MicroscopeSketch: Accurate Sliding Estimation Using Adaptive Zooming

Zheng Zhong
Peking University
Beijing, China
zheng.zhong@pku.edu.cn

Jiale Chen
Peking University
Beijing, China
jiale_chen@pku.edu.cn

Shiqi Jiang
Peking University
Beijing, China
jiangshiqi@pku.edu.cn

Yutong Hu
Peking University
Beijing, China
huyutong@pku.edu.cn

Tong Yang
Peking University & Pengcheng Laboratory
Beijing, China
yangtongemail@gmail.com

Bin Cui
Peking University
Beijing, China
bin.cui@pku.edu.cn

Uhlig Steve
Queen Mary University of London
London United Kingdom
steve.uhlig@gmail.com

ABSTRACT

[1] Approximate estimation of data streams provides real-time information for various applications. Estimation based on a sliding window has attracted much interest recently. The state-of-the-art approaches cannot achieve high accuracy, generality in algorithms and models, with limited memory usage, all at once. We propose MicroscopeSketch and its key technique, adaptive zooming. The key idea of adaptive zooming is to automatically zoom in or zoom out, to get a better view of the window. We theoretically analyze the error bound, the unilateral error, and the unbiased rounding error of MicroscopeSketch. We apply MicroscopeSketch to three tasks: frequency estimation, finding top-k frequent items, and finding top-k heavy changes. Compared to existing works, our experimental results show that the error of MicroscopeSketch is 363 times smaller in estimating the frequency and 15.2 smaller in finding top-k items than the existing works. All the related source code is (anonymously) provided open-source on Github.

ACM Reference format:
Zheng Zhong, Jiale Chen, Shiqi Jiang, Yutong Hu, Tong Yang, Bin Cui, and Uhlig Steve. 2021. MicroscopeSketch: Accurate Sliding Estimation Using Adaptive Zooming. In *Proceedings of ACM Conference, Washington, DC, USA, July 2017 (Conference'17),* 14 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1]Zheng Zhong, Jiale Chen contribute equally to this paper, and they together with Shiqi Jiang, Yutong Hu complete this work under the guidance of the corresponding author: Tong Yang.

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

PVLDB Reference Format:
Zheng Zhong, Jiale Chen, Shiqi Jiang, Yutong Hu, Tong Yang, Bin Cui, and Uhlig Steve. MicroscopeSketch: Accurate Sliding Estimation Using Adaptive Zooming. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/MicroscopeSketch/MicroscopeSketch.

1 INTRODUCTION

1.1 Background and Motivation

Approximate estimation of data streams provides real-time information for various applications [1–23]. Typical approximate estimation tasks include frequency estimation, finding the top-k frequent items, and finding the top-k heavy changes. Frequency estimation refers to reporting the estimated frequency of any item. Finding the top-k frequent items refer to estimating the most frequent k items in the data stream. Finding top-k heavy changes refers to estimating the k items whose frequency changed most significantly across two adjacent time windows. For these tasks, the sketch, a probabilistic data structure [1–38], has been widely considered thanks to its small and controllable error as well as its compactness, making it small enough to fit inside CPU caches.

Time window models for sketches in data streams rely on two models: fixed window and sliding window. In the sliding window model, we only consider recent data, *e.g.*, data in the recent three minutes, while the fixed window model considers all historical data.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

Compared with fixed window model, sliding window model is time-aware, focusing on recent information and discarding outdated information. Therefore, the sliding window model is more popular in practical applications. We refer to the above three typical tasks in the sliding window model as **Sliding Estimation**.

1.2 Prior works

Approximate estimation in sliding windows has caught the attention of the research community recently. The state-of-the-art algorithm is the Sliding sketches [39]. Sliding sketches divide the sliding window into some sub-windows and use the method of hopping window to approximate the sliding window. The error caused by the approximation is called window error. The more portion of the latest sub-window the sliding window covers, the smaller window error the sliding sketches have. To minimize the window error, the Sliding sketches make d sketches work asynchronously, so there is always one sketch's sliding window covers a large portion of the latest sub-window, which has a small window error. The Sliding sketches design a scanning operation to achieve it. Sliding sketches achieve great accuracy in the tasks of estimating frequency and finding top-k frequent items. However, its design focuses on the time dimension (divide the sliding window and discarded outdated data asynchronously). In contrast, our MicroscopeSketch focuses on both the time and frequency dimension, which we will elaborate on later.

Our goal in this paper is to design a sketch for sliding window estimation that satisfies as much as possible the following four requirements: 1) small memory usage; 2) high accuracy with limited memory; 3) generality in applicability: applicable to any counting algorithm and multiple kinds of mining tasks. 4) generality in models: works for both time-based and count-based sliding window models. To the best of our knowledge, no existing work is able to meet all the above requirements.

1.3 Proposed Solution

We propose MicroscopeSketch (MicroSketch for short) for three typical estimation tasks in the sliding window model. MicroSketch has the following three properties: 1) Accuracy: its error is significantly smaller than the state-of-the-art. 2) Compactness: its memory usage is small enough to be held in CPU caches. 3) Generality: it can be applied to all sketches that use counters for estimation in time-based or count-based sliding window.

Our first technique is two-dimension quantization. The ideal approach for the sliding window model is to collect and store a continuous frequency curve, in which each point represents the number of items that arrived at each moment. However, storing a continuous curve requires infinite space, which is impossible to implement, in the same way as a picture cannot record every detail in a scene. MicroSketch first uses a *shutter counter* to record the exact frequency. Then, similar to *image resolution*, it quantizes the exact frequency curve in two dimensions: time and frequency. The exact frequency curve will therefore be quantized into many pixels to store the curve efficiently. Note that only the pixels that intersect the curve will be stored as a counter. Given an item, we set a fixed number of pixels (fixed memory) to record its frequency curve, just like images set a fixed number of pixels to store a scene.

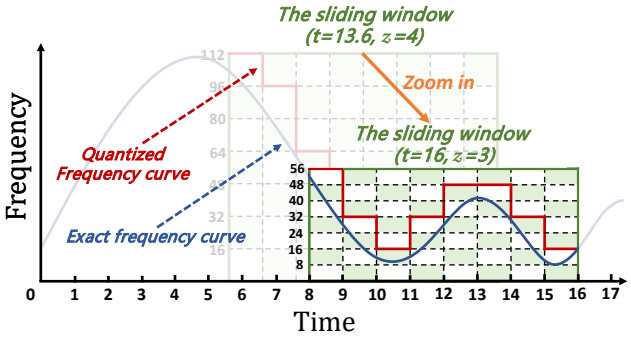


Figure 1: An example of MicroscopeSketch's zoom-in operation. (We will elaborate MicroscopeSketch in Section 3) We use 3-bit for each sub-window, which can represent the range from 0 to 7. At time 13.6, the max frequency in the sliding window is high, so the zooming counter Z is set to a high value 4 (each pixel represents $2^Z = 2^4 = 16$ items) to hold the frequency. At time 16, the max frequency in the sliding window drops below 56, so we zoom in on the frequency curve: decrement Z to 3 (each pixel represents $2^Z = 2^3 = 8$ items) to get a more accurate estimation.

The second and more important technique is called *adaptive zooming*. MicroSketch bears a resemblance to a *photographic microscope*, as we explain below. We take a picture at every $\frac{1}{T}$ sliding window to record the frequency curve, where T is the number of pictures we take in a sliding window. When we use a microscope to observe a curve, if the curve is small relative to the microscope's field of view, we will zoom into the curve to improve the resolution. Figure 1 provides an example of the zoom-in operation. If the curve is too large for the scope of the microscope's view, we will zoom out to get a better perspective of the entire curve and then take a picture of the entire curve. This way, we can accurately record the curve. From an information-theoretic perspective, each image has a fixed number of pixels. Therefore, when the image just covers the entire target, it covers the largest number of pixels. In this way, it has the target's maximum information. MicroSketch automatically traces the change in the frequency curve and zooms in or out over time to record it as accurately as possible.

1.4 Key contribution

1) We propose the MicroscopeSketch, which is accurate and generic. It works well in the tasks of estimating frequency, finding top-K items, and finding top-K heavy changes (MicroscopeSketch is the first work focus in this task in sliding window) in time-based or count-based sliding window.

2) We theoretically prove that MicroscopeSketch can achieve overestimation, underestimation, or unbiased rounding error by using different querying approaches. We also prove that the estimation made by MicroscopeSketch is unbiased and analyze the error bound under the assumption of stable frequency at the edge of the sliding window.

3) We conduct extensive experiments to show MicroscopeSketch's performance. Compared to the existing works, its error achieves at most 363 times smaller in estimating the frequency and 15.2 smaller in finding top-k items than the existing works.

All the related source code is (anonymously) provided open-source on Github [2].

2 BACKGROUND AND RELATED WORK

This paper focuses on three kinds of measurement tasks: frequency estimation, finding top-k frequent items, and finding top-k heavy changes. Below, we describe the sliding window models, the measurement tasks, and existing works. Regarding existing works, we focus on sketches applied to the sliding window model. For algorithms in fixed windows, we refer the reader to [1, 2, 4, 5, 7–11, 40–42].

2.1 Preliminaries

Frequency estimation is a fundamental building block for sliding window counting tasks. Two main sliding window models exist: *count-based sliding window model* and *time-based sliding window model*. The difference is how the size of a window is defined, either using the number of items or the time length.

PROBLEM 1. ***(Count-based Frequency Estimation)**. Given an item e in a data stream, count the number of appearances of e in the recent W items.*

PROBLEM 2. ***(Time-based Frequency Estimation)**. Given an item e in a data stream, at any time t, count the number of appearances of e from time $t - W$ to t.*

The definition of finding top-k frequent items and finding top-k heavy changes in the count-based and time-based sliding window are similar.

