A Universal Sketch for Estimating Heavy Hitters and Per-Element Frequency Moments in Data Streams with Bounded Deletions

LIANG ZHENG, Southeast University, China
QINGJUN XIAO*, Southeast University, China and Purple Mountain Laboratories, China
XUYUAN CAI, The Hong Kong Polytechnic University, China

In the field of data stream processing, there are two prevalent models, i.e., insertion-only, and turnstile models. Most previous works were proposed for the insertion-only model, which assumes new elements arrive continuously as a stream, and neglects the possibilities of removing existing elements. In this paper, we make a *bounded deletion* assumption, putting a constraint on the number of deletions allowed. For such a turnstile stream, we focus on a new problem of *universal measurement* that estimates multiple kinds of statistical metrics simultaneously using limited memory and in an online fashion, including per-element frequency, heavy hitters, frequency moments, and frequency distribution. There are two key challenges for processing a turnstile stream with bounded deletions. Firstly, most previous methods for detecting heavy hitters cannot ensure a bounded detection error when there are deletion events. Secondly, there is still no prior work to estimate the per-element frequency moments under turnstile model, especially in an online fashion. In this paper, we address the former challenge by proposing a Removable Augmented Sketch, and address the latter by a Removable Universal Sketch, enhanced with an Online Moment Estimator. In addition, we improve the accuracy of frequency estimation by a compressed counter design, which can halve the memory cost of a frequency counter and support addition/minus operations. Our experiments show that our solution outperforms other algorithms by 16% $\sim$ 69% in F1 Score of heavy hitter detection, and improves the throughput of frequency moment estimation by 3.0×10^4 times.

CCS Concepts: • **Theory of computation → Sketching and sampling**.

Additional Key Words and Phrases: Data streams, Turnstile Model, Sketch, Universal Measurement

ACM Reference Format:
Liang Zheng, Qingjun Xiao, and Xuyuan Cai. 2024. A Universal Sketch for Estimating Heavy Hitters and Per-Element Frequency Moments in Data Streams with Bounded Deletions. *Proc. ACM Manag. Data* 2, 6 (SIGMOD), Article 224 (December 2024), 28 pages. https://doi.org/10.1145/3698799

1 INTRODUCTION

Background. In many big data scenarios, data arrives as a continuous stream and at a high speed, such as Internet traffic logs [1, 10, 15, 25, 39, 56, 73, 74, 83], sensor network readings [12, 32, 58, 80, 88], and social media messages [3, 11, 37, 46, 49, 51]. Extensive prior studies have been devoted to designing time-efficient one-pass algorithms for data streams. For example, in network traffic

*Qingjun Xiao is the corresponding author.

Authors' addresses: Liang Zheng, liangzheng@seu.edu.cn, Southeast University, School of Cyber Science and Engineering, Nanjing, China, 211189; Qingjun Xiao, csqjxiao@seu.edu.cn, Southeast University, School of Cyber Science and Engineering, Nanjing, China, 211189 and Purple Mountain Laboratories, Nanjing, China, 211111; Xuyuan Cai, xuyuan.cai@connect.polyu.hk, The Hong Kong Polytechnic University, Department of Computing, HongKong, China.

ACM 2836-6573/2024/12-ART224
https://doi.org/10.1145/3698799

measurement, AT&T collects terabytes of NetFlow [23] data from its production network every day. Numerous network flow logs constitute a data stream, which is processed by one pass to extract valuable statistical information for monitoring IP network services [22].

Universal Sketch. Many kinds of statistical metrics can be measured for a data stream. The first measurement task is to estimate the per-element frequency [14, 69, 73, 92], e.g., counting the number of packets originated from each source IP. The second task is to identify ε-heavy hitters, i.e., the elements whose frequencies exceed ε percent of the total frequency [9, 14, 39]. The third task is to measure the aggregated information of all elements, called the moments of element frequencies [18, 24, 70, 75]. For instance, the 0th-order moment is the number of distinct elements. The fourth task is to reconstruct the distribution of per-element frequencies, which can be achieved by fitting the generic moments [4, 21, 65, 86].

However, most existing algorithms are proposed to estimate only one type of the above statistics [9, 14, 17, 31, 52, 71, 91]. If each type of metric is measured by one dedicated algorithm, it will consume extensive memory and computation resources. Hence, *universal sketches* have been proposed to measure multiple metrics by one algorithm [39, 69, 73, 80]. They are developed based on a *recursive summation technique* to estimate generic moments [6]. This technique uses a two-phase framework: (a) In the online updating phase, the data stream is recursively sampled into multiple substreams by different probabilities $1, \frac{1}{2}, \frac{1}{4}, \ldots$. For each substream, a *subsketch* is utilized to capture the IDs of heavy hitters, and their frequencies. (b) In the offline estimation phase, the moment of each sampled substream is calculated recursively from the captured heavy hitters.

Bounded Deletion Stream. The previous universal sketches [39, 69, 80, 86] are designed to process an *insertion-only* stream. As a result, when deletion events arrive, they are unable to correspondingly update the multiple statistics. In real-world scenarios, there are two prevalent data stream models, namely the *insertion-only model* and the *turnstile model*. An *insertion-only* stream contains only insertion events, i.e., element IDs with frequency increments. A *turnstile* stream contains both insertion events and deletion events, i.e., frequency decrements of existing element IDs. For example, a proportion of NetFlow records may be deleted by network administrators, since they are collected during network attack campaigns.

In this paper, we focus on a new problem of designing a universal sketch that scans a *turnstile* stream by one pass and estimates the above statistics in an online manner. For the *turnstile* stream, a plausible assumption is that all the past insertions can be deleted. However, this incurs dramatically higher memory cost and longer update time than those of the *insertion-only* stream [7, 67, 87].

According to Jayaram et al. [27], for many *turnstile* data streams in practice, there are only a small fraction of element deletions. For example, the NetFlow logs deleted by network administrators due to their relation with network attacks only occupy a small fraction in a network log database. As a result, they proposed the *bounded deletion model* to ensure the number of deletions does not exceed $1 - \frac{1}{\zeta}$ fraction of the number of prior insertions, where ζ is a pre-configured constant no smaller than 1. When ζ is infinity, all past inserted elements can be deleted, and detecting heavy hitters needs significant memory cost linear to the massive number of elements. As ζ decreases and approaches 1, the memory cost can be greatly reduced [5, 27, 87, 89]. In this paper, we measure multiple kinds of statistics under the *bounded deletion model* and in an online manner.

Motivation. For generic measurement of a data stream, researchers have proposed the framework of universal sketch [39, 70]. As shown in Fig. 1, it consists of multiple components: substream sampler, heavy hitter (HH) detector for each substream, generic moment estimator, and the distribution fitter. Among them, the most important components are the HH detector and the moment estimator. These components face the following challenges when online processing a *bounded deletion* stream with millions or even billions of elements.

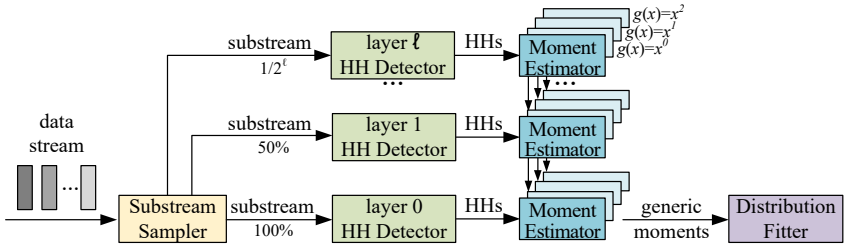

Fig. 1. Components of a general universal sketch

Challenge 1 (HH Detection in the Bounded Deletion Model). As shown in Fig. 1, a HH detector is to process a substream and identify the elements whose frequencies exceed a threshold. However, to detect the HHs in a bounded deletion substream, existing methods adopt a *simple structure* with a hash table to hold HHs' element-frequency pairs [45]. Therefore, when an insertion/deletion event arrives carrying an element ID untracked by the hash table, an existing element has to be chosen as a victim for replacement. This inevitably causes information loss, since the victim element has to be evicted from the table. So we design a *composite structure* combining the hash table with a backend data sketch to hold the non-HHs.

Challenge 2 (Improvement of HH Detection Accuracy). Traditionally, the backend data sketch uses a matrix of frequency counters, and each counter is implemented by a 32-bit integer. To improve the counters' memory efficiency, several recent studies propose a compression technique that implements each counter with 16-bit memory [80, 92]. This can double the number of counters in the sketch under the same memory budget, thereby improving HH detection accuracy. However, the previous counter designs are still inadequate. They either have a small counting range [80], e.g., $(-2^{15}, +2^{15})$, or lack support for the subtraction operation [69, 73], which is necessary for a turnstile stream with element deletions.

Challenge 3 (Online Moment Estimation). As shown in Fig. 1, for each substream, a HH detector is used to sample the heaviest elements. A moment estimator is also deployed to take a set of HHs as input, and estimate the moments of the frequencies of all elements in the substream. Previous work uses a *recursive summation technique* for moment estimation [6], which however has high computational cost and must be performed in an offline manner. This technique, as shown in Fig. 1, calculates the moment estimation by scanning the HHs of all substreams, starting from the ℓth sampling layer downwards to the 0th layer with 100% sampling probability.

Our Approach. In this paper, we propose a Removable Universal Sketch (RUS), which incorporates multiple improved designs.

To address **Challenge 1**, we design a *composite data structure* named Removable Augmented Sketch (RAS) that combines a pre-filtering *element-frequency table* and a backend sketch. The table holds both the IDs and the frequencies of heavy hitters. The backend sketch does not store the element IDs, but holds the frequencies of all elements in a compressed manner. The main advantage of this composite structure is that, when insertion or deletion events arrive carrying the elements untracked by the table, the backend sketch can update the frequencies of these untracked elements.

To address **Challenge 2**, we propose a novel compressed counter design, called the Removable Active Counter (RAC). RAC has a low memory cost with only 16 bits. It has a sign bit, an exponent part and a coefficient part, providing a wide counting range of $(-2^{27}, +2^{27})$. It outperforms other designs that halve the memory footprint at the cost of significantly narrowing the counting range [80]. Furthermore, our RAC supports both addition and subtraction operations.

To address **Challenge 3**, we propose the Online Moment Estimator (OME). It adapts the *recursive summation technique* [6], so that the generic moment estimation can be updated incrementally when the set of heavy hitters changes. Since the generic moments can be estimated online, it is possible to reconstruct the element frequency distribution online using the method of moments [4, 86].

In summary, we propose Removable Universal Sketch (RUS) to collect multiple statistics online in the bounded deletion model in Section 5. The key contributions of this paper are as follow.

- We propose a composite data structure named Removable Augmented Sketch (RAS) in Section 6, which can track Heavy Hitters (HHs) in the bounded deletion model with high accuracy.
- We propose a 16-bit Removable Active Counter (RAC) in Section 7, which supports both addition and subtraction operations, and has a wide counting range.
- We propose the Online Moment Estimator (OME) in Section 8, which allows us to update the generic moment estimation incrementally in an online manner.
- We conduct extensive experiments based on real-world datasets in Section 9. The results show that our sketch designs improve the F1 Score of HHs detection by 16% $\sim$ 69% than state-of-the-art, and increase the moment estimation speed by 3×10^4 times than UnivMon [39] and LUS [69, 73].

2 BACKGROUND

This section presents the data stream models and the problems we study. We summarize the notations commonly used in Table 1.

2.1 Data Stream Models

Formally, a data stream $S = \langle (e_1, w_1), (e_2, w_2), \ldots, (e_t, w_t) \rangle$ with a current length of t is a sequence of tuples, where each e_i indicates an element ID, and w_i represents the weight of e_i. Let E be the universe set of element IDs and we have $e_i \in E$. An element ID is allowed to appear multiple times in a data stream, which means the condition as $e_i = e_j$ with $i \neq j$ may occur. Depending on the domain of w_i, data streams can be classified into two major categories [47].

- **Insertion-only model** assumes all w_i are positive, which implies that the frequency of an element e_i can only increase and not decrease. This model is the most commonly used model.
- **Turnstile model** allows w_i to be positive or negative, for a tuple (e_i, w_i). It is an insertion if $w_i > 0$, and a deletion if $w_i < 0$. So the frequency of an element can both increase and decrease.

We assume a **bounded deletion model** with an upper-bounded $D : I$ ratio, which is the number of deletions D divided by the number of insertions I. More formally, $D : I \leq 1 - \frac{1}{\zeta}$, where ζ is a pre-configured constant no smaller than 1. This model definition is flexible. When $\zeta = \infty$, this model degrades to the *turnstile model* that allows all previously inserted elements to be deleted. When ζ is close to 1, the $D : I$ ratio is bounded, and the memory and time costs for detecting the ε-heavy hitters can be greatly reduced [5, 27, 87, 89].

2.2 Problem Definition

In this paper, we aim to measure multiple kinds of data stream statistics within a bounded deletion stream setting and in an online manner. We define these statistics as follows.

Per-Element Frequency. Let f_e be the frequency of an element e. When a tuple (e, w) arrives, we have $f_e = f_e + w$. Thus $f_e = \sum_{e_i = e} w_i$. Let $F = \sum_{e \in E} f_e$ be the total frequency of all elements in set E.

Frequency Moment. The g-moment of the stream is the functional sum of the frequency f_e for all the elements e with arrival tuples:

$$L = \sum_{e \in E} g(f_e), \tag{1}$$

where $g(x)$ is a monotonic function bounded by x^2. Typical definitions of the function g are as below.

• If $g(x) = x^0 = 1$, then L is called the 0th-order moment. It is equal to the cardinality, i.e., the number of distinct elements $|E|$.
• If $g(x) = x$, then L is called the 1st-order moment. It is equal to the total frequency of all elements, namely, L equals $\sum_{e \in E} f_e$.
• If $g(x) = x \log x$, then L is the entropy of the frequencies of all elements, indicating the diversity of the frequency distribution.
• If $g(x) = x^2$, then L is the 2nd-order moment of the frequencies of all elements, indicating the variance of the frequency distribution.

