Approaching 100% Confidence in Stream Summary through ReliableSketch

Yuhan Wu*
Peking University
Beijing, China
yuhan.wu@pku.edu.cn

Hanbo Wu*
Peking University
Beijing, China
wuhanbo@pku.edu.cn

Xilai Liu
Institute of Computing Technology, Chinese Academy of Sciences
Beijing, China
liuxilai20121013@gmail.com

Yuxuan Tian
Peking University
Beijing, China
tianyuxuan@stu.pku.edu.cn

Yikai Zhao
Peking University
Beijing, China
zyk@pku.edu.cn

Tong Yang†
Peking University
Beijing, China
yangtong@pku.edu.cn

Rui Qiu
Peking University
Beijing, China
ruiqiu@pku.edu.cn

Kaicheng Yang
Peking University
Beijing, China
ykc@pku.edu.cn

Sha Wang
National University of Defense Technology
Changsha, China
ws0623zz@163.com

Tao Li
National University of Defense Technology
Changsha, China
taoli@gmail.com

Lihua Miao
Huawei Technologies Co., Ltd.
Shenzhen, China
miaolihua4@huawei.com

Gaogang Xie
Computer Network Information Center, Chinese Academy of Sciences
Beijing, China
xie@cnic.cn

Abstract

To approximate sums of values in key-value data streams, sketches are widely used in databases and networking systems. They offer high-confidence approximations for any given key while ensuring low time and space overhead. While existing sketches are proficient in estimating individual keys, they struggle to maintain this high confidence across all keys collectively, an objective that is critically important in both algorithm theory and its practical applications. We propose ReliableSketch, the first to control the error of all keys to less than Λ with a small failure probability Δ, requiring only $O(1 + \Delta \ln\ln(\frac{N}{\Lambda}))$ amortized time and $O(\frac{N}{\Lambda} + \ln(\frac{1}{\Delta}))$ space. Furthermore, its simplicity makes it hardware-friendly, and we implement it on CPU servers, FPGAs, and programmable switches. Our experiments show that under the same small space, ReliableSketch not only keeps all keys' errors below Λ but also delivers competitive throughput among accuracy-oriented baselines, outperforming competitors with thousands of uncontrolled estimations. We have made our source code publicly available.

CCS Concepts

• **Theory of computation → Sketching and sampling**.

Keywords

data streams; sketches; error bounds; confidence

ACM Reference Format:
Yuhan Wu, Hanbo Wu, Xilai Liu, Yuxuan Tian, Yikai Zhao, Tong Yang, Rui Qiu, Kaicheng Yang, Sha Wang, Tao Li, Lihua Miao, and Gaogang Xie. 2025. Approaching 100% Confidence in Stream Summary through ReliableSketch. In *Proceedings of the 2025 ACM Internet Measurement Conference (IMC '25), October 28–31, 2025, Madison, WI, USA.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3730567.3764459

1 Introduction

In data stream processing, stream summary [17] is a simple but challenging problem: within a stream of key-value pairs, query a key "e" for its value sum $f(e)$ — the sum of all values associated with that key. The problem is typically addressed by "sketches" [25, 29, 44], a kind of approximate algorithm that can answer an estimated sum $\hat{f}(e)$ with small time and space consumption. In terms of accuracy, existing sketches ensure that the absolute error of $\hat{f}(e)$ is less than Λ with a high probability $1 - \delta$. This can be formally expressed as:

$$\text{For arbitrary key } e, \Pr\left[\left|\hat{f}(e) - f(e)\right| \leqslant \Lambda\right] \geqslant 1 - \delta,$$

*Both authors contributed equally to this research.
†Corresponding author: Tong Yang (yangtong@pku.edu.cn)

IMC '25, Madison, WI, USA

ACM ISBN 979-8-4007-1860-1/25/10
https://doi.org/10.1145/3730567.3764459

where there are two critical parameters: the error tolerance Λ, under which the absolute error is considered controllable, and $\hat{f}(e)$ is deemed sufficiently accurate; otherwise, the key e is referred to as an outlier. The individual *Confidence Level (CL)*, $1-\delta$, represents the lower bound probability that $\hat{f}(e)$ is sufficiently accurate.

Existing sketches, effective for individual queries, struggle with accurately answering multiple queries at once. When N keys are queried collectively, *the overall CL*, denoted as $1-\Delta$, that all answers are sufficiently accurate equals $(1-\delta)^N$. The overall CL rapidly decreases as N increases: from $(1-\delta) = 95\%$ for a single key to 90.25% for two keys, and further diminishes to just 1% for 90 keys. Furthermore, when all keys are queried collectively, as in a million-key scenario, a significant absolute number—about the δ-fraction of these keys—are expected to be outliers. These outliers, which users cannot distinguish from other keys, undermine confidence in the results' reliability and pose real-world challenges. For example, in network devices, sketches are used to identify if a key is frequent (if its value is large enough). In a dataset with 1 million infrequent and 1,000 frequent keys, even with a 99% individual CL, approximately 10,000 infrequent keys might be wrongly labeled as frequent, leading to a high false positive rate of 90.9%. Such misidentification can cause serious issues in network applications, such as placing critical control signals into low-priority queues, which can result in the loss of these signals during network congestion.

Our target problem is to accurately answer an unlimited number of queries collectively, with a negligible failure probability Δ. This can be formally stated as:

$$\Pr\left[\forall\ key\ e, \left|\hat{f}(e) - f(e)\right| \leqslant \Lambda\right] \geqslant 1-\Delta,$$

As Table 1 illustrates, existing sketches, both counter-based and heap-based, can hardly address our target problem with small time and space consumption. Counter-based sketches [9, 14, 17, 29], which only record counters, increase confidence by repeating experiments and creating multiple sub-sketch copies. To achieve the individual CL of $1-\delta$, they create $\ln(\frac{1}{\delta})$ copies, which results in a time and space cost multiplied by $\ln(\frac{1}{\delta})$. For keys that can potentially reach a number up to N ($N := \sum f(e)$), accurately estimating all keys requires setting δ to a very small fraction of $\frac{\Delta}{N}$. This results in a significant increase in time and space costs. Counter-based sketches are divided into two types based on complexity: those using the L1 norm (e.g., CM [17], CU [19], Elastic [40]) and the L2 norm (e.g., Count [14], UnivMon [27], Nitro [26]). Given the challenge in assuming data characteristics, these types are generally not directly comparable. Our research focuses on optimizing L1 norm-based sketches. Heap-based sketches, such as Space Saving (SS) [30] and Frequent [18], are more adept at dealing with outliers but suffer from slower data insertion due to their logarithmic time complexity $\left(O(\ln(\frac{N}{\Lambda}))\right)$ heap structures. Additionally, they face compatibility challenges with high-speed hardware like FPGAs.

To address the target problem, we propose the ReliableSketch with versatile theoretical and practical advantages:

- Confidence: ReliableSketch guarantees that the error for all keys is controllable with an overall CL of $1-\Delta$, where Δ is a small quantity that can be easily reduced to below 10^{-10}. This ensures that not a single outlier will occur even after many years of the algorithm's operation. Here, "controllable" refers to keeping the error less than Λ.
- Speed: Our time complexity is lower than existing solutions. The amortized time cost for inserting each key-value pair is only $O(1+\delta\ln\ln(\frac{N}{\Lambda}))$. In practice, $\ln\ln(\frac{N}{\Lambda})$ is generally much smaller than $\frac{1}{\delta}$, making the time complexity effectively O(1) most of the time.
- Space: Our space complexity is $O(\frac{N}{\Lambda}+\ln(\frac{1}{\delta}))$, which is efficient because the result is additive and $\frac{N}{\Lambda}$ is generally much greater than $\ln(\frac{1}{\delta})$. We can easily set δ to be very small without worrying about increased space and time overhead.
- Compatibility with High-Speed Hardware: Our design is friendly to high-performance hardware architectures like FPGA, ASIC, and Tofino, adhering to the programming constraints of pipeline architecture. Our data structure does not require pointers, sorting, or dynamic memory allocation.

Compared to counter-based sketches, we have improved in terms of confidence, speed, and space complexity. We ensure high overall confidence for all keys collectively, reducing the amortized insertion complexity and transforming the space cost from a multiplicative $O(\frac{N}{\Lambda}\times\ln(\frac{1}{\delta}))$ to an additive one. Compared to heap-based sketches, we have made significant improvements in speed, optimizing the non-parallel $O(\ln(\frac{N}{\Lambda}))$ time to amortized O(1), while achieving nearly the same level of confidence and space efficiency.

Our main strategy is to sense errors in all keys in real-time, controlling those with larger errors to prevent any from becoming outliers. There are two key challenges involved: how to measure errors and how to control them. In response, we address them by two key techniques respectively: the Error-Sensible bucket and the Double Exponential Control.

Challenge 1: Measure Errors. Existing sketches, like the count-min sketch, do not know the error associated with a key because they mix the values of different keys in the same counter upon hash collisions. We harness an often-undervalued feature of the widely adopted election technology [18, 36, 40], particularly emphasizing the role of the negative vote. In elections, negative votes provide an ideal way to observe the hash collision and sense the error for each key during estimation. We replace the counter with an Error-Sensible bucket that contains an election mechanism, enabling real-time observation of each key's error.

Challenge 2: Controlling Errors. Existing sketches control errors by constructing d identical sub-tables for repeated experiments. This approach not only incurs significant overhead, as repeating the experiment d times requires d times the resources in terms of time and space, but it can only eliminate $O(e^d)$ outliers. Our Double Exponential Control technique not only eliminates $O(e^{(e^d)})$ outliers (with d=8, $e^{(e^d)} > 10^{1294}$) among all keys using d sub-tables, but also maintains a steady time and space cost, which does not increase linearly with the growth of d.

Our key contributions are as follows:

- We devise **ReliableSketch** to accurately answer the queries for all keys collectively, with a negligible failure probability Δ.
- We theoretically prove that ReliableSketch outperforms state-of-the-art in both space and time complexity.
- We implement ReliableSketch on multiple platforms including CPU, FPGA, and Programmable Switch. Under the same small

	Counter-based (L1-Norm)	**Counter-based (L2-Norm)**	**Heap-based**	**Reliable Sketch (Ours)**
Overall Confidence	Moderate: $(1-\delta)^N$	Moderate: $(1-\delta)^N$	Optimal: 100%	Near Optimal: $1-\Delta$
Speed	Moderate: $O(\ln(\frac{1}{\delta}))$	Moderate: $O(\ln(\frac{1}{\delta}))$	Low: $O(\ln(\frac{N}{\Lambda}))$	High: $O(1+\Delta\ln\ln(\frac{N}{\Lambda}))$
Space	Low: $O(\frac{N}{\Lambda}\times\ln(\frac{1}{\delta}))$	Low: $O(\frac{N_2^2}{\Lambda^2}\times\ln(\frac{1}{\delta}))$	Optimal: $O(\frac{N}{\Lambda})$	Near Optimal: $O(\frac{N}{\Lambda}+\ln(\frac{1}{\delta}))$
Compatibility	High	High	Low	High

Table 1: Comparative Analysis of Counter-based, Heap-based, and Reliable Sketches: Here, $N=\sum f(e)$, $N_2=\sqrt{\sum(f(e))^2}$, and Λ is the error tolerance. Additionally, δ and Δ represent the probabilities of failure for individual and all keys, respectively.

memory, ReliableSketch not only eliminates outliers, but also achieves competitive throughput, while its competitors have thousands of outliers.

2 Background and Motivation

In this section, we start with the problem definition and then discuss the limitations of existing sketches, using the typical CM Sketch as an example.

2.1 Problem Definition

Stream Summary Problem [17]. In a key-value data stream $S=\{\langle e_1,v_1\rangle,\langle e_2,v_2\rangle,\dots\}$, the sketch algorithm processes each pair (or data item) in real-time. At any moment, for a user query about a specific key e, the sketch can rapidly estimate the aggregate sum of values for all pairs containing e. The actual sum and the sketch's estimated sum for a key e are denoted as $f(e)$ and $\hat{f}(e)$, respectively. Within a predefined error tolerance Λ, a key e whose estimation error exceeds Λ, expressed as $\left|\hat{f}(e)-f(e)\right|>\Lambda$, is defined as an outlier.

We aim to accurately answer all queries collectively and eliminating outliers, under user-specified hyperparameters Λ (the error tolerance) and Δ (upper limit of failure probability):

$$\Pr\left[\forall\ key\ e, \left|\hat{f}(e)-f(e)\right|\leqslant\Lambda\right]\geqslant 1-\Delta.$$

2.2 Limitation of Existing Solutions

Here, for readers unfamiliar with sketches, we begin with the simplest CM sketch to explain why existing sketches struggle to eliminate outliers. A detailed complexity analysis has already been discussed in Section 1 and Table table 1, and will not be reiterated.

The CM sketch is a prime example of the design philosophy behind most counter-based sketches. It comprises d arrays, $A_i[.]$, each containing w counters. For a key e, CM selects d *mapped counters* using independent hashing. The i-th mapped counter is $A_i[h_i(e)]$, where $h_i(\cdot)$ is the hash function. When an item arrives with key e and a positive value v, CM increments these mapped counters by v. To query the value sum of e, CM reports the smallest counter among e's mapped counters as the estimate. However, when other keys collide with the same counter as e (a collision), they add their value to the counter, causing an error. The key insight is that the smallest mapped counter is the most accurate due to the fewest collisions.

However, even the minimal counter, with the least error, can have significant inaccuracies when every mapped counter experiences severe hash collisions. Thus, CM and other counter-based sketches can only maintain a high confidence level for a single query. When querying a large number of keys, it's not guaranteed that each key's error will be small. Similarly, other counter-based sketches, including Count, CU, Univmon, and others, share this design limitation. The complexity of counter-based sketches is based on the L1 norm $N=\sum f(e)$ and the L2 norm $N_2=\sqrt{\sum(f(e))^2}$. Since the relative sizes of N and N_2 depend on dataset characteristics, the complexity of these two types of sketches cannot be directly compared. Our research focuses on optimizing L1 norm-based sketches, while solutions based on the L2 norm's complexity for our target problem are left for future work..

Heap-based sketches, like Space Saving and Frequent, use heap structures to maintain high-frequency elements but suffer from slower data insertion due to their logarithmic time complexity ($\ln(\frac{N}{\Lambda})$). This complexity cannot be accelerated through parallelization. Additionally, the pointer operations they require become an obstacle in implementing them on high-speed hardware platforms like FPGA programmable switches. Only when $v=1$ can these heap structures be implemented with $O(1)$ complexity using linked lists, but we aim to address a broader range of stream summary problems where v is not equal to 1.