2.2 Frequency Estimation

Frequency estimation refers to estimating the frequency of arbitrary items. Prior works are Sliding sketches [39] and ECM [43]. Sliding sketches is a framework, which can apply to many algorithms. To estimating frequency for arbitrary item, Sliding sketches apply CM sketch [1] and CU [2] sketch. Similar to our time quantization, the Sliding sketches divide the sliding window into some sub-windows, and use the method of hopping window to approximate the sliding window. The error caused by the approximation is call window error. The more portion of the latest sub-window the sliding window cover, the smaller window error the sliding sketches have. In contrast, the error caused by hash collisions in the sketch is call sketch error. And a sketch's total error is the sum of its window error and sketch error. Normally, to minimize the error, sketch algorithms will build d sketches, and reported the estimation of the sketches with minimal error. However, if the d sketches work synchronously, their window error will be similar, so the "report the estimation with minimal error" method can only minimize sketch error. To minimize the window error, the Sliding sketches make d sketches work asynchronously, so there is always one sketch's sliding window cover a large portion of the latest sub-window, which has a small window error. The Sliding sketches design a scanning operation to achieve it. However, it just focuses on the design of the time dimension, while the MicroscopeSketch considers two dimensions: time and frequency.

[2] https://github.com/MicroscopeSketch/MicroscopeSketch

ECM sketch [43] is based on the CM sketch [1], and it uses Exponential Histograms (EH) to replace each counter in the CM sketch. Exponential Histograms (EH) [44] is a classic algorithm for counting in sliding windows, which can be used to count the frequency of a specified item in this task. EH divides a sliding window into many smaller sub-windows, and uses buckets to record the frequencies in these sub-windows. Specifically, the bucket of EH records two information, $<cnt, ts>$, where cnt means frequency count and ts is the timestamp of the last arrived item in this bucket. EH is initialized with no buckets. After recording for a while, it will accumulate several buckets. When a target item arrives, it appends a new bucket $<1, t_{now}>$. Then, it may merge some buckets by some rules. To merge $<cnt_1, ts_1>$ and $<cnt_2, ts_2>$, the new bucket is $<cnt_1 + cnt_2, \min ts_1, ts_2>$. When a bucket's ts is out of the current sliding window, the bucket is dropped. However, because EH records too many timestamps, when the memory is limited, ECM's accuracy is poor.

2.3 Finding Top-k Frequent Items

Finding top-k frequent items refers to finding k items with the highest frequency. Prior works are sliding sketches [39] and WCSS [45]. Sliding sketches apply HeavyKeeper [46] and use the asynchronous method mentioned above to achieve great accuracy in the task of finding top-K in the sliding window. WCSS, which is based on Compact Space-Saving (CSS) structure, records each item and its frequency during the recent window. Supposing the sliding window size is W, WCSS divides the window into k sub-windows, each of which is of size $(\frac{W}{k})$. When an item is updated in CSS, and its frequency count reaches a multiple of the sub-window size, it is said that the item *overflows*. WCSS employs a queue to record the overflows in each block, and maintain $(k + 1)$ queues for the recent $(k + 1)$ blocks. To estimate the frequency of an item, WCSS count the number of overflows of that item in the recent window, denote it as d, and query the frequency count, y, in CSS. Then, WCSS multiply d by the block size $(\frac{W}{k})$, and add the remainder in CSS, *i.e.*, $(d \times \frac{W}{k} + y\%\frac{W}{k})$. To guarantee no under-estimation occurs in WCSS, two more block size is increased. Therefore, the estimated frequency is $((d + 2) \times \frac{W}{k} + y\%\frac{W}{k})$. However, when the memory is insufficient, the accuracy of WCSS is poor.

2.4 Finding Top-k Heavy Changes

Finding top-k heavy changes refers to finding k items whose frequency changes most across successive windows. Suppose there is an item e and two adjacent sliding windows w_1 and w_2. The frequency of e in w_1 and w_2 are f_1, f_2, respectively. The top-k heavy changes refer to the k items such that $|f_1 - f_2|$ is the highest. To the best of our knowledge, no existing work can be directly used to find the top-k heavy changes (or finding heavy changes) in the sliding window model.

3 THE MICROSCOPESKETCH

In this section, we present the details of our MicroscopeSketch algorithm, including the data structure, insertion, query, and deletion operations. The symbols frequently used in this paper are shown in Table 1.

Table 1: Symbols used in this paper.

Notation	Meaning
P	Array of pixel counter, recording a of $a \times c^b$
Z	Zooming counter, recording b of $a \times c^b$
S	Shutter counter
c	A parameter of $a \times c^b$
l	Number of bits used for each pixel counter
T	Number of sub-windows per sliding window
n	Current sub-window
f	True frequency in the recent sliding window
$\hat{f}$	Estimated frequency given by MicroscopeSketch

3.1 Data Structure

We divide a sliding window into T sub-windows, and propose MicroscopeSketch (MicroSketch for short) to record the frequency in the recent $T+1$ sub-windows. MicroSketch consists of three parts: $T+2$ pixel counters $P[0], P[1], \cdots, P[T+1]$, a zooming counter Z, and a shutter counter S. We approximate the exact frequency by $a \times c^b$ (*e.g.*, $c = 2$). For each sub-window, we use a pixel counter to record the value of a, and all recent $T+1$ sub-windows share the same value of b, recorded by the zooming counter Z. To minimize the error caused by quantization, we use the shutter counter to record the sum of 1) the frequency of the current sub-window and 2) the remaining frequency of the previous sub-window. [3] Each pixel counter has l bits ($l < 32$), ranging from 0 to $2^l - 1$. The zooming counter has 5 bits, and the shutter counter has 32 bits. All counters are initialized to 0.

Now we explain two things: (1) Why we need $T+2$ pixel counters instead of T pixel counters? (2) Why we use $a \times c^b$ to approximate the frequency?

For the first question, as shown in Figure 2, the sliding window may span $T+1$ sub-windows. Therefore, $T+1$ sub-windows' pixel counters might be used simultaneously. We also need one more pixel counter, the *zero counter* (always 0), to safely clean the outdated information from the sliding window. For the second question, using $a \times c^b$ to approximate the frequency is easy and efficient. Thanks to its neat structure, we can share b for many a to save memory, and implement our zoom-in and zoom-out operation to raise accuracy in this approximation method.

3.2 Operations

Insertion: Let e be the incoming item, and n be the current sub-window. First, we locate the $n \bmod (T+2)^{th}$ pixel counter $P[n \bmod (T+2)]$. We call it P_{cur} for convenience. We locate the *zero counter* $P[(n+1) \bmod (T+2)]$. Second, we check whether the *zero counter* is 0 and clear it otherwise. Third, we increment the shutter counter by 1, then do the carry-in operation: if the shutter counter reaches the threshold c^Z, we increment P_{cur} by 1 and clear the shutter counter to 0. After incrementing, if P_{cur} reaches its maximum value, *i.e.*, 2^l, we do the **zoom-out operation**: increasing the zooming counter Z by 1 and dividing all pixel counters $P[0], P[1], \cdots, P[T+1]$ by c, to adapt to the increasing frequency. When we divide a number that is not the multiples of c in a pixel counter, a rounding error will occur. We discuss this in Section 3.3. The pseudo-code is shown in Alg. 10.

Note that we normally do not clean the shutter counter at the end of each sub-window (expect the applications of the CU sketch [2], we will elaborate it in Section 4.1). Instead, we let the next sub-window inherit the remaining frequency. In this way, the remaining frequency will incur an underestimation error for this sub-window and an overestimation error for the next sub-window. Therefore, when we sum up the recent $T+1$ sub-windows to calculate the estimated frequency in the sliding window, these errors will almost be canceled out.

Algorithm 1: Insertion-MicroscopeSketch

```
Input: an item e; n, the current sub-window;
Locate P_cur for the current sub-window;
Clear expired statistics: P[(n + 1) mod (T + 2)] ← 0;
Increase the shutter counter: S ← S + 1;
if S = c^Z then
    S ← 0;
    P_cur ← P_cur + 1;
    if P_cur = 2^l then
        Z ← Z + 1;
        for i from 0 to (T + 1) do
            P[i] ← P[i]/c;
```

Zoom-in operation: In Insertion, we described how to spontaneously do the zoom-out operation when the frequency reaches the maximum value. Now we describe how to do the **zoom-in operation** when the frequency becomes small to achieve a more accurate estimation. At the end of every sub-window, we check whether all pixel counters $P[0], P[1], \cdots, P[T+1]$ are smaller than $\frac{2^Z}{c}$. If so, we multiple all the pixel counters by c and decrement the zooming counter by 1.

Query: We first compute the frequency of each sub-window. We use $\hat{f}_i$ to denote the frequency of the i^{th} sub-window. The detailed formula to compute $\hat{f}_i$ is shown below.

$$\hat{f}_i = P[i \bmod (T+2)] \times c^Z \tag{1}$$

The estimated frequency for a sliding window using MicroSketch is the sum of three parts: the value of the *shutter counter* S, the sum of the frequency of the recent T sub-windows, and an estimated frequency of the oldest sub-window in the sliding window (*i.e.*, the recent $(T+1)^{th}$ sub-window), which is denoted by Δf. The formula is shown in Eq. 2.

$$\hat{f} = S + \sum_{i=n-T+1}^{n} \hat{f}_i + \Delta f \tag{2}$$

There are three methods to compute Δf: linear approximation, overestimation, and underestimation. 1) The *linear approximation* is the most accurate one according to our experimental results. Let t be the current time and w be the size of a sub-window. We compute the proportion $p = 1 - \frac{t \bmod w}{w} \in [0, 1]$. p is the proportion of

[3] We will elaborate in Section 3.2

the oldest sub-window included in the sliding window. Then, we set $\Delta f = \hat{f}_{n-T} \times p$ to estimate the frequency of that part. 2) The overestimation method sets $\Delta f = \hat{f}_{n-T}$. 3) The underestimation method sets $\Delta f = -\hat{f}_{i_0}$, where i_0 be the minimum $i \geqslant 1$ such that $\hat{f}_{n-T+i} \neq 0$.