Heavy Hitters. A heavy hitter is an element $e \in E$, whose frequency f_e contributes at least ε fraction of the g-moment $L = \sum_{e \in E} g(f_e)$. We denote the set of all heavy hitters H as Eq. (2), where $0 < \varepsilon < 1$.

$$H = \{e \mid g(f_e) \geq \varepsilon L\} \tag{2}$$

We call H the first-order heavy hitters or $L1$ heavy hitters if $g(x) = x$, and second-order heavy hitters or $L2$ heavy hitters if $g(x) = x^2$.

Per-Element Frequency Distribution. The distribution considers the frequency of an element as a random variable. In the following paper, we refer to it as frequency distribution for brevity. The probability mass function of the distribution can be written as:

$$d(f^*) = \begin{cases} Pr(f_e = f^*), & f^* \in \mathcal{S}_f, \\ 0, & f^* \notin \mathcal{S}_f, \end{cases} \tag{3}$$

where $\mathcal{S}_f$ is the set of all possible per-element frequency values of a stream, and f^* is a random frequency value. $Pr(f_e = f^*)$ is the probability of an arbitrary element e to have a frequency value f^*.

Table 1. Symbols frequently used in this paper

Symbol	Description
S, S_j	Stream and substream sampled with probability $1 / 2^j$
(e, w)	A stream tuple with element ID e and weight w
f_e^{new}, f_e^{old}	The new frequency and old frequency of the element e
I, D	Number of insertion events and number of deletion events
ζ	The parameter to bound # deletions / # insertions (i.e., $D : I$)
L, L_j	Frequency moment of stream S and substream S_j
ε	The threshold to determine a heavy hitter
H, H_j	Set of all heavy hitters in stream S and substream S_j
C, ρ, α, β	Counter and its sign, exponent, and coefficient part
$Maxkicks$	The maximum number of kicks for KP-CF.

3 RELATED WORK

In this section, we review the related work on per-element frequency estimation, heavy hitter detection, and the universal sketch.

3.1 Per-Element Frequency Estimation

A data sketch is a technique to create compact summaries of large data streams [13]. It can be used to estimate per-element frequencies in a data stream, such as CountSketch (CS) [9], randomized error-reduction Sketch (rSkt) [63, 64], CountMin Sketch (CMS) [14], and Conservative Update Sketch (CUS) [17]. It uses hash functions to project the per-element frequency vector into a fixed-size structure with a sublinear memory cost to the number of distinct elements, allowing for efficient querying of an arbitrary element's frequency.

Most sketches allocate a matrix of 32-bit integers with d rows, where d is often set to 4. CMS (or CUS) can only provide an overestimation, since for each tuple (e, w), it updates by adding the weight w to all (or the minimum) of the hashed counters. By contrast, CS provides an unbiased estimation using the Tug-of-War technique [2]. Specifically, before adding w to the hashed counter in each of the d rows, CS multiplies w randomly by either +1 or −1. rSkt also offers unbiased estimation. To avoid multiplication, it uses two rows to implement the function of one row in CS.

Counter compression techniques boost sketch accuracy by allowing more counters to be allocated. Some of them involve altering the counting strategy. Diamond Sketch (DS) [81] uses 4-bit counters and utilizes a counting strategy similar to hexadecimal. FCM-Sketch (FCMS) [57] and TowerSketch [79] employ a counting strategy similar to DS but allocate different-sized counters in different rows. These strategies only work in the *insertion-only model*, thus DS allocates an extra CUS to record deletions. Others design counters for seamless integration into various sketches. The lossless encoding counter [80] uses integers in different sizes to represent numbers within various ranges, all of which are within -2^{15} to $+2^{15}$. Active Counter [44, 92] has a large counting range. ActiveCM [73] and GenericCM [73] use 16-bit active counters with a counting range of 0 to 2^{43} to enhance CMS. However, they do not support subtraction.

3.2 Heavy Hitter Detection

Existing approaches for identifying heavy hitters (HHs) can be categorized into counter-based and sketch-based algorithms [36].

Counter-Based Algorithms. These algorithms maintain an *element-frequency table* to hold the IDs and frequencies of the current HHs in a data stream. Canonical algorithms include Space-Saving (SS) [45], Unbiased SpaceSaving (USS) [61], SpaceSaving$^{\pm}$ (SS$^{\pm}$) [87], Lossy Counting [43], HeavyGuardian [82], HeavyKeeper [84], WavingSketch [34], and Cuckoo Filter along Kicking Path (KP-CF) [72, 74]. The KP-CF provides state-of-the-art update throughput and estimation accuracy. However, it only works in the *insertion-only* stream. Likewise, most existing methods, such as SS and USS, do not apply to the *bounded deletion model* due to Challenge 1 in Section 1.

SS$^{\pm}$ is an initial solution to handle bounded deletion streams [87]. When each tuple (e, w) arrives, e may not be cached by the table. If it is an insertion with $w > 0$, SS$^{\pm}$ adopts the same replacement strategy as SS. If it is a deletion with $w < 0$, SS$^{\pm}$ applies w to the element with the largest estimation error in the table. SS$^{\pm}$ proves that, if the capacity k of the table exceeds $\frac{\zeta}{\varepsilon}$, the ε-heavy hitter detection error can be bounded by a threshold with a high probability.

We give an example in Fig. 2, where an *element-frequency-error table* stores the elements e_1, e_2, and e_3. When an insertion event $(e_4, 2)$ arrives carrying an untracked element e_4, SS$^{\pm}$ finds the record $(e_1, 7, 0)$ with the smallest frequency. Then, the record becomes $(e_4, 9, 7)$, as its element ID is replaced by e_4, its frequency is increased by 2, and the estimation error grows to 7 by the replacement. Next, when a deletion event $(e_1, -1)$ arrives carrying an untracked element e_1, SS$^{\pm}$ finds the record $(e_4, 9, 7)$ with the largest error, and reduces its frequency and error by 1, resulting in $(e_4, 8, 6)$.

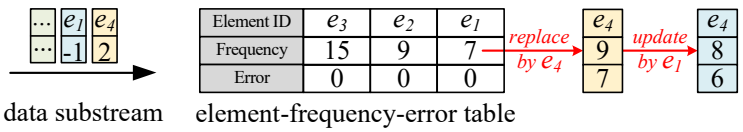

Fig. 2. Example of handling insertion and deletion events

However, the previous works, including SS$^{\pm}$ sketch [87], have suboptimal estimation accuracy of per-element frequency, which in turn greatly reduces the identification accuracy of HHs. The

main cause is that SS$^{\pm}$ adopts a simple data structure design with *one single element-frequency table* as shown in Fig. 2. Therefore, when an insertion or deletion event arrives carrying an element ID untracked by the table, it has to choose an existing element for replacement, causing accuracy loss inevitably. For example, in Fig. 2, the frequency of element e_4 is 2, but is overestimated as 8.

Sketch-Based Algorithms. These algorithms maintain a frequency counting sketch, as well as a HH sampling structure guided by the sketch. Traditionally, a min-heap is placed after the sketch to hold the current top-k HHs [73]. We call it a postfilter. However, this solution does not fully utilize the highly skewed distribution of element frequencies, thus resulting in suboptimal accuracy. Recent works propose to separate HHs from normal elements by a prefilter-based strategy [52, 69, 76]. These algorithms maintain a pre-filtering *element-frequency table* to track current HHs, and a backend sketch to hold the evicted non-HHs, which greatly improves the HH detection accuracy by gracefully swapping out the non-HHs.

This strategy was first proposed by Augmented Sketch (AS) [52]. AS implements the prefilter by a min-heap, with $O(k)$ lookup time cost when holding the top-k HHs. However, for a turnstile stream, the size of the prefilter might need to be set to tens of thousands to ensure accuracy. Several follow-up works propose to improve the throughput by hash-based prefilter structures with $O(1)$ time cost, such as Cold filter [91], ElasticSketch [83], MV-Sketch [60], and OneSketch [20]. However, they only work in *insertion-only streams*.

3.3 Universal Sketch

Universal sketch can estimate multiple kinds of statistical metrics simultaneously, such as per-element frequency, heavy hitters, and generic moments. UnivMon [39], Light-weight Universal Sketch (LUS) [73], Augmented LUS (ALUS) [69], and Joltik [80] can natively support generic moment estimation, which relies on the *recursive summation technique* [6]. We neglect ElasticSketch [83], FCMS [57], OneSketch [20], and Panakos [88], which can only estimate lower-order moments (e.g., cardinality and entropy) but not higher-order moments [30, 66]. Higher-order moments are essential for precisely recovering the per-element frequency distribution [21, 28, 53].

UnivMon is the first universal sketch. It works in the *insertion-only model* and uses the *hierarchical sampling* method to favor the heavy hitters in the long tail. Follow-up studies propose better moment estimators based on different optimization techniques. Both LUS [73] and Joltik [80] replace the *hierarchical sampling* method with the *progressive sampling technique*. This technique dramatically improves the time efficiency and estimation accuracy. To reduce the memory footprint, the previously mentioned ActiveCM [73] and lossless encoding counter [80] are employed in LUS and Joltik, respectively. ALUS utilizes a prefilter [52] to reduce the estimation error of LUS. Overall, the universal sketch algorithms have achieved significant progress. However, none of these algorithms can estimate moments online in the *bounded deletion model*.

In summary, we compare our work with existing methods in Table 2. It shows that our work has two key advantages: the ability to handle deletion events, and online estimation of generic moments.

4 PRELIMINARY

In this section, we first give an introduction to the universal sketch. Then, we present the design of KP-CF to track heavy hitters (HHs).

4.1 Preliminary about Universal Sketch

The universal sketch is constructed as a layered structure, and employs the two-phase framework from *recursive summation* [6] to estimate moments. We take UnivMon as an example to illustrate the layered structure and the two-phase framework. Similar to Fig. 1, we assume UnivMon has

Table 2. Comparison of existing methods with our solutions

Task	Method	Accuracy	Throughput	Deletion Support	Moment Estimation
Per-Element Frequency Estimation	Traditional Sketches	○	●	●	–
	w. Compressed Counter	●	○	○	–
	Sketches with Our RAC	●	◐	●	–
Heavy Hitter Detection	Counter-Based	○	●	◐	–
	Sketch-Based	●	○	○	–
	Our RAS	●	●	●	–
Moment Estimation	UnivMon Inspired	◐	◐	○	Offline
	Our RUS with OME	●	●	●	Online

●, ◐, and ○ represent perform well, moderately, and poorly, respectively.

multiple layers, whose indices range from 0 to ℓ. In each of the $\ell + 1$ layers, a CS combined with a postfilter is allocated to track heavy hitters, i.e., the *HH Detector* in Fig. 1, and we denote the combination in each jth layer as CS_j for simplicity.

Online Updating Phase. For an arrival tuple (e, w), the *hierarchical sampling* is performed, with a sampling probability $\frac{1}{2^j}$ assigned for the jth layer (as depicted in Fig. 1). Specifically, the element e is initially sampled at the 0th layer with a probability of 100%, then at the 1st layer with a probability of 50%, and the procedure continues until sampling fails at a particular layer. To clarify, the element is sampled for all subsequent layers from the 0th layer up to the topmost sampled layer $j_t(e)$, as determined by Eq. (4).

$$j_t(e) = \min\left(\ell,\ \arg\max_j \left\{ \bigwedge\nolimits_{0\le i\le j} [1|h(e)]_i = 1 \right\}\right), \tag{4}$$

where $h(e)$ represents the binary representation of e's hash, and $[1|h(e)]_i$ is the leftmost ith bit of the concatenation of 1 with $h(e)$. Hence, the expression $\wedge_{0\le i\le j}[1|h(e)]_i = 1$ implies that the leftmost j bits of $[1|h(e)]$ are all 1s. Since each bit of $h(e)$ has 50% chance of being 1, the sampling probability reduces by half with each layer, and it is 100% for the 0th layer as $[1|h(e)]_0$ is always 1.

After that, among the sampled substream S_j on the jth layer, $j \in [0, j_t(e)]$, the sketch CS_j identifies the heavy hitters H_j in Eq. (2).

Offline Estimation Phase. Firstly, the g-moment L_ℓ of the substream S_ℓ sampled for the highest layer ℓ is estimated by

$$\hat{L_\ell} = \sum\nolimits_{e\in\hat{H_\ell}} g(\hat{f_e}). \tag{5}$$

Then, from layer $\ell-1$ to 0, the *recursive summation technique* [6] defined in Eq. (6) is used to obtain the g-moment of the substream S_0:

$$\hat{L_j} = 2\,\hat{L}_{j+1} + \sum\nolimits_{e\in\hat{H_j}} \left(1 - 2\cdot \mathbf{1}_{j+1\le j_t(e)}\right)\cdot g(\hat{f_e}), \tag{6}$$

where $\mathbf{1}_{j+1\le j_t(e)}$ is an indicator function that returns 1 if the element e is sampled for the layer $j+1$, and 0 otherwise.

4.2 Preliminary about KP-CF

Cuckoo Filter along Kicking Path (KP-CF) consists of an array of buckets, each of which contains s slots and forms a bucket-level min-heap. The bucket-level min-heap is used to record the IDs and frequencies of heavy hitters (HHs), and ensures that the minimum frequency element within the min-heap is maintained at the root node. We give an example in Fig. 3 to explain its design.

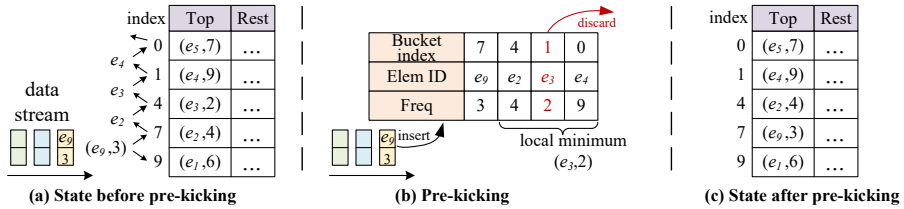

Fig. 3. An example of inserting a new element into KP-CF

State before Pre-Kicking. In Fig. 3(a), we only show the element at the root node of the bucket-level min-heap in the "Top" column to ease the presentation. For each arrival data stream tuple, it is hashed to two candidate buckets by Cuckoo hash [74]. For example, the tuple $(e_9, 3)$ is hashed to buckets 7 and 9. If both of the candidate buckets are full, the KP-CF chooses bucket 7 to kick, which contains the record $(e_2, 4)$ with the minimum frequency 4 in the two candidate buckets. So, as is illustrated in Fig. 3(a), the record $(e_2, 4)$ in bucket 7 is to be kicked to bucket 4, and the record $(e_3, 2)$ in bucket 4 is to be kicked to bucket 1, and so on. The procedure continues until an empty slot is found or a predefined number of kicks, which we call $Maxkicks$, is reached. In Fig. 3(a), we assume the $Maxkicks$ is 4 and all the 4 kicks encounter full buckets. At the end of the procedure, we can select the last kicked element, the element e_5 whose frequency is 7 in Fig. 3(a), as the victim element to be discarded from the KP-CF. However, the element e_3 has the lowest frequency among the set of all the kicked elements $\{e_2, e_3, e_4, e_5\}$. If we choose e_3 as the victim, we can achieve the best maintenance of the HHs under the constraints of the $MaxKicks$.