3 Reliable Sketch

In this section, we present ReliableSketch. We start from a new alternative to the counter of counter-based sketches, termed an 'Error-Sensible bucket'. This allows every basic counting unit within ReliableSketch to perceive the magnitude of its error, as discussed in § 3.1. Then, we demonstrate how to organize these buckets and set appropriate thresholds to control the error of every key within the user-defined threshold, as discussed in § 3.2.

3.1 Basic Unit: the Error-Sensible Bucket

The basic unit is the smallest cell in a sketch that performs the counting operation. In counter-based sketches, the basic unit is a standard counter. In our ReliableSketch, the basic unit is the "Error-Sensible Bucket" structure, which actively perceives the extent of hash collisions and reports the Maximum Possible Error (MPE). We demonstrate the workflow and a practical example in Figure 1 and 2, respectively.

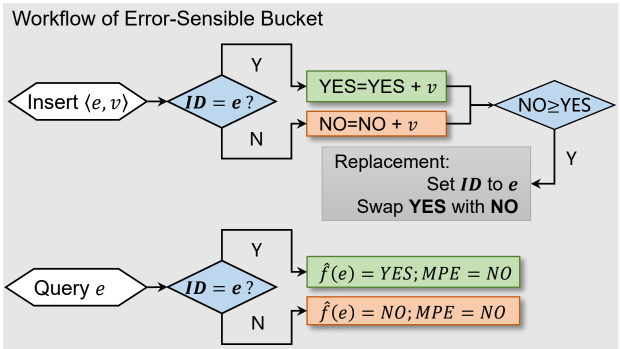


Figure 1: The workflow of the Error-Sensible Bucket, including insertion and querying processes.

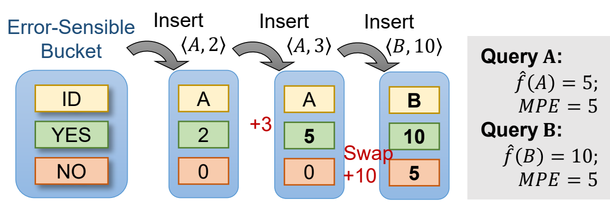


Figure 2: An example of how an Error-Sensible Bucket works: starting from empty, sequentially inserting three items followed by two queries.

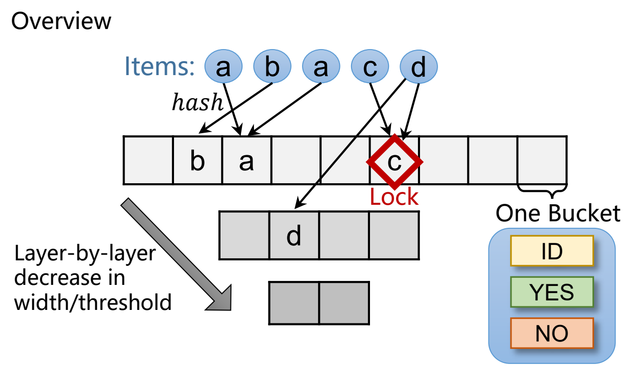


Figure 3: An Overview of ReliableSketch.

Formally, the bucket supports two operations: (1) Insert a key-value pair $\langle e, v\rangle$; (2) Query the value sum of a key e. Upon querying, the bucket returns two results, the estimated frequency $\hat{f}(e)$ along with its MPE, satisfying $\hat{f}(e) \in [f(e), f(e) + MPE]$.

Structure: The bucket has three fields: one ID field recording a candidate key, and two counters recording the positive and negative votes for the candidate, denoted as *ID*, *YES*, and *NO*, respectively. Initially, *ID* is null and both counters are set to 0.

Insert. When inserting an item $\langle e, v\rangle$, there are two phases: voting and replacement. If the newly arrived e is the same as *ID*, a positive vote is cast, setting YES to $YES + v$; otherwise, a negative vote is cast, making $NO = NO + v$. If the positive votes are less than or equal to the negative votes, a replacement occurs: *ID* is set to e, and the values of *YES* and *NO* are swapped.

Query. When querying the value sum of e, we first check if the recorded *ID* matches e. If it does, indicating e is the current candidate, we use *YES* to estimate its value sum. It can be proven inductively that *YES* is greater than or equal to $f(e)$, with its MPE being *NO*, i.e., $f(e) \in [YES - NO, YES]$. On the other hand, if *ID* is not equal to e, we use *NO* as the estimate, which is always greater than $f(e)$. The MPE remains *NO*.

Discussion—Correctness. A full inductive proof follows from the bucket's update rules; here we provide a compact intuition.

Basic facts. (i) YES + NO always equals the total value ever inserted into the bucket; (ii) after each swap, NO contains no contribution from the current ID.

Fix a key e_0 and let $f(e_0)$ be its true sum. Each arrival of e_0 casts one vote: it increases YES if ID $= e_0$ and NO otherwise. Hence YES + NO $\geq f(e_0)$. We now consider two query-time cases.

Case ID $= e_0$. Since NO records only collisions with keys $\neq e_0$, YES upper-bounds $f(e_0)$. Moreover, the increase of YES − NO is caused only by insertions of e_0 (including the swap that moves past negative votes of e_0 into YES). Therefore,

$$f(e_0) \in [\text{YES} - \text{NO},\ \text{YES}].$$

Case ID $\neq e_0$. All occurrences of e_0 must have contributed to NO and never to YES, so

$$f(e_0) \in [0,\ \text{NO}].$$

Why NO equals the collision mass. NO increases in exactly two situations. (1) *Without replacement:* inserting $\langle a, v\rangle$ while ID $= b \neq a$ adds v to NO, recording a collision between distinct keys. (2) *With replacement:* for a pre-swap bucket $\langle \text{ID} = b, \text{YES} = y_0, \text{NO} = n_0\rangle$, inserting $\langle a, v\rangle$ with $a \neq b$ raises NO and then triggers a swap; the net increase equals $y_0 - n_0$. Each unit of value participates in at most one such collision, so there is no double counting.

Complementary intuition (equivalent view). Only matching inserts can increase YES; non-matching inserts increase NO; and YES + NO is conserved. If at query time ID $= e$, then all historical inserts of e reside in YES and none in NO, giving YES−NO $\leq f(e) \leq$ YES. If ID $\neq e$, then $0 \leq f(e) \leq$ NO. Hence the bucket returns $\hat{f}$ together with the interval $[\hat{f} - \text{MPE}, \hat{f}]$ where MPE = NO.

3.2 Formal ReliableSketch

In this part, we propose how to organize Error-Sensible buckets and integrate them into a ReliableSketch that can control the error of all keys. Our key idea is to lock the error of a bucket when it reaches a critical threshold, diverting any further error-increasing insertions to the next layer. We first introduce the "lock" mechanism, then describe the data structure of ReliableSketch, as well as its insertion and querying operations.

Lock Mechanism. When the Maximum Possible Error (MPE) of a bucket reaches a threshold, i.e., $B.NO = \lambda\ \ \&\ \ B.YES > B.NO$, the bucket must be locked to halt the growth of MPE. Upon being locked, only two types of insertions are permitted, neither of which will increase the MPE ($B.NO$):

- If $B.ID = e$, it only increments $B.YES$.

- If $B.YES = B.NO$, a replacement occurs, and this also only increments $B.YES$.

In other scenarios, items cannot be inserted into the locked bucket. These items, at a higher risk of losing control, are redirected to other buckets for insertion.

Data structure. As depicted in Figure 3, ReliableSketch is composed of d layers, with each layer, indexed by i, containing w_i Error-Sensible buckets, where the width w_i diminishes progressively with an increase in i. In each layer, the j-th bucket is denoted as $B_i[j]$. Each layer is assigned a specific threshold, referred to as λ_i, used to determine when to lock a bucket in the i-th layer. The λ_i values also decrease with increasing i, and their cumulative sum does not surpass the user-defined error threshold, i.e., $\sum_i \lambda_i \leqslant \Lambda$. Furthermore, each layer utilizes an independent hash function $h_i(\cdot)$, which uniformly maps each key to a single bucket within the respective layer.

Parameter Configurations: ReliableSketch performs best when both w_i and λ_i are set to **decrease geometrically** (i.e., exponentially in i), which is our Key Technique II (Double Exponential Control). As established by our analysis in §4, with high probability the number of keys that proceed to the next layer decays exponentially (for both mice and elephant keys). Matching the parameter schedules to this decay keeps the expected per-update work bounded and makes the failure probability drop doubly exponentially across layers, yielding the stated complexity. In contrast, changing either sequence to an arithmetic progression would undermine these effects and deteriorate complexity. Practically, we set $w_i = \left\lceil \frac{W(R_w-1)}{R_w^i} \right\rceil$ and $\lambda_i = \left\lfloor \frac{\Lambda(R_\lambda-1)}{R_\lambda^i} \right\rfloor$, where W denote the total number of buckets. In our proofs (Theorem 4), we set $W = \frac{4(R_w R_\lambda)^6}{(R_w-1)(R_\lambda-1)} \cdot \frac{N}{\Lambda}$, which is with large constant. But based on our experiment results, we recommend to set $W = \frac{(R_w R_\lambda)^2}{(R_w-1)(R_\lambda-1)} \cdot \frac{N}{\Lambda}$, $R_w \in [2, 10]$, $R_l \in [2, 10]$, and $d \geqslant 7$. When then memory size is given without given Λ, we set $\Lambda = \frac{(R_w R_\lambda)^2}{(R_w-1)(R_\lambda-1)} \cdot \frac{N}{W}$.

Algorithm 1: Insert Operation.

1 **Procedure** `Insert`($\langle e, v\rangle$):
2   **for** *Layer* $i = 1, 2, \ldots, d$ **do**
3     $j \leftarrow h_i(e)$
4   **if** $B_i[j].ID = e$ **then**
5       $B_i[j].YES \leftarrow B_i[j].YES + v$
6   **Return**
7   **end**
8   **if** $B_i[j].NO + v > \lambda_i$ ***and*** $B_i[j].YES > \lambda_i$ **then**
9   ▷ Lock triggered
10       $B_i[j].NO \leftarrow \lambda_i$
11       $v \leftarrow v - (\lambda_i - B_i[j].NO)$
12   **Continue** ▷ Continue to next layers
13   **else**
14       $B_i[j].NO \leftarrow B_i[j].NO + v$
15   **if** $B_i[j].NO \geqslant B_i[j].YES$ **then**
16         $B_i[j].ID \leftarrow e$
17   Swap($B_i[j].NO$, $B_i[j].YES$)
18   **end**
19   **Return**
20   **end**
21   **end**

Insert Operation for Item $\langle e, v\rangle$ (Algorithm 1). The insertion into ReliableSketch is a layer-wise process, starting at the first layer and continuing until the value v is fully inserted. The operation, which may not involve all layers, includes these four steps in each layer:

(1) **Locating a Bucket (Line 3):** Utilizing the hash function $h_i()$ of the current layer i, we locate the j-th bucket, $B_i[j]$, where $j = h_i(e)$. We aim to insert $\langle e, v\rangle$ specifically into this bucket, without considering other buckets in the same layer.

(2) **Handling Matching ID (Line 4-7):** If $B_i[j].ID$ equals e, we increment YES by v and finish the insertion without moving to further layers.

(3) **Triggering Lock (Line 8-12):** This step follows if the ID doesn't match. Before increasing NO, we check if adding v triggers the layer threshold λ_i. The lock activates when $B_i[j].NO+v > \lambda_i$ (indicating certain lock activation without replacement) and when $B_i[j].YES > \lambda_i$ (signifying lock activation upon replacement). On triggering the lock, only a portion of $\langle e, v\rangle$ can be accommodated in $B_i[j]$, equal to the difference $\lambda_i - B_i[j].NO$. The excess value, $v - (\lambda_i - B_i[j].NO)$, is reserved for insertion into subsequent layers (Line 12). Consequently, $B_i[j].NO$ is adjusted to λ_i.

(4) **Adjusting NO and Checking for Replacement (Line 14-19):** If this step is reached, it means a negative vote is cast, and the lock would not be activated. $B_i[j].NO$ is incremented by v. Then, compare $B_i[j].NO$ with $B_i[j].YES$ to determine if a replacement occurs. If it does, perform the replacement. The insertion is finished, and no further layers are visited.

If, by the end of the final layer, there remains value that has not been inserted, we consider the insertion operation to have failed. Once insertion failure occurs, we cannot guarantee zero outliers. Fortunately, through our design and theoretical proofs, we have shown that the probability of such an failure is extremely low. For those still concerned about this scenario, refer to § 3.3 for emergency solutions.

Query Operation for Item e (Algorithm 2). In ReliableSketch, each query reports $\hat{f}(e)$ along with its Maximum Possible Error (MPE). The query operation is similar to insertion, requiring a layer-wise process to gather results layer by layer, stopping as soon as there's sufficient reason to do so (which usually happens quickly). Beginning from the first layer, we sequentially access the hashed bucket $B_i[j]$, $j = h_i(e)$ in each layer. If $B_i[j].ID$ equals e, we add $B_i[j].YES$ to $\hat{f}(e)$; otherwise, we add $B_i[j].NO$. For MPE, we always add $B_i[j].NO$. The query can be finished without accessing subsequent layers if any of the following conditions are met, as each indicates that e has not been inserted into subsequent layers: (1) $B_i[j].NO < \lambda_i$ indicates $B_i[j]$ is not locked, and e will not visit subsequent layers; (2) $B_i[j].YES = B_i[j].NO$ indicates a potential replacement, meaning e will not visit subsequent layers; (3) $B_i[j].ID = e$ indicates a match with e, and even if the current bucket is locked and cannot be replaced, e will not visit subsequent layers.

Algorithm 2: Query Operation.

