MicroscopeSketch: Accurate Sliding Estimation Using Adaptive Zooming

Yuhan Wu[*†]
Peking University
yuhan.wu@pku.edu.cn

Shiqi Jiang[‡]
Peking University
shiqijiang@pku.edu.cn

Siyuan Dong[§]
Peking University
dongsiyuan@pku.edu.cn

Zheng Zhong[§]
Peking University
zheng.zhong@pku.edu.cn

Jiale Chen[§]
Peking University
jiale_chen@pku.edu.cn

Yutong Hu[¶]
Peking University
huyutong@pku.edu.cn

Tong Yang[*†]
Peking University
yangtongemail@gmail.com

Steve Uhlig[‖]
Queen Mary University of London
steve.uhlig@quml.ac.uk

Bin Cui[*]
Peking University
bin.cui@pku.edu.cn

ABSTRACT

[1] High-accuracy real-time data stream estimations are critical for various applications, and sliding-window-based techniques have attracted wide attention. However, existing solutions struggle to achieve high accuracy, generality, and low memory usage simultaneously. To overcome these limitations, we present MicroscopeSketch, a high-accuracy sketch framework. Our key technique, called adaptive zooming, dynamically adjusts the granularity of counters to maximize accuracy while minimizing memory usage. By applying MicroscopeSketch to three specific tasks—frequency estimation, top-k frequent items discovery, and top-k heavy changes identification—we demonstrate substantial improvements over existing methods, reducing errors by roughly 4 times for frequency estimation and 3 times for identifying top-k items. The relevant source code is available in a GitHub repository.

CCS CONCEPTS

• **Theory of computation** → **Sketching and sampling**; • **Networks** → *Network measurement*.

[*]National Key Laboratory for Multimedia Information Processing, School of Computer Science, Peking University, Beijing, China
[†]Peng Cheng Laboratory, Shenzhen, China
[‡]School of Mathematical Sciences, Peking University, Beijing, China
[§]School of Electronic Engineering and Computer Science, Peking University, Beijing, China
[¶]School of Intelligence Science and Technology, Peking University, Beijing, China
[‖]School of Electronic Engineering and Computer Science, London, U.K.
[1]Yuhan Wu, Shiqi Jiang, and Siyuan Dong contribute equally to this paper. Corresponding author: Tong Yang (yangtongemail@gmail.com).

KDD '23, August 6–10, 2023, Long Beach, CA, USA

ACM ISBN 979-8-4007-0103-0/23/08…$15.00
https://doi.org/10.1145/3580305.3599432

KEYWORDS

Data stream mining, Sketch, Data structure, Sliding window, Approximate query

ACM Reference Format:
Yuhan Wu, Shiqi Jiang, Siyuan Dong, Zheng Zhong, Jiale Chen, Yutong Hu, Tong Yang, Steve Uhlig, and Bin Cui. 2023. MicroscopeSketch: Accurate Sliding Estimation Using Adaptive Zooming. In *Proceedings of the 29th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '23), August 6–10, 2023, Long Beach, CA, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3580305.3599432

1 INTRODUCTION

Approximate estimation of data streams provides real-time information for various applications, such as anomaly detection [1–3], network measurement [4, 5], network accounting [6], joining tables [7, 8], and more [9–22]. Such estimation typically involves tasks such as frequency estimation [23–27], finding the top-k frequent items [28–31], and finding the top-k heavy changes [32–34]. Specifically, in a data stream, each item carries a key/identifier, *e.g.*, the source IP address of a data packet in the network stream. Frequency estimation refers to approximating the number of items with the same key/identifier. Finding the top-k frequent items aims to identify the k-most frequent items. Similarly, finding top-k heavy changes aims to identify the k items whose frequency underwent the most significant change between two adjacent time windows. For these tasks, the sketch [35–40], a probabilistic data structure, has been widely considered thanks to its small and controllable error as well as its memory efficiency.

For data streams, there are two primary time window models—fixed window and sliding window, which have been extensively studied in literature [41–46]. The sliding window model focuses on recent data items, such as those in the last three minutes. As it better captures the real-time changes in data stream characteristics, we focus on the sliding window model over the fixed window model.

The state-of-the-art solution for approximate estimation in sliding windows is sliding sketches [45, 47]. Sliding Sketches divide the sliding window into smaller sub-windows, utilizing the consolidation of the most recent ones in order to approximate the overall

window. Specifically, sliding sketches partition time into several stationary sub-windows of identical size, and ensure that by combining the most recent $T+1$ sub-windows, the entire target sliding window is completely covered. Subsequently, Sliding sketches employ $T+1$ sketches to individually record the frequency of items in the most recent $T+1$ sub-windows. As time passes, the oldest sub-window exits the sliding window, prompting the corresponding outdated sketch structure to clear itself and begin tracking a new incoming sub-window. Although Sliding sketches achieve good accuracy, their design concentrates solely on the time dimension (*e.g.*, dividing the sliding window and discarding old data) without taking into account the frequency similarities across the $T+1$ sub-windows. As a result, their accuracy is hindered from achieving further improvement.

