# SketchConf: A Framework for Automatic Sketch Configuration

Ruijie Miao*, Fenghao Dong*, Yikai Zhao*, Yiming Zhao*, Yuhan Wu*, Kaicheng Yang*, Tong Yang*†, Bin Cui*
* School of Computer Science, and National Engineering Laboratory for Big Data Analysis Technology and Application, Peking University, Beijing, China
† Peng Cheng Laboratory, Shenzhen, China

***Abstract*—Sketches have risen as promising solutions for frequency estimation, which is one of the most fundamental tasks in approximate data stream processing. In many scenarios, users have a strong demand to apply sketches under the expected error constraints. In this paper, we explore how to configure sketch parameters to satisfy user-defined error constraints. We propose SketchConf, an automatic sketch configuration framework, which efficiently generates memory-optimal configurations for the first time. We show that SketchConf can be applied to order-independent sketches, including CM, Count, Tower, and Nitro sketches. We further discuss how to deal with the unknown and changeable workloads when applying SketchConf to the real scenarios of streaming data processing. Experimental results show that SketchConf can be up to $715.51$ times faster than the baseline algorithm, and the outputted configurations save up to $99.99\%$ memory and achieve up to $27.44$ times throughput, compared with the theory-based configurations. The code is open sourced at Github.**

***Index Terms*—Data Stream; Frequency Estimation; Sketch; Configuration**

## I. Introduction

Approximate data stream processing [1] has received considerable research interests, and frequency estimation is one of the most fundamental tasks. With sub-linear space complexity and $O(1)$ update/query time, sketching algorithms are considered as promising solutions for frequency estimation [2], [3], [4], [5], [6], [7]. Take the CM sketch [8], one of the most widely used sketches, as an example. The CM sketch consists of a $d \times w$ matrix of counters. For an insertion, each row uses a hash function to locate and increment one counter. For a query, the same hash functions are used in $d$ rows to locate $d$ counters, and the minimum value is reported. Although the estimated frequency of sketches has errors, users in many scenarios have a strong demand to apply sketching algorithms within the expected error constraints. Such scenarios include streaming databases [9], [10], natural language processing [11], [12], network measurement [13], [14], and more [15], [16], [17], [18], [19], [20], [21], [22].

In this paper, we study the following problem: providing appropriate sketch parameters configuration in order to satisfy user-defined error constraints. Typically, the sketch parameters include the size of rows and columns of the counter matrix, as $d$ and $w$ in the CM sketch. At the first glance, this problem seems simple. However, appropriate sketch configurations are actually critical for applications of sketches in most scenarios where error constraints exist. An inappropriate configuration may lead to violation of error constraints or drop in sketch performance. For instance, the existing work on finding top-k items [23] uses sketches to count the frequency of items and maintain a heap of size $k$ for top-k items. If the $d$ and $w$ are set too small, the accuracy of sketches decreases, which may misclassify the infrequent items as top-k items. If the $d$ and $w$ are set too large, sketches suffer from not only high memory overhead, but also low throughput, resulting in inefficiency when dealing with large data. Although existing solutions claim themselves to perform well, their performances are often measured against well-chosen configurations, and how to choose configurations is not well addressed.

To provide appropriate sketch parameters configuration, two problems should be dealt with. First, given a parameter configuration, we need to answer whether the error constraints can be satisfied. Second, typical sketching algorithms (*e.g.*, CM sketches) involve multiple parameters that should be configured simultaneously, which means a large choice space. Among the configurations that satisfy error constraints, the optimal one should be selected. Below we show how existing solutions address these two problems, and analyze their weakness as well.

Existing solutions on sketch configuration can be divided into two categories: theory-based solutions and expert-tuned solutions. Theory-based solutions enable users to calculate sketch configurations with simple mathematical formulas. However, theory-based solutions are based on theoretical error bounds, which are usually much looser than real errors in most workloads. Therefore, theory-based solutions provide conservative answers to the question whether a configuration satisfies error constraints. As a consequence, we observe significant memory overhead and throughput drop for theory-based configurations. With error constraints that $Pr\{Error > 500\} < 0.1$, for the synthetic dataset with Zipf skewness parameter $0.5$, the theory-based CM sketch takes $2.08$MB, with an insertion throughput of $19.42$M items/s, while a manually-adjusted configuration takes $395$KB, with an insertion throughput of $46.07$M items/s. Similar phenomenon exists for Count sketches, and more detailed comparison is provided in §II-C.