Pre-Kicking. This procedure aims to identify the aforementioned victim element, namely, the kicked element with the lowest frequency on the kick-out path of a cuckoo hash table. So KP-CF uses a pre-kicking queue in Fig. 3(b) to record the information of kicked elements in each round in the form of 3-tuples. For example, the arrival tuple $(e_9, 3)$, which is to be inserted into the bucket 7, is recorded by $(7, e_9, 3)$. The record $(e_2, 4)$, which is currently in bucket 7 and is to be kicked to the bucket 4, is recorded by $(4, e_2, 4)$. After that, KP-CF chooses the local minimum element in the pre-kicking queue as the victim to discard, which is e_3 in Fig. 3(b).

State after Pre-Kicking. In Fig. 3(c), based on the pre-kicking queue, KP-CF inserts $(e_9, 3)$ into the top slot of bucket 7, $(e_2, 4)$ into the top slot of bucket 4, and adjusts the min-heaps of these buckets.

5 REMOVABLE UNIVERSAL SKETCH

We first describe the structure and workflow of Removable Universal Sketch (RUS), followed by an introduction to its applications.

5.1 Overview of RUS

As depicted in Fig. 4(a), RUS is facilitated by four sequentially interconnected modules. Each module in Fig. 4(a) processes the output from its predecessor, beginning with the *Substream Sampler*, which receives a tuple (e, w) consisting of an element e and a weight w from the bounded deletion stream. The tuple represents an insertion event when w is positive and a deletion event when w is negative.

Substream Sampler. As shown in Fig. 4(a), the *HH Detector* in RUS is constructed as a layered structure, and we employ the *Substream Sampler* to sample arrival stream tuples to different layers, forming substreams for each layer. Specifically, for each arrival tuple (e, w), the sampler uses Eq. (4) to calculate the topmost sampled layer of the element e (e.g., layer 5 in Fig. 4), and delivers the tuple (e, w) to the *HH Detector* at this layer, indicating that e is sampled to the layers from the 0th layer to the topmost sampled layer.

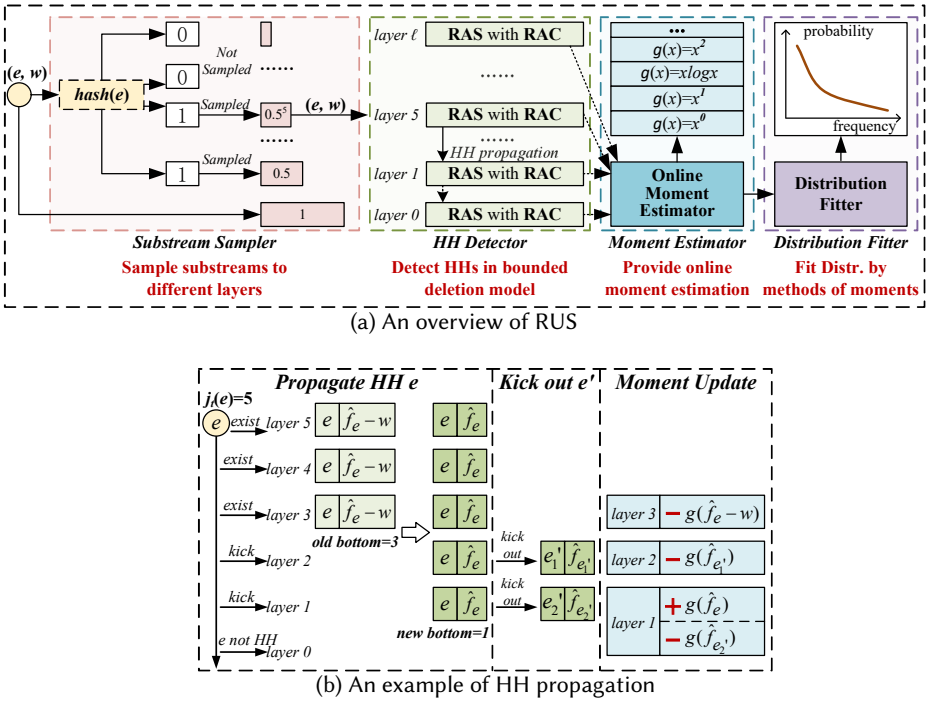

Fig. 4. An overview of RUS and an example of HH propagation

HH Detector. As shown in Fig. 4(a), there are $\ell + 1$ layers in the *HH Detector*, where each layer is used to identify the heavy hitters from the substream sampled to this layer. For example, in Fig. 4(a), the topmost sampled layer of the element e is layer 5, so the element e is in the substreams sampled for layer 0 to layer 5, and the *HH Detector* has to identify whether the element e is a heavy hitter within the substreams sampled for these layers. To achieve this, firstly, the 5th-layer of *HH Detector* identifies whether the element e is a heavy hitter at this layer. Then, if yes, the *HH Detector* propagates the element e with its frequency $\hat{f}_e$ estimated from layer 5 to lower layers, to see if the element e is also a heavy hitter at these layers. We detail the two procedures as below.

1) HH Identification. We propose the Removable Augmented Sketch (RAS), detailed in **Section 6**, which functions as a layer within the multi-layered *HH Detector* to identify HHs from a substream sampled from the *Substream Sampler* module. In our design, RAS comprises an *element-frequency table* and a CountSketch (CS), as shown in Fig. 2(b). For the *element-frequency table*, we design it based on the KP-CF to improve query efficiency. For the CS, we substitute the conventional 32-bit integer counter with our proposed 16-bit active counter, namely the Removable Active Counter (RAC), to double the number of allocated counters and thereby enhance the frequency estimation accuracy [73, 74]. In addition to the common addition operation, RAC supports subtraction operation to handle deletions in the *bounded deletion model* (see **Section 7** for details).

2) HH Propagation. Fig. 4(b) shows an example of *HH propagation*. We assume that, the element e has been identified as a heavy hitter at layers 3 to 5, with frequency estimation $\hat{f}_e - w$. Consider an arrival tuple (e, w), where w is positive, and $j_t(e) = 5$ denotes the topmost sampled layer of element e. We first update the frequency of the element e recorded at layer 5 from $\hat{f}_e - w$ to $\hat{f}_e$.

Then, we propagate $(e, \hat{f}_e)$ to lower layers. For layers 4 and 3, we directly update the frequency of e from $\hat{f}_e - w$ to $\hat{f}_e$ as e is already recorded as a heavy hitter at these layers. For lower layers, if the *element-frequency table* used to record heavy hitters is full, we check if $\hat{f}_e$ is greater than any of the heavy hitters at these layers. Subsequently, we find the frequency of the elements e'_1 and e'_2 is lower than $\hat{f}_e$ at layers 2 and 1, respectively. So we kick out the two elements from the *table* to make room for e. For layer 0, we do nothing since $\hat{f}_e$ is not larger than any of the frequencies of recorded heavy hitters.

Online Moment Estimator. After obtaining HHs from each of the $\ell+1$ layers of the *HH Detector*, the *recursive summation technique* shown in Eq. (6) can be used to estimate generic moments. However, as introduced in Section 4.1, this technique is limited to offline mode due to its requirement for a recursive scan of all HHs from the ℓth-layer down to the 0th-layer. So we propose an Online Moment Estimator (OME) in **Section 8** to update the moment estimation as soon as the set of HHs changes, i.e., the *HH Detector* finds a new HH, or finds that a HH element no longer qualifies as a HH.

Distribution Fitter [86]. The frequency moments can be utilized to reconstruct the frequency distribution in real time by employing the distribution fitter based on the method of moments and the cut-then-rejoin strategy. For further details, please refer to [86].

5.2 Applications of RUS

Network Measurement. Network measurement utilizes only the size-limited SRAM (usually in MBs) on their line cards to measure multiple kinds of statistics, including per-packet frequency, heavy hitters, network flow cardinality, network flow frequency distribution and its entropy [69]. In addition, network administrators may delete the network flow records relating to network attacks, which only account for a small fraction of all the records collected. So, our RUS can be used to monitor and manage IP network services [22].

Database Query Optimization. Query optimization aims to identify an execution plan for a query statement that minimizes intermediate results. To achieve this, DBMS has to collect basic statistics for each column, such as per-value frequency (i.e., per-element frequency), the number of distinct values (i.e., the 0th-order moment), the top frequency histogram (i.e., HHs), and quantile information (i.e., frequency distribution), while managing record insertions and partial deletions in high speed [26]. Considering the database's size, there are limitations on the memory resources to retrieve these statistics, e.g., memory in MBs in the Join Order Benchmark [16]. So our methods show promise for application in query optimization.

N-Gram Mining. N-gram mining is extensively employed in natural language processing applications [37, 41, 59], such as sequence embedding [40, 48] and sequence classification [54]. A significant challenge in n-gram mining is the combinatorial explosion of token combinations. For instance, a corpus with only 300 distinct words can yield up to 90,000 distinct bigrams. Consequently, the memory footprint required to exactly record the statistics of these n-grams could be substantial. Additionally, the enforcement of data protection laws and the right to be forgotten [38, 50, 62] require the removal of specific information from the corpus in the trained models or collected statistics, highlighting the necessity for deletion support. Our RUS meets the needs of this scenario.

6 REMOVABLE AUGMENTED SKETCH

In this section, we propose a Removable Augmented Sketch (RAS) to track heavy hitters (HHs) under the *bounded deletion model*.

6.1 Design Rationale

For the problem of detecting ε-heavy hitters in the bounded deletion model, a naïve solution is to use a min-heap to hold the top-k heavy hitters. However, this method is impractical due to its low element processing throughput. Whenever an element arrives, we need to search the element ID in the min-heap, which involves a linear scan with $O(k)$ time cost. The number k of HHs that need to be held in the min-heap, under the bounded deletion model, can be very large, e.g., $k = 8192$ when $\zeta = 12$ and $\varepsilon = 2^{-13}$, as shown in Section 6.4.

An improved solution is to additionally maintain a hashtable to reduce the key lookup time complexity to $O(1)$, like SpaceSaving$^{\pm}$ (SS$^{\pm}$) [45, 87]. However, in addition to the extra memory cost for the hashtable, the heap maintenance is still time-consuming. Essentially, the min-heap is a sorting data structure for quickly locating the *global minimum element* at the root. When updating the frequency of an existing element or inserting a new element, it needs to restore the min-heap property, involving $O(\log k)$ sift up/down operations.

Thus, we embrace the design of the KP-CF [74], which searches only the local minimum along the kicking path and uses it as the victim to make room for the new element. KP-CF is memory efficient as its load factor can be up to 95% [19], and it has both $O(1)$ lookup and insertion time costs, the same as the Cuckoo Filter. However, its shortcoming is to kick only the path-level local minimum element for replacement. We address this by using the KP-CF as the prefilter and using a backend sketch to store the kicked victim elements.

6.2 Overview of RAS

As shown in Fig. 5, our RAS has two main components: a prefilter and a sketch. The prefilter consists of two parts: a Cuckoo Filter along Kicking Path (KP-CF) and a tiny histogram. The KP-CF serves as the function of the table in Fig. 2(b), which keeps the IDs and frequencies of the HHs. The histogram is used to track the minimum frequency of the HHs, since KP-CF cannot be queried for the minimum frequency of the HHs as the min-heap does.

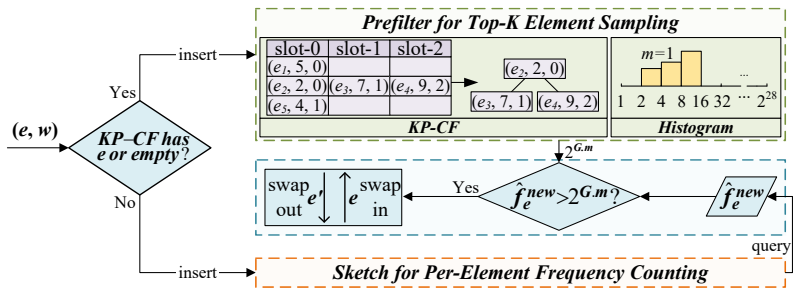

Fig. 5. Structure and workflow of our RAS

We explain the necessity of the histogram through an example. Suppose we use the KP-CF in Fig. 3 as a prefilter to track HHs. If an insertion event $(e_{10}, 1)$ arrives carrying an untracked element and a weight lower than the frequencies of all the tracked elements, the element e_{10} is not going to be recorded by the KP-CF. However, the pre-kicking will be triggered blindly as the KP-CF does not maintain the global minimum frequency of the HHs. This blind attempt will cause significant computational waste, as most elements (e.g., e_{10}) are not HHs. Thus, we use a tiny histogram to track the *approximate global minimum frequency* of the HHs. We insert an element into the KP-CF only if its frequency exceeds this *minimum frequency*.

KP-CF Overview. As shown in Fig. 5, the KP-CF we use is almost the same as the version shown in Fig. 3, except that each slot maintains three attributes of an element: (id, newFreq, oldFreq). The newFreq of an element denotes its current frequency. The oldFreq of an element is the frequency accumulated during its residence in the sketch before being inserted into the KP-CF, which is 0 if the element has never resided in the sketch. The difference between oldFreq and newFreq of an element is the exact aggregated frequency that is accumulated during its residence in the KP-CF [52].