Our goal is to design a sketch for sliding window estimation with following four requirements: 1) High accuracy; 2) Small memory usage; 3) Versatility in terms of application to a variety of estimation tasks and algorithms; 4) Generalizable 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.

In this paper, we propose MicroscopeSketch (MicroSketch for short) for three typical estimation tasks in the sliding window model. MicroSketch offers 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 or network devices. 3) Generality: it can be applied to all sketches that use counters for estimation in time-based or count-based sliding window.

Our key technique is named *adaptive zooming*, which dynamically alters the granularity of count tracking to obtain the highest possible accuracy in frequencies while minimizing storage space usage. We approximate the time window in the same way as the sliding sketch, using $T+1$ independent counters for tracking the frequencies of a single key's presence in each of the corresponding sub-windows. We discovered that, for each of most item keys (whether frequent or not), their count in the $T+1$ sub-windows is often quite similar in most cases, *i.e.*, the size is within the same order of magnitude. This property enables a more compact representation of the $T+1$ numbers. In this regard, we introduce adaptive zooming that emulates the floating-point number representation (*i.e.*, partitioning each number into a fixed-precision integer referred to as the significand and an integer exponent). We then consolidate the $T+1$ integer exponents into a single exponent, thereby requiring only a small additional significand to be stored for each number.

In summary, We propose MicroSketch, a precise and versatile approach for three estimation tasks in the sliding window model, including frequency estimation, finding top-k item, and top-k heavy change detection. Through theoretical analysis, we demonstrate that different querying approaches can allow MicroSketch to achieve overestimation, underestimation, or unbiased rounding. Our evaluation shows that MicroSketch has considerably smaller error rates than prior works, with frequency estimation errors reduced by around 4 times and top-k item searches by around 3 times smaller than existing approaches. Our source code is available on Github [48].

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, which we will elaborate on in Section 2.2, 2.3 and 2.4. For algorithms in fixed windows, we refer the reader to [49, 50].

2.1 Preliminaries

The data stream [45]: A data stream is a sequence of items that arrive in real time, where each item has its own identifier/key (*e.g.*, each network packet item has a source IP address as the key). Whenever an item arrives, the processing algorithm (*i.e.*, the sketch we are targeting) will insert it into the data structure. When a user submits a query, we will query the data structure and return the results.

The sliding window model [51]: In the data stream, users often query the items arrived in the latest period of time, where the period is called sliding window. Depending on how the sliding window is defined, there are two kinds of sliding window models: *count-based* window is the most recent W items, and *time-based* window is the period from time $t-W$ to t, where t is the current time and W is the window size. For example, in the sliding window, users can perform frequency estimation, that is, count how many items have the same key:

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 (Section 2.3 and 2.4).

Sketch: Sketch is a probabilistic data structure employing hashing that can quickly process items in a data stream, using a small amount of memory [23, 52, 53].

2.2 Frequency Estimation

Frequency estimation refers to estimating the frequency of arbitrary items. Prior works are Sliding sketches [45] and ECM [54]. Sliding sketches is a framework, which can apply to many algorithms. To estimating frequency for arbitrary item, Sliding sketches apply CM sketch [23] and CU [6] 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.

ECM sketch [54] is based on the CM sketch [23], and it uses Exponential Histograms (EH) to replace each counter in the CM sketch. Exponential Histograms (EH) [41] 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, $\langle cnt,ts\rangle$, 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 $\langle 1,t_{now}\rangle$. Then, it may merge some buckets by some rules. To merge $\langle cnt_1,ts_1\rangle$ and $\langle cnt_2,ts_2\rangle$, the new bucket is $\langle cnt_1+cnt_2,\min ts_1,ts_2\rangle$. 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 [45] and WCSS [55]. Sliding sketches apply HeavyKeeper [56] 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 heavy changes is to detect traffic anomalies by looking for significant changes in a short time that are inconsistent with its normal behavior. This is a crucial task in a lot of big data scenarios, such as anomalies detection [57, 58], web search engines [59], time series forecasting and outlier analysis [60], etc. Nevertheless, normal heavy changes needs users to predefine the threshold of frequency difference. To skip the trouble, We devise MicroSketch to find the 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 first present the data structure of our solution in Section 3.1. Then we introduce the operations of MicroSketch in Section 3.2. Finally, we showcase the optimization method of unbiased rounding in Section 3.3. 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. c is a predefined base number. We will elaborate on the usage of a, b and c in Section 3.2. 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 (We will elaborate in Section 3.2). 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.

Here we explain two design choices briefly: (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 1, 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 [6], 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.[2] 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 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* (The difference between the real value and the value after rounding) 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.

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.