Function `Query(e)`:
  $\hat{f}(e) \leftarrow 0$ ▷ Estimator
  $MPE \leftarrow 0$ ▷ Maximum Possible Error
  for $Layer\, i = 1, 2, \ldots, d$ **do**
    $j \leftarrow h_i(e)$
    if $B_i[j].ID = e$ **then**
      $\hat{f}(e) \leftarrow \hat{f}(e) + B_i[j].YES$
    else
      $\hat{f}(e) \leftarrow \hat{f}(e) + B_i[j].NO$
    end
    $MPE \leftarrow MPE + B_i[j].NO$
    if $(B_i[j].NO < \lambda_i$ ***or*** $B_i[j].YES = B_i[j].NO$ ***or*** $B_i[j].ID = e)$ **then**
      break ▷ Stop collecting value.
    end
  end
  return $\langle \hat{f}(e), MPE \rangle$

Novelty of ReliableSketch. The fundamental novelty of ReliableSketch lies in its key idea: identifying keys with significant errors and effectively controlling these errors to completely eliminate outliers. This approach is supported by two innovative techniques we have developed: the Error-Sensible Bucket for error measurement and the Double Exponential Control for error management:

- Key Technique I (Error-Sensible Bucket). Although the voting technique itself is not novel, tracing back to the classic majority vote [31] algorithm of 1981, our unique contribution lies in demonstrating that the *NO* value can effectively limit the extent and impact of collisions. In integrating the Error-Sensible Bucket as the fundamental unit of a sketch, we developed a lock strategy that is directly informed by the size of *NO*. This approach contrasts with existing works like Majority, MV, Elastic [31, 36, 40], which primarily concentrate on identifying high-frequency keys but do not fully exploit the capability of *NO* in guiding error control.
- Key Technique II (Double Exponential Control). To control errors effectively for all keys, it's critical to address the outliers resulting from insertion failures. A crucial strategy is to limit the number of keys advancing to the next layer at each layer. Typically, a layer might halt about half of the keys, which significantly reduces the probability of outlier occurrence, denoted as $\mathcal{P}$, to $\frac{1}{2^d}$. Our research indicates that when both the width w_i and the layer threshold λ_i decrease exponentially, for example, $\frac{1}{2^{2^d}}$ (with $d = 8$, the probability is approximately 8.6×0.1^{78}), the failure probability $\mathcal{P}$ diminishes at a double exponential rate. This marked decrease in probability effectively reduces the number of keys that can potentially become outliers, thereby eliminating outliers with an extremely high probability.

3.3 Optimizations and Extensions

Exception Handling. For completeness, we provide an optional fallback for insertion failures. When an update $\langle e, v\rangle$ (or a residual part $\langle e, v'\rangle$) cannot be accommodated across the first d layers, we divert it to a small auxiliary structure—either a hash table or a SpaceSaving instance—so that the uninserted portion is recorded explicitly. This mechanism is straightforward to implement on CPU servers. On FPGAs or network devices, maintaining the auxiliary structure can be delegated to a CPU-based control plane. We have implemented this exception-handling path, but we exclude it from our accuracy evaluation (see § 6), in order to present the performance of ReliableSketch on its own more clearly.

Accuracy Optimization. The first layer of ReliableSketch, which occupies more than 50% of the entire structure, is its largest. However, this layer can be inefficient when the dataset contains a significant proportion of mice keys (*i.e.*, keys with a small value sum). This is because mice keys sharply increase the NO counters, leading to the locking of most buckets in the first layer, resulting in many buckets being inefficiently used to record these mice keys. Given that NO counters do not exceed λ_1, we propose replacing the first layer with an existing sketch where each counter records up to λ_1. This involves substituting each bucket with a counter representing NO, updated with every insertion until reaching λ_1. We employed a commonly used CU sketch [19] for this purpose. In practice, 8-bit counters are adequate for the filter. Compared to a layer consisting of 72-bit error-sensible buckets, this filter can reduce the space requirement of the first layer by nearly 10 times, while introducing only small, manageable errors.

4 Mathematical Analysis

In this section, we provide the key results and key proof steps. We have placed the details of non-key steps in our open-source repository, as they are exceedingly complex and we are certain they cannot be fully included in the paper.

4.1 Key Results

We aim to prove the following two key claims.

Claim 1: The algorithm can achieve the following two goals by using $O\left(\frac{N}{\Lambda} + \ln(\frac{1}{\Delta})\right)$ space:

$$\Pr\left[\forall\ key\ e, \left|\hat{f}(e) - f(e)\right| \leqslant \Lambda\right] \geqslant 1 - \Delta$$

and

$$\forall\ key\ e, \Pr\left[\left|\hat{f}(e) - f(e)\right| \leqslant \Lambda\right] \geqslant 1 - \Delta$$

Claim 2: The algorithm can achieve the above two goals with $O\left(1 + \Delta \ln\ln(\frac{N}{\Lambda})\right)$ amortized time.

4.2 Key Steps

Generally, the key steps seek to prove that as i increases, the number of items entering the i-th layer during insertion diminishes rapidly. In the i-th layer, we categorize keys that enter the i-th layer based on their value size, into elephant keys and mice keys. For elephant keys, we show that their numbers decrease quickly. For mice keys, we show a rapid reduction in their aggregate value. This analysis forms the basis for determining the algorithm's time and space complexity.

Before explaining the proof sketch and key steps, we introduce some basic terms and symbols for clarity. Generally, we assume all values are 1. This means each data item's insertion adds a value of "1" to the sketch. The sum $f(e)$ equals how many times key e is inserted, its frequency. We first prove this for values of 1. Extending the proof to other values is trivial.

When an item is inserted to the i-th layer and stops the loop, it "enters" layers $1, 2 \dots i$ and "leaves" layers $1, 2 \dots i-1$. $f_i(e)$ represents the times an item with key e enters layer i. We compare $f_i(e)$ with $\frac{\lambda_i}{2}$ to categorize keys into two groups at layer i: Mice keys $\mathcal{S}_i^0$ and Elephant keys $\mathcal{S}_i^1$. We aim to show that the total frequency of mice keys F_i and the number of distinct elephant keys C_i decrease quickly with increasing i. These symbols are detailed in lines 1-5 of Table 2. For detailed analysis within a bucket, these symbols (lines 2-5) are adapted for the j-th bucket at layer i (lines 6-9).

Table 2: Common symbols

Symbol	Description
(1) $f_i(e)$	The number of times that key e enters the i-th layer.
(2) $\mathcal{S}_i^0$	$\{e \mid e \in \mathcal{S}_i \wedge f_i(e) \leqslant \frac{\lambda_i}{2}\}$, the set of mice keys.
(3) $\mathcal{S}_i^1$	$\{e \mid e \in \mathcal{S}_i \wedge f_i(e) > \frac{\lambda_i}{2}\}$, the set of elephant keys.
(4) F_i	$\sum_{\{e \in \mathcal{S}_i^0\}} f_i(e)$, the total frequency of mice keys in $\mathcal{S}_i^0$.
(5) C_i	$\lvert\mathcal{S}_i^1\rvert$, the number of elephant keys in $\mathcal{S}_i^1$.
(6) $\mathcal{S}_{i,j}^0$	$\{e \mid e \in \mathcal{S}_i^0 \wedge h(e) = j\}$, the set of mice keys that are mapped to the j-th bucket.
(7) $\mathcal{S}_{i,j}^1$	$\{e \mid e \in \mathcal{S}_i^1 \wedge h(e) = j\}$, the set of elephant keys that are mapped to the j-th bucket.
(8) $F_{i,j}$	$\sum_{\{e \in \mathcal{S}_{i,j}^0\}} f_i(e)$, the total frequency of mice keys in $\mathcal{S}_{i,j}^0$.
(9) $C_{i,j}$	$\lvert\mathcal{S}_{i,j}^1\rvert$, the number of elephant keys in $\mathcal{S}_{i,j}^1$.
(10) $\mathcal{P}_{i,k}$	$\{e_1, \cdots, e_k\}$, a subset of $\mathcal{S}_i$ composed of the first k keys.
(11) $f_{i,k}^P$	$\sum_{\{e \in \mathcal{P}_{i,k-1} \cap \mathcal{S}_{i,h(e_k)}^0\}} f_i(e)$, the total frequency of mice keys with a smaller index that conflicts with key e_k.
(12) $c_{i,k}^P$	$\left\lvert\{e \mid e \in \mathcal{P}_{i,k-1} \cap \mathcal{S}_{i,h(e_k)}^1\}\right\rvert$, the number of elephant keys with a smaller index that conflicts with key e_k.

Proof sketch: The proof consists of the following four steps. The first three steps focus on a single layer (the i-th layer), and analyze the relationship between the i-th layer and the $(i+1)$-th layer. The fourth step traverses all layers to draw a final conclusion.

- **Step 1 (Bound mice and elephant keys leaving the i-th layer with Xi and Yi, respectively)**. Our analysis must be applicable regardless of the order in which any item is inserted into the sketch. We start by analyzing, among the items entering the i-th layer, how many will proceed to the $(i+1)$-th layer, thereby leaving the i-th layer. We construct two time-order-independent random variables X_i and Y_i to bound mice keys and elephant keys, respectively: X_i bounds the total frequency of the mice keys leaving the i-th layer, and Y_i bounds the number of distinct elephant keys leaving the i-th layer (Theorem 1).
- **Step 2 (Double exponential decrease of X_i and Y_i)**: We prove that if the number of mice keys F_i and elephant keys C_i in the i-th layer decrease double exponentially, then the quantity of keys leaving the i-th layer, *i.e.*, X_i and Y_i, will also decrease double exponentially (Theorem 2).
- **Step 3 (Double exponential decrease of F_{i+1} and C_{i+1})**: Although the quantities of X_i and Y_i leaving the i-th layer are within limits, it does not directly imply that the number of mice keys F_{i+1} and elephant keys C_{i+1} in the $(i+1)$-th layer are few. This is because the criteria for categorizing an elephant key differ across layers, and a mice key from the i-th layer may become an elephant key upon entering the $(i+1)$-th layer. Fortunately, by using a Concentration inequality, we prove that this situation is controllable. That is, if F_i and C_i decrease double exponentially, F_{i+1} and C_{i+1} in the next layer will also decrease double exponentially (Theorem 3).
- **Step 4 (Combine all layers)**: Based on step 3, by using Boole's inequality, we combine the results from each layer and prove that there is a high probability $(1-\Delta)$ that the final conclusion holds (Theorem 4).

The results of step 1.

Theorem 1. *Let*

$$X_{i,k} = \begin{cases} 0 & C_{i,h(e_k)} = 0 \wedge f_{i,k}^P \leqslant \frac{\lambda_i}{2} \\ f_i(e_k) & C_{i,h(e_k)} = 0 \wedge f_{i,k}^P > \frac{\lambda_i}{2} \\ f_i(e_k) & C_{i,h(e_k)} > 0 \end{cases}, \quad X_i = \sum_{\{e_k \in \mathcal{S}_i^0\}} X_{i,k}.$$

The total frequency of the mice keys leaving the i-th layer does not exceed X_i, i.e.,

$$F_{i+1} \leqslant \sum_{\{e \in \mathcal{S}_i^0 \cap \mathcal{S}_{i+1}\}} f_{i+1}(e) \leqslant X_i.$$

Let

$$Y_{i,k} = \begin{cases} 0 & c_{i,k}^P = 0 \wedge F_{i,h(e_k)} \leqslant \lambda_i, \\ 2 & c_{i,k}^P = 0 \wedge F_{i,h(e_k)} > \lambda_i \\ 2 & c_{i,k}^P > 0. \end{cases}, \qquad Y_i = \sum_{e_k \in \mathcal{S}_i^1} Y_{i,k}.$$

The number of distinct elephant keys leaving the i-th layer does not exceed Y_i, i.e.,

$$\left|\mathcal{S}_i^1 \cap \mathcal{S}_{i+1}^1\right| \leqslant Y_i.$$

The results of step 2.

Theorem 2. *Let $W = \frac{4N(R_w R_\lambda)^6}{\Lambda(R_w-1)(R_\lambda-1)}$, $\alpha_i = \frac{\|F\|_1}{(R_w R_\lambda)^{i-1}}$, $\beta_i = \frac{\alpha_i}{\frac{\lambda_i}{2}}$, $\gamma_i = (R_w R_\lambda)^{(2^{i-1}-1)}$, and $p_i = (R_w R_\lambda)^{-(2^{i-1}+4)}$. Under the conditions of $F_i \leqslant \frac{\alpha_i}{\gamma_i}$ and $C_i \leqslant \frac{\beta_i}{\gamma_i}$, we have*

$$\Pr\left(X_i > (1+\Delta)\frac{p_i \alpha_i}{\gamma_i}\right) \leqslant \exp\left(-(\Delta - (e-2))\frac{2p_i\alpha_i}{\lambda_i \gamma_i}\right).$$

and

$$\Pr\left(Y_i > (1+\Delta)\frac{3}{2}\frac{p_i \beta_i}{\gamma_i}\right) \leqslant \exp\left(-(\Delta - (e-2))\frac{3p_i\beta_i}{4\gamma_i}\right).$$

The results of step 3.

Theorem 3. *Let $R_w R_\lambda \geqslant 2$, $W = \frac{4N(R_w R_\lambda)^6}{\Lambda(R_w-1)(R_\lambda-1)}$, $\lambda_i = \frac{\Lambda(R_\lambda - 1)}{R_\lambda^i}$, $\alpha_i = \frac{N}{(R_w R_\lambda)^{i-1}}$, $\beta_i = \frac{\alpha_i}{\frac{\lambda_i}{2}}$, $\gamma_i = (R_w R_\lambda)^{(2^{i-1}-1)}$, and $p_i =$*

$(R_w R_\lambda)^{-(2^{i-1}+4)}$. *We have*

$$\Pr\left(F_{i+1} > \frac{\alpha_{i+1}}{\gamma_{i+1}} \mid F_i \leqslant \frac{\alpha_i}{\gamma_i} \wedge C_i \leqslant \frac{\beta_i}{\gamma_i}\right) \leqslant \exp\left(-(9-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right).$$

$$\Pr\left(C_{i+1} > \frac{\beta_{i+1}}{\gamma_{i+1}} \mid F_i \leqslant \frac{\alpha_i}{\gamma_i} \wedge C_i \leqslant \frac{\beta_i}{\gamma_i}\right)$$

$$\leqslant \exp\left(-(5-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right) + \exp\left(-(\frac{11}{3}-e)\frac{3p_i\beta_i}{4\gamma_i}\right).$$

The results of step 4.

Theorem 4. *Let* $R_w R_\lambda \geqslant 2$, $W = \frac{4N(R_w R_\lambda)^6}{\Lambda(R_w-1)(R_\lambda-1)}$, $\lambda_i = \frac{\Lambda(R_\lambda-1)}{R_\lambda^i}$, $\alpha_i = \frac{N}{(R_w R_\lambda)^{i-1}}$, $\beta_i = \frac{\alpha_i}{\frac{\lambda_i}{2}}$, $\gamma_i = (R_w R_\lambda)^{(2^{i-1}-1)}$, *and* $p_i = (R_w R_\lambda)^{-(2^{i-1}+4)}$. *For given* Λ *and* $\Delta < \frac{1}{4}$, *let* d *be the root of the following equation*

$$\frac{R_\lambda^d}{(R_w R_\lambda)^{(2^d+d)}} = \Delta_1 \frac{\Lambda}{N} \ln(\frac{1}{\Delta}).$$

And use an SpaceSaving of size $\Delta_2 \ln(\frac{1}{\Delta})$ *(as the* $(d+1)$*-layer), then*

$$\Pr\left(\forall\ key\ e, \left|\hat{f}(e) - f(e)\right| \leqslant \Lambda\right) \geqslant 1 - \Delta,$$

where

$$\Delta_1 = 2R_w^2 R_\lambda^2 (R_\lambda - 1), \qquad \Delta_2 = 3\left(\frac{R_w R_\lambda^2}{R_\lambda - 1}\right)\Delta_1 = 6R_w^3 R_\lambda^4.$$

Complexity of ReliableSketch.