Histogram Overview. As shown in Fig. 5, the histogram is an array, and its ith entry keeps the number of HHs whose frequency falls in the range of $[2^i, 2^{i+1})$, $0 \le i \le 27$. We set up an indicator m to record the index of the first non-zero entry. So 2^m represents the *approximate global minimum* (*approx-min* for short) frequency of the elements in the KP-CF. The histogram has three operations:

- **Insert**: If a new element e with frequency f is inserted into the KP-CF, we calculate $i = \lfloor \log_2(f) \rfloor$ and increase the ith entry of the histogram by 1. Then, we assign i to m if i is less than m.
- **Remove**: If an old element e with frequency f is removed from the KP-CF, we decrease the ith entry of the histogram by 1, where $i = \lfloor \log_2(f) \rfloor$. Then, if m equals i and the ith entry becomes 0, we update m to the index of the next non-zero entry.
- **Update**: If an existing element e in the KP-CF has its frequency updated, this operation can be seen as one remove and one insert.

With this histogram, we can easily decide whether to insert an element into the KP-CF by comparing its frequency with 2^m.

6.3 Algorithm for Updating RAS

We present the update procedure of RAS in Algorithm 1. At Line 1, we use two hash functions, $h_1(\cdot)$ and $h_2(\cdot)$, to map elements to candidate buckets. We then describe the algorithm in four phases.

Lookup *e*. When a stream tuple (e, w) arrives, at Line 2, we call `lookupFilter`(R, e) to check if the element e is present in the KP-CF $R.B$ or if any of e's candidate buckets have an empty slot.

Update when *e* Exists. At Line 3, if the index u and v are not **NIL**, and the *id* in the slot $R.B[u][v]$ is e, we confirm that e is present in the slot $R.B[u][v]$. Consequently, to update e's newFreq, we call `updateFilter`$(R, u, v, \hat{f}_e^{new}{=}w{+}R.B[u][v].newFreq)$ at Line 4. Finally, the procedure ends as the condition at Line 9 does not hold.

Update when an Empty Slot is Found. If Line 5 holds, e's candidate buckets have at least one empty slot. So we insert the element e into the empty slot by `insertFilter`$(R, e, \hat{f}_e^{new}{=}w, \hat{f}_e^{old}{=}0)$. At Line 20, we initialize the pre-kicking queue $R.Q$. Then, as we have confirmed that an empty slot is in one of e's candidate buckets, no more elements will be inserted into the queue $R.Q$ at Line 21. Thus, at Line 25, the element e is inserted into an empty slot, and the histogram $R.G$ is updated accordingly. Next, as the condition at Line 26 holds, we return $\langle \textbf{NIL}, 0, 0 \rangle$ to indicate that the insertion succeeds and no element is kicked out. Finally, the procedure ends.

Update after Querying Sketch. In this case, at Line 7, we first insert the tuple (e, w) into the sketch M and query for e's current frequency $\hat{f}_e^{new}$ from it. Then, if $\hat{f}_e^{new}$ does not exceed the *approx-min* $2^{R.G.m}$ recorded by the histogram $R.G$, the procedure ends as the frequency of item e is not greater than that of any item in the KP-CF. Otherwise, we try to insert the element e into the KP-CF $R.B$ by `insertFilter`$(R, e, \hat{f}_e^{new}, \hat{f}_e^{old}{=}\hat{f}_e^{new})$ at Line 8. At Line 21, after initializing the pre-kicking queue at Line 20, we fill the queue until the number of kicks exceeds *Maxkicks* or an empty slot is recorded.

If no empty slot is recorded at Line 21, the condition at Line 22 holds, and the limit *Maxkicks* is reached. At Line 23, we find the element e' with the minimum frequency $\hat{f}_{e'}^{new}$ in the queue $R.Q$. If the minimum frequency element e' is e, we return $\langle e, \hat{f}_e^{new}, \hat{f}_e^{old} \rangle$ at Line 24 to indicate that the

Algorithm 1: Update procedure of RAS

Input: an arrival tuple consisting of element ID e and weight w
State: prefilter R with KP-CF B and histogram G, sketch M

1 $h_1(e) = hash(e),\ h_2(e) = h_1(e) \oplus hash(e, salt)$
2 $\langle u, v\rangle = \texttt{lookupFilter}(R, e), \quad \langle e', \hat{f}_{e'}^{new}, \hat{f}_{e'}^{old}\rangle = \langle \textbf{NIL}, 0, 0\rangle$
3 **if** $u \neq \textbf{NIL} \wedge v \neq \textbf{NIL} \wedge R.B[u][v].id == e$ **then** // e exists
4 $\quad \texttt{updateFilter}(R, u, v, \hat{f}_e^{new} = w + R.B[u][v].newFreq)$
5 **else if** $u \neq \textbf{NIL} \wedge v \neq \textbf{NIL}$ **then** // an empty slot exists
6 $\quad \langle e', \hat{f}_{e'}^{new}, \hat{f}_{e'}^{old}\rangle = \texttt{insertFilter}(R, e, \hat{f}_e^{new} = w, \hat{f}_e^{old} = 0)$
7 **else if** $(\hat{f}_e^{new} = \texttt{updateSketch}(M, e, w)) > 2^{R.G.m}$ **then** // there is no e or empty slot, but e's freq. $\hat{f}_e^{new}$> minimum
8 $\quad \langle e', \hat{f}_{e'}^{new}, \hat{f}_{e'}^{old}\rangle = \texttt{insertFilter}(R, e, \hat{f}_e^{new}, \hat{f}_e^{old} = \hat{f}_e^{new})$
9 **if** $e' \neq \textbf{NIL}$ **then** $\texttt{updateSketch}(M, e', \hat{f}_{e'}^{new} - \hat{f}_{e'}^{old})$ // kick e'
10 **Function** $\texttt{lookupFilter}(R, e)$**:** // prefilter, element
11 $\quad$ **foreach** $u \in \{h_1(e), h_2(e)\}, v \in [0, s)$ **do**
12 $\quad\quad$ **if** $R.B[u][v].id == e$ **then return** $\langle u, v\rangle$
13 $\quad$ **if** $R.B[h_1(e)][0].id == \textbf{NIL}$ **then return** $\langle h_1(e), 0\rangle$
14 $\quad$ **if** $R.B[h_2(e)][0].id == \textbf{NIL}$ **then return** $\langle h_2(e), 0\rangle$
15 $\quad$ **return** $\langle \textbf{NIL}, \textbf{NIL}\rangle$
16 **Function** $\texttt{updateFilter}(R, u, v, \hat{f}_e^{new})$**:** // bkt idx,slot idx, freq.
17 $\quad$ update the histogram $R.G$ and $R.B[u][v].newFreq = \hat{f}_e^{new}$
18 $\quad$ restore the min-heap property of the bucket $R.B[u]$
19 **Function** $\texttt{insertFilter}(R, e, \hat{f}_e^{new}, \hat{f}_e^{old})$**:** // id, new&old freq.
20 $\quad$ initialize $R.Q$ with $R.Q[0]$ recording $\langle e, \hat{f}_e^{new}, \hat{f}_e^{old}\rangle$
21 $\quad$ fill pre-kicking queue $R.Q$ until an empty slot is recorded by $R.Q$ or the number of attempts reaches $Maxkicks$
22 $\quad$ **if** *no empty slot is recorded by* $R.Q$ **then**
23 $\quad\quad$ find the minimum record $\langle e', \hat{f}_{e'}^{new}, \hat{f}_{e'}^{old}\rangle$ in queue $R.Q$
24 $\quad\quad$ **if** $e' == e$ **then return** $\langle e, \hat{f}_e^{new}, \hat{f}_e^{old}\rangle$ // insert failed
25 $\quad$ update the histogram $R.G$ and update KP-CF $R.B$ by $R.Q$
26 $\quad$ **if** *an empty slot is recorded by* $R.Q$ **then return** $\langle \textbf{NIL}, 0, 0\rangle$
27 $\quad$ **else return** $\langle e', \hat{f}_{e'}^{new}, \hat{f}_{e'}^{old}\rangle$ // old element replaced by e

insertion fails. Otherwise, if e' is not e, the element e can be inserted into the KP-CF $R.B$, so we update the histogram $R.G$ at Line 25 and update the KP-CF $R.B$ by the pre-kicking queue $R.Q$ as illustrated in Fig. 3(c). Finally, at Line 27, we return the information $\langle e', \hat{f}_{e'}^{new}, \hat{f}_{e'}^{old}\rangle$ on the kicked element e', in order to swap out the element e' to the sketch M at Line 9.

If an empty slot is recorded at Line 21, the condition at Line 22 does not hold. At Line 25, we update the histogram $R.G$ and update the KP-CF $R.B$ by $R.Q$, and return $\langle \textbf{NIL}, 0, 0\rangle$ at Line 26 to indicate that the insertion succeeds and no element is kicked out. Then, the procedure ends as the condition at Line 9 does not hold.

6.4 Analysis of Size Lower Bound for Prefilter

Theorem 1 shows the lower bound of the prefilter size k to detect ε-HHs in the bounded deletion model. Its formal proof is omitted in this paper and can be found in our technical report [90]. Here, we provide only the outline of the proof. Let η be the skewness of element frequency distribution. We have $\eta \geq 1$ in most scenarios [29, 52], and $\eta = 1$ indicates low skewness. Under this assumption, we proved in [90] that the prefilter size can be set to $\frac{\zeta}{12\varepsilon}$. The error bounds for HH detection and frequency estimation are also given.

Theorem 1 (Prefilter capacity lower bound). *Assume that a bounded deletion stream has N distinct elements with $D:I$ ratio upper bounded by $1-\frac{1}{\zeta}$, and has its per-element frequency following a Zipf distribution with skewness η. To detect the ε-HHs in this data stream, the capacity k of the prefilter must be at least $\left\lfloor \sqrt[\eta]{\zeta/\varepsilon \sum_{i=1}^{N}(1/i)^{\eta}} \right\rfloor$.*

Proof outline. In the bounded deletion stream, after all I insertions and D deletions, the frequency of each ε-HH is at least $\frac{\varepsilon}{\zeta}I$, given that $\varepsilon(I-D) = \frac{\varepsilon}{\zeta}I$. Thus, the prefilter must retain all elements with a frequency exceeding $\frac{\varepsilon}{\zeta}I$ before deletions occur. Otherwise, if such an element is not retained before deletions occur, it may become an untracked HH afterward. Therefore, the lower bound is the maximum value of k satisfying $f(k;\eta,N) \geq \frac{\varepsilon}{\zeta}$, where $f(i;\eta,N) = 1/\left(i^{\eta}\sum_{j=1}^{N} j^{-\eta}\right)$ is the probability mass function of Zipf distribution. By solving $f(k;\eta,N) \geq \frac{\varepsilon}{\zeta}$ for k, we have Theorem 1.

7 REMOVABLE ACTIVE COUNTER

We present the Removable Active Counter (RAC), which supports not only the common addition operation but also the subtraction operation to accommodate deletions in the bounded deletion model. We first show the internal representation of RAC, followed by an example to explain its subtraction operation. Finally, we discuss an optimization to improve the processing speed when using RAC. For the pseudocode of the addition and subtraction operations, as well as mathematical analysis, please refer to our technical report [90].

7.1 Internal Representation of RAC

An RAC is a compressed 16-bit representation of a signed integer. The internal structure of an RAC C consists of three parts:

- The **sign bit** $C.\boldsymbol{\rho}$ is the leading bit;
- The **exponent part** $C.\boldsymbol{\alpha}$ occupies the subsequent $\mathcal{L}_\alpha = 4$ bits;
- The **coefficient part** $C.\boldsymbol{\beta}$ contains the remaining $\mathcal{L}_\beta = 11$ bits.

To further save memory, for the coefficient $C.\beta$, we use a leading-one convention, which assumes that its leftmost bit is always 1 and is not stored explicitly. We denote the coefficient under the leading-one convention as $\tilde{\beta} = 2^{\mathcal{L}_\beta} + C.\beta$, where $\mathcal{L}_\beta$ is the length of the coefficient part, defaulting to 11. To find the represented integer value of an RAC C, the evaluation function V can be used:

$$V(C) = (2\,C.\rho - 1) \cdot \left((2^{\mathcal{L}_\beta} + C.\beta) \cdot 2^{C.\alpha} - 2^{\mathcal{L}_\beta}\right), \tag{7}$$

where $(2\,C.\rho - 1)$ converts the sign bit $C.\rho$ to ± 1. The initial values of $C.\alpha$, $C.\beta$ and $\tilde{\beta}$ are 0, 0 and $2^{\mathcal{L}_\beta}$, respectively, since $\tilde{\beta}$ carries an implicit leftmost 1 bit ahead of $C.\beta$. The last term $-2^{\mathcal{L}_\beta}$ provides a bias correction to ensure the initial value of $V(C)$ is zero. The maximum values of $C.\alpha$ and $\tilde{\beta}$ are $2^4 - 1 = 15$ and $2^{12} - 1 = 4095$, respectively. Thus, the counting range of our RAC is within $\pm(4095 \cdot 2^{15} - 2^{11})$, which is roughly $\pm 2^{27}$, sufficient for most practical applications.

7.2 Example of Subtraction Operation in RAC

Fig. 6 shows an example to subtract $|w|$ from an RAC C. Assume an RAC C with $\rho = 1b$ (denoted by +), $\alpha = 001b$, $\tilde{\beta} = ①001b$, and $\mathcal{L}_\beta = 3$, where ① represents the implicit leftmost 1 bit. The weight w in our example is $-16 = -1 \cdot 2^{001b} \cdot 1000b$. Since the counter value $V(C)$ has been multiplied by $2^{C.\alpha}$ in Eq. (7), we need to divide the increment w by $2^{C.\alpha}$, so that it can be added to the coefficient $\tilde{\beta}$. We denote the division result as $\Delta\tilde{\beta}$, yielding $-1 \cdot 1000b$ in our example in Fig. 6. We show the example in three phases.