Deletion: Similarly to Insertion, we first locate P_{cur}. Then, there are three cases: 1) If the shutter counter is not 0, we decrement the shutter counter by 1. 2) If the shutter counter is 0, but P_{cur} is not 0, we decrement P_{cur} by 1 and set shutter counter to $2^c - 1$. 3) If both the shutter counter and P_{cur} are 0, we find the first non-zero pixel counter before P_{cur}, which is denoted by P_*. Then we decrement P_* by 1 and set the shutter counter to $2^c - 1$.

Analysis: The error of MicroSketch comes from three parts: 1) the fixed window algorithm to which MicroSketch is applied; 2) the estimation of Δf; 3) the *rounding error*[4] of pixel counters. For the first part, because MicroSketch can be applied to many algorithms, we can choose the best existing fixed window algorithm to minimize it. For the second part, we can increase T (the number of sub-windows in a sliding window) to decrease it. For the third part, we can give more space to each pixel counter (*i.e.*, increase l) to decrease it. Increasing T or l is a trade-off between memory usage and accuracy.

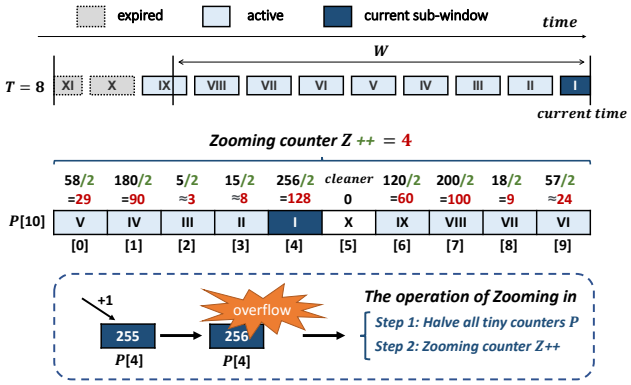


Figure 2: An example of MicroscopeSketch ($c = 2$). $I \sim XI$ are ten sub-windows. I is the current sub-window. We use 8 bits for each pixel counter, so their maximum value is 255. After insertion, the value of $P[4]$ is going to be 256, which will cause an overflow. Therefore, we increment the zooming counter by 1 and halve all pixel counters. If we query at this time, the frequency is $(29 + 90 + 3 + 8 + 128 + 0 + 60 + 100 + 9 + 24) \times 2^4 = 7216$.

3.3 Optimizations: Unbiased Rounding

In Section 3.2, we mentioned that when we halve a pixel counter in the zoom-out operation, the rounding error will occur when the counter value is not the multiples of c. The reason is that after the division of c, the counter value cannot be represented by an integer. We have two straightforward options: always round up the pixel counter, bringing an overestimation rounding error, and always round down the pixel counter, bringing an underestimation rounding error. To minimize the error, we propose the unbiased method: When we divide a pixel counter $P[i]$ by c, if $P[i] \ mod \ c \neq 0$, we round it up with the probability of $\frac{P[i] \ mod \ c}{c}$. Otherwise, we round it down. In this way, we obtain an unbiased rounding error, which is more accurate than always rounding up or always rounding down.

[4]The difference between the real value and the value after rounding.

4 APPLYING MICROSCOPESKETCH TO THREE TASKS

4.1 Frequency Estimation

4.1.1 *MicroscopeSketch-CM sketch.*

To estimate the frequency of items in a data stream, the Count-Min sketch [1] (CM sketch) is a well-known and effective data structure, thanks to its modest memory usage and high accuracy. The data structure of a CM sketch consists of d (*e.g.*, $d = 3$) counter arrays $D_1, D_2, \cdots, D_d$, and d hash functions $h_1(.), h_2(.), \cdots, h_d$. The size of each counter array is m. Each item e is mapped to d counters, which are $D_i[h_i(e) \bmod w]$ $(1 \leqslant i \leqslant d)$, where $D_i[j]$ refers to the j^{th} counter in array D_i. When inserting an item e, the d mapped counters of e are incremented by 1. When querying the frequency of e, the minimal value of the d mapped counters is reported as the estimated frequency. The reason of reporting the minimal value is, because the CM sketch always overestimates items, the minimal value has the smallest error.

The CM sketch naturally supports frequency estimation in fixed windows. To adapt it to sliding windows, we use a MicroSketch to replace each counter of the CM sketch. We call this adapted CM sketch **MicroSketch-CM**. The operations of the MicroSketch-CM are similar to the CM sketch. Specifically, each item has d mapped MicroSketches.

Insertion: When inserting an item, we first use the insertion operation of the CM. When incrementing a mapped MicroSketch, we use the insertion operation of the MicroSketch.

Query: When querying the frequency of an item, we use the query operation of the CM sketch to get d mapped MicroSketches. We report the frequency of the smallest MicroSketch's estimation by using the query operation of the MicroSketch. MicroSketch-CM can achieve the only overestimation by using the overestimation query of MicroSketch.

4.1.2 *MicroscopeSketch-CU sketch.*

The CU sketch [2] slightly modifies the CM sketch's insertion operation and can reduce its error significantly. When inserting item e, the CU sketch increments the smallest counter among the d mapped counters of e by 1. To adapt it to sliding windows, we use a MicroSketch to replace each counter of the CU sketch. We call this adapted CU sketch **MicroSketch-CU**. The operations of the MicroSketch-CU are similar to the MicroSketch-CM. There are three modifications, though: 1) When inserting an item e, we increment the MicroSketch with the smallest frequency in the current sub-window among the d mapped ones. The frequency in the current counter is $S + P_{cur} \times 2^Z$; 2) We clear the shutter counter S at the end of each sub-window and increment P_{cur} by 1 with probability $\frac{S}{c^Z}$; 3) When querying an item e, we report the sum of three parts: the smallest shutter counter S among the d mapped ones, the sum of $\underline{P_i}$ in the recent T sub-windows, and the smallest Δf among the d mapped ones, where $\underline{P_i}$ is the smallest pixel counter in the i^{th} sub-window among the d MicroSketches. MicroSketch-CU can

achieve the only overestimation by using the overestimation query of MicroSketch.

4.2 Finding Top-k Frequent Items

4.2.1 MicroscopeSketch-HeavyGuardian.

Among existing works for finding the top-k items in a data stream, HeavyGuardian [9] achieves very high accuracy with small memory usage. The key idea of HeavyGuardian is to split the top-k items away from the others. To achieve this, it randomly divides items into different buckets through hashing. In each bucket, it uses θ (*e.g.*, $\theta = 8$) key-value (KV) pairs **<ID, frequency>** to maintain the statistics of the θ most frequent items. Among the θ recorded frequent items, the most frequent one is called the *king*, while the others are called the *guardians*. For each new item, the weakest guardian (the guardian with the lowest frequency) might be decayed with a probability, called *Exponential Decay*. When a guardian cell becomes 0, the new item will be recorded there.

We adapt HeavyGuardian using MicroSketch to find the top-k frequent items in sliding windows. We use a MicroSketch to replace each value of the KV pairs in HeavyGuardian. We call the new data structure **MicroSketch-HG**.

Insertion: When inserting an item e, we first use a hash function $h(e)$ to map e to a bucket. Now assume that e is mapped to the i^{th} bucket. Then, there are three cases.

- *1:* If e already exists in the bucket, its frequency will be increased by using the *Insertion* operation of MicroSketch.
- *2:* If e is not recorded in the bucket, but the bucket has an empty cell, e will be inserted into the empty cell with a frequency of 1.
- *3:* If e is not recorded in the bucket and the bucket is full, we will decay the weakest guardian with a probability. Assuming that the frequency of the weakest MicroSketch guardian is f_i^*, we use the deletion operation of MicroSketch to decrease f_i^* by 1 with probability $\beta^{-f_i^*}$ where β is a constant (*e.g.*, $\beta = 1.08$). After the decay, if f_i^* becomes 0, we insert e into the weakest MicroSketch guardian.

Top-k Report: To get the top-k frequent items from MicroSketch-HG, we traverse all KV pairs and report the k KV pairs with the highest frequency. MicroSketch-HG can achieve the only underestimation by using the underestimation query of MicroSketch.

4.2.2 MicroscopeSketch-SpaceSaving.

Besides exponential decay, another choice is to use SpaceSaving to evict the weakest KV pair from a bucket. The main idea of SpaceSaving is to always use the incoming item to replace the weakest guardian and increment it by 1. We only need to modify the third case above of MicroSketch-HG as follows: *If e is not recorded in the bucket and the bucket is full, we increase the frequency of the weakest guardian by 1. Then, we replace the ID of the weakest guardian by e.* We call the new data structure **MicroSketch-SS**. MicroSketch-SS can achieve the only overestimation by using the overestimation query of MicroSketch.

4.3 Finding Top-k Heavy Changes

To find heavy changes in sliding windows, we still use the data structure and operations of **MicroSketch-HG**. However, we need a slight modification to the MicroSketch data structure. We extend the number of pixel counters to $2T + 2$, where T is the number of sub-windows of the sliding window. Assume that the current time is t, we compute the frequency of e in the sliding window ranging from time $t - 2 \cdot W$ to $t - W$, which is denoted by $\hat{f}_1$, and the frequency from time $t - W$ to t, which is denoted by $\hat{f}_2$, using the query operation of MicroSketch.

Top-k Report: To get the top-k heavy changes from MicroSketch-HG, we traverse all KV pairs and report the k KV pairs whose $|\hat{f}_1 - \hat{f}_2|$ is highest.