Theorem 5. *Using the same settings as Theorem 4, the space complexity of the algorithm is* $O(\frac{N}{\Lambda} + \ln(\frac{1}{\Delta}))$, *and the time complexity of the algorithm is amortized* $O(1 + \Delta \ln\ln(\frac{N}{\Lambda}))$.

5 Implementations

We have implemented ReliableSketch on three platforms: CPU server, FPGA, and Programmable Switch. Given the challenging nature of implementations on the latter two platforms, due to various hardware constraints, we provide a brief introduction here. Our source code is available on GitHub [2].

5.1 FPGA

We implement the ReliableSketch on an FPGA network experimental platform (Virtex-7 VC709). The FPGA integrated with the platform is xc7vx690tffg1761-2 with 433200 Slice LUTs, 866400 Slice Register, and 1470 Block RAM Tile. The implementation mainly consists of three hardware modules: calculating hash values (hash), Error-Sensible Buckets (ESbucket), and a stack for emergency solution (Emergency). ReliableSketch is fully pipelined, which can input one key in every clock, and complete the insertion after 41 clocks. According to the synthesis report (see Table 1), the clock frequency of our implementation in FPGA is 340 MHz, meaning the throughput of the system can be 340 million insertions per second.

5.2 Programmable Switch

To implement ReliableSketch on programmable switches (*e.g.*, Tofino), we need to solve the following three challenges.

Table 3: FPGA Implementation Results.

Module	CLB LUTs (count)	CLB Registers (count)	Block RAM (tiles)	Frequency (MHz)
Hash	85	130	0	339
ESbucket	2521	2592	258	339
Emergency	48	112	1	339
Total	2654	2834	259	339
Usage	0.61%	0.33%	17.62%	

Challenge I: Circular Dependency. Programmable switches limit SALU access to a pair of 32-bit data per stage, but each ReliableSketch bucket contains three fields (`ID`, `YES`, `NO`), creating dependencies that exceed this limit. To resolve this, we simplify the dependencies by using the difference between `YES` and `NO` (`DIFF`) for replacement decisions. This adjustment allows us to align `DIFF` and `ID` in the first stage and `NO` in the second stage, breaking the dependency cycle.
Challenge II: Backward Modification. When `NO` surpasses a threshold, the bucket must be locked, preventing updates to `ID`. However, due to pipeline constraints, a packet can't modify the `LOCKED` flag within its lifecycle. Our solution involves recirculating the packet that first exceeds the threshold, allowing it to re-enter the pipeline and update the flag.
Challenge III: Three Branches Update and Output Limitation. Weighted updates to `DIFF` could result in three different values, but switches can only support two variations. To accommodate this limitation and the 32-bit output constraint per stage, we simplify the update process. When not matching `ID`, `DIFF` is updated using saturated subtraction. In replacement scenarios, `DIFF` is reduced to zero, and `ID` is replaced upon the arrival of the next packet identifying `DIFF` as zero.

Table 4: H/W Resources Used by ReliableSketch.

Resource	Usage (unit)	Percentage
Hash Bits (bits)	541	10.84%
SRAM (blocks)	138	14.37%
Map RAM (blocks)	119	20.66%
TCAM (blocks)	0	0%
Stateful ALU (count)	12	25.00%
VLIW Instr (count)	23	5.99%
Match Xbar (count)	109	7.10%

Hardware resource utilization: After solving the above three challenges, we have fully implemented ReliableSketch on Edgecore Wedge 100BF-32X switch (with Tofino ASIC). Table 4 lists the utilization of various hardware resources on the switch. The two most used resources of ReliableSketch are Map RAM and Stateful ALU, which are used 20.66% and 25% of the total quota, respectively. These two resources are mainly used by the multi-level bucket arrays in ReliableSketch. For other kinds of resources, ReliableSketch uses up to 14.37% of the total quota.

6 Experiment Results

In this section, we present the experiment results for ReliableSketch. We begin with the setup of the experiments (§ 6.1). Following this, we perform a comparative analysis of ReliableSketch against existing solutions in terms of accuracy (§ 6.2) and speed (§ 6.3). Finally, we evaluate ReliableSketch in detail, including the impact of its parameters on performance (§ 6.4), its capability in error sensing and control, and industrial deployment performance (§ 6.5). The source code is available on GitHub [2].

6.1 Experiment Setup

6.1.1 Implementation. Our experiments are mostly based on C++ implementations of ReliableSketch and related algorithms. Here we use fast 32-bit Murmur Hashing [1], and different hash functions that affect accuracy little. Each bucket of ReliableSketch consists of a 32-bit YES counter, a 16-bit NO counter, and a 32-bit ID field. Mice filter occupies 20% of total memory, and bucket size of it is fixed to 2 bits unless otherwise noted. According to the study in § 6.4, we set R_w to 2 and R_λ to 2.5 by default. The memory size is 1MB and the user-defined threshold Λ is 25 by default. All the experiments are conducted on a server with 18-core CPU (36 threads, Intel CPU i9-10980XE @3.00 GHz), which has 128GB memory. Programs are compiled with O2 optimization.

6.1.2 Datasets. We use four large-scale real-world streaming datasets and one synthetic dataset, with the first dataset being the default.

- **IP Trace (Default):** An anonymized dataset collected from [4], comprised of IP packets. We use the source and destination IP addresses as the key. The first 10M packets of the whole trace are used to conduct experiments, including about 0.4M distinct keys.
- **Web Stream:** A dataset built from a spidered collection of web HTML documents [3]. The first 10M items of the entire trace are used to conduct experiments, including about 0.3M distinct keys.
- **University Data Center:** An anonymized packet trace from university data center [11]. We fetch 10M packets of the dataset, containing about 1M distinct keys.
- **Hadoop Stream:** A dataset built from real-world traffic distribution of HADOOP. The first 10M packets of the whole trace are used to conduct experiments, including about 20K distinct keys.
- **Synthetic Datasets:** We generate [33] several synthetic datasets according to a Zipf distribution with different skewness for experiments, each of them consists of 32M items.

By default, the value is set to 1 to allow for comparison with existing methods, unless the unit of the value is explicitly mentioned.

6.1.3 Evaluation Metrics. We evaluate the performance of ReliableSketch and its competitors using the following four metrics. Given our objective to control all errors below the user-defined threshold, our accuracy evaluation focuses more on the first metric, # Outliers, rather than metrics like AAE.

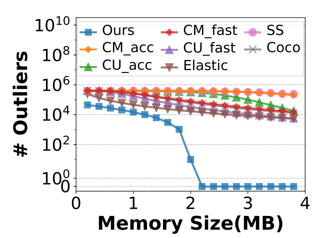
- **The Number of Outliers (# Outliers):** The number of keys whose absolute error of estimation is greater than the user-defined threshold Λ.
- **Average Absolute Error (AAE):** $\frac{1}{|U|}\sum_{e_i \in U}|f(e_i) - \hat{f}(e_i)|$, where U is the set of keys, $f(e_i)$ is the true value sun of key e_i, and $\hat{f}(e_i)$ is the estimation.
- **Average Relative Error (ARE):** $\frac{1}{|U|}\sum_{e_i \in U}\frac{|f(e_i)-\hat{f}(e_i)|}{f(e_i)}$.
- **Throughput:** $\frac{N}{T}$, where N is the number of operations and T is the elapsed time. Throughput describes the processing speed of an algorithm, and we use Million of packets per second (Mpps) to measure the throughput.

6.1.4 Implementation of Competitors. We conduct experiments to compare the performance of ReliableSketch ("Ours" in figures) with seven competitors, including CM [17], CU [19], SS [30], Elastic [40], Coco [41], HashPipe [35], and PRECISION[10]. Together they cover the categories in Table 1—counter-based (CM/CU), heap-based (SS), modern counter-based with light/heavy or partial-key components (Elastic/Coco), and switch-pipeline designs (HashPipe/PRECISION)—so the set is representative for our setting. For CM and CU, we provide fast (CM_fast/CU_fast) and accurate (CM_acc/CU_acc) two versions, implementing 3 and 16 arrays respectively. For Elastic, its light/heavy memory ratio is 3 as recommended [40]. For Coco, we set the number of arrays d to 2 as recommended [41]. For HashPipe, we set the number of pipeline stages d to 6 as recommended [35]. And for PRECISION, we set the number of pipeline stages d to 3 for best performance [10].

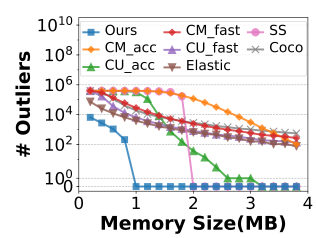
6.2 Accuracy Comparison

ReliableSketch controls error efficiently as our expectation and achieves the best accuracy compared with competitors. In evaluating accuracy, we considered three aspects: the number of outliers in all keys, the number of outliers in frequent keys, and average estimation error of values.

6.2.1 Number of Outliers in All Keys. Under various Λ values and across different datasets, we consistently achieve zero outliers with more than 2 times memory saving.



(a) Λ=5



(b) Λ=25

Figure 4: # Outliers in Different Λ.

Impact of Threshold Λ (Figure 4a, 4b): We vary Λ and count the number of outliers. As the figures show, ReliableSketch takes the lead position regardless of Λ. When Λ=25, ReliableSketch achieves zero outlier within 1MB memory, while the others still report over 5000 outliers.

Zero-Outlier Memory Consumption (Figure 5): We further explore the precise minimum memory consumption to achieve

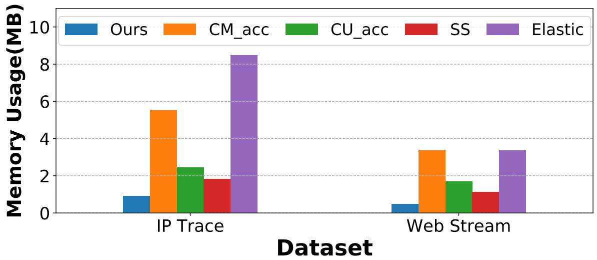
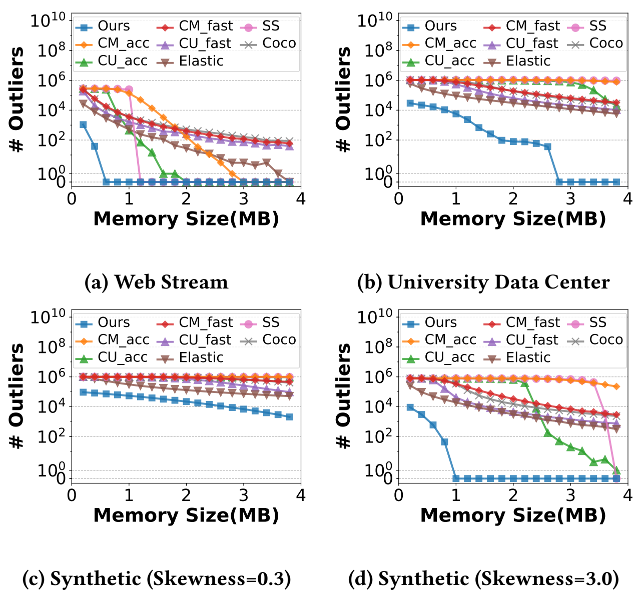


Figure 5: Memory Consumption under Zero Outlier.

zero outlier for all algorithms, Λ is fixed to 25 and experiments are conducted on different datasets. For the IP Trace dataset, memory consumption of ReliableSketch is 0.91MB, about 6.07, 2.69, 2.01, 9.32 times less than CM (accurate), CU (accurate), Space-Saving, and Elastic respectively. CM (fast), CU (fast) and Coco cannot achieve zero outlier within 10MB memory. Besides, CM, CU and Elastic usually require more memory than the minimum value, otherwise they cannot achieve zero outlier stably.



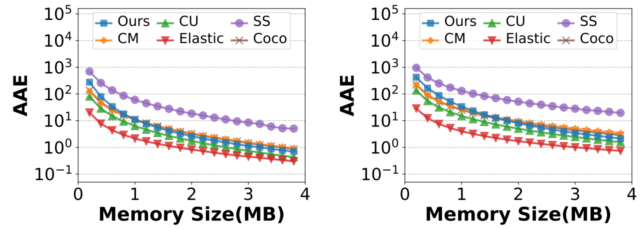
(a) Web Stream **(b) University Data Center**

(c) Synthetic (Skewness=0.3) **(d) Synthetic (Skewness=3.0)**

Figure 6: # Outliers on Different Datasets.

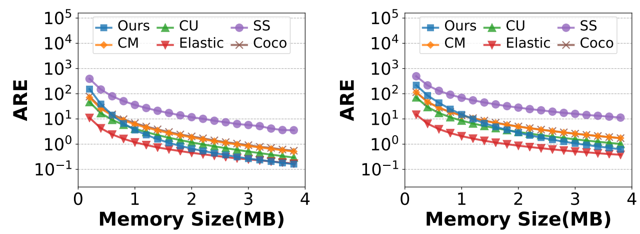
Impact of Dataset (Figure 6a,6b,6c,6d): We fix Λ to 25 and change the dataset. The figures illustrate that ReliableSketch has the least memory requirement regardless of the dataset. For synthetic dataset with skewness=0.3, no algorithm achieves zero outlier within 4MB memory, while the number of outliers of ReliableSketch is over 50 times less than others.

6.2.2 Average Error. Average error measures the average difference between estimated and actual values. ReliableSketch is comparable to the best solutions in this regard. However, optimizing average error is not our primary goal, because its correlation with confidence is relatively low.



(a) IP Trace **(b) Synthetic (Skewness=3.0)**

Figure 7: AAE on Different Datasets.



(a) IP Trace **(b) Synthetic (Skewness=3.0)**

Figure 8: ARE on Different Datasets.

AAE *vs.* Memory Size (Figure 7a, 7b): It is shown that when memory size is up to 4MB, ReliableSketch has a comparable AAE with Elastic and CU in two datasets, is about 1.59 ~ 2.01 times lower than CM, 1.34 ~ 1.69 times lower than Coco, and 9.10 ~ 11.48 times lower than Space-Saving.