[2] It does not imply that we can simultaneously provide overestimation and underestimation for the target frequency f.

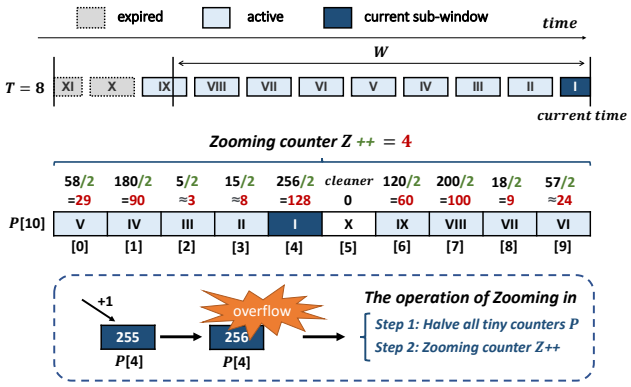


Figure 1: An example of MicroscopeSketch ($c = 2$). $I \sim XI$ are ten sub-windows. I is the current sub-window. We use 8 bits for each pixel counter, so their maximum value is 255. After insertion, the value of $P[4]$ is going to be 256, which will cause an overflow. Therefore, we increment the zooming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

4 APPLYING MICROSCOPE 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 [23] (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 [6] 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 [31] 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 $\langle ID, frequency\rangle$ 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

Due to space limitation, the contents of error analysis are provided in our GitHub repository [48]. We will first show how MicroSketch can achieve either overestimation or underestimation by using different querying strategies. 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.

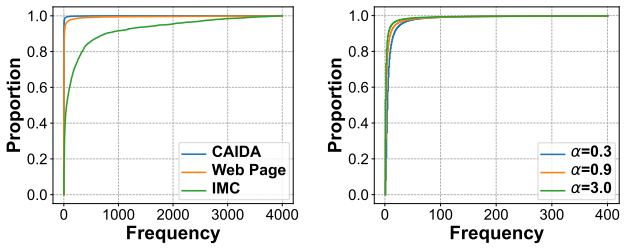
6 EXPERIMENT RESULTS

In this section, we evaluate the performance of MicroSketch for three tasks: frequency estimation, finding top-k frequent items, and finding top-k heavy changes. Due to the space constraints, we have placed the experiment results for selecting the optimal parameters for MicroSketch in our Appendix. The relevant source code is available in our GitHub repository [48].

6.1 Experiment 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 four kinds of datasets in our experiments: the CAIDA anonymized Internet traces, the IMC data center traffic traces, synthetic stream datasets that follow a Zipf distribution, and the Web Page dataset of distinct terms in a web page. We show their frequency distributions in Figure 2.



(a) CAIDA, Web Page and IMC. (b) Zipf.

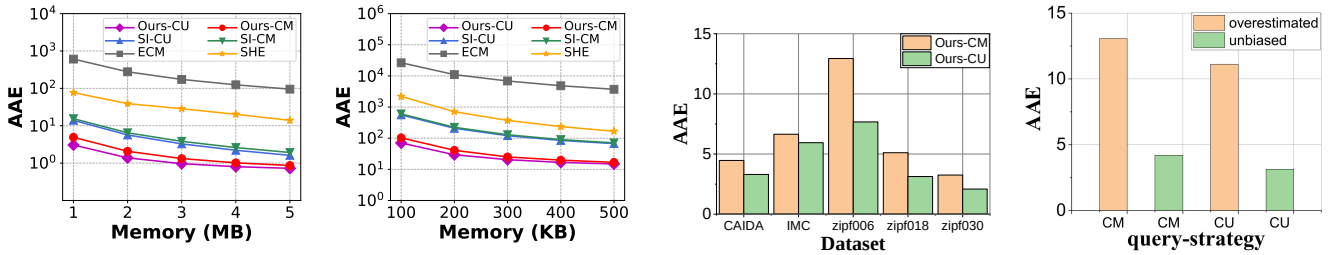
Figure 2: Distribution of studied datasets.

1) **CAIDA:** This is a set of IP packet streams, coming from the *CAIDA Anonymized Internet Trace 2018* [61]. 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.
2) **IMC:** The second dataset comes from one of the data centers studied in *Network Traffic Characteristics of Data Centers in the Wild* [62]. 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.
3) **Zipf:** We also generate three synthetic datasets following the Zipf 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.
4) **Web Page:** We build the fourth dataset from a collection of web pages. The Web Page dataset is downloaded from the website[3]. Each item (4 bytes) represents the number of distinct terms in a web page.

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, to fit different memory settings, we only change the number of counters and fix all remaining parameters. We show the algorithms compared to ours in each task below.