Co-primary authors: Ruijie Miao, Fenghao Dong, Yikai Zhao. The corresponding author: Tong Yang (yangtongemail@gmail.com).

On the other hand, expert-tuned solutions ask for experts to manually tune sketch parameters. To check whether error constraints can be satisfied for a specific configuration, expert-tuned solutions repeatedly conduct experimental testing on a test dataset sampled from real world workloads. With careful parameter tuning, the result sketch configuration can be much better than the theory-based one, reducing memory consumption and speeding up insertion. However, as the choice space is large, the fine-grained parameter tuning procedure is slow and involves a large amount of manual labor. Besides, existing work on workload analysis shows that workload characteristics vary over time [24], [25] or when the applied scenarios change [26], [27]. In response to the workload change, experts should reconfigure the sketch, which requires even more manual work.

Ideally, the sketch configuration framework should achieve three following requirements:

- **Automatic**. The configuration procedure should not involve manual labor.
- **Optimal**. The provided configuration should maximize the sketch performance.
- **Fast**. The time cost of configuration procedure should be low.

In this paper we aim to generate memory-optimal configuration, which minimizes the total memory consumption. To search for memory-optimal configuration, a naive solution is to automate the procedure of manually tuning parameters: conduct binary search on memory, and for each memory value enumerate parameter choices and check whether the error constraints are satisfied, until the minimum feasible memory value is found. For the check of error constraints, the naive solution inserts testing dataset to sketches and compares the query results with ground truth, and repeats this process for multiple times. We find the naive solution suffers from poor time efficiency, because: (1) the check of error constraints is time-consuming, and (2) the total search time scales up with the number of checked configurations.

To overcome the low time efficiency of the naive solution, we propose our sketch configuration framework, SketchConf. For the fast check of error constraints, SketchConf utilizes **Monte Carlo simulation** technique to build an efficient error predictor for *order-independent sketches*. Typical sketching algorithms are *order-independent* [1], which means that the estimation does not depend on the arriving order of items. This property enables efficient Monte Carlo simulation for sketch errors. First, for order-independent sketches, the items in the stream can be swapped, and we can logically group items with the same ID together as $< iD, frequency >$ pairs. Instead of running simulation item-by-item, we can simulate pair-by-pair, which greatly reduce complexity. Second, for order-independent sketches, the update of counters is independent with each other. Therefore, Monte Carlo simulation of colliding items in a single counter is enough to predict errors, which avoids unnecessary hash computation and improves cache utility.

To break the scaling time cost with the checked configurations, we propose the technique of **reusing simulation results**. After analysis of the Monte Carlo simulation process, we find that the tail error conditioning on the number of distinct colliding items $n$ (*e.g.*, $Pr\{Error > X \mid n\}$) depends only on the workload characteristics, which remains unchanged across the search. This implies that we can break tail error probability into conditional probabilities, simulate conditional probabilities only once, and reuse the simulation results throughout the searching process. By reusing Monte Carlo simulation results, the time cost is considerably reduced.

We propose an optimization to further accelerate Monte Carlo simulation for skewed data distribution, which is common in the real world datasets [2], [26], [30], [31], [32]. Infrequent items contribute little to the errors but require much time to simulate, and we can separate the simulation of frequent and infrequent items. For infrequent items, we conduct limited rounds of simulation, and hence the time cost can be further reduced. We further discuss how to deal with the challenges when applying SketchConf in real scenarios of data stream processing, including: (1) unknown workload characteristics, (2) overhead of frequent re-configuration of sketches, and (3) violation of users demand due to the abrupt change of workload characteristics. We provide solutions for estimating the workload information, sketch error monitoring and urgent re-configuration of sketches, respectively.

Our main contribution can be summarized as follows:

- We propose a Monte-Carlo-based error predictor to estimate errors for order-independent sketches. Based on the error predictor, we propose SketchConf, an automatic sketch configuration framework that generates memory-optimal configurations efficiently (§III).
- We show how to apply our framework to order-independent sketches, including sketches of CM, Count, Tower and Nitro (§IV).
- We discuss the challenges and solutions when applying SketchConf in the real scenarios with unknown and changeable workloads (§V).
- We evaluate SketchConf in the synthetic datasets and real world datasets (§VI). Our experimental results show that SketchConf can be up to $715.51$ times faster than the baseline algorithm, and the outputted configurations save up to $99.99\%$ memory and achieve up to $27.44$ times throughput, compared with the theory-based configurations.