ARE *vs.* Memory Size (Figure 8a, 8b): It is shown that when memory size is up to 4MB, ReliableSketch achieves a comparable ARE with Elastic in two datasets, and is 1.63 ~ 2.75 times lower than CU, 2.78 ~ 5.23 times lower than CM, 2.76 ~ 5.05 times lower than Coco, and 18.07 ~ 36.67 times lower than Space-Saving.

6.3 Speed Comparison.

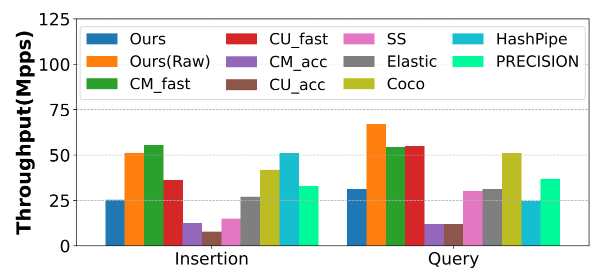


Figure 9: Throughput Evaluation. Ours(Raw) denotes ReliableSketch without the mice filter.

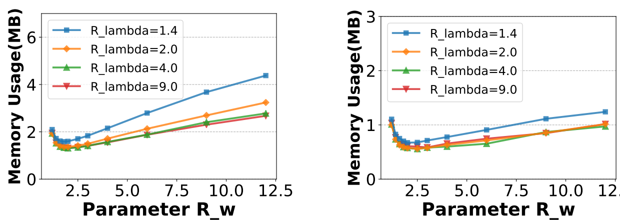
We find that ReliableSketch is not only highly accurate but also fast. In our tests involving 10 million insertions and queries, we compare its throughput with that of other algorithms. An alternative version of ReliableSketch, without the mice filter ("Raw" in the figure), is also presented, sacrificing a tolerable level of accuracy for a significant increase in speed.

Figure 9 shows that the insertion throughputs for ReliableSketch and the Raw version are 25.40 Mpps and 51.29 Mpps, respectively; for queries, they are 31.29 Mpps and 66.89 Mpps. Among baselines that target higher accuracy (SS, Elastic, CM_acc, CU_acc), ReliableSketch achieves comparable or higher throughput. Compared with the fastest-throughput designs (e.g., CM_fast, Coco, HashPipe), ReliableSketch trades some speed for the order-of-magnitude accuracy gains reported in Figure 4 and Figure 6. The Raw ablation further indicates the headroom of our underlying data structure when prioritizing speed over accuracy; we include it as a tradeoff reference rather than our default configuration.

6.4 Impact of Parameters

We explore the impact of various parameters on the accuracy of ReliableSketch, including R_w, R_λ, and the error threshold Λ, and analyze the trends in ReliableSketch's speed changes. At the same time, we provide recommended parameter settings.

6.4.1 Impact of Parameter R_w. When adjusting R_w (*i.e.*, the parameter of the decreasing speed of array size), we find ReliableSketch performs best when $R_w = 2$.



(a) IP Trace **(b) Web Stream**

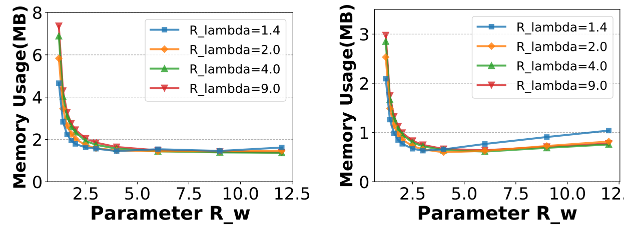
Figure 10: Impact of R_w under Zero Outlier.

Memory Usage under Zero Outlier (Figure 10a, 10b): We conduct experiments on IP Trace and Web Stream datasets. We set the user-defined threshold Λ to 25, and compare the minimum memory consumption achieving zero outlier under different R_w. It is shown that ReliableSketch with $R_w = 2 \sim 2.5$ requires less memory than the ones with other R_w. When R_w is lower than 1.6 or higher than 3, the memory consumption increases rapidly.

Memory Usage under the Same Average Error (Figure 11a, 11b): We conduct experiments on IP Trace and Web Stream datasets, set the target estimation AAE to 5, and compare the memory consumption when R_w varies. The figures show that the higher R_w goes with less memory usage. However, the memory consumption is quite close to the minimum value when $R_w = 2 \sim 6$.

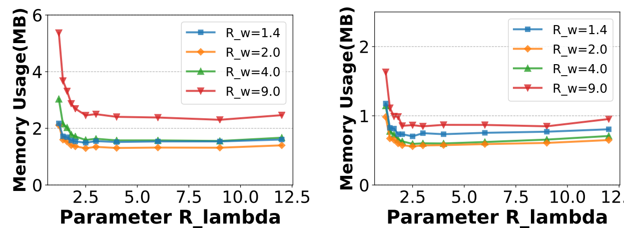
6.4.2 Impact of Parameter R_λ. When adjusting R_λ (*i.e.*, the parameter of the decreasing speed of error threshold), we find ReliableSketch performs best when we set $R_\lambda = 2.5$.

Memory Usage under Zero Outlier (Figure 12a, 12b): We conduct experiments and set the target Λ to 25. As the figures show, memory consumption drops down rapidly when R_λ grows from 1.2 to 2 and finally achieve the minimum when $R_\lambda = 2$. There is no significant change when R_λ is higher than 2.5, only some jitters due to the randomness of ReliableSketch.



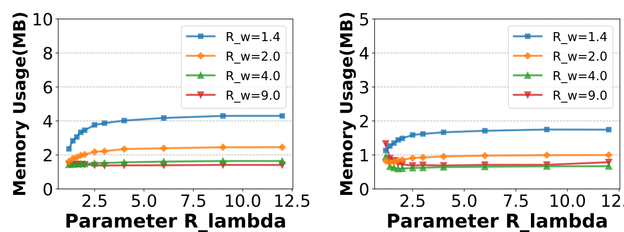
(a) IP Trace **(b) Web Stream**

Figure 11: Impact of R_w under the Same Average Error.



(a) IP Trace **(b) Web Stream**

Figure 12: Impact of R_λ under Zero Outlier.



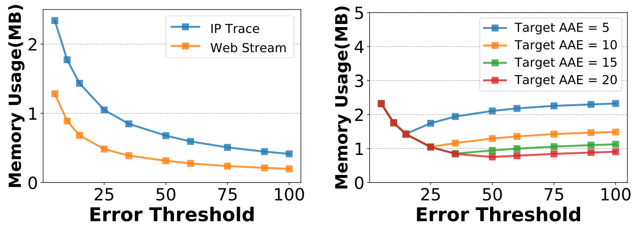
(a) IP Trace **(b) Web Stream**

Figure 13: Impact of R_λ under the Same Average Error.

Memory Usage under the Same Average Error (Figure 13a, 13b): To explore the impact of parameter R_λ on average error, we set the target estimation AAE to 5 and compare the memory consumption when R_λ varies. It is shown that when R_w is low, the higher R_λ is, the less memory ReliableSketch uses. When R_w is greater than 4, R_λ affects little.

6.4.3 Error Threshold Λ. We find that the user-defined error threshold Λ, which denotes the maximum estimated error ReliableSketch guaranteed, is almost inversely proportional to the memory consumption.

Memory Usage under Zero Outlier (Figure 14a): In this experiment, we fix the parameter R_w to 2, R_λ to 2.5, and conduct it on three different datasets. It is shown that memory usage monotonically decreases, which means the optimal Λ is exactly the maximum

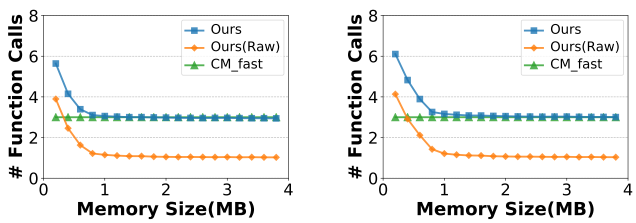


(a) Zero Outlier (b) Overall AAE=5

Figure 14: Memory Usage for Different Λ.

tolerable error. On the other hand, reducing Λ blindly will lead to an extremely high memory cost.

Memory Usage under the Same Average Error (Figure 14b): In this experiment, we fix the parameter R_w to 2, R_λ to 2.5, and conduct it on IP Trace dataset. The figure shows that optimal Λ increases as target AAE increases, and optimal Λ is about $2 \sim 3$ times greater than target AAE. For target AAE=5/10/15/20, the optimums are 15/25/35/50, requiring 1.43/1.05/0.85/0.75MB memory respectively.



(a) Item Insertion (b) Frequency Query

Figure 15: Average Number of Hash Function Calls.

6.4.4 *Trend of speed changes.*

The number of hash function calls, directly proportional to consumed time, fundamentally indicates the trend of speed changes. In ReliableSketch, this number dynamically varies during insertions and queries due to its multi-layer structure. To explore the relationship between memory size and the average number of hash function calls, we conducted experiments using the IP Trace dataset.

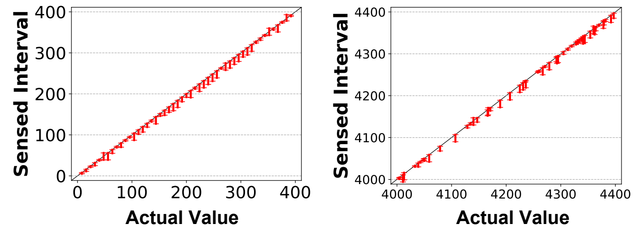
Figures 15a and 15b reveal that the average hash function calls for the raw version of ReliableSketch decrease rapidly with increasing memory, eventually stabilizing at 1. ReliableSketch with a 2-array mice filter eventually stabilizes at 3 due to 2 additional calls in the filter. Smaller ReliableSketch instances record fewer keys in the earlier layers, leading to more hash function calls and reduced throughput. For this reason, unless memory is exceptionally scarce, we recommend allocating more space to gain faster processing speeds.

6.5 In-depth Observations of ReliableSketch

We show how ReliableSketch performs in SenseCtrlErr comprehensively, and compare it with prior algorithms.

6.5.1 *Error-Sensing Ability.*

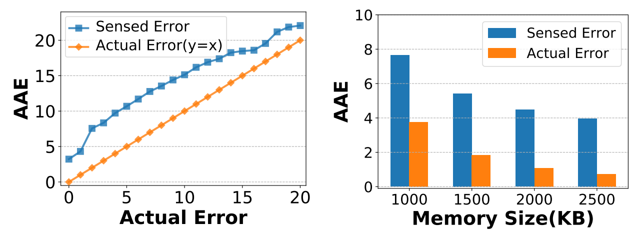
ReliableSketch can confidently and accurately sense the error, the MPE it reports, using the default parameters.



(a) Mice Keys (b) Elephant Keys

Figure 16: Illustration of Sensed Error and Intervals.

Sensed Interval (Figure 16a, 16b): We examine keys with both large and small values to ensure their true values fall within the range [estimated value - MPE, estimated value], thus corroborating that the estimation error is well-controlled within MPE.



(a) vs. Actual Error (b) vs. Memory Size

Figure 17: Experiments on Sensed Error.

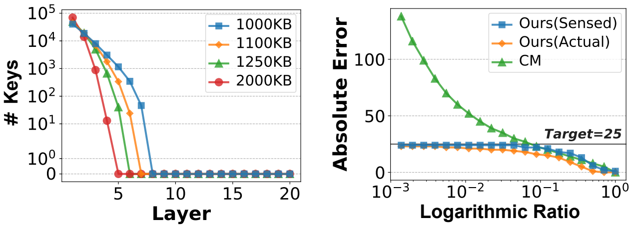
Actual Error *vs.* Sensed Error (Figure 17a): As we query the values of all keys in ReliableSketch, we classify these keys by their actual absolute error, and calculate the average sensed error respectively. The result shows that the average sensed error keeps close to the actual error no matter how it changes, which means ReliableSketch can sense error accurately and stably.

Sensed Error *vs.* Memory Size (Figure 17b): We further vary the memory size from 1000KB to 2500KB, and study how errors change. The figure shows that sensed error decreased when memory grows.

6.5.2 *Error-Controlling Ability.*

ReliableSketch controls error efficiently as our expectation.

Layer Distribution (Figure 18a): When the latest-arriving item of a key concludes its insertion in a particular layer, we categorize the key as belonging to that layer. Through repeated experiments, we calculate the distribution of keys across layers. The results, as depicted in the figure, indicate that the number of keys associated with each layer diminishes at a rate faster than exponential. This suggests that ReliableSketch is capable of effectively controlling errors with only a few layers, and the remaining layers contribute to eliminating potential outliers.

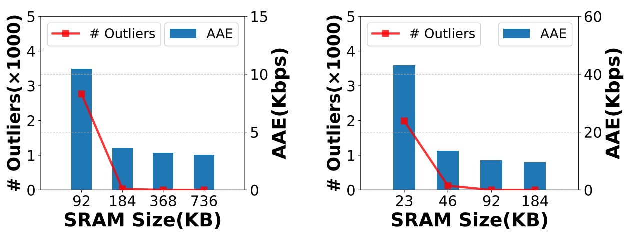


(a) Layer Distribution

(b) Error Distribution

Figure 18: Illustration of Error-Controlling.

Error Distribution (Figure 18b): We count absolute errors of all keys, and sort them in descending order. The figure shows that errors of ReliableSketch are controlled within Λ completely, while most traditional sketch algorithms cannot control the error of all keys, such as CM.



(a) IP Trace

(b) Hadoop

Figure 19: Accuracy on TestBed Deployment.

6.5.3 Deployment. Compared to the flexible CPU platforms, high-performance devices impose more restrictions on algorithm implementation. The more complex the operations used by an algorithm, the less likely it is to be feasibly deployed in reality. We evaluate the accuracy of ReliableSketch implemented on a programmable switch known as Tofino. We send 40 million packets selected from the IP Trace and Hadoop datasets at a link speed of 40Gbps from an end-host connected to the Tofino switch. The evaluation focuses on AAE and the number of outliers for ReliableSketch using SRAM of different sizes.

As depicted in Figure 19, for the IP Trace dataset, ReliableSketch requires more than 368KB of SRAM to ensure zero outliers, maintaining an AAE within 4Kbps. For the Hadoop dataset, more than 92KB of SRAM is necessary for ReliableSketch to guarantee no outliers, with an AAE within 10Kbps.