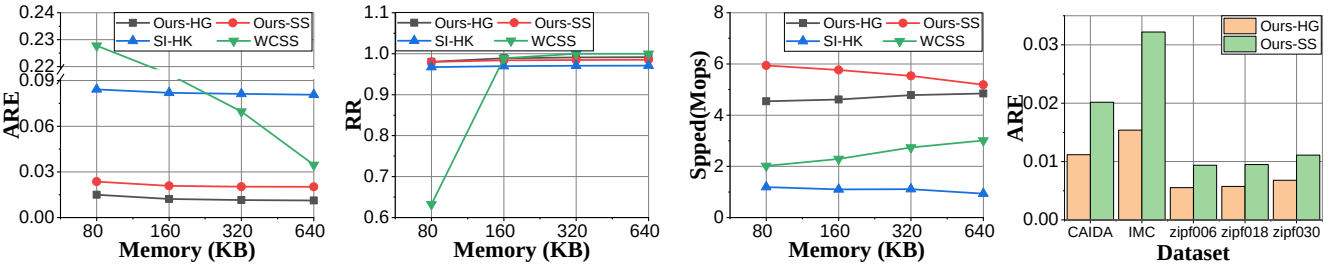
- Frequency Estimation: We choose the state-of-the-art, Sliding sketches [45], SHE [44], and ECM [54], for comparison. Sliding sketches is a framework that can be used on different algorithms. We combine Sliding sketches (Sl) [45] with CM and CU (Sl-CM, Sl-CU) for comparison. According to the results of the ECM original paper, to achieve high accuracy, the parameter u in EH [41] is set to 2. We set the number of hash functions to 3 in every algorithm.
- Finding Top-k Frequent Items: We compare with the state-of-the-art algorithms: Sliding sketches with HeavyKeeper (Sl-HK) [56], and WCSS [55].
- Finding Top-k Heavy Changes: We evaluate the MicroSketch-HG under different memory sizes. In this case, we cannot

[3] http://fimi.ua.ac.be/data/



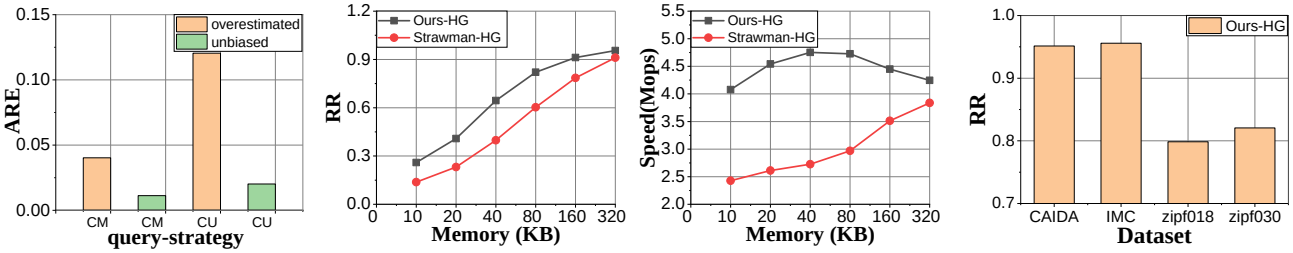
(a) AAE *v.s.* Algorithms (Count-Based) (b) AAE *v.s.* Algorithms (Time-Based) (c) AAE *v.s.* Datasets (d) AAE *v.s.* query strategies

Figure 3: Performance of algorithms and query strategies on the frequency estimation task on CAIDA.



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

Figure 4: Performance of algorithms on the top-k task on CAIDA.



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

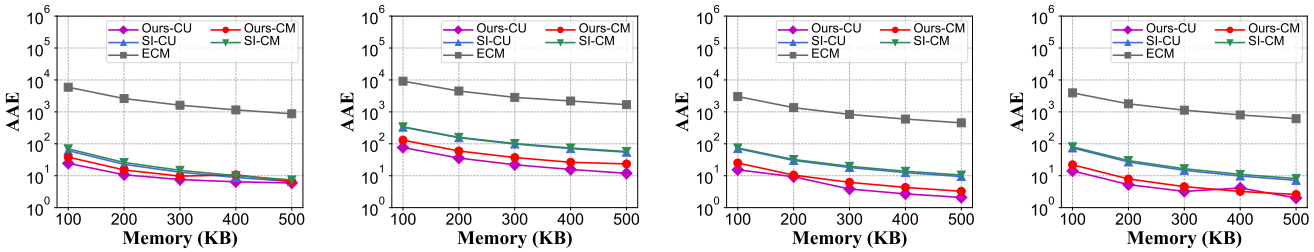
Figure 5: Query strategies on the top-k task on CAIDA.

Figure 6: Performance of algorithms on the heavy change task on CAIDA.

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.

