s,t}[k]D_{\mathbf{g}}^{s,t}[k])^2]\le\frac{1}{w}\|\mathbf{f}_t\|_2\|\mathbf{g}_t\|_2. \tag{13}$$

Plugging (10), (11), (12) and (13) into (8) follows that

$$\mathrm{Var}[\hat{I}(\mathbf{f}_t,\mathbf{g}_t)]\le\frac{1}{w}\|\mathbf{f}_t\|_2^2\|\mathbf{g}_t\|_2^2+\Delta_{\mathbf{f}}^2\|\mathbf{g}\|_2^2+\Delta_{\mathbf{g}}^2\|\mathbf{f}\|_2^2+w\Delta_{\mathbf{f}}^2\Delta_{\mathbf{g}}^2$$
$$\le\frac{1}{w}\left(\|\mathbf{f}_t\|_2^2+w\Delta_{\mathbf{f}}^2\right)\left(\|\mathbf{g}\|_2^2+w\Delta_{\mathbf{g}}^2\right).$$

Using Chebyshev's inequality, it follows that

$$\Pr\left[\left|\hat{I}(\mathbf{f}_t,\mathbf{g}_t)-I(\mathbf{f}_t,\mathbf{g}_t)\right|\ge 2\sqrt{\frac{1}{w}\left(\|\mathbf{f}_t\|_2^2+w\Delta_{\mathbf{f}}^2\right)\left(\|\mathbf{g}\|_2^2+w\Delta_{\mathbf{g}}^2\right)}\right]$$
$$\le\mathrm{Var}[\hat{I}(\mathbf{f}_t,\mathbf{g}_t)]\Big/4\left(\frac{1}{w}\left(\|\mathbf{f}_t\|_2^2+w\Delta_{\mathbf{f}}^2\right)\left(\|\mathbf{g}\|_2^2+w\Delta_{\mathbf{g}}^2\right)\right)\le 1/4.$$

By setting $w\ge 4/\varepsilon^2$, we have

$$\Pr\left[\left|\hat{I}(\mathbf{f}_{s,t},\mathbf{g}_{s,t})-I(\mathbf{f}_{s,t},\mathbf{g}_{s,t})\right|\le\mathcal{E}\right]\le 1/4,$$

where $\mathcal{E}=\varepsilon\sqrt{\left(\|\mathbf{f}_t\|_2^2+(\frac{\Delta_{\mathbf{f}}}{\varepsilon})^2\right)\left(\|\mathbf{g}\|_2^2+(\frac{\Delta_{\mathbf{g}}}{\varepsilon})^2\right)}$. Finally, recall that we have $d=O(\log 1/\delta)$ estimators $\hat{I}_j(\mathbf{f}_t,\mathbf{g}_t)$ for $j=1,\ldots,d$, and we take the median $\hat{I}(\mathbf{f}_t,\mathbf{g}_t)=\mathrm{median}_{j\in[d]}\{\hat{I}_j(\mathbf{f}_t,\mathbf{g}_t)\}$ as the final estimator. By standard Chernoff bound, we have

$$\Pr\left[\left|\hat{I}(\mathbf{f}_t,\mathbf{g}_t)-I(\mathbf{f}_t,\mathbf{g}_t)\right|\ge\mathcal{E}\right]\ge 1-\delta.$$

This complete the proof of Theorem 4.2. □

## A.6 Proof of Theorem 5.3

PROOF. By the proof of Theorem 3.3, the number of segment for a single row in epoch $[t_{i-1},t_i)$ is $O(\frac{t_i-t_{i-1}}{\Delta_{t_{i-1}}^2})$,thus the total size of the historical lists for this row is $O(\sum_{k=1}^{s}\frac{t_i-t_{i-1}}{\Delta_{t_{i-1}}^2})$. Since $\varepsilon\|\mathbf{f}_t\|_1=\Theta(\Delta_{t_{i-1}})$ for $t_{i-1}\le t<t_i$, for $k=1,\ldots,s$. It follows that the total size of the historical lists for this row is

$$O\left(\sum_{k=1}^{s}\frac{t_i-t_{i-1}}{\Delta_{t_{i-1}}^2}\right)=O\left(\sum_{k=1}^{s}\sum_{j=t_{i-1}}^{t_i-1}\frac{1}{\Delta_{t_{i-1}}^2}\right)=O\left(\sum_{k=1}^{m}\frac{1}{\varepsilon^2\|\mathbf{f}_t\|_1^2}\right).$$

In the random streaming model, we have $\|\mathbf{f}_t\|_1^2=t^2$, and thus the number of segments becomes $O\left(\sum_{k=1}^{m}\frac{1}{\varepsilon^2t^2}\right)=O(1/\varepsilon^2)$. Summing up all $d=O(\log\frac{1}{\delta})$ rows, and the Theorem follows. □

## A.7 Proof of Theorem 5.6

PROOF. By the proof of Theorem 4.1, the size of the historical lists for this row in epoch $[t_{i-1},t_i)$ is $O(\frac{t_i-t_{i-1}}{\Delta_{t_{i-1}}})$, thus the total size of the historical lists for this row is $O(\sum_{k=1}^{s}\frac{t_i-t_{i-1}}{\Delta_{t_{i-1}}})$. Since $\varepsilon\|\mathbf{f}_t\|_2=\Theta(\Delta_{t_{i-1}})$ for $t_{i-1}\le t<t_i$, for $k=1,\ldots,s$. It follows that the total size of the historical lists for this row is $O\left(\sum_{k=1}^{s}\sum_{j=t_{i-1}}^{t_i-1}\frac{1}{\varepsilon\|\mathbf{f}_t\|_2}\right)=O\left(\sum_{k=1}^{m}\frac{1}{\varepsilon\|\mathbf{f}_t\|_2}\right)$.

In the standard streaming model, we have $\|\mathbf{f}_t\|_2\ge\sqrt{\|\mathbf{f}_t\|_1}=\sqrt{t}$, thus the size of historical lists becomes $O\left(\sum_{k=1}^{m}\frac{1}{\varepsilon\sqrt{t}}\right)=O\left(\frac{\sqrt{m}}{\varepsilon}\right)$. Summing up all $d=O(\log\frac{1}{\delta})$ rows, and the Theorem follows. □