5 MICROSCOPESKETCH ERROR ANALYSIS

In this section, we analyze the unilateral error properties of MicroSketch. By using different querying strategies, MicroSketch can achieve either overestimation or underestimation. Below, we show how MicroSketch achieves it. We also prove that MicroSketch can achieve unbiased rounding error and show its variance. In addition, under the assumption of stable frequency at the edge of the sliding window, we prove that the estimation by MicroSketch is unbiased and show the error bound. The symbols frequently used in this section are shown in Table 2.

Table 2: Symbols used in this section.

Notation	Meaning
$\hat{f}_i$	Estimated frequency given by the i^{th} sub-window
f^*	Exact result recorded if there was no rounding
d	Number of MicroSketches arrays in MicroSketch-CM
m	Size of each MicroSketches array
$f(t_1, t_2)$	Frequency during period $(t_1, t_2]$
$f^{(i)}$	True frequency of item e_i in the recent sliding window
$\hat{f}^{(i)}$	Estimated frequency of item e_i by our MicroSketch-CM
$\hat{f}^{(i)}_{CM}$	Frequency of item e_i in the recent sliding window by a corresponding CM sketch

5.1 Unilateral Error (Overestimation)

To achieve overestimation only, we round up the pixel counters when zooming out. We apply the overestimation method in Eq. 2 and use the following formula for querying:

$$\hat{f} = S + \sum_{i=n-T}^{n} \hat{f}_i \tag{3}$$

Theorem 5.1. *If we use Eq. 3 for querying, $\hat{f} \geqslant f$.*

Proof. From Eq. 3, we use S and the statistics in the latest $T + 1$ sub-windows for approximation. For the latest T sub-windows, items falling in this period must be contained in the pixel counters. For the latest $(T+1)^{th}$ sub-window, the frequency must be overestimated because S could be over 0 when the sub-window starts. Also, because we round up the pixel counters when zooming out, the frequency represented by the pixel counters could be overestimated. Therefore, the reported frequency must be no less than the true frequency. □

5.2 Unilateral Error (Underestimation)

To achieve underestimation only, the query operation of MicroSketch must be changed. When we zoom out, we round them down. We apply the underestimation method from Eq. 2 and use the following formula for querying.

$$\hat{f} = S + \sum_{i=n-T+1}^{n} \hat{f_i} - \hat{f_{i_0}} \tag{4}$$

Here i_0 is the minimum $i \geqslant 1$ such that $\hat{f}_{n-T+i} \neq 0$.

Theorem 5.2. *If we use Eq. 4 for querying, $\hat{f} \leqslant f$.*

Proof. During insertion, instead of directly increasing pixel counters, we increase the standard counter S first. As a result, when starting a new sub-window, S leads to overestimation because it may not be 0. Let i_0 be the minimum $i \geqslant 1$ such that $\hat{f}_{n-T+i} \neq 0$. If the overestimation caused by S occurs, it must be recorded in $P\,[(n-T+i_0) \bmod (T+2)]$. Also, because we round down the pixel counters when zooming out, the frequency recorded in the pixel counters can only be lower than the true frequency during the corresponding period.

We'll get underestimation if we eliminate the overestimated part, by removing $\hat{f_{i_0}}$ from the estimation $S+\sum_{i=n-T+1}^{n} \hat{f_i}$. Therefore, $\hat{f} \leqslant f$, *i.e.*, our MicroSketch achieves unilateral error (underestimation) by Eq. 4. □

5.3 Unbiased rounding error

Here unbiased rounding error implies that rounding does not produce bias, *i.e.*, the frequency is given by MicroSketch is an unbiased estimate of the result if we do not do the rounding when zooming out. To achieve unbiased rounding error, we apply the optimization of unbiased rounding from Section 3.3. When zooming out , we round it up with a probability of $P \bmod c$; otherwise we round up it.

Theorem 5.3. *Given an item e, let $\hat{f}$ be the estimated frequency given by our MicroSketch and f^* the exact result that should be recorded if there was no rounding. $E(\hat{f}) = f^*$, i.e., our MicroSketch has a property of unbiased rounding error.*

Proof. In a sub-window, the estimated frequency will change when we zoom out and do rounding. Let P be the number recorded in the pixel counter and Z the zooming counter before halving. We do rounding only when P is odd. The pixel counter after halving P' has a probability of $\frac{P \bmod c}{c}$ to be $\frac{P-P \bmod c}{c} + 1$ and a probability of $\frac{1-P \bmod c}{c}$ to be $\frac{P-P \bmod c}{c}$. At the same time, Z is increased by 1. The expectation is

$$\begin{aligned} E(P' \cdot c^{Z+1}) =& \frac{P \bmod c}{c} \cdot (\frac{P - P \bmod c}{c} + 1) \cdot c^{Z+1} \\ &+ \frac{1 - P \bmod c}{c} \cdot (\frac{P - P \bmod c}{c}) \cdot c^{Z+1} \\ =& P \cdot c^Z \end{aligned} \tag{5}$$

The expectation will not change after we halve the pixel counter. Therefore, our estimated frequency in each pixel counter is unbiased, *i.e.*, $E(\hat{f_i}) = f_i$, where f_i is the frequency without rounding during the i^{th} sub-window. Since $\hat{f}$ is a linear combination of the estimated frequency of pixel counters, we have $E(\hat{f}) = f^*$. In other words, our MicroSketch has a property of unbiased rounding error. □

In subsection 5.4 we will show that under the assumption that the increment of the frequency is stable at the edge of the sliding window, the error caused by the sliding window is also unbiased, and thus MicroSketch is unbiased.

When we adopt a data structure with MicroSketch, we actually do further estimation about each counter with a MicroSketch based on the original structure. For those structures where elements are inserted into fixed counters, when we use MicroSketches to replace each counter, since MicroSketch is unbiased, thus the bias will not increase further than the original structure. Particularly, if the estimate by the original data structure is unbiased, then the adapted structure also achieves unbiased results.

Below we show the variance of MicroSketch when we apply linear approximation from Eq. 2 and use unbiased rounding.

Theorem 5.4. *Let R_i be $f_i \bmod c^Z$. The variance of $\hat{f}$ satisfies*

$$Var(\hat{f}) = \sum_{i=n-T+1}^{n} R_i \cdot (c^Z - R_i) + p^2 \cdot R_{n-T} \cdot (c^Z - R_{n-T}) \tag{6}$$

Here Z^* is the number recorded in the zooming counter when querying.

Proof. Since randomness comes from rounding, f_i and Z^* are not random. Thus, R_i is also a fixed value. Let P_0 be $\frac{f_i - R_i}{c^Z}$. The number recorded in the pixel counter $P\,[i \bmod (T+2)]$ is either P_0 or P_0+1. Hence, $\hat{f_i} = P\,[i \bmod (T+2)] \times c^Z = f_i - R_i$ or $f_i - R_i + c^Z$. Since $E(\hat{f_i}) = f_i$, $\hat{f_i}$ is $f_i - R_i + c^Z$ with a probability $\frac{R_i}{c^Z}$, and $f_i - R_i$ with a probability $1 - \frac{R_i}{c^Z}$. The variance of $\hat{f_i}$ satisfies

$$\begin{aligned} Var(\hat{f_i}) = E(\hat{f_i} - E(\hat{f_i}))^2 &= \frac{R_i}{c^Z} \cdot (c^Z - R_i)^2 + (1 - \frac{R_i}{c^Z}) \cdot R_i^2 \\ &= R_i \cdot (c^Z - R_i) \end{aligned} \tag{7}$$

The rounding in all pixel counters is independent of each other. Therefore, we have

$$\begin{aligned} Var(\hat{f}) &= Var(S + \sum_{i=n-T+1}^{n} \hat{f_i} + p \cdot \hat{f}_{n-T}) \\ &= \sum_{i=n-T+1}^{n} Var(\hat{f_i}) + p^2 \cdot Var(\hat{f}_{n-T}) \\ &= \sum_{i=n-T+1}^{n} R_i \cdot (c^Z - R_i) + p^2 \cdot R_{n-T} \cdot (c^Z - R_{n-T}) \end{aligned} \tag{8}$$

□

5.4 Error Bound

Below, we introduce the error caused by the sliding window and do further analysis. For the current time t, we assume that the increment of the frequency is stable at the edge of the sliding window, *i.e.*, the frequency is proportional to the duration. Let v be the frequency in normalized time at the edge of the sliding window.

First, we prove that $\hat{f}$ is unbiased, *i.e.*, $E(\hat{f}) = f$. From previous analysis, $E(\hat{f} - f^*) = 0$. Let $f(t_1, t_2)$ be the frequency during period

$(t_1, t_2]$. f is the frequency in a window. We have

$$\begin{aligned} f &= f(t-W,t) \\ &= f(t-Tw, t-t \bmod w-(T-1)w) + f(t-t \bmod w-(T-1)w, t) \end{aligned} \tag{9}$$

f^* is the exact result that should be recorded if there was no rounding. We have

$$\begin{aligned} f^* =& f(t-t \bmod w-(T-1)w, t) \\ &+ p \cdot f(t-t \bmod w-Tw, t-t \bmod w-(T-1)w) \end{aligned} \tag{10}$$

Thus, we have

$$\begin{aligned} E(f^*-f) =& p \cdot f(t-t \bmod w-Tw, t-t \bmod w-(T-1)w) \\ &- f(t-Tw, t-t \bmod w-(T-1)w) \\ =& p \cdot wv - (w-t \bmod w)v \\ =& 0 \end{aligned} \tag{11}$$

Therefore, under our assumption, $E(\hat{f})-f = E(\hat{f}-f^*)-E(f^*-f) = 0$, *i.e.*, the estimated frequency by MicroSketch is unbiased. Similarly to the previous proof, the variance

$$Var(\hat{f}) = \sum_{i=n-T+1}^{n} R_i \cdot (c^Z - R_i) + p^2 \cdot R_{n-T} \cdot (c^Z - R_{n-T}) \tag{12}$$

According to Chebyshev inequality, we get the error bound of MicroSketch: for $\epsilon \geqslant 0$, we have

$$\begin{aligned} &Pr\left[\left|\hat{f}-f\right| \geqslant \epsilon\right] \leqslant \frac{1}{\epsilon^2} Var(\hat{f}) \\ =& \frac{1}{\epsilon^2}\left(\sum_{i=n-T+1}^{n} R_i \cdot (c^Z - R_i) + p^2 \cdot R_{n-T} \cdot (c^Z - R_{n-T})\right) \end{aligned} \tag{13}$$

Now we consider the error bound of MicroSketch-CM with d MicroSketches arrays and the size of each array is m. Let $e_1, e_2, \cdots, e_M$ be the items inserted into MicroSketch-CM and their true frequency from time $(t-W)$ to t is $f^{(1)}, f^{(2)}, \cdots, f^{(M)}$, and let $\boldsymbol{f} = (f^{(1)}, f^{(2)}, \cdots, f^{(M)})$. For an item e_i, let $\hat{f}^{(i)}$ be the estimated frequency by our MicroSketch-CM and $\hat{f}^{(i)}_{CM}$ be the frequency in the recent sliding window,*i.e.*, from time $(t-W)$ to t, by a corresponding CM sketch .