- **Obtain new $\tilde{\beta}$**. As shown in Fig. 6, we obtain the new value of $\tilde{\beta}$ by summing $\Delta\tilde{\beta}$ with $\tilde{\beta}$, resulting in $-1 \cdot 1000b + ①001b = ⓪001b$.
- **Update ρ and $\tilde{\beta}$**. After the update of $\tilde{\beta}$, the value of $V(C)$ becomes negative as $\tilde{\beta} \cdot 2^{C.\alpha} - 2^{\mathcal{L}_\beta}$ falls below 0. However, $2C.\rho - 1$ is still +1, contradicting the representation of RAC, which assumes that the term $\tilde{\beta} \cdot 2^{C.\alpha} - 2^{\mathcal{L}_\beta}$ is non-negative, and the sign of $V(C)$ matches $2C.\rho - 1$. So, in Fig. 6, we flip the sign bit $C.\rho$ from $1b$ to $0b$ to ensure $2C.\rho - 1$ equals -1. For the term $\tilde{\beta} \cdot 2^{C.\alpha} - 2^{\mathcal{L}_\beta}$, we recalculate the value of $\tilde{\beta}$ according to Eq. (8), so that the term adheres to the representation in Eq. (7) while yielding a value equal to $-(\tilde{\beta} \cdot 2^{C.\alpha} - 2^{\mathcal{L}_\beta})$. So, in Fig. 6, we assign ⓪$111b$ to $\tilde{\beta}$.

$$-(\tilde{\beta} \cdot 2^{C.\alpha} - 2^{\mathcal{L}_\beta}) \rightarrow \underbrace{(2^{\mathcal{L}_\beta - \alpha + 1} - \tilde{\beta})}_{new\ value\ of\ \tilde{\beta}} \cdot 2^{\alpha} - 2^{\mathcal{L}_\beta} \tag{8}$$

- **Correct underflow**. The resulting $\tilde{\beta}$ experiences underflow because its left-most implicit bit is ⓪. To correct it, we adjust the value of $\tilde{\beta}$ and α to ensure the implicit bit of $\tilde{\beta}$ becomes ①. As illustrated on the right-hand side of Fig. 6, when $\tilde{\beta}$ becomes ⓪$111b$, we left-shift $\tilde{\beta}$ by 1 bit, and decrement α by 1 accordingly.

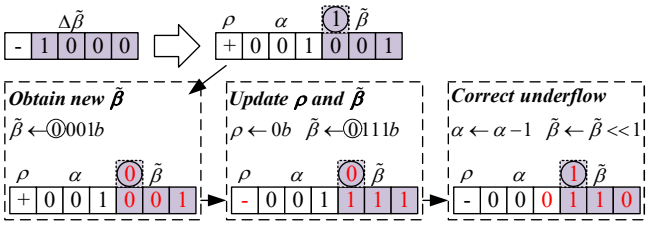

Fig. 6. Example of updating an RAC C

7.3 Optimization

With RAC, the estimation accuracy of traditional sketches, such as CountSketch (CS) and CountMin Sketch, can be improved. However, for each insertion or deletion, these sketches update one counter per row, resulting in the update of d (e.g., 4, 8) RACs. This leads to a higher processing overhead than using traditional integer counters, due to the complexity of the RAC's update procedure.

To mitigate this, a good practice is to limit updates to only one counter per insertion or deletion, as demonstrated by Virtual Active Counter [44, 92]. Thus, we introduce a variant of CS, Single-update CountSketch (SuCS), which diverges from CS in three primary aspects: first, SuCS maintains a one-dimensional array; second, it uses d hashes to find d counters, but randomly updates only one of them; third, it returns 1 if the query result of frequency is less than 0, to align with the bounded deletion model. In RAS, we replace CountSketch with SuCS, and then apply RAC to SuCS, yielding RA-SuCS. Our experiments show that, RA-SuCS provides an effective trade-off between estimation accuracy and processing speed. The mathematical analysis is available in our technical report [90].

8 ONLINE MOMENT ESTIMATOR

We first show the rationale behind the Online Moment Estimator (OME), followed by an example of online frequency moment estimation. The pseudocode is available in our technical report [90].

8.1 Rationale behind Online Moment Estimator

The *recursive summation technique* [6] estimates the frequency moment of all layers recursively from layer ℓ to layer 0 by Eq. (6). We present the following general formula to estimate the 0th-layer moment $\hat{L_0}$ without recursion. This is achieved by applying the ℓth-layer moment $\hat{L_\ell} = \sum_{e\in\hat{H_\ell}} g(\hat{f_e})$ to its lower-layer moment $\hat{L}_{\ell-1}$ in Eq. (6), and then recursively to $\hat{L}_{\ell-2}$, and so on.

$$\hat{L_0} = 2^\ell \sum\nolimits_{e\in\hat{H_\ell}} g(\hat{f_e}) + \sum\nolimits_{j=0}^{\ell-1} 2^j \sum\nolimits_{e\in\hat{H_j}} \left(1 - 2\cdot \mathbf{1}_{j+1\le j_t(e)}\right) g(\hat{f_e})$$

According to $j_t(e)$ defined in Eq. (4), an element e is never sampled on layer $\ell+1$, i.e., $\mathbf{1}_{\ell+1\le j_t(e)} \equiv 0$. So we rewrite this equation to

$$\hat{L_0} = \sum\nolimits_{j=0}^{\ell} 2^j \sum\nolimits_{e\in\hat{H_j}} \left(1 - 2\cdot \mathbf{1}_{j+1\le j_t(e)}\right) g(\hat{f_e}). \tag{9}$$

It implies that the moment can be estimated by summing up the moment contribution of each element on each layer, e.g., the contribution of the element e on the jth layer is $2^j\cdot\left(1-2\cdot\mathbf{1}_{j+1\le j_t(e)}\right)\cdot g(\hat{f_e})$.

Then, consider a case when all the elements are present in multiple adjacent layers. For example, an element e is present in each layer from $j_t(e)$ to $j_b(e)$, where $j_t(e)$ is e's topmost sampled layer, and $j_b(e)$ is the lowest layer among all the layers that regard e as a heavy hitter. We call $j_b(e)$ the *bottom layer* of the element e:

$$j_b(e) = \min j, \quad \text{subject to } 0 \le j \le \ell \ \wedge\ e \in \hat{H}_j. \tag{10}$$

In this case, Eq. (9) can be converted to

$$\begin{aligned}\hat{L_0} &= \sum\nolimits_{e\in\cup\hat{H_j}} \sum\nolimits_{j=j_b(e)}^{j_t(e)} 2^j \left(1 - 2\cdot \mathbf{1}_{j+1\le j_t(e)}\right) g(\hat{f_e}), \\ &= \sum\nolimits_{e\in\cup\hat{H_j}} \left(2^{j_t(e)} - \sum\nolimits_{j=j_b(e)}^{j_t(e)-1} 2^j\right) g(\hat{f_e}) = \sum\nolimits_{e\in\cup\hat{H_j}} 2^{j_b(e)} g(\hat{f_e})\end{aligned} \tag{11}$$

where $\cup\hat{H_j}$ is the union of all sets of heavy hitters from layer ℓ to 0.

So, we obtain OME as expressed by Eqs. (9) and (11). For an arrival tuple (e, w) carrying an element e that appears in multiple adjacent layers (e.g., *HH propagation*), we can incrementally update the moment $\hat{L_0}$ by Eq. (11), using the old values of $\hat{f_e}$ and $j_b(e)$ before the tuple's arrival and the new values afterward. In addition, if an element e' is no longer a heavy hitter on the jth layer (e.g., e' is kicked out during the *HH propagation*), we update the moment $\hat{L_0}$ by removing the contribution of e' on layer j using Eq. (9).

8.2 Example of Online Moment Updating

Combining the *HH Propagation* described in Section 5.1, we illustrate an example of the online moment updating in Fig. 4(b). Suppose an RUS with layers indexed from 0 to ℓ, and an arrival tuple (e, w), whose topmost sampled layer $j_t(e)$ is layer 5.

Update Moment by Arrival Element *e*. We show it in two phases.

- **Remove old contribution.** As shown on the left-hand side of Fig. 4(b), the element e is delivered to layer 5 and then propagates to layer 1. Before the arrival of the tuple (e, w), the bottom layer of the element e was layer 3, and it becomes layer 1 afterward. Thus, in the column "Moment Update" of Fig. 4(b), we remove the contribution of element e at layer 3 using Eq. (11), which is

$2^{j_b^{old}(e)} g(\hat{f}_e - w)$, where $j_b^{old}(e)$ denotes e's old bottom layer, namely layer 3, and $\hat{f}_e - w$ denotes e's frequency before the arrival of the tuple (e, w).

• **Add new contribution.** As shown on the left-hand side of Fig. 4(b), layer 1 is the new bottom layer of e after propagation. So, as depicted in the "Moment Update" column of Fig. 4(b), we add the contribution of the element e at layer 1 using Eq. (11), calculated as $2^{j_b^{new}(e)} g(\hat{f}_e)$, where $j_b^{new}(e)$ represents e's new bottom layer, namely layer 1, and $\hat{f}_e$ denotes the current frequency of e.

Update Moment by Kicked Elements e_1', e_2'. During the propagation, an element e_1' is kicked out at layer 2. Consequently, as illustrated in the "Moment Update" column of Fig. 4(b), we remove the contribution of element e_1' at layer 2 using Eq. (9), which is $2^j \cdot \left(1 - 2 \cdot \mathbf{1}_{j+1 \le j_t(e_1')}\right) \cdot g(\hat{f}_{e_1'})$, where j is the current layer, i.e., 2, and $j_t(e_1')$ denotes the topmost sampled layer of e_1'. Similarly, we remove the moment contribution of the element e_2' at layer 1 by Eq. (9).

9 EXPERIMENTAL EVALUATION

This section evaluates the performance of our proposed solutions for estimating per-element frequency, heavy hitters, and frequency moments in the *bounded deletion model*. Due to the page limit, we omit the experiments on parameter setting and frequency distribution estimation, which are detailed in our technical report [90].

9.1 Experimental Setup

Computation Platform. We conduct all the experiments on a 12-core CPU server (Intel i7-12700) with 128GB of memory.

Implementation. We implement the Single-update CountSketch (SuCS), CountSketch [9] with RAC (RA-CS), SuCS with RAC (RA-SuCS), Removable Augmented Sketch (RAS), Removable Universal Sketch (RUS), and all competing methods in different tasks in C++.

• **Per-element frequency estimation:** CMS [14], CS [9], randomized error-reduction Sketch (rSkt) [64], and Diamond Sketch (DS) [81]. We compare them with RA-SuCS, RA-CS, and SuCS. For DS, we set all parameters as suggested in the original paper [81]. We set the number of rows to 1 for SuCS and 4 for other methods.

• **Heavy hitter detection:** CMS/CS/DS/rSkt + MH (sketch with min-heap postfilter), MH + CMS/CS/DS/rSkt (sketch with min-heap prefilter) based on the frequency estimation methods, as well as SpaceSaving$^{\pm}$ (SS$^{\pm}$) [87]. We compare these methods with RAS.

• **Moment estimation:** UnivMon [39] and Off-RUS (an offline version of RUS that estimates moments using an offline moment estimator). We compare UnivMon and Off-RUS with our RUS.

Datasets. We use three real-world datasets and one synthetic dataset.

1) CAIDA Traces [8]. The CAIDA traces are collected from the Equinix Chicago high-speed monitor. We use the trace with a monitoring interval of 60s. The trace contains about 30.1M packets, originating from 1.1M distinct IP flows identified by (srcIP, srcPort, dstIP, dstPort, Protocol). We extract the ID of the IP flow from each packet to form the data stream, so there are about 30.1M insertions, and the insertions and deletions are represented as (flow ID, ±1).

2) IMDB Dataset [33]. The dataset includes multiple relations used for the Join Order Benchmark [33]. We select a high-cardinality column *person_role_id*, containing about 17.6M rows with 3.1M distinct values, from the largest table *cast_info*. We extract the value ID to construct the data stream, so there are 17.6M insertions, with the insertions and deletions in the form of (value ID, ±1). In database systems, the statistics collected from a column can be used to estimate the result size of all sub-plans of each query by a query optimizer, such as the Selinger query optimizer [55, 68].

3) Yelp Reviews Dataset [85]. The dataset includes around 7.0M business reviews in chronological order. We perform bigram text mining on each review, and identify about 22.0M distinct bigrams with a total frequency of 733.9M. The corresponding data stream is formed by the bigrams extracted from each of the ordered reviews. Overall, there are about 733.9M insertions, and the insertion and deletion events are represented as (bigram ID, ±1).

4) Synthetic Zipf Distribution Dataset [87]. The dataset contains about 4.9M distinct items with a total frequency of 20.0M, where items' frequencies follow Zipf's Law [42] with skewness $\eta = 1$. The corresponding data stream includes 20.0M insertions, with the insertion and deletion events represented as (item ID, ±1). We explore two arrival patterns of the dataset to show the robustness of our solution. In *shuffled pattern*, we shuffle the dataset so that items arrive in a random order. In *sorted pattern*, we sort the dataset to allow the items with lower frequencies to arrive first.

Deletion Patterns. For the CAIDA traces, IMDB, and Zipf distribution datasets, deletions in the corresponding data streams are randomly chosen from the insertions. For the Yelp reviews dataset, deletions are made from the oldest reviews. Therefore, the insertion events that arrive first in the data stream are deleted first.

Metrics. We use the Average Absolute Error (AAE), Average Relative Error (ARE), F1 Score (F1), Relative Error (RE), and Throughput in Millions of Operations per Second (Mops) as our evaluation metrics. These metrics are detailed in our technical report [90].

9.2 Performance of Frequency Estimation

On the three real-world datasets, we compare RA-SuCS against RA-CS, SuCS, CMS, CS, DS, and rSkt under different memory allocations, with the $D:I$ ratio set to 0.5. We set the memory allocation ranges from 128KB to 1024KB for the CAIDA traces and IMDB datasets, and from 5MB to 50MB for the Yelp reviews dataset, as the number of distinct bigrams extracted significantly exceeds the count of unique elements in the first two datasets. The AAE and ARE for the three real-world datasets are shown in Figs. 7a–7f.