## II. Background and Related work

### A. Problem Statement

We define the problem of *sketch parameters configuration*, which takes the following information as input:

- **Workload**. The required workload information includes the frequency distribution of data stream and the number of distinct items.

[1]Previous work [28], [29] has proposed similar definition.

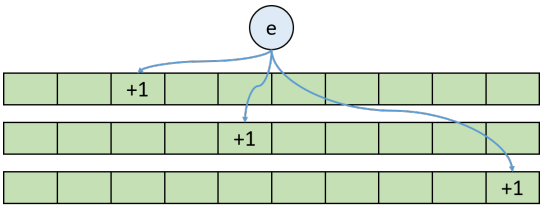


Fig. 1: A CM sketch with $d = 3$ and $w = 10$.

- **Sketch type**. Users specify the sketch type, *e.g.*, the CM sketch, the Count sketch. In this paper we consider sketches that are order-independent.
- **Error constraints**. The error constraints are conveyed in the form of tail error bounds, *e.g.*, $Pr\{Error > X\} < \delta$. This means that with probability less than $\delta$, the error of estimation will be larger than $X$. There can be multiple error constraints for one running sketch, *e.g.*, $Pr\{Error > X_1\} < \delta_1, Pr\{Error > X_2\} < \delta_2, \cdots, Pr\{Error > X_n\} < \delta_n$.

The output of sketch parameters configuration is the memory-optimal configuration. If multiple feasible configurations have the same memory, the one with minimal hash computation is outputted.

We explain the reasons behind the problem definition in the follows. First, we decide to minimize memory, because it improves the throughput at the same time. Existing work [33] shows that memory access time comprises a large portion of sketch processing time, and minimizing memory accelerates memory access by improving cache utility. Second, the error constraints are expressed in the form of one or multiple tail error bounds. A more general way is to convey a complete error distribution. However, in real scenario users usually only cares some tail error probabilities. For example, users may expect $99\%$ quantile of error to be less than $100$, which can be express by tail error probability $Pr\{Error > 100\} < 0.01$. It is also widely accepted in the research community, such as [34]. Third, workload characteristics are acquirable. We notice that existing work [35], [36] propose diverse methods to acquire accurate estimation of the number of distinct items and frequency distribution. And we will further discuss how to deal with the unknown workload characteristics in §V.

### B. Related work

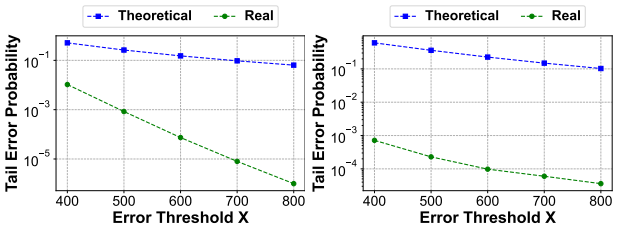
*1) Sketching algorithms:*
Sketches are probabilistic data structure, which can provide approximate answers for frequency estimation tasks. Many sketches have similar data structure and operation design. Figure 1 shows one typical sketch, CM sketches [8]. The CM sketch comprises of a $d \times w$ counter matrix. When an item arrives, each row uses an independent hash function to map it to one counter and increments it by 1. To query an item's frequency, the CM sketch reports the minimum value of $d$ hashed counters. We find CM sketches are order-independent: swapping two items in the data stream does not impact the result value of the counter matrix, and therefore does not impact the frequency estimation. In addition to CM sketches, there are many typical sketches that are order-independent, including sketches of Count [37], Tower [38], Nitro [33], and more [39], [40].

Sketching algorithms can be applied in diverse downstream applications [41]. One typical application is finding top-k items, which aims to find $k$ elements with the highest frequency. Sketch-based solutions [23] will use sketches as auxiliary data structures to estimate items' frequency, and maintain a heap of size $k$ for top-k items. The primary goal is to find top-k items accurately and fast, and the sketch configurations are vital to the performance. While the state-of-the-art solutions perform well, their performances are often measured under well-chosen configurations. Therefore, the downstream applications will benefit from an automatic sketch configuration framework. Similar applications using sketches also includes join size estimation [42], feature extraction [43].