For the error bound of the estimated frequency of e_i, we can consider it as two parts:

$$\begin{aligned} Pr\left[\left|\hat{f}^{(i)}-f^{(i)}\right| \geqslant \epsilon\|\boldsymbol{f}\|_1\right] \leqslant& Pr\left[\left|\hat{f}^{(i)}_{CM}-f^{(i)}\right| \geqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1\right] \\ &+ Pr\left[\left|\hat{f}^{(i)}_{CM}-f^{(i)}\right| \leqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1, \left|\hat{f}^{(i)}-\hat{f}^{(i)}_{CM}\right| \geqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1\right] \end{aligned} \tag{14}$$

From the theorem about the error bound of the CM sketch[1], when $m \geqslant \frac{2e}{\epsilon}$ we have

$$Pr\left[\left|\hat{f}^{(i)}_{CM}-f^{(i)}\right| \geqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1\right] < e^{-d} \tag{15}$$

The estimation is given by the MicroSketch with the minimal value among the d mapped MicroSketches. In that MicroSketch, from equation 13 we can get that

$$\begin{aligned} &Pr\left[\left|\hat{f}^{(i)}_{CM}-f^{(i)}\right| \geqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1\right] \\ \leqslant& \frac{4}{\epsilon^2\|\boldsymbol{f}\|_1^2}\left(\sum_{i=n-T+1}^{n} R_i \cdot (c^Z - R_i) + p^2 \cdot R_{n-T} \cdot (c^Z - R_{n-T})\right) \\ \leqslant& \frac{4}{\epsilon^2\|\boldsymbol{f}\|_1^2}\left(T \cdot \left(\frac{c^Z}{2}\right)^2 + p^2 \cdot \left(\frac{c^Z}{2}\right)^2\right) \\ \leqslant& \frac{1}{\epsilon^2\|\boldsymbol{f}\|_1^2}(T+1) \cdot c^{2Z} \end{aligned} \tag{16}$$

When $\left|\hat{f}^{(i)}_{CM}-f^{(i)}\right| \leqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1$, we have $c^Z \cdot c^{l-1} \leqslant c^Z \cdot \max P \leqslant \hat{f}^{(i)}_{CM} \leqslant f^{(i)} + \frac{\epsilon}{2}\|\boldsymbol{f}\|_1$, thus we can get the bound of Z that $Z \leqslant \left\lfloor \ln\left(f^{(i)} + \frac{\epsilon}{2}\|\boldsymbol{f}\|_1\right)/\ln c \right\rfloor - l + 1$. Let $Z_0 = \left\lfloor \ln\left(f^{(i)} + \frac{\epsilon}{2}\|\boldsymbol{f}\|_1\right)/\ln c \right\rfloor - l + 1$. Then we have

$$\begin{aligned} &Pr\left[\left|\hat{f}^{(i)}_{CM}-f^{(i)}\right| \leqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1, \left|\hat{f}^{(i)}-\hat{f}^{(i)}_{CM}\right| \geqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1\right] \\ \leqslant& Pr\left[\left|\hat{f}^{(i)}-\hat{f}^{(i)}_{CM}\right| \geqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1 \middle| \left|\hat{f}^{(i)}_{CM}-f^{(i)}\right| \leqslant \frac{\epsilon}{2}\|\boldsymbol{f}\|_1\right] \\ \leqslant& \sup_{Z \leqslant Z_0} \frac{1}{\epsilon^2\|\boldsymbol{f}\|_1^2}(T+1) \cdot c^{2Z} \\ \leqslant& \frac{1}{\epsilon^2\|\boldsymbol{f}\|_1^2}(T+1) \cdot c^{2Z_0} \end{aligned} \tag{17}$$

Therefore, for any $\epsilon \geqslant 0$, when $m \geqslant \frac{2e}{\epsilon}$ we can get the error bound of our MicroSketch-CM

$$Pr\left[\left|\hat{f}^{(i)}-f^{(i)}\right| \geqslant \epsilon\|\boldsymbol{f}\|_1\right] \leqslant e^{-d} + \frac{1}{\epsilon^2\|\boldsymbol{f}\|_1^2}(T+1) \cdot c^{2Z_0} \tag{18}$$

Here $Z_0 = \left\lfloor \ln\left(f^{(i)} + \frac{\epsilon}{2}\|\boldsymbol{f}\|_1\right)/\ln c \right\rfloor - l + 1$.

6 EXPERIMENTAL RESULTS

In this section, we evaluate the performance of MicroscopeSketch (MicroSketch for short) for three tasks: frequency estimation, finding top-k frequent items, and finding top-k heavy changes.

Table 3: Abbreviations used in our experiments.

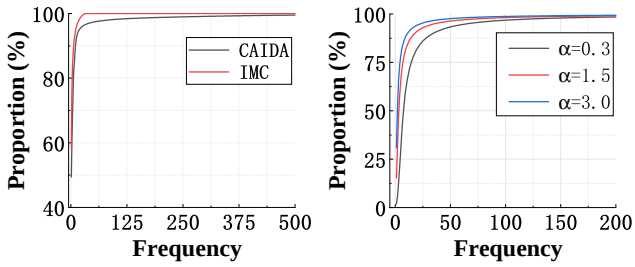
Abbreviation	Full Name
Ours	MicroscopeSketch (Microsketch)
Sl	Sliding sketches [39]
SS	SpaceSaving [7]
HG	HeavyGuardian [9]
HK	HeavyKeeper [46]

6.1 Experimental Setup

Server: Our experiments are performed on a computer with a 6-core CPU (12 threads, Intel(R) Core(TM) i7-8750H CPU @ 2.20GHz). Each core has three levels of caches: 384KB L1 cache, 1.5MB L2 cache, and 9MB L3 cache.

Datasets: We use three kinds of datasets in our experiments: the CAIDA anonymized Internet traces, the IMC data center traffic traces, and synthetic datasets that follow a Zipf distribution.

1) **CAIDA:** This is a set of IP packet streams, coming from the *CAIDA Anonymized Internet Trace 2018* [47]. Each item is a packet in the stream, distinguished by its 5-tuple (13 bytes), *i.e.*, the combination of source/destination IP addresses, source/destination ports and protocol type. Each trace contains about 27M items with around 1.3M total distinct items. The frequency distribution are shown in Figure 3(a).
2) **IMC:** The second dataset comes from one of the data centers studied in *Network Traffic Characteristics of Data Centers in the Wild* [48]. Similar to the CAIDA traces, each item ID is a 5-tuple. There are about 14M items in the IMC dataset, with around 3.3M total distinct items. The frequency distribution are shown in Figure 3(a).
3) **Zipf:** We also generate three synthetic datasets following the Zipf [49] distribution. The skewness (denoted by α) varies from 0.3 to 3.0. 4-byte IDs distinguish items in this dataset. There are about 32M items in this dataset, with around 1~10M total distinct items depending on the skewness. The frequency distribution are shown in Figure 3(b).



(a) CAIDA and IMC. (b) Zipf.

Figure 3: Distribution of studied datasets.

Algorithm Comparisons and Implementations: Experiments were carried out for three tasks. All queries in the experiments are made once every $\frac{1}{100}$ sliding window to test the average performance of different algorithms. For MicroSketch, there are four parameters: M, D, T, l. In each task, we fix $D = 8$, find the best T and l, and adjust M to fit different memory settings. We show the performance of the algorithms compared to ours below.

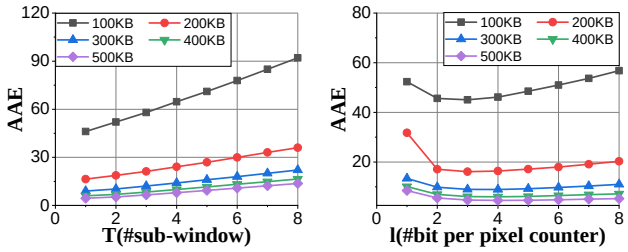
- Frequency Estimation: We choose the state-of-the-art, Sliding sketches [39] and ECM [43], for comparison. Sliding sketches is a framework that can be used on different algorithms. We combine Sliding sketches with CM and CU (Sl-CM, Sl-CU) for comparison. According to the results of the ECM original paper, to achieve high accuracy, the parameter u in EH [44] used in ECM should be set to 2. We set the number of hash functions to 3 in every algorithm.
- Finding Top-k Frequent Items: We choose the state-of-the-art algorithms, Sliding sketches (Sl-HK) and WCSS [45], for comparison.
- Finding Top-k Heavy Changes: We test the performance of MicroSketch-HG under different memory sizes. In this case, we cannot compare our algorithm to the state-of-the-art, because no existing work so far is able to find heavy changes in sliding windows. Therefore, we only provide the comparison between MicroSketch-HG and the straw man solution, which we will elaborate in Section 6.4.