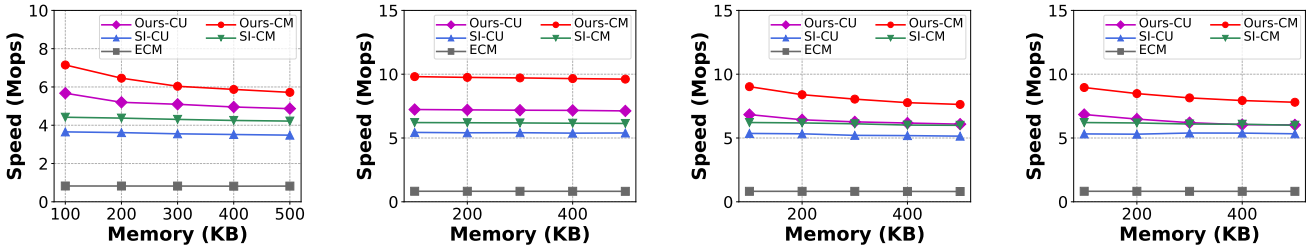
Metrics: AAE (Average Absolute Error), ARE (Average Relative Error), RR (Recall Rate) and Speed (the Million operations per second or Mops).

- **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.
- **Speed:** We test the Million operations (insertions) per second (Mops) of different algorithms.



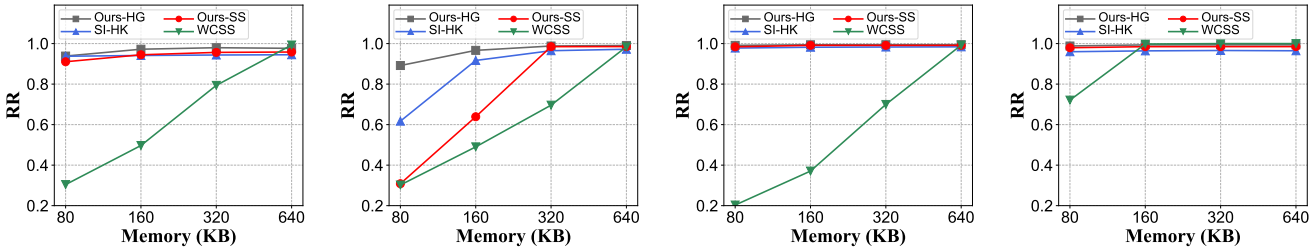
(a) AAE on IMC (b) AAE on Zipf-0.3 (c) AAE on Zipf-3.0 (d) AAE on Webpage

Figure 7: Accuracy comparison experiments for frequency estimation tasks across various datasets.



(a) Speed on CAIDA (b) Speed on IMC (c) Speed on Zipf-3.0 (d) Speed on Webpage

Figure 8: Speed comparison experiments for frequency estimation tasks across various datasets.



(a) RR on IMC (b) RR on Zipf-0.3 (c) RR on Zipf-3.0 (d) RR on Webpage

Figure 9: Accuracy comparison experiments for top-k tasks across various datasets.

6.2 Frequency Estimation

We compare six algorithms: MicroSketch-CM, MicroSketch-CU, Sl-CM, Sl-CU, SHE, and ECM, in the count-based model and time-based model. The sliding window size is set to 1M packets. Different datasets are used to compare the algorithms' AAE and speed under five different memory sizes. We also show the robust performance of MicroSketch-CM and MicroSketch-CU on different datasets.

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.

MicroSketch-CM, MicroSketch-CU *v.s.* Sl-CM, Sl-CU, ECM (Figure 3(a), 3(b), 7, 8). MicroSketch-CM and MicroSketch-CU achieve both lower AAE and faster speed than Sl-CU, Sl-CM, SHE and ECM. On CAIDA dataset, MicroSketch achieves about 130 ~ 200 times smaller error rate and about 6.96 ~ 8.65 times faster speed compared to ECM. Compared to Sliding sketches, MicroSketch achieves 2.23 ~ 5.06 times smaller error rate and 1.36 ~ 1.95 times faster speed. Compared to SHE, MicroSketch achieves 19.2 ~ 29.5 times smaller error rate. On other datasets, MicroSketch always achieves smaller error rate and faster speed compared to ECM, SHE and Sliding sketches in all memory settings.

MicroSketch-CM, MicroSketch-CU *v.s.* Dataset (Figure 3(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.

MicroSketch-CM, MicroSketch-CU *v.s.* Query Strategy (Figure 3(d)). 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.

Summary and Analysis: According to the above results, compared to the state-of-the-art Sliding sketches, MicroSketch achieves $2.23 \sim 5.06$ times smaller AAE and $1.36 \sim 1.95$ times faster speed. Also, compared to ECM, MicroSketch achieves $130 \sim 200$ times smaller AAE and $6.96 \sim 8.65$ times faster speed. Compared to ECM, MicroSketch achieves $19.2 \sim 29.5$ times smaller AAE. Such an improvement stems from adaptive zooming, which improve memory efficiency.

6.3 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.