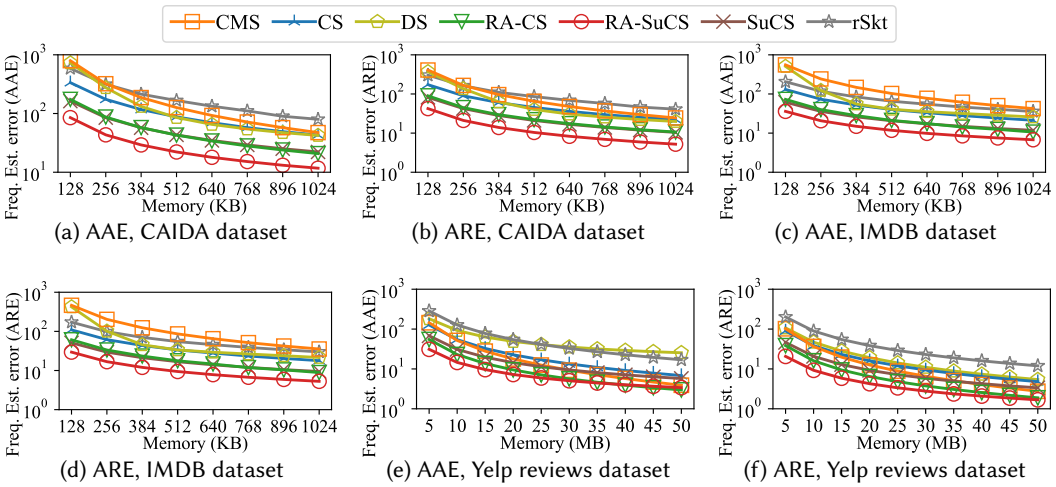

Fig. 7. Frequency estimation errors on three datasets

AAE (Figs. 7a, 7c and 7e). Compared with CMS, CS, RA-CS, DS, SuCS, and rSkt, the AAE of RA-SuCS is 12.4, 3.7, 2.0, 11.3, 1.8, and 6.3 times lower, respectively, on average under 128KB of memory for the CAIDA traces and IMDB datasets; it is 4.6, 4.1, 1.8, 5.8, 2.2, and 9.2 times lower under 5MB of memory for the Yelp reviews dataset.

ARE (Figs. 7b, 7d and 7f). Compared with CMS, CS, RA-CS, DS, SuCS, and rSkt, the ARE of RA-SuCS is 12.7, 3.8, 2.0, 11.5, 1.8, and 6.5 times lower, respectively, on average under 128KB of memory for the CAIDA traces and IMDB datasets; it is 4.9, 4.3, 1.9, 5.2, 2.2, and 9.7 times lower under 5MB of memory for the Yelp reviews dataset.

Analysis of Frequency Estimation. (a) SuCS outperforms other methods that use 32-bit integer counters, including CMS, CS, and rSkt, in our evaluation. This is because it uses the Tug-of-War technique like CS to reduce the cumulative estimation error, rather than adding up all frequencies into the hashed counter like CMS, which increases the cumulative error. Besides, SuCS sets the estimation to 1 if it falls below 0, which complies with the bounded deletion model where the deletion is bounded. (b) RA-SuCS shows the best performance, since it combines SuCS with 16-bit RAC, further enhancing the estimation accuracy by doubling the number of counters. (c) DS also proposes a counter compression technique, but it does not show a performance advantage. This is because it is designed for low-skew (e.g., skewness between 0 and 1) data streams carrying low-frequency elements (e.g., frequencies less than 2^{16}) [81].

9.3 Performance of Heavy Hitter Detection

We compare RAS with CMS/CS/DS/rSkt+MH, MH+CMS/CS/DS/rSkt, and SS$^{\pm}$ in HH detection. We conduct two experiments to show the performance under fixed and varying $D:I$ ratios, respectively. In both experiments, we set the threshold ε for HH detection as defined in Eq. (2) to 2^{-14}, and set the filter size to $\frac{\zeta}{12\varepsilon}$ by Section 6.4.

1) We fix the $D:I$ ratio (i.e., $1-\frac{1}{\zeta}$) to 0.5 and use the CAIDA traces dataset, IMDB dataset, and Yelp reviews dataset for evaluation. We set the memory allocation to range from 256KB to 1024KB.

2) With 256KB of memory, we use the synthetic Zipf distribution dataset to evaluate the performance of HH detection when the $D:I$ ratio ranges from 0.1 to 0.9, which reflects the bounded deletion scenario, where deletions exist and the $D:I$ ratio is bounded.

9.3.1 ***Performance under Fixed $D:I$ Ratio.*** The results for the experiments under a fixed $D:I$ ratio are shown in Figs. 8a–8f.

F1 Score of HH Detection (Figs. 8a, 8c and 8e). Compared with CMS+MH, CS+MH, DS+MH, MH+CMS, MH+CS, MH+DS, MH+rSkt, SS$^{\pm}$, and rSkt+MH, the F1 Score of our RAS is 21%, 53%, 55%, 20%, 16%, 18%, 35%, 59%, and 69% higher, respectively, on average under 256KB of memory for the three real-world datasets.

ARE of HH Frequency Estimation (Figs. 8b, 8d and 8f). Compared with CMS+MH, CS+MH, DS+MH, MH+CMS, MH+CS, MH+DS, MH+rSkt, SS$^{\pm}$, and rSkt+MH, the ARE of our RAS is 30.6, 32.2, 37.6, 23.4, 7.1, 6.3, 21.7, 28.3, and 39.5 times lower, respectively, on average under 256KB of memory for the three real-world datasets.

Analysis of HH Detection under Fixed $D:I$ Ratio. (a) The prefilter-based methods outperform both the postfilter- and counter-based methods. In postfilters, the frequencies recorded are solely queried from the sketch, whereas in prefilters, updates to elements can be precisely recorded once they are loaded. For the counter-based method SS$^{\pm}$, when the table is full, it has to replace the newly arrived element with one already recorded in the table, leading to information loss and resulting in poorer overall accuracy of HH detection compared to methods that combine a sketch with a prefilter. (b) RAS performs better than the conventional prefilter-based methods, because we adopt a KP-CF as a prefilter and use RA-SuCS as the backend sketch. The prefilter improves the frequency estimation accuracy, and RA-SuCS outperforms other backend sketches, thus showing superior performance compared to conventional methods.

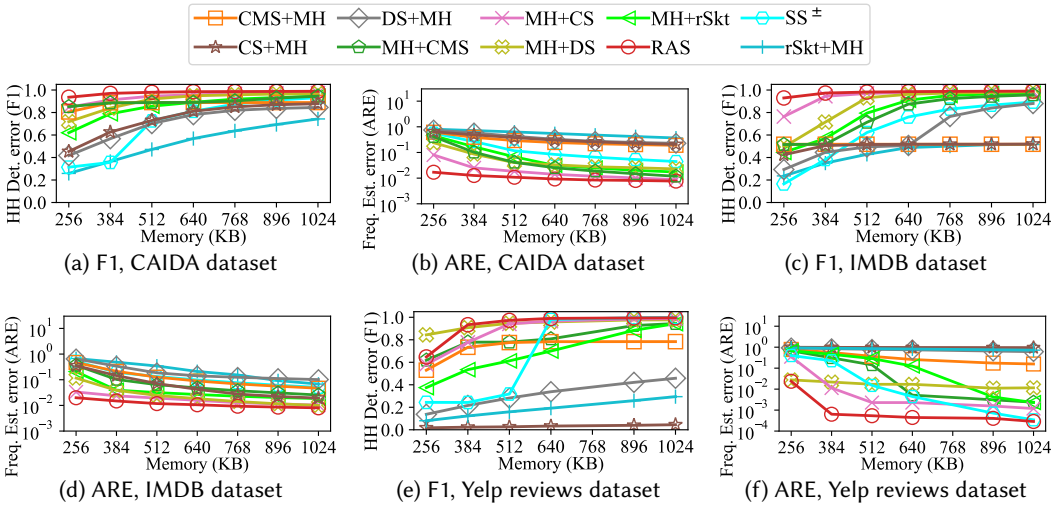

Fig. 8. HH detection errors on three real-world datasets

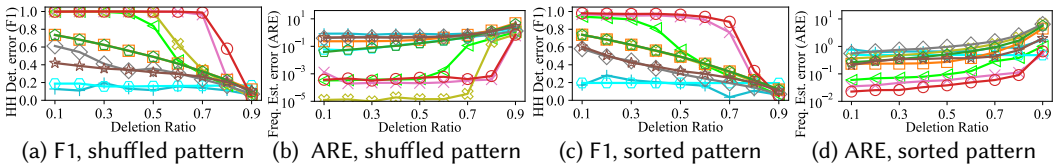

Fig. 9. HH detection errors under varying deletion ratios on Zipf distribution dataset

9.3.2 ***Performance under Varying D:I Ratios.*** The results for the shuffled pattern and sorted pattern of the Zipf distribution dataset described in Section 9.1 are shown in Figs. 9a–9d.

Analysis of HH Detection under Varying $D:I$ Ratios. (a) As shown in Fig. 9, as the $D:I$ ratio increases, the performance of the HH detection quantified by the F1 Score and ARE degrades accordingly. The reason is that, the lower bound of the prefilter size increases with the $D:I$ ratio, which reduces the memory budget for the backend sketch, thereby increasing the frequency estimation error and adversely affecting HH detection. (b) In the shuffled pattern, the frequency estimation accuracy of MH+DS is superior to that of other methods, whereas in the sorted pattern, it is not. The root cause is that, in the shuffled pattern, the high-frequency elements occur sooner than in the sorted pattern where the elements with low frequencies arrive first. Therefore, the true heavy hitters are more easily captured by the prefilter, which reduces the skewness of the data streams fed into the sketch, where DS excels [81]. However, in the sorted pattern, the heavy hitter elements must be stored in the sketch initially, leading to poorer performance of DS.

9.4 Performance of Moment Estimation

We compare RUS with UnivMon and Off-RUS in the 0th-, 2nd-, and 3rd-order moments estimation, as well as entropy estimation. We exclude the 1st-order moment, as it can be precisely computed by adding up all updating weights. We conduct two experiments to show the performance under fixed and varying $D:I$ ratios, respectively. We set the relative threshold ε defined in Eq. (2) to 2^{-14}, and the size of the prefilter in each layer to $\frac{\zeta}{12\varepsilon}$ as specified in Section 6.4.

1) We fix the $D:I$ ratio (i.e., $1-\frac{1}{\zeta}$) to 0.5 and evaluate the performance on the CAIDA traces, IMDB, and Yelp reviews datasets. We set the memory allocation to range from 768KB to 2560KB for the first two datasets, and from 5MB to 50MB for the last dataset.

2) We evaluate using the Zipf distribution dataset under 2560KB of memory, with $D:I$ ratio ranging from 0.1 to 0.9, to reflect the scenario where deletions exist and the $D:I$ ratio is bounded.

Setting of the Number of Hierarchical Layers $\ell+1$. Let N be the number of distinct elements in a data stream. Suppose the number of layers in the RUS is $\ell+1$, and the prefilter size on each layer is set to k. On average, the number of elements sampled on the ℓth layer by hierarchical sampling is $N/2^{\ell}$. Then, $N/2^{\ell} \le k$, which means the prefilter on the ℓth layer should be capable of holding all the elements sampled to this layer. For setting the smallest ℓ satisfying this, we have $\ell = \lceil \log_2(N/k) \rceil$. The reason is detailed below.

1) When the number of layers is greater than $\lceil \log_2(N/k) \rceil + 1$, the elements sampled to higher layers can be found in the prefilter on the $\lceil \log_2(N/k) \rceil$th layer. Therefore, these higher layers do not contribute to the moment estimation, as an element e contributes only at its lowest sampled layer $j_b(e)$ by Eqs. (10) and (11). Thus, these layers result in wasted memory, which degrades the HH detection accuracy and reduces the moment estimation accuracy.

2) When the number of layers is fewer than $\lceil \log_2(N/k) \rceil + 1$, according to Eq. (11), the error in moment estimation will be much higher than the real value, and even higher than when the number of layers exceeds $\lceil \log_2(N/k) \rceil + 1$. This is because the information of the HHs, which is expected to be stored in the absent layers required to achieve $\lceil \log_2(N/k) \rceil + 1$ layers, is completely lost.

Due to the deletions and estimation errors, the optimal value in practice may not exactly match $\lceil \log_2(N/k) \rceil + 1$. Therefore, we select the optimal value within $[\lceil \log_2(N/k) \rceil - 1, \lceil \log_2(N/k) \rceil + 3]$ (i.e., $\lceil \log_2(N/k) \rceil + 1 \pm 2$). Due to page limit, we include the results for different number of layers in our technical report [90]. Based on these results, we set the number of layers to 9, 10, 15, and 12 for the CAIDA traces, IMDB, Yelp reviews, and Zipf datasets, respectively.

Performance under Fixed D:I Ratio (Figs. 10a–10l). Compared with UnivMon, the RE of RUS for the 0th-, 2nd-, 3rd-order moment and entropy estimation is 17.2, 2.6×10^3, 1.4×10^6, and 2.1 times lower, respectively, on average under 2560KB of memory for the CAIDA traces and IMDB datasets; it is 1.8, 2.3×10^2, 1.5×10^4, and 5.0 times lower under 50MB of memory for the Yelp reviews dataset.

Analysis of Moment Estimation under Fixed D:I Ratio. (a) The moment estimation error of RUS and Off-RUS shows negligible difference. This is because the OME used by RUS to incrementally update the moment estimation online does not introduce additional error, as described in Section 8.1. (b) The moment estimation error with our RUS is lower than that with UnivMon. This is because in moment estimation, our RUS uses the RAS method to identify HHs and estimate their frequencies, which shows superior performance over the CS+MH method used by UnivMon, as shown in Fig. 8.

Analysis of Moment Estimation under Varying D:I Ratios. (a) As shown in Fig. 11, the estimation error increases with the $D:I$ ratio. This is because Section 9.3.2 indicates that the HH detection accuracy decreases as the $D:I$ ratio increases under a fixed memory allocation, resulting in decreased moment estimation accuracy as indicated by Eq. (11), which calculates the moment by aggregating the weighted frequencies of all HHs. (b) The estimation error of RUS and Off-RUS on the Zipf distribution dataset in the shuffled pattern is lower than that in the sorted pattern. For example, Fig. 11c shows that the estimation error of the 3rd-order moment for RUS on the Zipf distribution dataset in the shuffled pattern can reach as low as 10^{-10}, compared to 10^{-5} in the sorted pattern shown in Fig. 11g. This is because Section 9.3.2 shows that, the HH detection accuracy for the synthetic dataset in the sorted pattern is lower than that in the shuffled pattern, which increases the moment estimation error.