Recently the idea of learning-based sketches has been proposed [44] to enhance the performance of sketches. Learning-based sketches utilize machine learning to classify heavy-hitters, and record heavy-hitters with buckets and the non-heavy-hitters with sketching algorithms. For learning-based sketches, the sketch part of non-heavy-hitters still needs appropriate configurations.

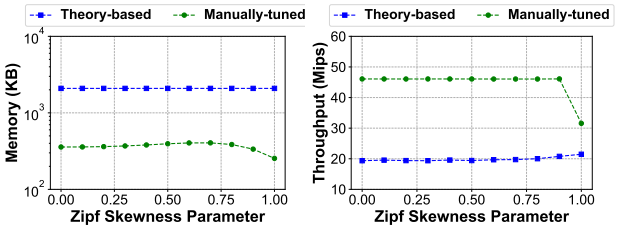
*2) Error estimation for sketches:*
Prior work on error estimation for sketches can be divided into two categories: theoretical and statistical. Most efforts towards error estimation for sketches reside on the theoretical analyses. Sketches usually provide theoretical error bounds [8], [37], [38], [33]. Moreover, prior work [45] also presents refined error bounds for the Count sketch, which is based on asymptotic analysis. For statistical methods, Rusu et. al [42] proposes to use statistical analysis and argues that in some cases theoretical bounds can be orders of magnitude worse than statistical results. Chen et. al [46] proposes a solution, which provides precise error estimation of sketch results at the query time. To the best of our knowledge, it is so far the most efficient algorithms to estimate error of sketch results.

*3) Sketch configuration:*
Providing sketch configuration under user specified requirements has been studied by the research community. OpenSketch [13] automatically allocates resources across sketches. The goal of OpenSketch is to minimize the error rate with the given total memory, which is based on the theoretical error bounds. HeteroSketch [34] explores sketch configuration under heterogeneity of hardware. For error constraints, HeteroSketch also uses theoretical error bounds. When configuring one sketch on a single node, the above solutions are the same as theory-based configurations, which are far from optimal. SCREAM [47] proposes a dynamic resource allocator, which dynamically adjusts the allocated resource for each sketch according to the instantaneous accuracy. If the accuracy is below the user specified accuracy bound, more resource is allocated; otherwise, more resource is allocated. SCREAM provides tighter but still non-optimal error bounds, as it does not rely on traffic characteristics. Besides, it is designed for heavy-hitter and heavy-change detection, while we focus on

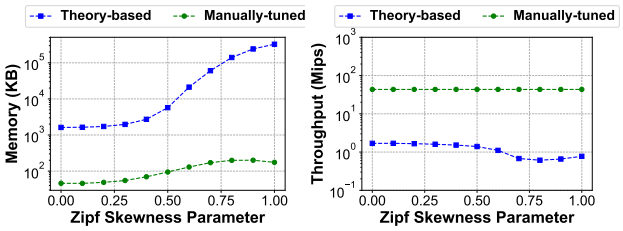



(a) CM sketches (b) Count sketches

Fig. 2: Gaps between theoretical error bounds and real tail errors.




(a) CM sketch - Memory (b) CM sketch - Throughput




(c) Count sketch - Memory (d) Count sketch - Throughput

Fig. 3: Comparison of theory-based and manually-tuned configurations.

frequency estimation. In a word, our SketchConf is different from prior work in the aspect of the optimal configuration.

### C. Motivation

Theory-based solutions are the state-of-the-art method to automatically configure sketches. We show limitations of theory-based configurations as follows.

**Gap between theoretical error bounds and real tail errors.** Theory-based configurations are based on the theoretical error bounds for $Pr\{Error > X\}$. However, there is a huge gap between theoretical error bounds and real tail errors. We conduct experiments on the synthetic dataset that follows Zipf distribution with skewness parameter $0.5$. We compare theoretical error bounds [8], [48] with real errors, and results are shown in Figure 2. For CM sketches, the theoretical error bounds are $1-4$ orders of magnitude looser than the real tail errors. Similarly, for Count sketches, the theoretical error bounds are $3-4$ orders of magnitude looser.

**Performance drop caused by theory-based configurations:** Theory-based configurations are far from optimal. we measure the memory overhead and drop of insertion throughput caused by theory-based configurations. We use the synthetic datasets with Zipf skewness parameter ranging from $0.0$ to $1.0$. We set the error constraint $Pr\{Error > 500\} < 0.1$, and compare the theory-based configurations [8], [48] with manually-tuned configurations that exactly satisfy the error constraint. As shown in Figure 3, theory-based configurations consumes more memory and have much lower throughput. For CM sketches, compared with manually-tuned configurations, the theory-based configurations consume $5.2-8.2\times$ memory, and have $1.5-2.4$ times lower throughput. For Count sketches, the theory-based configurations are even worse, consuming $35.3-1869.7\times$ memory and having $25.6-70.8$ times lower throughput.