MicroSketch-HG, MicroSketch-SS *v.s.* Sl-HK, WCSS (Figure 4(a), 4(b), 4(c), 9). Our results show that MicroSketch-HG and MicroSketch-SS beat Sl-HK and WCSS in most cases. On CAIDA dataset, 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 4(a), we observe that MicroSketch-HG performs the best for all memory sizes in terms of ARE and MicroSketch-SS also performs better than WCSS and Sl-HK. From Figure 4(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 4(c), we observe that MicroSketch-HG and MicroSketch-SS reach about 5 Mops, higher than others in all memory settings.

From Figure 9, we observe that on IMC, Zipf-3.0 and Webpage, MicroSketch-HG and MicroSketch-SS perform better than Sl-HK in all memory settings and better than WCSS when memory is tight. On Zipf-0.3, when the memory size reaches 320KB, the RR of MicroSketch-HG and MicroSketch-SS reach around 98.5%, beating the other two algorithms.

MicroSketch-HG, MicroSketch-SS *v.s.* Dataset (Figure 4(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 4(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 5). 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.

Summary and Analysis: Compared with Sl-HK, MicroSketch-HG achieves $5.6 \sim 7.2$ times smaller error rate and $3.8 \sim 5.2$ times faster speed, while MicroSketch-SS achieves $3.56 \sim 4$ times smaller error rate and $4.97 \sim 5.57$ times faster speed. Compared with WCSS, MicroSketch-HG achieves $3 \sim 15.1$ times smaller error rate and $1.61 \sim 2.25$ times faster speed, while MicroSketch-SS achieves $1.7 \sim 9.6$ times smaller error rate and $1.73 \sim 2.94$ times faster speed. We use adaptive zooming and therefore achieve better memory efficiency.

6.4 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.

In Figure 6, our results show that the technique we use in MicroSketch-HG improves accuracy, 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.

7 CONCLUSION

In this paper, we focus on approximate statistics estimation in data streams using sliding windows. We propose a generic framework, MicroscopeSketch, and its key technique, adaptive zooming for estimation in sliding windows. MicroscopeSketch achieves high accuracy with limited memory usage, making it more suitable for a wide set of practical applications. We apply it to several 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 around 4 times smaller in estimating the frequency and around 3 times smaller in finding top-k items.

ACKNOWLEDGMENT

We thank all anonymous reviewers for their help in improving this paper. This work is supported by Key-Area Research and Development Program of Guangdong Province 2020B0101390001, and National Natural Science Foundation of China (NSFC) (No. U20A20179 and 61832001).

REFERENCES

[1] SIGKDD. Spotlight: Detecting anomalies in streaming graphs. 2018.

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

[3] Emaad Manzoor, Sadegh M Milajerdi, and Leman Akoglu. Fast memory-efficient anomaly detection in streaming heterogeneous graphs. In *SIGKDD*, 2016.

[4] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, Peter Steenkiste, G Liu, A Li, C Canel, AA Philip, R Ware, et al. Sketchlib: Enabling efficient sketch-based monitoring on programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 743–759, 2022.

[5] Rui Ding, Shibo Yang, Xiang Chen, and Qun Huang. Bitsense: Universal and nearly zero-error optimization for sketch counters with compressive sensing. In *Proc. of SIGCOMM*, 2023.

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

[7] Florin Rusu and Alin Dobra. Sketches for size of join estimation. *ACM Transactions on Database Systems (TODS)*, 33(3):1–46, 2008.

[8] Aécio Santos, Aline Bessa, Fernando Chirigati, and etal. Correlation sketches for approximate join-correlation queries. In *SIGMOD*, 2021.

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

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

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

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

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

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

[15] Takuya Akiba and Yosuke Yano. Compact and scalable graph neighborhood sketching. In *SIGKDD*, 2016.

[16] Pinghui Wang, Yiyan Qi, Yuanming Zhang, Qiaozhu Zhai, Chenxu Wang, John CS Lui, and Xiaohong Guan. A memory-efficient sketch method for estimating high similarities in streaming sets. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 25–33, 2019.

[17] Runze Lei, Pinghui Wang, Rundong Li, and etal. Fast rotation kernel density estimation over data streams. In *SIGKDD*, 2021.

[18] Dingqi Yang, Paolo Rosso, Bin Li, and etal. Nodesketch: Highly-efficient graph embeddings via recursive sketching. In *SIGKDD*, 2019.

[19] 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.

[20] 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.

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

[22] 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.

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

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

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

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

[27] Bohan Zhao, Xiang Li, Boyu Tian, and etal. Dhs: Adaptive memory layout organization of sketch slots for fast and accurate data stream processing. In *SIGKDD*, 2021.

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

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

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

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

[32] Qun Huang, Patrick PC Lee, and Yungang Bao. Sketchlearn: relieving user burdens in approximate measurement with automated statistical inference. In *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 576–590, 2018.

[33] Robert Schweller, Zhichun Li, Yan Chen, Yan Gao, Ashish Gupta, Yin Zhang, Peter A Dinda, Ming-Yang Kao, and Gokhan Memik. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Transactions on Networking*, 15(5):1059–1072, 2007.