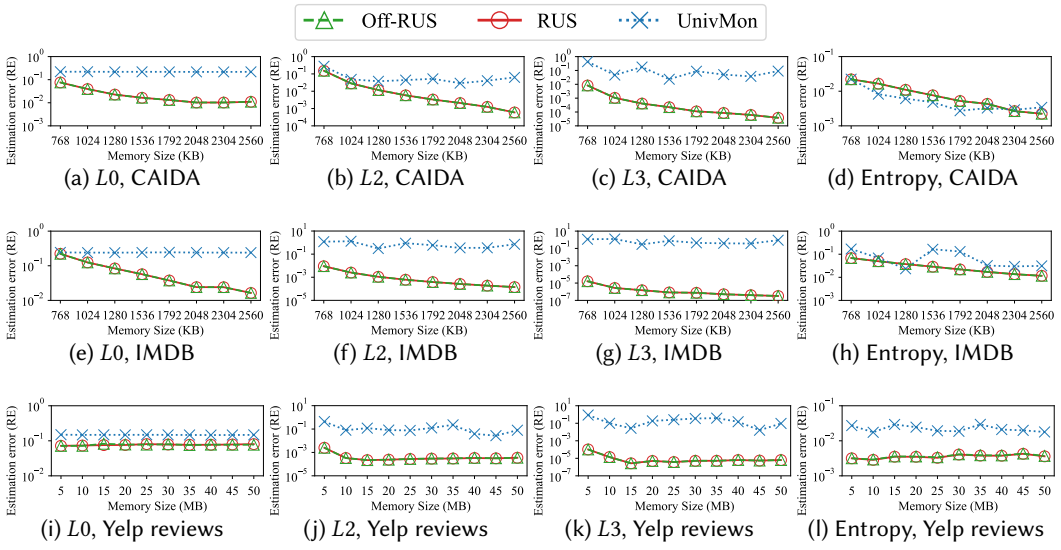

Fig. 10. Moment estimation errors when ***D*:*I*** ratio is 0.5

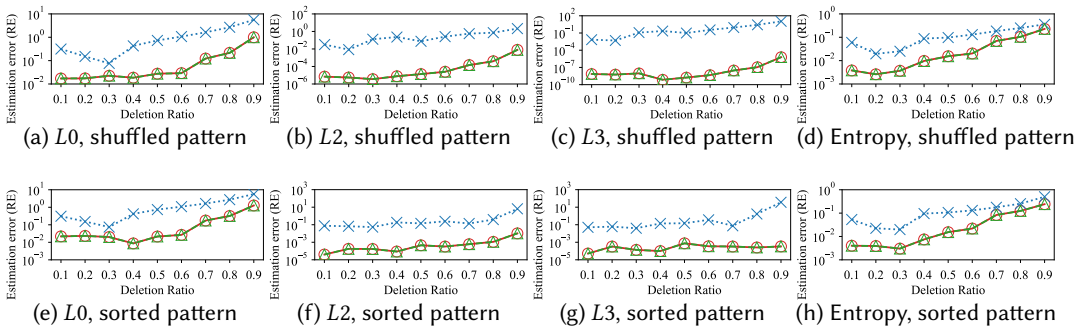

Fig. 11. Moment estimation errors under varying $D:I$ ratios on Zipf distribution dataset

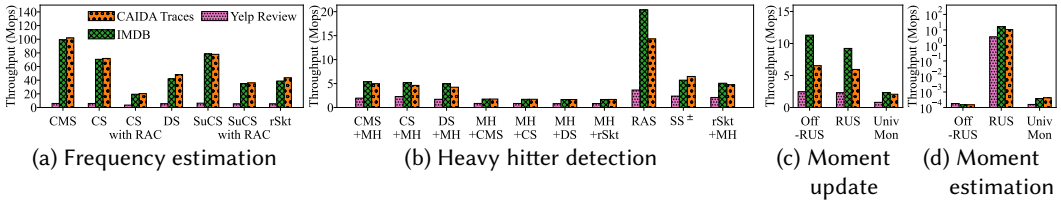

Fig. 12. Throughput of methods in per-element frequency estimation, HH detection, and moment estimation

9.5 Processing Speed

We show the average throughput across different memory allocations of methods in per-element frequency estimation (Section 9.2), HH detection (Section 9.3), as well as moment update and estimation (Section 9.4) on three real-world datasets in Figs. 12a–12d.

Analysis of Throughput for Frequency Estimation (Fig. 12a). (a) The throughput of CMS, CS, DS, SuCS, and rSkt is higher than that of RA-SuCS because they use only the integer counters to

record frequencies. (b) Compared to RA-CS on three real-world datasets, the throughput of RA-SuCS is 1.7 times higher, as it updates fewer RACs per insertion and deletion compared to RA-CS.

Analysis of Throughput for HH Detection (Fig. 12b). (a) The prefilter-based methods have the lowest throughput compared to the postfilter- and counter-based methods. Specifically, compared with RAS, the throughput of MH+CMS/CS/DS/rSkt is 8.0, 8.2, 8.4, 8.3 times lower. The throughput of CMS/CS/DS/rSkt+MH is 2.9, 2.9, 3.2, 2.9 times lower, and the throughput of SS$^{\pm}$ is 2.4 times lower. This is because the prefilter-based methods must query for the element from the prefilter once an element carried by a stream tuple arrives, with $O(k)$ lookup time cost for the min-heap. For the postfilter-based methods, the element is queried from the postfilter only when it is regarded as a HH, with $O(k)$ lookup time cost for the min-heap. For the counter-based method, SS$^{\pm}$, it achieves higher throughput as it maintains a hash table to accelerate the query speed of the min-heap. However, the min-heap is relatively large due to the *single element-frequency table design*, so the sift-up and sift-down operations, which cost $O(\log k)$, are still significant. (b) RAS shows the highest throughput, since we use KP-CF as a prefilter, which offers $O(1)$ query and update time costs, making the overall throughput of RAS higher than that of the comparison methods.

Analysis of Throughput for Moment Estimation (Figs. 12c and 12d). (a) The update throughput of RUS is higher (i.e., 3.2 times higher) than that of UnivMon. This is because for each arrival tuple, both algorithms have to update multiple HH detectors, and the HH detector used by RUS, namely RAS, offers higher update throughput than that of UnivMon, i.e., CS+MH. (b) The update throughput of RUS is slightly lower (i.e., 1.1 times lower) than that of Off-RUS. This is because RUS utilizes OME, which needs to update the moment estimation during the handling of the data stream tuples. (c) For the same reason, RUS achieves significantly higher throughput for moment estimation than that of Off-RUS and UnivMon. Specifically, the throughput of RUS for moment estimation is 3.0×10^4 and 6.6×10^4 times higher than that of UnivMon and Off-RUS, respectively. Note that Off-RUS and UnivMon utilize the *recursive summation technique*, which requires a linear scan of all the HHs from the highest layer downwards to the lowest layer upon queries.

10 CONCLUSION AND FUTURE WORK

For a data stream in the *bounded deletion model*, we propose RUS to simultaneously collect multiple kinds of statistics online: per-element frequency, HHs, frequency moments, and frequency distribution. Three main components of our RUS are the RAS to detect HHs in the *bounded deletion stream*, the OME to estimate the frequency moments online, and the 16-bit RAC to further improve the frequency estimation accuracy. Our experiments show that our RUS improves the F1 Score of detecting HHs by $16\% \sim 69\%$, and increases the throughput of moment estimation by 3.0×10^4 times.

In the future, we plan to adapt our solution to various scenarios described in Section 5.2, including network measurement, pattern mining in text corpora, and database query optimization. For network measurement, we will focus on exploring ways to apply multiple RUSs to collect statistics in distributed settings [77]. For pattern mining, we will extend RUS from mining frequent n-grams to the more general case of frequent sequential patterns [78]. For database query optimization, we will focus on integrating our RUS with the data-driven cardinality estimation model [35, 68] using statistics across multiple relations, to determine the cardinality of queries.

ACKNOWLEDGMENTS

This work was supported by National Natural Science Foundation of China under Grant 62372106, by Nsfocus Information Technology Co., Ltd. under Grant CCF-NSFOCUS202206, and by the Jiangsu Provincial Scientific Research Center of Applied Mathematics under Grant No. BK20233002.

REFERENCES

[1] Sugam Agarwal, Murali Kodialam, and TV Lakshman. 2013. Traffic engineering in software defined networks. In *Proc. IEEE INFOCOM*. 2211–2219.

[2] Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. 1999. Tracking join and self-join sizes in limited storage. In *Proc. of ACM PODS*. 10–20.

[3] Foteini Alvanaki and Sebastian Michel. 2014. Tracking set correlations at large scale. In *Proc. of ACM SIGMOD*.

[4] Animashree Anandkumar, Daniel Hsu, and Sham M Kakade. 2012. A method of moments for mixture models and hidden Markov models. In *Proc. of the ACM COLT*, Vol. 23. 33.1–33.34.

[5] Vladimir Braverman, Stephen R Chestnut, Nikita Ivkin, Jelani Nelson, Zhengyu Wang, and David P Woodruff. 2017. Bptree: An l2 heavy hitters algorithm using constant memory. In *Proc. of ACM PODS*. 361–376.

[6] Vladimir Braverman and Rafail Ostrovsky. 2013. Generalizing the layering method of indyk and woodruff: Recursive sketches for frequency-based vectors on streams. In *Proc. of APPROX Workshop*, Vol. 8096. 58–70.

[7] Vladimir Braverman, David Woodruff, and Lin Yang. 2018. Revisiting Frequency Moment Estimation in Random Order Streams. In *Proc. of ICALP*, Vol. 107. 25:1–25:14.

[8] CAIDA. 2016. The CAIDA Anonymized Internet Traces. http://www.caida.org/data/overview/

[9] Moses Charikar, Kevin Chen, and Martin Farach-Colton. 2004. Finding frequent items in data streams. *Theoretical Computer Science* 312, 1 (2004), 3–15.

[10] Peiqing Chen, Dong Chen, Lingxiao Zheng, Jizhou Li, and Tong Yang. 2021. Out of many we are one: Measuring item batch with clock-sketch. In *Proc. of ACM SIGMOD*. 261–273.

[11] Zhida Chen, Gao Cong, and Walid G Aref. 2020. STAR: A distributed stream warehouse system for spatial data. In *Proc. of ACM SIGMOD*. 2761–2764.

[12] Jeffrey Considine, Feifei Li, George Kollios, and John Byers. 2004. Approximate aggregation techniques for sensor databases. In *Proc. of IEEE ICDE*. 449–460.

[13] Graham Cormode. 2022. Current trends in data summaries. *ACM SIGMOD Record* 50, 4 (2022), 6–15.

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

[15] Hetong Dai, Heng Li, Che-Shao Chen, Weiyi Shang, and Tse-Hsun Chen. 2020. Logram: Efficient log parsing using n n-gram dictionaries. *IEEE Transactions on Software Engineering* 48, 3 (2020), 879–892.

[16] Kyle B Deeds, Dan Suciu, and Magdalena Balazinska. 2023. SafeBound: A Practical System for Generating Cardinality Bounds. In *Proc. of ACM SIGMOD*, Vol. 1. 1–26.

[17] Cristian Estan and George Varghese. 2002. New directions in traffic measurement and accounting. In *Proc. of ACM SIGCOMM*. 323–336.

[18] Cristian Estan, George Varghese, and Mike Fisk. 2003. Bitmap algorithms for counting active flows on high speed links. In *Proc. of ACM SIGCOMM*. 153–166.

[19] Bin Fan, Dave G Andersen, Michael Kaminsky, and Michael D Mitzenmacher. 2014. Cuckoo filter: Practically better than bloom. In *Proc. of ACM CoNEXT*. 75–88.

[20] Zhuochen Fan, Ruixin Wang, Yalun Cai, Ruwen Zhang, Tong Yang, Yuhan Wu, Bin Cui, and Steve Uhlig. 2023. OneSketch: A Generic and Accurate Sketch for Data Streams. *IEEE Transactions on Knowledge and Data Engineering* 35, 12 (2023), 12887–12901.

[21] Edward Gan, Jialin Ding, Kai Sheng Tai, Vatsal Sharan, and Peter Bailis. 2018. Moment-Based Quantile Sketches for Efficient High Cardinality Aggregation Queries. In *Proc. of VLDB Endow.*, Vol. 11. 1647–1660.

[22] Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. 2016. *Data stream management: processing high-speed data streams*. Springer.

[23] Michael Geller and Pramod Nair. 2018. 5G security innovation with Cisco. *Whitepaper Cisco Public* (2018), 1–29.

[24] Stefan Heule, Marc Nunkesser, and Alexander Hall. 2013. Hyperloglog in practice: Algorithmic engineering of a state of the art cardinality estimation algorithm. In *Proc. of Springer EDBT*. 683–692.

[25] He Huang, Jiakun Yu, Yang Du, Jia Liu, Haipeng Dai, and Yu-E Sun. 2023. Memory-Efficient and Flexible Detection of Heavy Hitters in High-Speed Networks. In *Proc. of ACM SIGMOD*, Vol. 1. 1–24.

[26] Yesdaulet Izenov, Asoke Datta, Florin Rusu, and Jun Hyung Shin. 2021. COMPASS: Online sketch-based query optimization for in-memory databases. In *Proc. of ACM SIGMOD*. 804–816.

[27] Rajesh Jayaram and David P Woodruff. 2018. Data streams with bounded deletions. In *Proc. of ACM PODS*. 341–354.

[28] Piotr Jurkiewicz, Grzegorz Rzym, and Piotr Boryło. 2021. Flow length and size distributions in campus Internet traffic. *Computer Communications* 167 (2021), 15–30.

[29] Aarati Kakaraparthy, Jignesh M Patel, Brian P Kroth, and Kwanghyun Park. 2022. VIP hashing: adapting to skew in popularity of data on the fly. In *Proc. of VLDB Endow.*, Vol. 15. 1978–1990.

[30] Abhishek Kumar, Minho Sung, Jun Xu, and Jia Wang. 2004. Data streaming algorithms for efficient and accurate estimation of flow size distribution. *ACM SIGMETRICS Performance Evaluation Review* 32, 1 (2004), 177–188.