7 Related Work

Sketches—probabilistic data structures—have been widely adopted for the stream summary problem; when all values equal one, the task reduces to frequency estimation. These approximate aggregations also underpin network and big-data systems (e.g., parameter-server load balancing and large-scale log anomaly detection), where controlling error across all keys—not only per-key—is crucial [13, 23]. We group prior work into three families—counter-based sketches, heap-based sketches, and other sketches—and complement the discussion above with additional references.

Counter-based Sketches are composed of counters, including CM [17], CU [19], Count [14], Elastic [40], UnivMon [27], Coco [41], SALSA [9], DHS [42], SSVS [29] and more [5, 21, 22, 32]. Among them, the work most relevant to ReliableSketch is the Elastic [40], which likewise employs election, with two counters resembling YES and NO. But because Elastic's purpose is to find frequent keys, it resets the NO-like counter to 1 when a replacement occurs, making it incapable of sensing errors. ReliableSketch and Elastic are similar only in appearance, as they differ greatly in their target problems and underlying ideas.

Heap-based Sketches include Frequent [18], Space Saving [30], Unbiased Space Saving [37], SpaceSaving$^{\pm}$ [43] and more [12]. The insertions of these solutions rely on a heap structure, resulting in slower speeds. Compared to their logarithmic time complexity, ReliableSketch achieves an amortized complexity of $O(1 + \Delta \ln \ln(\frac{N}{\Lambda}))$. Heap-based Sketches can only be optimized using a linked list in the special case where the value equals 1, achieving an insertion efficiency of O(1). However, even in the scenario where the value is 1, ReliableSketch still retains its unique advantages, being more suitable for high-performance hardware implementation, while heaps or linked lists are too hard to be implemented [8].

Other Sketches. Besides stream summary, sketches can address various tasks, including estimating cardinality [20, 39], quantiles [24, 28, 34, 45], and join sizes [6, 7, 16, 38], among other tasks [15].

8 Conclusion

We focus on approximating sums of values in data streams for all keys with a high degree of confidence, aiming to prevent the occurrence of outliers with excessive errors. To this end, we have developed ReliableSketch, which provides high-confidence guarantees for all keys. It features near-optimal amortized insertion time of $O(1+\Delta \ln \ln(\frac{N}{\Lambda}))$, near-optimal space complexity $O(\frac{N}{\Lambda} + \ln(\frac{1}{\Delta}))$, and excellent hardware compatibility. Compared to counter-based sketches, ReliableSketch optimizes confidence, speed, and space usage; and against heap-based sketches, it offers superior speed and, despite theoretically larger space requirements, demonstrates more efficient space utilization in practice.

The key idea of ReliableSketch is to identify keys with significant errors and effectively control these errors to completely eliminate outliers. These two steps are facilitated by our key techniques, "Error-Sensible Bucket" and "Double Exponential Control". We have implemented ReliableSketch on CPU servers, FPGAs, and programmable switches. Our experiments indicate that under the same limited space, ReliableSketch not only maintains errors for all keys below Λ but also achieves competitive throughput among accurate baselines, surpassing competitors that struggle with thousands of outliers.

Acknowledgments

This work was supported by the National Key Research and Development Program of China under Grant No. 2024YFB2906601, and in part by the National Natural Science Foundation of China (NSFC) (No. 62372009, 624B2005).

References

[1] [n. d.]. Murmur Hashing source codes. https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp.

[2] [n. d.]. Source code related to ReliableSketch. https://github.com/ReliableSketch/ReliableSketch.

[3] [n. d.]. Frequent itemset mining dataset repository. http://fimi.ua.ac.be/data/.

[4] [n. d.]. The CAIDA Anonymized Internet Traces. http://www.caida.org/data/overview/.

[5] Anup Agarwal, Zaoxing Liu, and Srinivasan Seshan. 2022. {HeteroSketch}: Coordinating network-wide monitoring in heterogeneous and dynamic networks. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 719–741.

[6] Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. 1999. Tracking join and self-join sizes in limited storage. In *Proceedings of the eighteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 10–20.

[7] Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 20–29.

[8] Ran Ben Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. 2020. Designing heavy-hitter detection algorithms for programmable switches. *IEEE/ACM Transactions on Networking* 28, 3 (2020), 1172–1185.

[9] Ran Ben Basat, Gil Einziger, Michael Mitzenmacher, and Shay Vargaftik. 2021. SALSA: self-adjusting lean streaming analytics. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. IEEE, 864–875.

[10] Ran Ben-Basat, Xiaoqi Chen, Gil Einziger, and Ori Rottenstreich. 2018. Efficient measurement on programmable switches using probabilistic recirculation. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*. IEEE, 313–323.

[11] Theophilus Benson, Aditya Akella, and David A Maltz. 2010. Network traffic characteristics of data centers in the wild. In *Proceedings of the 10th ACM SIGCOMM conference on Internet measurement*. 267–280.

[12] Robert S Boyer and J Strother Moore. 1991. MJRTY—a fast majority vote algorithm. In *Automated Reasoning*. Springer, 105–117.

[13] Weibo Cai, Shulin Yang, Gang Sun, Qiming Zhang, and Hongfang Yu. 2023. Adaptive load balancing for parameter servers in distributed machine learning over heterogeneous networks. *ZTE Communications* 21, 1 (March 2023), 72–80. doi:10.12142/ZTECOM.202301009

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

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

[16] Graham Cormode and Minos Garofalakis. 2005. Sketching streams through the net: Distributed approximate query tracking. In *Proceedings of the 31st international conference on Very large data bases*. 13–24.

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

[18] Erik D Demaine, Alejandro López-Ortiz, and J Ian Munro. 2002. Frequency estimation of internet packet streams with limited space. In *European Symposium on Algorithms*. Springer.

[19] Cristian Estan and George Varghese. 2002. New directions in traffic measurement and accounting. In *Proceedings of the 2002 conference on Applications, technologies, architectures, and protocols for computer communications*. 323–336.

[20] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. 2007. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm.

[21] Qun Huang, Xin Jin, Patrick PC Lee, Runhui Li, Lu Tang, Yi-Chao Chen, and Gong Zhang. 2017. Sketchvisor: Robust network measurement for software packet processing. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*. 113–126.

[22] Qun Huang, Siyuan Sheng, Po-Han Wang, Xiang Chen, Chi Zhang, Isaac Pedisich, Yungang Bao, Zhaoyang Han, Dinesh Bharadia, Rui Zhang, et al. 2021. Toward nearly-zero-error sketching via compressive sensing. In *18th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 21)*. 1027–1044.

[23] Yuhe Ji, Jing Han, Yongxin Zhao, Shenglin Zhang, and Zican Gong. 2023. Log anomaly detection through GPT-2 for large scale systems. *ZTE Communications* 21, 3 (Sept. 2023), 70–76. doi:10.12142/ZTECOM.202303010

[24] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*. IEEE, 71–78.

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

[26] Zaoxing Liu, Ran Ben-Basat, Gil Einziger, Yaron Kassner, Vladimir Braverman, Roy Friedman, and Vyas Sekar. 2019. Nitrosketch: Robust and general sketch-based monitoring in software switches. In *Proceedings of the ACM Special Interest Group on Data Communication*. 334–350.

[27] 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 *SIGCOMM*.

[28] Charles Masson, Jee E Rim, and Homin K Lee. [n. d.]. DDSketch: A Fast and Fully-Mergeable Quantile Sketch with Relative-Error Guarantees. *Proceedings of the VLDB Endowment* 12, 12 ([n. d.]).

[29] Dimitrios Melissourgos, Haibo Wang, Shigang Chen, Chaoyi Ma, and Shiping Chen. 2024. Single Update Sketch with Variable Counter Structure. *Proceedings of the VLDB Endowment* 16, 13 (2024).

[30] Ahmed Metwally, Divyakant Agrawal, and Amr El Abbadi. 2005. Efficient computation of frequent and top-k elements in data streams. In *International Conference on Database Theory*. Springer.

[31] J Strother Moore. 1981. A fast majority vote algorithm. *Automated Reasoning: Essays in Honor of Woody Bledsoe* (1981).

[32] Hun Namkung, Zaoxing Liu, Daehyeok Kim, Vyas Sekar, Peter Steenkiste, Guyue Liu, Ao Li, Christopher Canel, Adithya Abraham Philip, Ranysha Ware, et al. 2022. Sketchlib: Enabling efficient sketch-based monitoring on programmable switches. NSDI.

[33] Alex Rousskov and Duane Wessels. 2004. High-performance benchmarking with Web Polygraph. *Software: Practice and Experience* (2004).

[34] Rana Shahout, Roy Friedman, and Ran Ben Basat. 2023. Together is Better: Heavy Hitters Quantile Estimation. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–25.

[35] Vibhaalakshmi Sivaraman, Srinivas Narayana, Ori Rottenstreich, S Muthukrishnan, and Jennifer Rexford. 2017. Heavy-hitter detection entirely in the data plane. In *SOSR*. ACM.

[36] 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 *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*. IEEE, 2026–2034.

[37] Daniel Ting. 2018. Data Sketches for Disaggregated Subset Sum and Frequent Item Estimation. In *SIGMOD Conference*.

[38] Feiyu Wang, Qizhi Chen, Yuanpeng Li, Tong Yang, Yaofeng Tu, Lian Yu, and Bin Cui. 2023. JoinSketch: A Sketch Algorithm for Accurate and Unbiased Inner-Product Estimation. *Proceedings of the ACM on Management of Data* 1, 1 (2023), 1–26.

[39] 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 (TODS)* 15, 2 (1990).

[40] 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 *SIGCOMM*. 561–575.

[41] Yinda Zhang, Zaoxing Liu, Ruixin Wang, Tong Yang, Jizhou Li, Ruijie Miao, Peng Liu, Ruwen Zhang, and Junchen Jiang. 2021. CocoSketch: high-performance sketch-based measurement over arbitrary partial key query. In *Proceedings of the 2021 ACM SIGCOMM 2021 Conference*. 207–222.

[42] Bohan Zhao, Xiang Li, Boyu Tian, Zhiyu Mei, and Wenfei Wu. 2021. Dhs: Adaptive memory layout organization of sketch slots for fast and accurate data stream processing. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 2285–2293.

[43] Fuheng Zhao, Divyakant Agrawal, Amr El Abbadi, and Ahmed Metwally. 2021. SpaceSaving $^{\pm}$: An Optimal Algorithm for Frequency Estimation and Frequent items in the Bounded Deletion Model. *arXiv preprint arXiv:2112.03462* (2021).

[44] Fuheng Zhao, Punnal Ismail Khan, Divyakant Agrawal, Amr El Abbadi, Arpit Gupta, and Zaoxing Liu. 2023. Panakos: Chasing the Tails for Multidimensional Data Streams. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1291–1304.

[45] Fuheng Zhao, Sujaya Maiyya, Ryan Wiener, Divyakant Agrawal, and Amr El Abbadi. 2021. Kll±approximate quantile sketches over dynamic datasets. *Proceedings of the VLDB Endowment* 14, 7 (2021), 1215–1227.

APPENDIX

A Mathematical Proofs

A.1 Preliminaries

We derive a lemma to bound the sum of n random variables. This lemma is similar to the Hoeffding bound but cannot be replaced by Hoeffding.

LEMMA 1. *Let $X_1, \ldots, X_n$ be n random variables such that*

$$X_i \in \{0, s_i\}, \qquad \Pr(X_i = s_i \mid X_1, \cdots, X_{i-1}) \leqslant p,$$

where $0 \leqslant s_i \leqslant 1$. Let $X = \sum_{i=1}^n X_i$, and $\mu = \sum_{i=1}^n ps_i = nmp$.

$$Pr\ (X > (1+\Delta)\mu) \leqslant e^{-(\Delta-(e-2))nmp}$$

PROOF. For any $t > 0$, by using the Markov inequality we have

$$\Pr(X > (1+\Delta)\mu) = \Pr(e^X > e^{(1+\Delta)\mu}) \leqslant \frac{E(e^X)}{e^{(1+\Delta)\mu}}.$$

According to the conditions, we have

$$\begin{aligned} E(e^X) &= E\left(E(e^{\sum_{i=1}^n X_i} \mid X_1, \cdots, X_{n-1})\right) \\ &= E\begin{pmatrix} e^{\sum_{i=1}^{n-1} X_i} \cdot \Pr(X_n = 0 \mid X_1, \cdots, X_{n-1}) \\ + e^{s_n + \sum_{i=1}^{n-1} X_i} \cdot \Pr(X_n = s_n \mid X_1, \cdots, X_{n-1}) \end{pmatrix} \\ &\leqslant E\left(e^{\sum_{i=1}^{n-1} X_i}\right) \cdot (1 + p\,(e^{s_n} - 1)) \leqslant \cdots \leqslant \prod_{i=1}^n (1 + p\,(e^{s_i} - 1)) \end{aligned}$$

Because of $1 + x < e^x$, we have

$$E(e^X) \leqslant \prod_{i=1}^n e^{p(e^{s_i}-1)}.$$

Since for $s_i \leqslant 1$, there is $e^{s_i} - 1 \leqslant (e-1)s_i$, so there is

$$E(e^X) \leqslant e^{\sum_{i=1}^n p(e-1)s_i} = e^{(e-1)mnp}.$$

That is

$$Pr(X > (1+\Delta)\mu) \leqslant \frac{e^{(e-1)nmp}}{e^{(1+\Delta)nmp}} = e^{-(\Delta-(e-2))nmp}$$

□

A.2 Definition of Symbols

(1) $\mathcal{S}_i$: $\{e_1, \cdots, e_{N_i}\}$, the set of keys entering the i-th layer, where $N_i = |\mathcal{S}_i|$.

(2) $f_i(e)$: the number of times that key e enters the i-th layer.

(3) $\mathcal{S}_i^0$: $\{e \mid e \in \mathcal{S}_i \wedge \forall i' \leqslant i, f_{i'}(e) \leqslant \frac{\lambda_{i'}}{2}\}$, the set of mice keys.

(4) $\mathcal{S}_i^1$: $\{e \mid e \in \mathcal{S}_i \wedge \exists i' \leqslant i, f_{i'}(e) > \frac{\lambda_{i'}}{2}\}$, the set of elephant keys.

(5) F_i: $\sum_{\{e \in \mathcal{S}_i^0\}} f_i(e)$, the total frequency of mice keys in $\mathcal{S}_i^0$.

(6) C_i: $|\mathcal{S}_i^1|$, the number of elephant keys in $\mathcal{S}_i^1$.

(7) $\mathcal{S}_{i,j}^0$: $\{e \mid e \in \mathcal{S}_i^0 \wedge h(e) = j\}$, the set of mice keys that are mapped to the j-th bucket.