Metrics: AAE (Average Absolute Error), ARE (Average Relative Error), RR (Recall Rate), PR (Precision Rate) and Speed (the Million operations per second or Mops).

- **AAE** (Average Absolute Error): $\frac{1}{n}\sum_{i=1}^{n}|\hat{f}_i - f_i|$, where n is the number of relevant items, f_i represents the real frequency, and $\hat{f}_i$ represents the estimated frequency.
- **ARE** (Average Relative Error): $\frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{f}_i - f_i|}{f_i}$, where n is the number of relevant items, f_i represents the real frequency, and $\hat{f}_i$ represents the estimated frequency.
- **RR** (Recall Rate): $\frac{|\Omega \cap \Psi|}{|\Psi|}$, where Ω is the set of reported top-k items of frequent items (or heavy changes), and Ψ is the set of real top-k frequent items (or heavy changes). RR represents the ratio of the number of correctly reported top-k frequent items (or reported top-k heavy change items) to k.
- **PR** (Precision Rate): $\frac{|\Omega \cap \Psi|}{|\Omega|}$, where Ω and Ψ represent the same as in RR. PR represents the ratio of the number of correctly reported top-k frequent items (or heavy change items) to the reported set size. For finding top-k frequent items and top-k heavy changes, PR=RR. Therefore, we just use RR as our metrics in the experiments.
- **Speed:** We test the Million operations (insertions) per second (Mops) of different algorithms.

6.2 Performance on Frequency Estimation

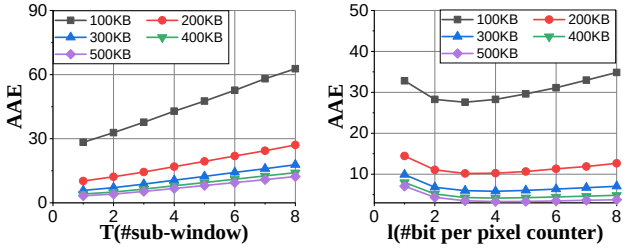
We compare five algorithms: MicroSketch-CM, MicroSketch-CU, Sl-CM, Sl-CU, and ECM, in the count-based model. The sliding window size is set to 1M packets. The CAIDA dataset is used to compare the algorithms' AAE and speed under five different memory sizes. The reason for using AAE in the task of estimating frequency is that, for infrequent items, e.g., a frequency of 5, an error of 500 leads to an RE (relative error) of 100, which will significantly increase the ARE (average relative error). However, for frequent items, e.g., of frequency 5000, an error of 500 only leads to an RE of 0.1. This makes the infrequent items have to be high in ARE. In practice, frequent items are more important. Therefore, compared to ARE, AAE is better at describing the algorithms' accuracy in frequency estimation. We also show the robust performance of MicroSketch-CM and MicroSketch-CU on different datasets.



(a) AAE *v.s.* T (b) AAE *v.s.* l

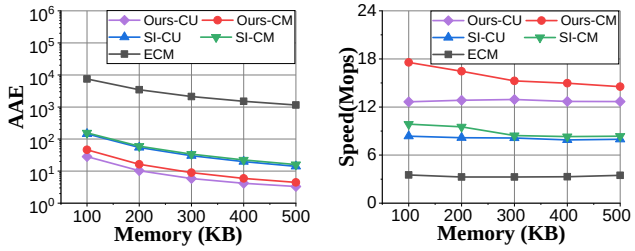
Figure 4: Effect of T and l on MicroSketch-CM.

First, we will set the parameters of MicroSketch-CM and MicroSketch-CU. In the figures, we use #sub-window to denote the number of

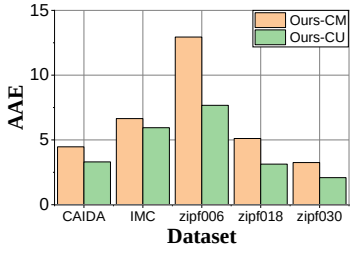


(a) AAE *v.s.* T (b) AAE *v.s.* l

Figure 5: Effect of T and l on MicroSketch-CU.



(a) AAE *v.s.* Algorithms (b) Speed *v.s.* Algorithms



(c) AAE *v.s.* Datasets

Figure 6: Performance of algorithms on the frequency estimation task.

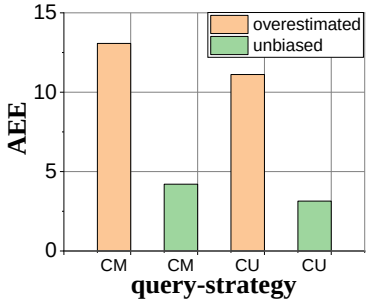
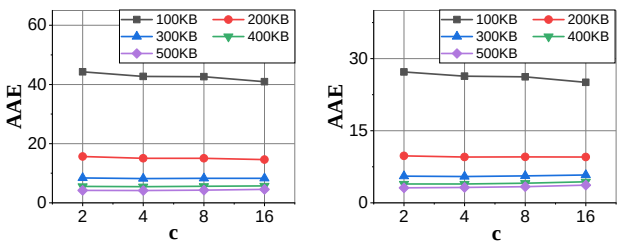


Figure 7: Effect of different query strategies on the frequency estimation task.

sub-windows within the sliding window, i.e., the parameter T. Similarly, we use #bit per pixel counter to denote the number of bits used for each pixel counter, i.e., the parameter l. As shown in Figure 4 and 5, the AAEs both increase as T grows. So we set $T = 1$ for both algorithms. As for l, there is not an optimal one, but $l = 4$ performs well on all memory settings. So for both algorithms, we set $l = 4$.

MicroSketch-CM, MicroSketch-CU *v.s.* Sl-CM, Sl-CU, ECM (Figure 6). As shown in Figure 6, MicroSketch-CM and MicroSketch-CU achieve both lower AAE and faster speed than Sl-CU, Sl-CM, and



(a) MicroSketch-CM (b) MicroSketch-CU

Figure 8: Effect of c on the frequency estimation task.

ECM. When the memory goes up to 500KB, the AAE of MicroSketch-CU and MicroSketch-CM are 3.3 and 4.5, while Sl-CU's and Sl-CM's are about 14 and 16, and ECM's is above 1150. This supports the much better performance of MicroSketch on frequency estimation. MicroSketch achieves about 200-300 times smaller error rate and about 4 times faster speed compared to ECM. Compared to Sliding sketches, MicroSketch achieves 4 times smaller error rate and 1-2 times faster speed.

MicroSketch-CM, MicroSketch-CU *v.s.* Dataset (Figure 6(c)). We find that the AAE of MicroSketch-CM is lower than 13, and MicroSketch-CU's AAE is lower than 8 on all datasets. For frequent items, which usually have a frequency higher than a thousand in a sliding window, the AAE of 13 or 8 is relatively small and has a limited impact. Therefore, our results support the good performance of MicroSketch in the frequency estimation task.

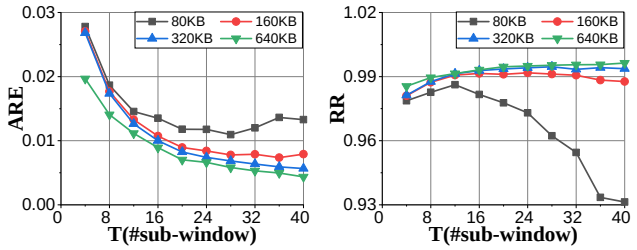
MicroSketch-CM, MicroSketch-CU *v.s.* Query Strategy (Figure 7). Since originally, CM sketch and CU sketch are overestimated, we only compare the overestimated and unbiased version of MicroSketch-CM, MicroSketch-CU. We find that the AAE of the overestimated version of MicroSketch-CM and MicroSketch-CU is about 13 and 11, respectively, while the unbiased version's AAE is about 4 and 3.

MicroSketch-CM, MicroSketch-CU *v.s.* c (Figure 8). We change c from 2 to 16 and find that it has little impact on the algorithm performce, since the line is almost horizontal, especially when the memory is relatively large.

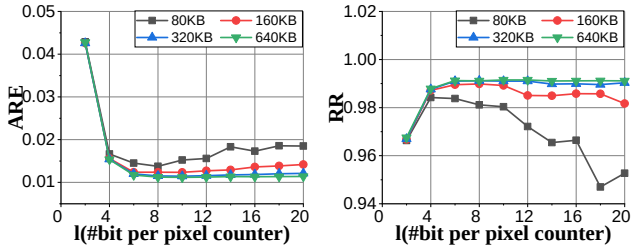
Summary and Analysis: According to the above results, compared to the state-of-the-art Sliding sketches, MicroSketch achieves 4.25 ~ 5.12 times smaller AAE and 1 ~ 2 times faster speed. Also, compared to ECM, MicroSketch achieves 265 ~ 348 times smaller AAE and 4 ~ 5 times faster speed. Such an improvement stems from two-dimensional quantization and adaptive zooming, which improve memory efficiency.

6.3 Performance on Finding Top-k Frequent Items

We use RR, ARE, and speed to evaluate the performance of MicroSketch-HG, MicroSketch-SS, Sl-HK, and WCSS, in finding the top-500 frequent items. Since WCSS only works in the count-based window model, we also use the count-based model and set the window size to 1M packets. We first compare the 4 algorithms under 4 different memory settings using the CAIDA dataset. Then, we test the robustness of MicroSketch-HG and MicroSketch-SS on various datasets, including CAIDA, IMC, and Zipf with varying skewness.