[31] Ashwin Lall, Vyas Sekar, Mitsunori Ogihara, Jun Xu, and Hui Zhang. 2006. Data streaming algorithms for estimating entropy of network traffic. *ACM SIGMETRICS Performance Evaluation Review* 34, 1 (2006), 145–156.
[32] Alexandru Lavric and Valentin Popa. 2017. Internet of things and LoRa™ low-power wide-area networks: a survey. In *Proc. of IEEE ISSCS*. 1–5.
[33] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. 2015. How good are query optimizers, really?. In *Proc. of VLDB Endow.*, Vol. 9. 204–215.
[34] Jizhou Li, Zikun Li, Yifei Xu, Shiqi Jiang, Tong Yang, Bin Cui, Yafei Dai, and Gong Zhang. 2020. Wavingsketch: An unbiased and generic sketch for finding top-k items in data streams. In *Proc. of ACM SIGKDD*. 1574–1584.
[35] Pengfei Li, Wenqing Wei, Rong Zhu, Bolin Ding, Jingren Zhou, and Hua Lu. 2023. ALECE: An Attention-based Learned Cardinality Estimator for SPJ Queries on Dynamic Workloads. In *Proc. of VLDB Endow.*, Vol. 17. 197–210.
[36] Weihe Li and Paul Patras. 2024. Stable-Sketch: A Versatile Sketch for Accurate, Fast, Web-Scale Data Stream Processing. In *Proc. of ACM WWW*. 4227–4238.
[37] Jialu Liu, Jingbo Shang, Chi Wang, Xiang Ren, and Jiawei Han. 2015. Mining quality phrases from massive text corpora. In *Proc. of ACM SIGMOD*. 1729–1744.
[38] Yi Liu, Lei Xu, Xingliang Yuan, Cong Wang, and Bo Li. 2022. The right to be forgotten in federated learning: An efficient realization with rapid retraining. In *Proc. of IEEE INFOCOM*. IEEE, 1749–1758.
[39] Zaoxing Liu, Antonis Manousis, Gregory Vorsanger, Vyas Sekar, and Vladimir Braverman. 2016. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proc. of ACM SIGCOMM*. 101–114.
[40] Qiang Long, Wei Wang, Jinfu Deng, Song Liu, Wenhao Huang, Fangying Chen, and Sifan Liu. 2019. A distributed system for large-scale n-gram language models at Tencent. In *Proc. of VLDB Endow.*, Vol. 12. 2206–2217.
[41] Jiaheng Lu, Chunbin Lin, Wei Wang, Chen Li, and Haiyong Wang. 2013. String similarity measures and joins with synonyms. In *Proc. of ACM SIGMOD*. 373–384.
[42] Nishad Manerikar and Themis Palpanas. 2009. Frequent items in streaming data: An experimental evaluation of the state-of-the-art. *Data & Knowledge Engineering* 68, 4 (2009), 415–430.
[43] Gurmeet Singh Manku and Rajeev Motwani. 2002. Approximate frequency counts over data streams. In *Proc. of VLDB Endow.* 346–357.
[44] Dimitrios Melissourgos, Haibo Wang, Shigang Chen, Chaoyi Ma, and Shiping Chen. 2023. Single Update Sketch with Variable Counter Structure. In *Proc. of VLDB Endow.*, Vol. 16. 4296–4309.
[45] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *Proc. of Springer ICDT*, Vol. 3363. 398–412.
[46] Jayanta Mondal and Amol Deshpande. 2014. Eagr: Supporting continuous ego-centric aggregate queries over large dynamic graphs. In *Proc. of ACM SIGMOD*. 1335–1346.
[47] Shanmugavelayutham Muthukrishnan et al. 2005. Data streams: Algorithms and applications. *Foundations and Trends® in Theoretical Computer Science* 1, 2 (2005), 117–236.
[48] Dang Nguyen, Wei Luo, Tu Dinh Nguyen, Svetha Venkatesh, and Dinh Phung. 2019. Sqn2vec: Learning sequence representation via sequential patterns with a gap constraint. In *Proc. of ECML PKDD*. 569–584.
[49] Thanh Tam Nguyen, Thanh Trung Huynh, Hongzhi Yin, Matthias Weidlich, Thanh Thi Nguyen, Thai Son Mai, and Quoc Viet Hung Nguyen. 2023. Detecting rumours with latency guarantees using massive streaming data. *The VLDB Journal* 32, 2 (2023), 369–387.
[50] Stuart L Pardau. 2018. The california consumer privacy act: Towards a european-style privacy regime in the united states. *J. Tech. L. & Pol'y* 23 (2018), 68.
[51] Debjyoti Paul, Yanqing Peng, and Feifei Li. 2019. Bursty event detection throughout histories. In *Proc. of IEEE ICDE*.
[52] Pratanu Roy, Arijit Khan, and Gustavo Alonso. 2016. Augmented sketch: Faster and more accurate stream processing. In *Proc. of ACM SIGMOD*. 1449–1463.
[53] Tony Saad and Giovanna Ruai. 2019. PyMaxEnt: A Python software for maximum entropy moment reconstruction. *SoftwareX* 10 (2019), 100353.
[54] Khaled Mohammed Saifuddin, Corey May, Farhan Tanvir, Muhammad Ifte Khairul Islam, and Esra Akbas. 2023. Seq-HyGAN: Sequence Classification via Hypergraph Attention Network. In *Proc. of ACM CIKM*. 2167–2177.
[55] P Griffiths Selinger, Morton M Astrahan, Donald D Chamberlin, Raymond A Lorie, and Thomas G Price. 1979. Access path selection in a relational database management system. In *Proc. of ACM SIGMOD*. 23–34.
[56] HyungBin Seo and MyungKeun Yoon. 2023. Generative intrusion detection and prevention on data stream. In *Proc. of USENIX Security*. 4319–4335.
[57] Cha Hwan Song, Pravein Govindan Kannan, Bryan Kian Hsiang Low, and Mun Choon Chan. 2020. FCM-Sketch: Generic Network Measurements with Data Plane Support. In *Proc. of ACM CoNEXT*. 78–92.
[58] Zehua Sun, Huanqi Yang, Kai Liu, Zhimeng Yin, Zhenjiang Li, and Weitao Xu. 2022. Recent advances in LoRa: A comprehensive survey. *ACM Transactions on Sensor Networks* 18, 4 (2022), 1–44.

[59] Kai Sheng Tai, Vatsal Sharan, Peter Bailis, and Gregory Valiant. 2018. Sketching linear classifiers over data streams. In *Proc. of ACM SIGMOD*. 757–772.
[60] Lu Tang, Qun Huang, and Patrick PC Lee. 2019. MV-Sketch: A Fast and Compact Invertible Sketch for Heavy Flow Detection in Network Data Streams. In *Proc. of IEEE INFOCOM*. 2026–2034.
[61] Daniel Ting. 2018. Data Sketches for Disaggregated Subset Sum and Frequent Item Estimation. In *Proc. of ACM SIGMOD*.
[62] Paul Voigt and Axel Von dem Bussche. 2017. The eu general data protection regulation (gdpr). *A Practical Guide, 1st Ed., Cham: Springer International Publishing* 10, 3152676 (2017), 10–5555.
[63] Haibo Wang, Chaoyi Ma, Olufemi O Odegbile, Shigang Chen, and Jih-Kwon Peir. 2021. Randomized error removal for online spread estimation in data streaming. In *Proc. of VLDB Endow.*, Vol. 14. 1040–1052.
[64] Haibo Wang, Chaoyi Ma, Olufemi O Odegbile, Shigang Chen, and Jih-Kwon Peir. 2022. Randomized Error Removal for Online Spread Estimation in High-Speed Networks. *IEEE/ACM TON* 31, 2 (2022), 558–573.
[65] Larry Wasserman. 2004. *All of statistics: a concise course in statistical inference*. Vol. 26. Springer.
[66] Kyu-Young Whang, Brad T Vander-Zanden, and Howard M Taylor. 1990. A linear-time probabilistic counting algorithm for database applications. *ACM Transactions on Database Systems* 15, 2 (1990), 208–229.
[67] David P Woodruff and Samson Zhou. 2021. Separations for Estimating Large Frequency Moments on Data Streams. In *Proc. of ICALP*, Vol. 198. 112:1–112:21.
[68] Ziniu Wu, Parimarjan Negi, Mohammad Alizadeh, Tim Kraska, and Samuel Madden. 2023. FactorJoin: a new cardinality estimation framework for join queries. In *Proc. of ACM SIGMOD*. 1–27.
[69] Qingjun Xiao, Xuyuan Cai, Yifei Qin, Zhiying Tang, Shigang Chen, and Yu Liu. 2023. Universal and Accurate Sketch for Estimating Heavy Hitters and Moments in Data Streams. *IEEE/ACM TON* 31, 5 (2023), 1919–1934.
[70] Qingjun Xiao, Yuexiao Cai, Yunpeng Cao, and Shigang Chen. 2023. Accurate and O(1)-Time Query of Per-Flow Cardinality in High-Speed Networks. *IEEE/ACM TON* 31, 6 (2023), 2994–3009.
[71] Qingjun Xiao, Shigang Chen, You Zhou, Min Chen, Junzhou Luo, Tengli Li, and Yibei Ling. 2017. Cardinality estimation for elephant flows: A compact solution based on virtual register sharing. *IEEE/ACM TON* 25, 6 (2017), 3738–3752.
[72] Qingjun Xiao, Yifei Li, and Yeke Wu. 2023. Finding recently persistent flows in high-speed packet streams based on cuckoo filter. *Computer Networks* 237 (2023), 110097.
[73] Qingjun Xiao, Zhiying Tang, and Shigang Chen. 2020. Universal online sketch for tracking heavy hitters and estimating moments of data streams. In *Proc. of IEEE INFOCOM*. 974–983.
[74] Qingjun Xiao, Haotian Wang, and Guannan Pan. 2022. Accurately Identify Time-decaying Heavy Hitters by Decay-aware Cuckoo Filter along Kicking Path. In *Proc. of IEEE/ACM IWQoS*. 1–10.
[75] Qingjun Xiao, You Zhou, and Shigang Chen. 2017. Better with fewer bits: Improving the performance of cardinality estimation of large data streams. In *Proc. of IEEE INFOCOM*. 1–9.
[76] Guorui Xie, Qing Li, Guanglin Duan, Yong Jiang, Zhuyun Qi, Shuo Liu, and Qiaoling Wang. 2023. Efficient Flow Recording with InheritSketch on Programmable Switches. In *2023 IEEE 43rd International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 1–11.
[77] Yuchen Xu, Wenfei Wu, Bohan Zhao, Tong Yang, and Yikai Zhao. 2023. MimoSketch: A Framework to Mine Item Frequency on Multiple Nodes with Sketches. In *Proc. of ACM SIGKDD*. 2838–2849.
[78] Da Yan, Wenwen Qu, Guimu Guo, Xiaoling Wang, and Yang Zhou. 2022. PrefixFPM: a parallel framework for general-purpose mining of frequent and closed patterns. *The VLDB Journal* 31, 2 (2022), 253–286.
[79] Kaicheng Yang, Sheng Long, Qilong Shi, Yuanpeng Li, Zirui Liu, Yuhan Wu, Tong Yang, and Zhengyi Jia. 2023. SketchINT: Empowering int with towersketch for per-flow per-switch measurement. *IEEE Transactions on Parallel and Distributed Systems* 34, 11 (2023), 2876–2894.
[80] Mingran Yang, Junbo Zhang, Akshay Gadre, Zaoxing Liu, Swarun Kumar, and Vyas Sekar. 2020. Joltik: enabling energy-efficient" future-proof" analytics on low-power wide-area networks. In *Proc. of ACM MobiCom*. 1–14.
[81] Tong Yang, Siang Gao, Zhouyi Sun, Yufei Wang, Yulong Shen, and Xiaoming Li. 2019. Diamond sketch: Accurate per-flow measurement for big streaming data. *IEEE Transactions on Parallel and Distributed Systems* 30, 12 (2019).
[82] Tong Yang, Junzhi Gong, Haowei Zhang, Lei Zou, Lei Shi, and Xiaoming Li. 2018. Heavyguardian: Separate and guard hot items in data streams. In *Proc. of ACM SIGKDD*. 2584–2593.
[83] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. 2018. Elastic sketch: Adaptive and fast network-wide measurements. In *Proc. of ACM SIGCOMM*. 561–575.
[84] Tong Yang, Haowei Zhang, Jinyang Li, Junzhi Gong, Steve Uhlig, Shigang Chen, and Xiaoming Li. 2019. HeavyKeeper: an accurate algorithm for finding Top-k elephant flows. *IEEE/ACM TON* 27, 5 (2019), 1845–1858.
[85] Yelp. 2021. Yelp Open Dataset. https://www.yelp.com/dataset/
[86] Quanwei Zhang, Qingjun Xiao, and Yuexiao Cai. 2023. A generic sketch for estimating super-spreaders and per-flow cardinality distribution in high-speed data streams. *Computer Networks* 237 (2023), 110059.

[87] Fuheng Zhao, Divyakant Agrawal, Amr El Abbadi, and Ahmed Metwally. 2022. SpaceSaving$^{\pm}$: an optimal algorithm for frequency estimation and frequent items in the bounded-deletion model. In *Proc. of VLDB Endow.*, Vol. 15. 1215–1227.
[88] Fuheng Zhao, Punnal Ismail Khan, Divyakant Agrawal, Amr El Abbadi, Arpit Gupta, and Zaoxing Liu. 2023. Panakos: Chasing the Tails for Multidimensional Data Streams. In *Proc. of VLDB Endow.*, Vol. 16. 1291–1304.
[89] Fuheng Zhao, Sujaya Maiyya, Ryan Wiener, Divyakant Agrawal, and Amr El Abbadi. 2021. KLL$^{\pm}$approximate quantile sketches over dynamic datasets. In *Proc. of VLDB Endow.*, Vol. 14. 1215–1227.
[90] Liang Zheng, Qingjun Xiao, and Xuyuan Cai. 2024. A Universal Sketch for Estimating Heavy Hitters and Per-Element Frequency Moments in Data Streams with Bounded Deletions [technical report]. https://github.com/usoop/Removaleb_Universal_Sketch/blob/main/TechnicalReport.pdf.
[91] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. 2018. Cold filter: A meta-framework for faster and more accurate stream processing. In *Proc. of ACM SIGMOD*. 741–756.
[92] You Zhou, Yian Zhou, Shigang Chen, and Youlin Zhang. 2018. Highly compact virtual active counters for per-flow traffic measurement. In *Proc. of IEEE INFOCOM*. 1–9.

Received April 2024; revised July 2024; accepted August 2024