For sketching algorithms, improving throughput means increased efficiency when dealing with large data. Meanwhile, reducing memory is also significant for the following reasons. First, reducing memory improves cache utility, and therefore is beneficial to the insertion throughput. Second, as the frequency estimation is a fundamental task, there often exist multiple instances of sketches on the same machine. Reducing memory consumption by tens or hundreds of MB for one sketch seems negligible, but can add up to big savings if there are hundreds of sketches running concurrently. Third, deploying sketches in the special hardware to achieve in-network acceleration is a popular design choice, where the available memory is limited. Reducing memory is key for such scenarios, as some special ASICs contains only 10MB available memory.

## III. SketchConf Algorithm

In this section, we take the CM sketch as a case study, and describe how SketchConf generates configuration for CM sketches. We first consider that, given a sketch configuration, how to design an error predictor that precisely predicts tail error probability (§III-A). Then, based on the error predictor, we describe how to search the optimal configurations efficiently (§III-B). We present the optimization for skewed data distribution to further speed up our algorithm (§III-C).

### A. From Configuration to Tail Error

We first consider that, given the frequency distribution $\mathcal{D}$, the number of distinct items $\mathcal{N}$ and a CM Sketch configuration, how to decide whether an error constraint (*e.g.*, $Pr\{Error > 100\} < 0.01$) can be satisfied. To solve the problem, we must provide precise prediction for $Pr\{Error > X\}$ with any $X$.

**Rationale:** In CM sketches, for the frequency estimation of item $e$ with real frequency $f$, we consider the counter that $e$ is hashed to. Suppose there are $Z$ other items hashed to this counter, namely $e_1, e_2, \cdots, e_Z$, and their frequencies are $f_1, f_2, \cdots, f_Z$. The counter provides an estimation of $(f + \sum_{i=1}^{Z} f_i)$ for $e$, and therefore the estimation error is $\sum_{i=1}^{Z} f_i$, *i.e.*, the sum frequency of hash colliding items. Theoretical error bounds use the expectation of the sum frequency and Markov inequality to derive bounds for $Pr\{Error > X\}$. The inequality is tight only in extreme frequency distribution, *e.g.*, all items have frequency $0$ except for one with frequency $X$. We instead use Monte Carlo simulation to sample and estimate the sum frequency precisely.

**Key technique i: Monte Carlo simulation (see Algorithm 1).** For one row of the CM sketch with $w$ counters, we

simulate $Pr\{Error > X\}$ for item $e$. For any one of the other $\mathcal{N}-1$ distinct items, the possibility that it collides with $e$ is $\frac{1}{w}$. Therefore, the number of distinct colliding items $Z$ follows binomial distribution $B(\mathcal{N}-1, \frac{1}{w})$. As $\mathcal{N}-1$ is usually large and $\frac{1}{w}$ is small, we approximate that $Z$ follows Poisson distribution, with $\lambda = \frac{\mathcal{N}-1}{w}$. We draw from Poisson distribution to sample $n$ from $Z$ (line 3). Then, we draw from frequency distribution $\mathcal{D}$ for $n$ times, and calculate the sum frequency (line 4). Here we should sample multiple times instead of setting error to $n*E(D)$, because we aim to estimate the distribution of the errors. The simulation will be conducted for multiple times, and the relative frequency that the sum frequency is larger than $X$ will be reported as our prediction (line 8). This method scales for multiple error constraints. For multiple tail error probabilities, we can estimate these tail errors in one Monte Carlo simulation process.

For a multi-row CM sketch with $d$ rows, as different rows are associated with independent hash functions, the estimation error in each row can be regarded as independent and identically distributed. Estimation error exceeding $X$ for $d$-row CM sketch happens when and only when error in each row exceeds $X$. Therefore, we run simulations to get prediction $pred$ for $Pr\{Error > X\}$ in one row, and report $pred^d$ (line 9).