(8) $\mathcal{S}_{i,j}^1$: $\{e \mid e \in \mathcal{S}_i^1 \wedge h(e) = j\}$, the set of elephant keys that are mapped to the j-th bucket.

(9) $F_{i,j}$: $\sum_{\{e \in \mathcal{S}_{i,j}^0\}} f_i(e)$, the total frequency of mice keys in $\mathcal{S}_{i,j}^0$.

(10) $C_{i,j}$: $|\mathcal{S}_{i,j}^1|$, the number of elephant keys in $\mathcal{S}_{i,j}^1$.

(11) $\mathcal{P}_{i,k}$: $\{e_1, \cdots, e_k\}$, a subset of $\mathcal{S}_i$ composed of the first k keys.

(12) $f_{i,k}^P$: $\sum_{\{e \in \mathcal{P}_{i,k-1} \cap \mathcal{S}_{i,h(e_k)}^0\}} f_i(e)$, the total frequency of mice keys with a smaller index that conflicts with key e_k.

(13) $c_{i,k}^P$: $\left|\{e \mid e \in \mathcal{P}_{i,k-1} \cap \mathcal{S}_{i,h(e_k)}^1\}\right|$, the number of elephant keys with a smaller index that conflicts with key e_k.

A.3 Properties in One Layer

This section aims to prove that only a small proportion of the keys inserted into the i-th layer will be inserted into the (i+1)-th layer.

THEOREM A.1. *(Theorem 1) Let*

$$X_{i,k} = \begin{cases} 0 & C_{i,h(e_k)} = 0 \wedge f_{i,k}^P \leqslant \frac{\lambda_i}{2} \\ f_i(e_k) & C_{i,h(e_k)} = 0 \wedge f_{i,k}^P > \frac{\lambda_i}{2} \\ f_i(e_k) & C_{i,h(e_k)} > 0 \end{cases}, \quad X_i = \sum_{\{e_k \in \mathcal{S}_i^0\}} X_{i,k}.$$

The total frequency of the mice keys in the i-th layer leaving it does not exceed X_i, i.e.,

$$F_{i+1} \leqslant \sum_{\{e \in \mathcal{S}_i^0 \cap \mathcal{S}_{i+1}\}} f_{i+1}(e) \leqslant X_i.$$

PROOF. For the mice keys in the j-th bucket of the i-th layer, let the number of times they leave be $F'_{i,j} = \sum_{\{e \in \mathcal{S}_{i,j}^0 \cap \mathcal{S}_{i+1}^0\}} f_{i+1}(e)$. Since a bucket can hold at least λ_i packets of the key, we have:

$$\begin{cases} F'_{i,j} = 0 & C_{i,j} = 0 \wedge F_{i,j} \leqslant \lambda_i \\ F'_{i,j} \leqslant F_{i,j} - \lambda_i & Cs_{i,j} = 0 \wedge F_{i,j} > \lambda_i \\ F'_{i,j} \leqslant F_{i,j} & C_{i,j} > 0 \end{cases}.$$

When $C_{i,j} = 0 \wedge F_{i,j} > \lambda_i$, exists k' satisfies

$$\sum_{\{e_k \in \mathcal{S}_{i,j}^0 \wedge k < k'\}} f_i(e_k) \leqslant \frac{\lambda_i}{2} \leqslant \sum_{\{e_k \in \mathcal{S}_{i,j}^0 \wedge k \leqslant k'\}} f_i(e_k) \leqslant \lambda_i.$$

Then for and only for any $e_k \in \mathcal{S}_{i,j}^0 \wedge k \leqslant k'$, there is $X_{i,k} = 0$, and

$$\begin{aligned} F'_{i,j} &\leqslant \left(\sum_{\{e_k \in \mathcal{S}_{i,j}^0 \wedge k \leqslant k'\}} f_i(e_k) + \sum_{\{e_k \in \mathcal{S}_{i,j}^0 \wedge k > k'\}} f_i(e_k)\right) - \lambda_i \\ &\leqslant 0 + \sum_{\{e_k \in \mathcal{S}_{i,j}^0 \wedge k > k'\}} f_i(e_k) \\ &\leqslant \sum_{\{e_k \in \mathcal{S}_{i,j}^0 \wedge k \leqslant k'\}} X_{i,k} + \sum_{\{e_k \in \mathcal{S}_{i,j}^0 \wedge k > k'\}} X_{i,k}. \end{aligned}$$

Then we have $F'_{i,j} \leqslant \sum_{\{e_k \in \mathcal{S}_{i,j}^0\}} X_{i,k}$, and

$$\sum_{\{e \in \mathcal{S}_i^0 \cap \mathcal{S}_{i+1}\}} f_{i+1}(e) = \sum_{j=1}^{w_i} F'_{i,j} \leqslant \sum_{j=1}^{w_i} \sum_{\{e_k \in \mathcal{S}_{i,j}^0\}} X_{i,k} = X_i.$$

□

Similarly, we have the following lemma.

Theorem A.2. *Let*

$$Y_{i,k}=\begin{cases}0 & c^P_{i,k}=0\wedge F_{i,h(e_k)}\leqslant\lambda_i,\\ 2 & c^P_{i,k}=0\wedge F_{i,h(e_k)}>\lambda_i\\ 2 & c^P_{i,k}>0.\end{cases},\qquad Y_i=\sum_{e_k\in\mathcal{S}_i^1}Y_{i,k}.$$

The number of distinct elephant keys in the i-th layer leaving it does not exceed Y_i*, i.e.,*

$$\left|\mathcal{S}_i^1\cap\mathcal{S}_{i+1}^1\right|\leqslant Y_i.$$

Proof. For the elephant keys in the j-th bucket of the i-th layer, $\sum_{\{e_k\in\mathcal{S}^1_{i,j}\}}Y_{i,k}<C_{i,j}$ if and only if $C_{i,j}=1\wedge F_{i,j}\leqslant\lambda_i$. In this case, the number of collisions in the bucket does not exceed λ_i, and no key enters the $(i+1)$-th layer. Thus we have $|\mathcal{S}^1_{i,j}\cap\mathcal{S}^1_{i+1}|\leqslant\sum_{\{e_k\in\mathcal{S}^1_{i,j}\}}Y_{i,j}$, and

$$\left|\mathcal{S}_i^1\cap\mathcal{S}_{i+1}^1\right|=\sum_{j=1}^{w_i}|\mathcal{S}^1_{i,j}\cap\mathcal{S}^1_{i+1}|\leqslant\sum_{j=1}^{w_i}\sum_{\{e_k\in\mathcal{S}^1_{i,j}\}}Y_{i,j}=Y_i.$$

□

Theorem A.3. *Let* $W=\frac{4N(R_wR_\lambda)^6}{\Lambda(R_w-1)(R_\lambda-1)}$, $\alpha_i=\frac{\|F\|_1}{(R_wR_\lambda)^{i-1}}$, $\beta_i=\frac{\alpha_i}{\frac{\lambda_i}{2}}$, $\gamma_i=(R_wR_\lambda)^{(2^{i-1}-1)}$, *and* $p_i=(R_wR_\lambda)^{-(2^{i-1}+4)}$. *Under the conditions of* $F_i\leqslant\frac{\alpha_i}{\gamma_i}$ *and* $C_i\leqslant\frac{\beta_i}{\gamma_i}$, *we have:*

$$\Pr\left(X_{i,k}>0\mid X_{i,1},\cdots,X_{i,k-1}\right)\leqslant p_i,\qquad \forall e_k\in\mathcal{S}_i^0.$$
$$\Pr\left(Y_{i,k}>0\mid Y_{i,1},\cdots,Y_{i,k-1}\right)\leqslant\frac{3}{4}p_i,\qquad \forall e_k\in\mathcal{S}_i^1.$$

Proof. By using Markov's inequality, we have

$$\begin{aligned}&\Pr\left(X_{i,k}>0\mid X_{i,1},\cdots,X_{i,k-1}\right)\\
=&\Pr\left(\begin{matrix}\left(C_{i,h(e_k)}=0\wedge f^P_{i,k}>\frac{\lambda_i}{2}\right)\\ \vee\quad C_{i,h(e_k)}>0\end{matrix}\mid X_{i,1},\cdots,X_{i,k-1}\right)\\
\leqslant&\begin{matrix}\Pr\left(C_{i,h(e_k)}>0\mid X_{i,1},\cdots,X_{i,k-1}\right)\\ +\quad\Pr\left(F_{i,h(e_k)}-f_i(e_k)>\frac{\lambda_i}{2}\mid X_{i,1},\cdots,X_{i,k-1}\right)\end{matrix}\\
\leqslant&\begin{matrix}\frac{E(C_{i,h(e_k)}\mid X_{i,1},\cdots,X_{i,k-1})}{1}\\ +\quad\frac{E(F_{i,h(e_k)}-f_i(e_k)\mid X_{i,1},\cdots,X_{i,k-1})}{\frac{\lambda_i}{2}}\end{matrix}\\
\leqslant&\frac{C_i}{w_i}+\frac{2F_i}{\lambda_iw_i}\end{aligned}$$

$$\begin{aligned}&\Pr\left(Y_{i,k}>0\mid Y_{i,1},\cdots,Y_{i,k-1}\right)\\
=&\Pr\left(\left(c^P_{i,k}=0\wedge F_{i,h(e_k)}>\lambda_i\right)\vee c^P_{i,k}>0\mid Y_{i,1},\cdots,Y_{i,k-1}\right)\\
\leqslant&\begin{matrix}\Pr\left(C_{i,h(e_k)}-1>0\mid Y_{i,1},\cdots,Y_{i,k-1}\right)\\ +\quad\Pr\left(F_{i,h(e_k)}>\lambda_i\mid Y_{i,1},\cdots,Y_{i,k-1}\right)\end{matrix}\\
\leqslant&\begin{matrix}\frac{E(C_{i,h(e_k)}-1\mid Y_{i,1},\cdots,Y_{i,k-1})}{1}\\ +\quad\frac{E(F_{i,h(e_k)}\mid Y_{i,1},\cdots,Y_{i,k-1})}{\lambda_i}\end{matrix}\\
\leqslant&\frac{C_i}{w_i}+\frac{F_i}{\lambda_iw_i}.\end{aligned}$$

Recall that $w_i=\lceil\frac{W(R_w-1)}{R_w^i}\rceil$ and $\lambda_i=\frac{\Lambda(R_\lambda-1)}{R_\lambda^i}$, under the conditions of $F_i\leqslant\frac{\alpha_i}{\gamma_i}$ and $C_i\leqslant\frac{\beta_i}{\gamma_i}$, we have

$$\begin{aligned}&\Pr\left(X_{i,k}>0\mid X_{i,1},\cdots,X_{i,k-1}\right)\\
\leqslant&\frac{\beta_i}{\gamma_iw_i}+\frac{2\alpha_i}{\gamma_i\lambda_iw_i}=\frac{4\alpha_i}{\gamma_i\lambda_iw_i}\leqslant\frac{1}{(R_wR_\lambda)^{2^{i-1}+4}}=p_i.\\
&\Pr\left(Y_{i,k}>0\mid Y_{i,1},\cdots,Y_{i,k-1}\right)\\
\leqslant&\frac{\beta_i}{\gamma_iw_i}+\frac{\alpha_i}{\gamma_i\lambda_iw_i}=\frac{3\alpha_i}{\gamma_i\lambda_iw_i}\leqslant\frac{3}{4(R_wR_\lambda)^{2^{i-1}+4}}\leqslant\frac{3}{4}p_i.\end{aligned}$$

□

Theorem A.4. *(Theorem 2) Let* $W=\frac{4N(R_wR_\lambda)^6}{\Lambda(R_w-1)(R_\lambda-1)}$, $\alpha_i=\frac{\|F\|_1}{(R_wR_\lambda)^{i-1}}$, $\beta_i=\frac{\alpha_i}{\frac{\lambda_i}{2}}$, $\gamma_i=(R_wR_\lambda)^{(2^{i-1}-1)}$, *and* $p_i=(R_wR_\lambda)^{-(2^{i-1}+4)}$. *Under the conditions of* $F_i\leqslant\frac{\alpha_i}{\gamma_i}$ *and* $C_i\leqslant\frac{\beta_i}{\gamma_i}$, *we have*

$$\Pr\left(X_i>(1+\Delta)\frac{p_i\alpha_i}{\gamma_i}\right)\leqslant\exp\left(-(\Delta-(e-2))\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right).$$

and

$$\Pr\left(Y_i>(1+\Delta)\frac{3}{2}\frac{p_i\beta_i}{\gamma_i}\right)\leqslant\exp\left(-(\Delta-(e-2))\frac{3p_i\beta_i}{4\gamma_i}\right).$$

Proof. According to Theorem A.3,

$$\Pr\left(\frac{X_{i,k}}{\frac{\lambda_i}{2}}=\frac{f_i(e_k)}{\frac{\lambda_i}{2}}\mid\frac{X_{i,1}}{\frac{\lambda_i}{2}},\cdots,\frac{X_{i,k-1}}{\frac{\lambda_i}{2}}\right)\leqslant p_i.$$
$$\Pr\left(\frac{Y_{i,k}}{2}=1\mid\frac{Y_{i,1}}{2},\cdots,\frac{Y_{i,k-1}}{2}\right)\leqslant\frac{3}{4}p_i.$$

According to Lemma 1,

$$\begin{aligned}&\Pr\left(X_i>(1+\Delta)\frac{p_i\alpha_i}{\gamma_i}\right)\leqslant\Pr\left(X_i>(1+\Delta)p_iF_i\mid F_i=\frac{\alpha_i}{\gamma_i}\right)\\
=&\Pr\left(\sum_{\{e_k\in\mathcal{S}_i^0\}}\frac{X_{i,k}}{\frac{\lambda_i}{2}}>(1+\Delta)p_i\sum_{\{e_k\in\mathcal{S}_i^0\}}\frac{f_i(e_k)}{\frac{\lambda_i}{2}}\mid F_i=\frac{\alpha_i}{\gamma_i}\right)\\
\leqslant&\exp\left(-(\Delta-(e-2))\frac{\alpha_i}{\gamma_i\frac{\lambda_i}{2}}p_i\right)=\exp\left(-(\Delta-(e-2))\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right).\\
&\Pr\left(Y_i>(1+\Delta)\frac{3}{2}\frac{p_i\beta_i}{\gamma_i}\right)\leqslant\Pr\left(Y_i>(1+\Delta)\frac{3}{2}p_iC_i\mid C_i=\frac{\beta_i}{\gamma_i}\right)\\
=&\Pr\left(\sum_{\{e_k\in\mathcal{S}_i^1\}}\frac{Y_{i,k}}{2}>(1+\Delta)\frac{3}{4}p_i\sum_{\{e_k\in\mathcal{S}_i^1\}}\frac{2}{2}\mid C_i=\frac{\beta_i}{\gamma_i}\right)\\
\leqslant&\exp\left(-(\Delta-(e-2))\frac{\beta_i}{\gamma_i}\frac{3}{4}p_i\right)=\exp\left(-(\Delta-(e-2))\frac{3p_i\beta_i}{4\gamma_i}\right).\end{aligned}$$