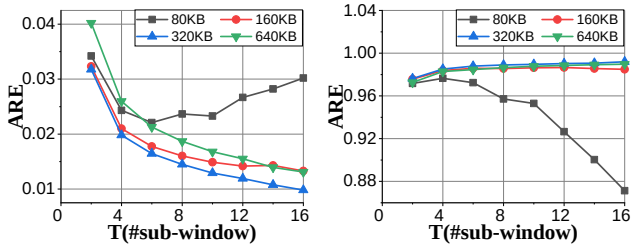


(a) ARE *v.s.* T (b) RR *v.s.* T

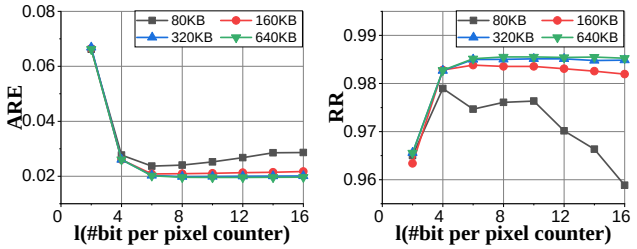


(c) ARE *v.s.* l (d) RR *v.s.* l

Figure 9: Effect of T and l on MicroSketch-HG.



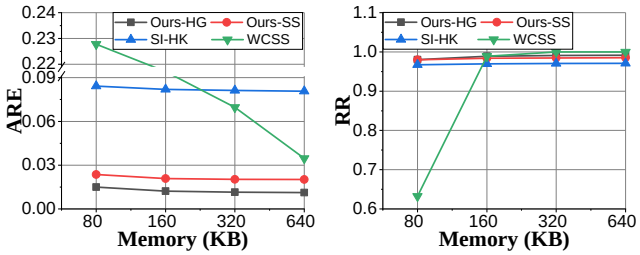
(a) ARE *v.s.* T (b) RR *v.s.* T



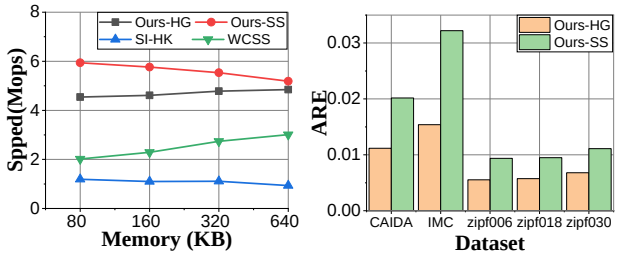
(c) ARE *v.s.* l (d) RR *v.s.* l

Figure 10: Effect of T and l on MicroSketch-SS.

As mentioned earlier, we need to set T and l for MicroSketch-HG and MicroSketch-SS. First, we consider MicroSketch-HG. As shown in Figure 9, at first, the RR goes up, and ARE goes down when T or l goes up. This is due to a larger T giving the hopping window a more accurate approximation of the sliding window, and a larger l makes the rounding error of MicroSketch smaller. However, the RR may go down and ARE go up when the parameters become too large. The reason is that the counters will consume much memory and leave little memory to store items. According to our results, the different memory settings require different values of T and l. Moreover, the better values of T and l grow larger when the memory is larger. Indeed, when we set $T = 12$ and $l = 8$, RR reaches more than 98%, and ARE goes below 0.014 for all considered memory sizes. Similarly, for MicroSketch-SS, we choose $T = 4$ and $l = 6$ according



(a) ARE *v.s.* Algorithms. (b) RR *v.s.* Algorithms.



(c) Speed *v.s.* Algorithms. (d) ARE *v.s.* Datasets.

Figure 11: Performance of algorithms on the top-k task.

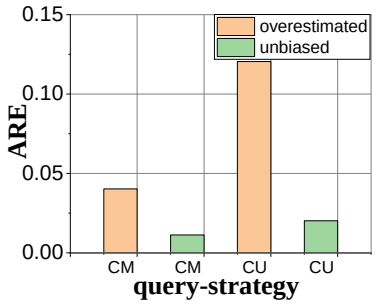
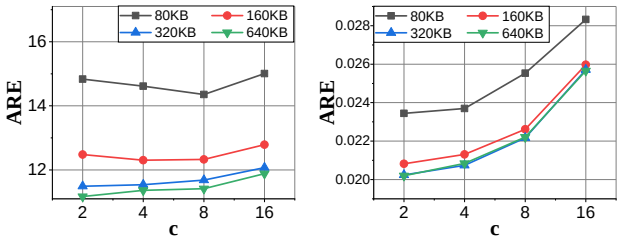


Figure 12: Performance of different query strategies on the top-k task.



(a) MicroSketch-HG (b) MicroSketch-SS

Figure 13: Effect of c on the top-k task.

to Figure 10. Using these parameters, MicroSketch-SS's RR reaches 97.4%, and its ARE goes below 0.026.

MicroSketch-HG, MicroSketch-SS *v.s.* Sl-HK, WCSS (Figure 11). Our results show that compared to Sl-HK and WCSS, MicroSketch-HG achieves at most 7 times and 15 times smaller error rate with at most 5.22 times and 2.25 faster speed, respectively, while MicroSketch-SS achieves at most 4 times and 9.65 times smaller error rate with at most 5.57 times and 2.94 times faster speed. From Figure 11(a), we observe that the ARE of MicroSketch-HG varies between 0.011 and 0.014, the ARE of MicroSketch-SS varies between 0.020 and 0.023, while the ARE of Sl-HK goes between 0.08 and 0.084 and the ARE

of WCSS varies between 0.03 and 0.228. MicroSketch-HG performs the best for all memory sizes in terms of ARE. MicroSketch-SS also performs better than WCSS and Sl-HK. From Figure 11(b), we observe that MicroSketch-HG and MicroSketch-SS obtain a relatively higher RR than others when memory is small. When the memory size reaches 320KB, all algorithms' RR reaches 97%. From Figure 11(c), we observe that MicroSketch-HG and MicroSketch-SS reach about 5 Mops, higher than others in all memory settings.

MicroSketch-HG, MicroSketch-SS *v.s.* Dataset (Figure 11(d)). Our results show that, under a memory size of 640KB, MicroSketch-HG achieves an ARE lower than 0.015 and MicroSketch-SS lower than 0.032 on all datasets. From Figure 11(d), on the IMC dataset, we observe a higher ARE for the algorithms, which comes from the flatter distribution of this dataset, i.e., it is harder to distinguish frequent items well in this dataset.

MicroSketch-HG, MicroSketch-SS *v.s.* Query Strategy (Figure 12). Since originally, HG is underestimated and SS is overestimated, we only compare the underestimated MicroSketch-HG, overestimated MicroSketch-SS, and their unbiased version. We find that the ARE of underestimated MicroSketch-HG and overestimated MicroSketch-SS is about 0.04 and 0.12, respectively, while the unbiased version's ARE is about 0.01 and 0.02.

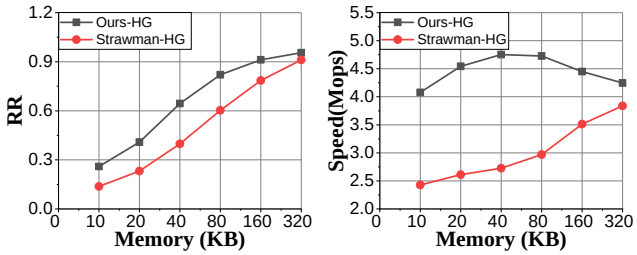
MicroSketch-HG, MicroSketch-SS *v.s.* c (Figure 8). Similar to what we do in the frequency estimation task, we change c from 2 to 16 and find that it has little impact on MicroSketch-HG's performance, since the line is almost horizontal. However, for MicroSketch-SS, the smaller the c is, the higher the accuracy, which is consistent with our choice 2.

Summary and Analysis: Compared with Sl-HK, MicroSketch-HG achieves 5.6 ∼ 7.2 times smaller error rate and 3.8 ∼ 5.2 times faster speed, while MicroSketch-SS achieves 3.56 ∼ 4 times smaller error rate and 4.97 ∼ 5.57 times faster speed. Compared with WCSS, MicroSketch-HG achieves 3 ∼ 15.1 times smaller error rate and 1.61 ∼ 2.25 times faster speed, while MicroSketch-SS achieves 1.7 ∼ 9.6 times smaller error rate and 1.73 ∼ 2.94 times faster speed. The explanation is that the key technique of WCSS and Sl-HK is one-dimensional quantization, while we use two-dimensional quantization and therefore achieve better memory efficiency.

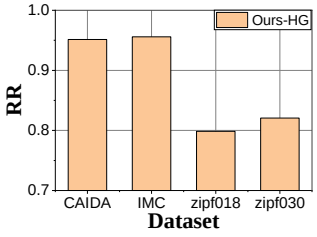
6.4 Performance on Finding Top-k Heavy Changes

We show the performance of MicroSketch-HG and our straw man solution to find the top-500 heavy changes in CAIDA datasets. The straw man solution splits the sliding window into sub-windows and keeps a counter for each sub-window. The difference between the straw man solution and MicroSketch-HG is that MicroSketch-HG additionally uses quantization and adaptive zooming in the frequency dimension. In this experiment, the window size is set to 1M packets. We test the robustness of MicroSketch-HG on various datasets, including CAIDA, IMC, and Zipf with varying skewness.

As shown in Figure 14, we observe that RR goes up with memory size. When the memory is set to 320KB, the RR of MicroSketch-HG reaches about 95.5%, while the RR of the straw man solution is about 91%. However, for relatively small memory sizes, the RR of MicroSketch-HG is almost twice the RR of the straw man solution. Our results show that the technique we use in MicroSketch-HG



(a) RR *v.s.* Algorithms. (b) Speed *v.s.* Algorithms.



(c) RR *v.s.* Datasets.

Figure 14: Performance of algorithms on the heavy change task.