**Algorithm 1:** Error Predictor for the CM Sketch

**Input:** the number of distinct items $\mathcal{N}$; frequency distribution $\mathcal{D}$; sketch depth $d$, width $w$; error threshold $X$;
**Output:** $Pr\{Error > X\}$

1 $cnt \leftarrow 0, simRounds \leftarrow 0, pred \leftarrow 0$;
2 **while** *$pred$ has not converged* **do**
3 draw $n$ from $Poisson(\frac{\mathcal{N}-1}{w})$;
4 $error \leftarrow$ sum of $n$ frequencies sampled from $\mathcal{D}$;
5 $simRounds \leftarrow simRounds + 1$;
6 **if** $error > X$ **then**
7 $cnt \leftarrow cnt + 1$;
8 $pred \leftarrow \frac{cnt}{simRounds}$;
9 **return** $pred^d$;

### B. From Tail Error to Configuration

With the error predictor prepared, we present our design to search for the memory-optimal configuration. The configurable parameters for the CM sketch include the sketch depth $d$ and width $w$. In the following we first describe a naive solution, and then show how to improve time efficiency by reusing simulation results.

**A naive solution:** the naive solution conducts binary search on memory. For each memory consumption, it traverses possible sketch depth $d$. Prior work [36] shows that the optimal configurations usually have a small $d$, *e.g.*, $3-4$, and therefore only small $d$ should be traversed, *e.g.*, $1-6$. With given memory consumption and number of sketch rows, the number of sketch columns can be uniquely determined, and the error predictor can be called to check whether all error constraints can be satisfied. The naive solution can find the memory-optimal configuration, but it suffers from poor time efficiency. Each time a new configuration is searched, the error predictor should be called to execute the time-consuming Monte Carlo simulation.

**Key technique ii: reuse simulation results (see Algorithm 2).** To accelerate the search, we break down the tail error:

$$Pr\{Error > X\} = \sum Pr\{Error > X \mid Z = n\} Pr\{Z = n\}$$

The former $Pr\{Error > X \mid Z = n\}$ means the probability that estimation error is larger than $X$ conditioning on $n$ colliding distinct items. This is the same as: given $n$ variables following distribution $\mathcal{D}$, the probability that their sum exceeds $X$. It is only associated with the frequency distribution $\mathcal{D}$, which is unchanged in the search. The latter $Pr\{Z = n\}$ can be fast calculated with the formula of Poisson distribution, as analyzed in III-A.

Therefore, we design our configuration searching with reusing simulation results. We conduct binary search on memory and enumerate possible configurations (line 2-6). When a new configuration is searched, we first ensure that the conditional probability $Pr\{Error > X \mid Z = n\}$ for $n \in [1, K]$ has been calculated (line 14-21). According to Theorem 1, when $K$ is set to $\frac{\mathcal{N}}{w} + \mathcal{N}\epsilon$, the sum of conditional probability $Pr\{Error > X \mid Z = n\}$ for $n \in [1, K]$ is close to $Pr\{Error > X\}$, and the difference is no more than $\delta$. Empirically, we set $\delta$ to $1\%$ to ensure the difference is small. Then we calculate the $\lambda$ of Poisson distribution for $Z$, and predict tail error probability (line 11-12). Note that for multiple tail error constraints, the conditional probability with the same $n$ can be estimated in one Monte Carlo process.

**Theorem 1.** *Let $\mathcal{N}$ be the number of distinct items, $w$ be the number of counters in a row. Then for a particular counter, define random variables $\Xi_1, \Xi_2, \ldots, \Xi_{\mathcal{N}}$ such that $\Xi_i \coloneqq \mathbb{1}\{$the $i$-th item is mapped into the counter$\}$. Denote $\xi \coloneqq \frac{5\sqrt{w}}{5\sqrt{w}+\sqrt{\mathcal{N}}}$, then for $\forall \delta > 0$, take $\epsilon = \max\left\{5\sqrt{\frac{1}{\mathcal{N}w}}, \frac{\ln\frac{1}{\delta}}{\mathcal{N}(\frac{\ln(1-\xi)}{-\xi}-1)}\right\}$, the following inequality holds:*

$$\Pr\left(\sum_{i=1}^{\mathcal{N}} \Xi_i > \frac{\mathcal{N}}{w} + \mathcal{N}\epsilon\right) \leqslant \delta$$