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

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

[36] Daniel Ting, Jonathan Malkin, and Lee Rhodes. Data sketching for real time analytics: Theory and practice. In *SIGKDD*, 2020.

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

[38] Yikai Zhao, Wenchen Han, Zheng Zhong, Yinda Zhang, Tong Yang, and Bin Cui. Double-anonymous sketch: Achieving fairness for finding global top-k frequent items. In *Proc. of ACM SIGMOD*, 2023.

[39] Peiqing Chen, Yuhan Wu, Tong Yang, Junchen Jiang, and Zaoxing Liu. Precise error estimation for sketch-based flow measurement. In *Proceedings of the 21st ACM Internet Measurement Conference*, pages 113–121, 2021.

[40] Haoyu Li, Qizhi Chen, Yixin Zhang, Tong Yang, and Bin Cui. Stingy sketch: a sketch framework for accurate and fast frequency estimation. *Proceedings of the VLDB Endowment*, 15(7):1426–1438, 2022.

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

[42] 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.

[43] 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*, 2019.

[44] Yuhan Wu, Zhuochen Fan, Qilong Shi, Yixin Zhang, Tong Yang, Cheng Chen, Zheng Zhong, Junnan Li, Ariel Shtul, and Yaofeng Tu. She: A generic framework for data stream mining over sliding windows. In *Proceedings of the 51st International Conference on Parallel Processing*, pages 1–12, 2022.

[45] 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.

[46] Eran Assaf, Ran Ben Basat, Gil Einziger, and Roy Friedman. Pay for a sliding bloom filter and get counting, distinct elements, and entropy for free. In *IEEE INFOCOM 2018-IEEE Conference on Computer Communications*, pages 2204–2212. IEEE, 2018.

[47] Xiangyang Gou, Yinda Zhang, Zhoujing Hu, Long He, Ke Wang, Xilai Liu, Tong Yang, Yi Wang, and Bin Cui. A sketch framework for approximate data stream processing in sliding windows. *IEEE Transactions on Knowledge and Data Engineering*, 2022.

[48] The source codes related to microscopesketch. https://github.com/MicroscopeSketch/MicroscopeSketch.

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

[50] Hui Han, Zheng Yan, Xuyang Jing, and Witold Pedrycz. Applications of sketches in network traffic measurement: A survey. *Information Fusion*, 82:58–85, 2022.

[51] Zijie Zeng, Lin Cui, Mimi Qian, Zhen Zhang, and Kaimin Wei. A survey on sliding window sketch for network measurement. *Computer Networks*, 226:109696, 2023.

[52] Graham Cormode. Sketch techniques for approximate query processing. *Foundations and Trends in Databases. NOW publishers*, 2011.

[53] Rana Shahout, Roy Friedman, and Dolev Adas. Cell: counter estimation for per-flow traffic in streams and sliding windows. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2021.

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

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

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

[57] Er Krishnamurthy, Subhabrata Sen, and Yin Zhang. Sketchbased change detection: Methods, evaluation, and applications. In *Proc. ACM SIGCOMM Internet Measurement Conference*, 2003.

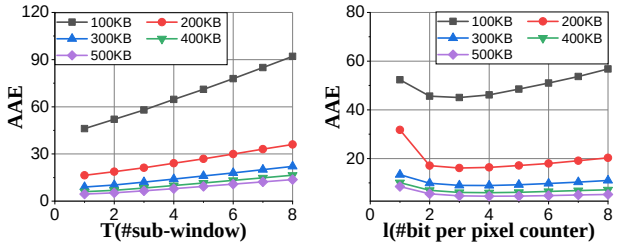
[58] Robert Schweller, Ashish Gupta, Elliot Parsons, and Yan Chen. Reversible sketches for efficient and accurate change detection over network data streams. In *Proc. Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, 2004.

[59] Monika Rauch Henzinger. Algorithmic challenges in web search engines. *Internet Mathematics,*, 1(1):115–123, 2004.

[60] Chung Chen and Lon-Mu Liu. Forecasting time series with outliers. *Journal of Forecasting*, 12(1):13–35, 1993.

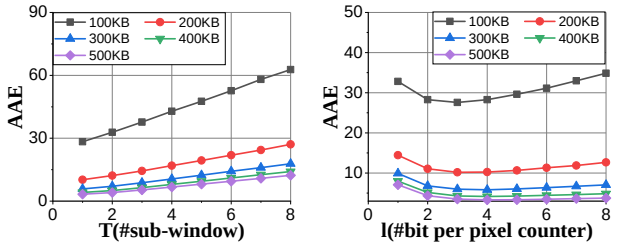
[61] The CAIDA UCSD Anonymized Internet Traces Dataset - 2018.03.15. http://www.caida.org/data/passive/passive_dataset.xml.