□

Theorem A.5. *(Theorem 3) Let $R_wR_\lambda \geqslant 2$, $W = \frac{4N(R_wR_\lambda)^6}{\Lambda(R_w-1)(R_\lambda-1)}$, $\alpha_i = \frac{\|F\|_1}{(R_wR_\lambda)^{i-1}}$, $\beta_i = \frac{\alpha_i}{\frac{\lambda_i}{2}}$, $\gamma_i = (R_wR_\lambda)^{(2^{i-1}-1)}$, and $p_i = (R_wR_\lambda)^{-(2^{i-1}+4)}$. We have*

$$\Pr\left(F_{i+1} > \frac{\alpha_{i+1}}{\gamma_{i+1}} \mid F_i \leqslant \frac{\alpha_i}{\gamma_i} \wedge C_i \leqslant \frac{\beta_i}{\gamma_i}\right) \leqslant \exp\left(-(9-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right).$$

$$\Pr\left(C_{i+1} > \frac{\beta_{i+1}}{\gamma_{i+1}} \mid F_i \leqslant \frac{\alpha_i}{\gamma_i} \wedge C_i \leqslant \frac{\beta_i}{\gamma_i}\right) \leqslant \exp\left(-(5-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right) + \exp\left(-(\frac{11}{3}-e)\frac{3p_i\beta_i}{4\gamma_i}\right).$$

Proof. According to settings, we have

$$p_i\frac{\alpha_i}{\gamma_i} = \frac{\|F\|_1}{(R_wR_\lambda)^{(2^i+i+2)}} \leqslant \frac{1}{8}\frac{\alpha_{i+1}}{\gamma_{i+1}}$$

$$p_i\frac{\beta_i}{\gamma_i} = p_i\frac{\alpha_i}{\gamma_i\frac{\lambda_i}{2}} \leqslant \frac{1}{8}\frac{\alpha_{i+1}}{\gamma_{i+1}\frac{\lambda_{i+1}}{2}} = \frac{1}{8}\frac{\beta_{i+1}}{\gamma_{i+1}}.$$

Recall that $C_{i+1} = |\mathcal{S}^1_{i+1} \cap \mathcal{S}^0_i| + |\mathcal{S}^1_{i+1} \cap \mathcal{S}^1_i|$, and

$$|\mathcal{S}^1_{i+1} \cap \mathcal{S}^0_i| \leqslant \frac{\sum_{\{e\in\mathcal{S}^0_i\cap\mathcal{S}_{i+1}\}} f_i(e)}{\frac{\lambda_{i+1}}{2}} \leqslant \frac{X_i}{\frac{\lambda_{i+1}}{2}}$$

Let $\Gamma_i = \left(F_i \leqslant \frac{\alpha_i}{\gamma_i} \wedge C_i \leqslant \frac{\beta_i}{\gamma_i}\right)$, according to Theorem A.1 and Theorem 2,

$$\Pr(F_{i+1} > \frac{\alpha_{i+1}}{\gamma_{i+1}} \mid \Gamma_i) \leqslant \Pr\left(X_i > 8p_i\frac{\alpha_i}{\gamma_i} \mid \Gamma_i\right) \leqslant \exp\left(-(9-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right).$$

According to Theorem A.2 and Theorem 2,

$$\begin{aligned}
&\Pr(C_{i+1} > \frac{\beta_{i+1}}{\gamma_{i+1}} \mid \Gamma_i)\\
=&\Pr(|\mathcal{S}^1_{i+1} \cap \mathcal{S}^0_i| + |\mathcal{S}^1_{i+1} \cap \mathcal{S}^1_i| > \frac{\beta_{i+1}}{\gamma_{i+1}} \mid \Gamma_i)\\
\leqslant&\Pr(|\mathcal{S}^1_{i+1} \cap \mathcal{S}^0_i| > \frac{\beta_{i+1}}{2\gamma_{i+1}} \vee |\mathcal{S}^1_{i+1} \cap \mathcal{S}^1_i| > \frac{\beta_{i+1}}{2\gamma_{i+1}} \mid \Gamma_i)\\
\leqslant&\Pr(\frac{X_i}{\frac{\lambda_{i+1}}{2}} > \frac{\beta_{i+1}}{2\gamma_{i+1}} \mid \Gamma_i) + \Pr(Y_i > \frac{\beta_{i+1}}{2\gamma_{i+1}} \mid \Gamma_i)\\
\leqslant&\Pr(X_i > 4p_i\frac{\alpha_i}{\gamma_i} \mid \Gamma_i) + \Pr(Y_i > 4p_i\frac{\beta_i}{\gamma_i} \mid \Gamma_i)\\
\leqslant&\exp\left(-(5-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right) + \exp\left(-(\frac{11}{3}-e)\frac{3p_i\beta_i}{4\gamma_i}\right).
\end{aligned}$$

□

A.4 Space and Time Complexity

Theorem A.6. *(Theorem 4) Let $R_wR_\lambda \geqslant 2$, $W = \frac{4N(R_wR_\lambda)^6}{\Lambda(R_w-1)(R_\lambda-1)}$, $\alpha_i = \frac{\|F\|_1}{(R_wR_\lambda)^{i-1}}$, $\beta_i = \frac{\alpha_i}{\frac{\lambda_i}{2}}$, $\gamma_i = (R_wR_\lambda)^{(2^{i-1}-1)}$, and $p_i = (R_wR_\lambda)^{-(2^{i-1}+4)}$. For given Λ and $\Delta < \frac{1}{4}$, let d be the root of the following equation*

$$\frac{R_\lambda^d}{(R_wR_\lambda)^{(2^d+d)}} = \Delta_1\frac{\Lambda}{N}\ln(\frac{1}{\Delta}).$$

And use an SpaceSaving of size $\Delta_2\ln(\frac{1}{\Delta})$ (as the $(d+1)$-layer), then

$$\Pr\left(\forall \text{ item } e, \left|\hat{f}(e) - f(e)\right| \leqslant \Lambda\right) \geqslant 1-\Delta,$$

where

$$\Delta_1 = 2R_w^2R_\lambda^2(R_\lambda-1), \qquad \Delta_2 = 3\left(\frac{R_wR_\lambda^2}{R_\lambda-1}\right)\Delta_1 = 6R_w^3R_\lambda^4.$$

Proof. Recall that $\Gamma_i = \left(F_i \leqslant \frac{\alpha_i}{\gamma_i} \wedge C_i \leqslant \frac{\beta_i}{\gamma_i}\right)$, When all conditions Γ_i (including Γ_{d+1}) are true, we have

$$C_{d+1} \leqslant \frac{\beta_{d+1}}{\gamma_{d+1}} = \frac{2NR_\lambda^{d+1}}{(R_wR_\lambda)^{(2^d+d-1)}(R_\lambda-1)\Lambda} = \left(\frac{2R_wR_\lambda^2}{R_\lambda-1}\right)\Delta_1\ln(\frac{1}{\Delta}).$$

$$F_{d+1} \leqslant \frac{\alpha_{d+1}}{\gamma_{d+1}} = \frac{\lambda_{d+1}}{2}\frac{\beta_{d+1}}{\gamma_{d+1}} = \lambda_{d+1}\left(\frac{R_wR_\lambda^2}{R_\lambda-1}\right)\Delta_1\ln(\frac{1}{\Delta}).$$

Since we use an SpaceSaving of size $\Delta_2\ln(\frac{1}{\Delta}) > C_{d+1}$, it can record all elephant keys without error, and the estimation error for mice keys does not exceed

$$\frac{F_{d+1}}{\Delta_2\ln(\frac{1}{\Delta}) - C_{d+1}} \leqslant \frac{\lambda_{d+1}\left(\frac{R_wR_\lambda^2}{R_\lambda-1}\right)\Delta_1\ln(\frac{1}{\Delta})}{\Delta_2\ln(\frac{1}{\Delta}) - \left(\frac{2R_wR_\lambda^2}{R_\lambda-1}\right)\Delta_1\ln(\frac{1}{\Delta})} = \lambda_{d+1}$$

Therefore, for any item e,

$$\left|\hat{f}(e) - f(e)\right| = \sum_{i=1}^{d}\lambda_i \leqslant \sum_{i=1}^{\infty}\frac{\Lambda(R_\lambda-1)}{R_\lambda^i} = \Lambda$$

Next, we deduce the probability that at least one condition Γ_i is false. Note that

$$\left.\begin{array}{l}
(\frac{11}{3}-e)\frac{3p_i\beta_i}{4\gamma_i}\\
(9-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\\
(5-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}
\end{array}\right\} \geqslant \frac{p_i\alpha_i}{\lambda_i\gamma_i}.$$

Then According to Theorem A.5, we have

$$\begin{aligned}
&\Pr\left(\neg\left(\bigwedge_{i=1}^{d}\Gamma_{i+1}\right)\right) = \Pr\left(\bigvee_{i=1}^{d}\neg\Gamma_{i+1}\right) = \Pr\left(\bigvee_{i=1}^{d}\left(\bigwedge_{j=1}^{i}\Gamma_j \wedge \neg\Gamma_{i+1}\right)\right)\\
\leqslant&\sum_{i=1}^{d}\Pr\left(\Gamma_i \wedge \neg\Gamma_{i+1}\right) \leqslant \sum_{i=1}^{d}\Pr\left(\neg\Gamma_{i+1} \mid \Gamma_i\right)\\
\leqslant&\sum_{i=1}^{d}\left(\begin{array}{r}\exp\left(-(\frac{11}{3}-e)\frac{3p_i\beta_i}{4\gamma_i}\right)\\ +\exp\left(-(9-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right) + \exp\left(-(5-e)\frac{2p_i\alpha_i}{\lambda_i\gamma_i}\right)\end{array}\right)\\
\leqslant&\sum_{i=1}^{d}3\exp\left(-\frac{p_i\alpha_i}{\lambda_i\gamma_i}\right).
\end{aligned}$$

Note that

$$\begin{aligned}\exp\left(-\frac{p_d\alpha_d}{\lambda_d\gamma_d}\right) &= \exp\left(-\frac{NR_\lambda^d}{(R_wR_\lambda)^{(2^d+d+2)}\Lambda(R_\lambda-1)}\right)\\ &= \exp\left(-\frac{1}{R_w^2R_\lambda^2(R_\lambda-1)}\Delta_1\ln(\frac{1}{\Delta})\right)\\ &= \Delta^{\left(\frac{1}{R_w^2R_\lambda^2(R_\lambda-1)}\Delta_1\right)} = \Delta^2.\end{aligned}$$

Since $\Delta \leqslant 1$, and the monotonicty of $\exp\left(-\frac{p_d\alpha_d}{\lambda_d\gamma_d}\right)$, we have

$$\begin{aligned}\exp\left(-\frac{p_i\alpha_i}{\lambda_i\gamma_i}\right) &= \exp\left(-\frac{p_{i+1}\alpha_{i+1}}{\lambda_{i+1}\gamma_{i+1}}\cdot R_w^{(2^i+1)}R_\lambda^{(2^i)}\right)\\ &\leqslant \exp\left(-\frac{p_{i+1}\alpha_{i+1}}{\lambda_{i+1}\gamma_{i+1}}\right)^{R_wR_\lambda} \leqslant \exp\left(-\frac{p_{i+1}\alpha_{i+1}}{\lambda_{i+1}\gamma_{i+1}}\right)^2\\ &\leqslant \Delta^2\exp\left(-\frac{p_{i+1}\alpha_{i+1}}{\lambda_{i+1}\gamma_{i+1}}\right)\end{aligned}$$

Therefore, we have

$$\sum_{i=1}^{d} 3\exp\left(-\frac{p_i\alpha_i}{\lambda_i\gamma_i}\right) \leqslant 3\sum_{i=1}^{d}\Delta^{2i} \leqslant \left(\frac{3\Delta}{1-\Delta^2}\right)\Delta \leqslant \Delta.$$

In other words,

$$\Pr\left(\forall\ item\ e, \left|\hat{f}(e)-f(e)\right| \leqslant \Lambda\right) \geqslant 1-\Delta,$$

which leads to a weaker conclusion,

$$\forall\ item\ e, \Pr\left(\left|\hat{f}(e)-f(e)\right| \leqslant \Lambda\right) \geqslant 1-\Delta.$$

□

Theorem A.7. *Using the same settings as Theorem A.6, the space complexity of the algorithm is $O(\frac{N}{\Lambda}+\ln(\frac{1}{\Delta}))$, and the time complexity of the algorithm is amortized $O(1+\Delta\ln\ln(\frac{N}{\Lambda}))$.*

Proof. Recall that d is the root of the equation

$$\frac{R_\lambda^d}{(R_wR_\lambda)^{(2^d+d)}} = \Delta_1\frac{\Lambda}{N}\ln(\frac{1}{\Delta}),$$

which means $d = O\left(\ln\ln(\frac{N}{\Lambda})\right)$. Therefore, total space used by the data structure is

$$\begin{aligned}\sum_{i=1}^{d} w_i + \Delta_1\ln(\frac{1}{\Delta}) &= \sum_{i=1}^{d}\lceil\frac{W(R_w-1)}{R_w^i}\rceil + O(\ln(\frac{1}{\Delta}))\\ &\leqslant \frac{4N(R_wR_\lambda)^6}{\Lambda(R_w-1)(R_\lambda-1)} + d + O(\ln(\frac{1}{\Delta}))\\ &= O(\frac{N}{\Lambda}+\ln(\frac{1}{\Delta}))\end{aligned}$$

Next, we analyze the time complexity. When all condition Γ_i are true, for a new item $e \notin \mathcal{S}_1$, the probability that it enters the $(i+1)$-th layer from the i-th layer is $\frac{\frac{F_i}{\frac{\lambda_2}{2}}+C_i}{w_i} \leqslant p_i$. Thus the time complexity of insert item e does noes exceed

$$(1-\Delta)\cdot(1+\sum_{i=1}^{d}p_i) + \Delta\cdot d = O(1+\Delta\ln\ln(\frac{N}{\Lambda})).$$

□

B Ethics

This work does not raise any ethical issues.