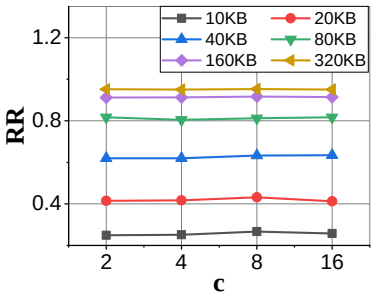


Figure 15: Effect of c on the heavy change task.

improves performance, especially when the memory is limited. Also, MicroSketch-HG is faster than the straw man solution. We also observe that MicroSketch-HG reaches more than 78% RR in Zipf datasets and more than 91% in CAIDA and IMC datasets when the memory setting is 320KB. We also have shown that the change of c has little impact on the RR of our algorithm in Figure 15.

7 CONCLUSION

In this paper, we consider the problem of mining statistics in data streams using sliding windows. We propose a generic framework, MicroscopeSketch, and its key technique, adaptive zooming for estimation in sliding windows. It achieves high accuracy with limited memory usage, making it more suitable for a wide set of practical applications. We apply MicroscopeSketch to well-known algorithms on multiple tasks, including estimating frequency, finding top-k frequent items, and finding top-k heavy changes. We also analyze its error bound, its unilateral error, and unbiased rounding error. We conduct experiments on the three tasks using multiple datasets. Compared to existing works, we find that MicroscopeSketch's error

is at most 363 times smaller in estimating the frequency and at most 15.2 smaller in finding top-k items than the existing works.

REFERENCES

[1] Graham Cormode and S Muthukrishnan. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms*, 2005.

[2] Cristian Estan and George Varghese. New directions in traffic measurement and accounting. *SIGMCOMM Conference*, 2002.

[3] Moses Charikar, Kevin Chen, and Martin Farach-Colton. Finding frequent items in data streams. In *Automata, Languages and Programming*. Springer, 2002.

[4] Daniel Ting. Data sketches for disaggregated subset sum and frequent item estimation. In *SIGMOD Conference*, 2018.

[5] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *SIGMOD Conference*, 2016.

[6] M. Gurmeet Singh and M. Rajeev. Approximate frequency counts over data streams. In *VLDB*, 2002.

[7] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. Efficient computation of frequent and top-k elements in data streams. In *ICDT*, 2005.

[8] G. Lukasz, D. David, D. Erik D, and etal. Identifying frequent items in sliding windows over on-line packet streams. In *ACM IMC*, 2003.

[9] Tong Yang, Junzhi Gong, Haowei Zhang, and etal. Heavyguardian: Separate and guard hot items in data streams. In *SIGKDD Conference*, 2018.

[10] Tong Yang, Jie Jiang, Peng Liu, and etal. Elastic sketch: Adaptive and fast network-wide measurements. In *SIGCOMM Conference*, 2018.

[11] Ran Ben-Basat, Gil Einziger, Roy Friedman, and etal. Randomized admission policy for efficient top-k and frequency estimation. In *INFOCOM Conference*, 2017.

[12] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and etal. Sketching linear classifiers over data streams. In *SIGMOD Conference*, 2018.

[13] Anshumali Shrivastava, Arnd Christian Konig, and Mikhail Bilenko. Time adaptive sketches (ada-sketches) for summarizing data streams. In *SIGMOD Conference*, 2016.

[14] Yanqing Peng, Jinwei Guo, Feifei Li, and etal. Persistent bloom filter: Membership testing for the entire history. In *SIGMOD Conference*, 2018.

[15] Nan Tang, Qing Chen, and Prasenjit Mitra. Graph stream summarization: From big bang to big crunch. In *SIGMOD Conference*, 2016.

[16] Zhewei Wei, Ge Luo, Ke Yi, and etal. Persistent data sketching. In *SIGMOD Conference*, 2015.

[17] Haipeng Dai, Muhammad Shahzad, Alex X Liu, and etal. Finding persistent items in data streams. *VLDB Endowment*, 2016.

[18] Daniel Ting. Count-min: Optimal estimation and tight error bounds using empirical error distributions. In *SIGKDD Conference*, 2018.

[19] Prashant Pandey, Michael A. Bender, Rob Johnson, and etal. A general-purpose counting filter: Making every bit count. In *SIGKDD Conference*, 2017.

[20] Alex D. Breslow and Nuwan S. Jayasena. Morton filters: Faster, space-efficient cuckoo filters via biasing, compression, and decoupled logical sparsity. *VLDB Endow.*, 2018.

[21] Minmei Wang, Mingxun Zhou, Shouqian Shi, and etal. Vacuum filters: More space-efficient and faster replacement for bloom and cuckoo filters. *VLDB Endow.*, 2019.

[22] Yuliang Li, Rui Miao, Changhoon Kim, and Minlan Yu. Flowradar: a better netflow for data centers. In *USENIX NSDI*, 2016.

[23] K. Balachander, S. Subhabrata, Z. Yin, and etal. Sketch-based change detection: methods, evaluation, and applications. In *IMC*, 2003.

[24] Peiqing Chen, Dong Chen, Lingxiao Zheng, Jizhou Li, and Tong Yang. Out of many we are one: Measuring item batch with clock-sketch. SIGMOD, 2021.

[25] Aécio Santos, Aline Bessa, Fernando Chirigati, Christopher Musco, and Juliana Freire. Correlation sketches for approximate join-correlation queries. In *SIGMOD*, 2021.

[26] Rundong Li, Pinghui Wang, Jiongli Zhu, Junzhou Zhao, Jia Di, Xiaofei Yang, and Kai Ye. Building fast and compact sketches for approximately multi-set multi-membership querying. In *SIGMOD*, 2021.

[27] Y. Izenov, A. Datta, F. Rusu, and JH Shin. Online sketch-based query optimization. 2021.

[28] Zheng Zhong, Shen Yan, Zikun Li, Decheng Tan, Tong Yang, and Bin Cui. Burstsketch: Finding bursts in data streams. In *SIGMOD*, 2021.

[29] Daniel Ting and Rick Cole. Conditional cuckoo filters. In *SIGMOD*, 2021.

[30] Prashant Pandey, Alex Conway, Joe Durie, Michael A Bender, Martin Farach-Colton, and Rob Johnson. Vector quotient filters: Overcoming the time/space trade-off in filter design. In *SIGMOD*, 2021.

[31] Gaurav Gupta, Minghao Yan, Benjamin Coleman, Bryce Kille, RA Leo Elworth, Tharun Medini, Todd Treangen, and Anshumali Shrivastava. Fast processing and querying of 170tb of genomics data via a repeated and merged bloom filter (rambo). In *SIGMOD*, 2021.

[32] Yesdaulet Izenov, Asoke Datta, Florin Rusu, and Jun Hyung Shin. Compass: Online sketch-based query optimization for in-memory databases. In *SIGMOD*, 2021.

[33] Benwei Shi, Zhuoyue Zhao, Yanqing Peng, Feifei Li, and Jeff M Phillips. At-the-time and back-in-time persistent sketches. In *SIGMOD*, 2021.

[34] Peng Jia, Pinghui Wang, Junzhou Zhao, Shuo Zhang, Yiyan Qi, Min Hu, Chao Deng, and Xiaohong Guan. Bidirectionally densifying lsh sketches with empty bins. In *SIGMOD*, 2021.

[35] Zhenwei Dai, Aditya Desai, Reinhard Heckel, and Anshumali Shrivastava. Active sampling count sketch (ascs) for online sparse estimation of a trillion scale covariance matrix. In *SIGMOD*, 2021.

[36] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. A learned sketch for subgraph counting. In *SIGMOD*, 2021.

[37] Chunyao Song, Xuanming Liu, Tingjian Ge, and Yao Ge. Top-k frequent items and item frequency tracking over sliding windows of any size. *Information Sciences*, 475:100–120, 2019.

[38] Nicoló Rivetti, Yann Busnel, and Achour Mostefaoui. Efficiently summarizing data streams over sliding windows. In *2015 IEEE 14th International Symposium on Network Computing and Applications*, pages 151–158. IEEE, 2015.

[39] Xiangyang Gou, Long He, Yinda Zhang, and etal. Sliding sketches: A framework using time zones for data stream processing in sliding windows. In *SIGKDD Conference*, 2020.

[40] Tong Yang, Yang Zhou, Hao Jin, and etal. Pyramid sketch: A sketch framework for frequency estimation of data streams. *VLDB Endow.*, 2017.

[41] Fan Deng and Davood Rafiei. New estimation algorithms for streaming data: Count-min can do more. *Webdocs. Cs. Ualberta. Ca*, 2007.

[42] Tao Li, Shigang Chen, and Yibei Ling. Per-flow traffic measurement through randomized counter sharing. *IEEE/ACM Transactions on Networking*, 2012.

[43] Odysseas Papapetrou, Minos Garofalakis, and Antonios Deligiannakis. Sketch-based querying of distributed sliding-window data streams. *arXiv preprint arXiv:1207.0139*, 2012.

[44] Mayur Datar, Aristides Gionis, Piotr Indyk, and etal. Maintaining stream statistics over sliding windows. *SICOMP*, 2002.

[45] Ran Ben-Basat, Gil Einziger, Roy Friedman, and etal. Heavy hitters in streams and sliding windows. In *INFOCOM*, 2016.

[46] Tong Yang, Haowei Zhang, Jinyang Li, and etal. Heavykeeper: An accurate algorithm for finding top-k elephant flows. *TON*, 2019.

[47] The CAIDA UCSD Anonymized Internet Traces Dataset - 2018.03.15. http://www.caida.org/data/passive/passive_dataset.xml.

[48] Theophilus Benson, Aditya Akella, and David A Maltz. Network traffic characteristics of data centers in the wild. In *SIGCOMM conference*, 2010.

[49] Applications and explanations of Zipf's law.