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



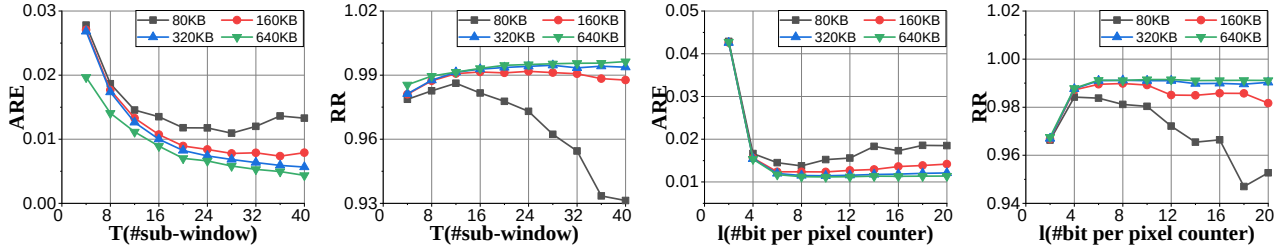
(a) AAE *v.s.* T (b) AAE *v.s.* l

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



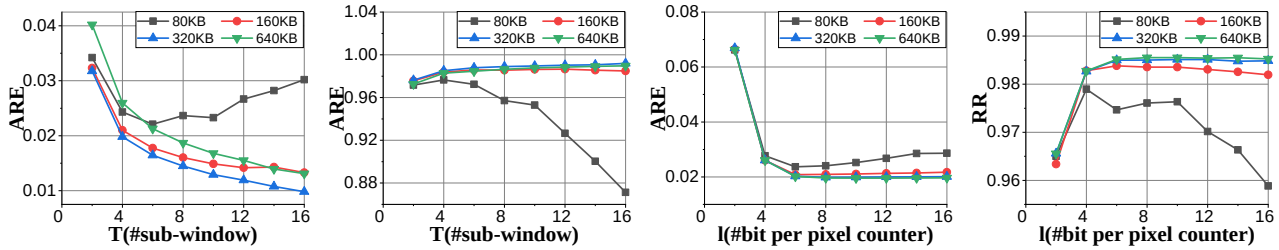
(a) AAE *v.s.* T (b) AAE *v.s.* l

Figure 11: Effect of T and l on MicroSketch-CU.



(a) ARE *v.s.* T (b) RR *v.s.* T (c) ARE *v.s.* l (d) RR *v.s.* l

Figure 12: 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 13: Effect of T and l on MicroSketch-SS.

A EXPERIMENTS ON PARAMETERS

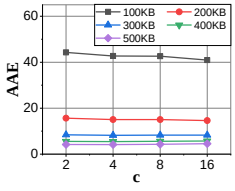
In Frequency Estimation Task, we employ MicroSketch-CM and MicroSketch-CU. In the figures, we use #sub-window to denote the number of sub-windows within the sliding window, i.e., the parameter T. Similarly, we use #bit per pixel counter to denote the number of bits used for each pixel counter, i.e., the parameter l. As shown in Figure 10 and 11, both AAEs increase with T, so we set $T=1$. As for l, there is not an optimal one, but $l=4$ performs well on all memory settings. So for both algorithms, we set $l=4$.

In Figure 14, we change c from 2 to 16 and find that it has little impact on the algorithm performance, since the line is almost horizontal, especially when the memory is relatively large.

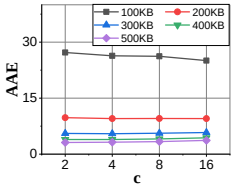
In Finding Top-k Frequent Items task, we consider MicroSketch-HG firstly. As shown in Figure 12, at first, the RR goes up, and ARE goes down when T or l goes up. This is due to a larger T giving the hopping window a more accurate approximation of the sliding window, and a larger l makes the rounding 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 to Figure 13. Using these parameters, MicroSketch-SS's RR reaches 97.4%, and its ARE goes below 0.026.

In Figure 15, 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.

In Finding Top-k Heavy Changes, We also have shown that the change of c has little impact on the RR of MicroSketch-HG in Figure 16.

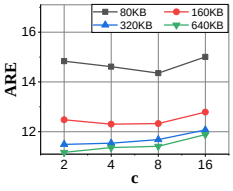


(a) MicroSketch-CM

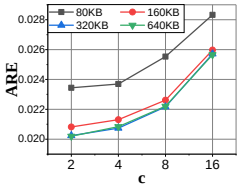


(b) MicroSketch-CU

Figure 14: Effect of c on frequency estimation.



(a) MicroSketch-HG



(b) MicroSketch-SS

Figure 15: Effect of c on top-k.

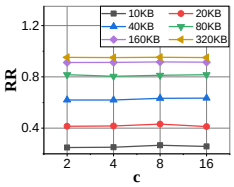


Figure 16: Effect of c on heavy change.