*Proof.* Since it's easy to see that $\Xi_1, \Xi_2, \ldots, \Xi_{\mathcal{N}}$ are iid and $\mathbb{E}(\Xi_i) = \frac{1}{w}$, denote $\mu \coloneqq \frac{1}{w}$. According to Chernoff bound, for $\forall \epsilon > 0$, we have:

$$\begin{aligned}\Pr\left(\sum_{i=1}^{\mathcal{N}} \Xi_i > \frac{\mathcal{N}}{w} + \mathcal{N}\epsilon\right) &= \Pr\left(\frac{1}{\mathcal{N}}\sum_{i=1}^{n} \Xi_i - \mu > \epsilon\right) \\ &= e^{-\mathcal{N}\left((\mu+\epsilon)\ln(\frac{\mu+\epsilon}{\mu}) + (1-\mu-\epsilon)\ln(\frac{1-\mu-\epsilon}{1-\mu})\right)} \\ &\leqslant e^{\mathcal{N}(\mu+\epsilon)\ln(1-\frac{\epsilon}{\mu+\epsilon})} \cdot e^{\mathcal{N}\epsilon}\end{aligned}$$

As we take $\epsilon \geqslant 5\sqrt{\frac{1}{\mathcal{N}w}}$, we have $\frac{\epsilon}{\mu+\epsilon} = \frac{1}{\frac{\mu}{\epsilon}+1} \geqslant \frac{5\sqrt{w}}{5\sqrt{w}+\sqrt{\mathcal{N}}} = \xi$. Since $f(x) = \ln(1+x)$ is concave in $(-1, +\infty]$ with $f(0) = 0$, we can know that $\ln(1+x) \leqslant \frac{\ln(1-\xi)}{-\xi}x$ holds for $\forall x \in (-1, -\xi]$. Therefore, we have:

$$\begin{aligned}\Pr\left(\sum_{i=1}^{\mathcal{N}} \Xi_i > \frac{\mathcal{N}}{w} + \mathcal{N}\epsilon\right) &\leqslant e^{-\frac{\ln(1-\xi)}{-\xi}\mathcal{N}\epsilon} \cdot e^{\mathcal{N}\epsilon} \\ &= e^{-\left(\frac{\ln(1-\xi)}{-\xi}-1\right)\mathcal{N}\epsilon} \leqslant \delta\end{aligned}$$

□

**Algorithm 2:** Configuration Search for the CM Sketch

**Input:** the number of distinct items $\mathcal{N}$; frequency distribution $\mathcal{D}$; error threshold $X$; the required number of conditional probability $K$;
**Output:** the memory-optimal configuration

```
1  probLen ← 0, Prob ← [],;
2  while binary search on memory is not stopped do
3      enumerate possible configurations (d, w) for current memory value;
4      call Error_Predictor(N, d, w, Prob) to check error constraints;
5      if a better configuration is found then
6          assign it to optConfig;
7  return optConfig;
8  Function Error_Predictor(N, d, w, &Prob):
9      Get_Conditional_Probability(X, K, Prob);
10     λ ← (N−1)/w;
11     pred ← Σ_{i=1}^{K} Poisson_λ(i) Prob[i];
12     return pred^d;
13 Function Get_Conditional_Probability(X, K, &Prob):
14     while probLen ≤ K do
15         cnt ← 0, simRounds ← 0, probLen ← probLen + 1;
16         while Prob[probLen] has not converged do
17             error ← sum of probLen frequencies sampled from D;
18             if error > X then
19                 cnt ← cnt + 1;
20             simRounds ← simRounds + 1;
21             Prob[probLen] ← cnt/simRounds;
```

### C. Optimization

Real world often witnesses skewed frequency distribution of workload [2], [26], which indicates that most items are infrequent, while only a few items are frequent. Infrequent items contribute little to the estimation error; besides, with much smaller variance, the converge of Monte Carlo needs much fewer rounds of simulation [49]. Therefore, we separate the simulation of frequent and infrequent items, by recording distributions of infrequent items with limited rounds of simulation. For highly skewed data, this optimization can largely reduce the amount of sampling for infrequent items.

As the pseudo code shown in Algorithm 3, the optimized version first simulates for infrequent items (line 2). Specifically, we simulate conditional probability $Pr\{Error > X \mid Z = n\}$ for infrequent items with different $n$, and repeat for $T$ times (line 10-14). We will show in §VI-D that the choice of $T$ has little impact on the performance of the optimizations. When simulating for $Pr\{Error > X \mid Z = n\}$, the sum frequency of frequent items is sampled and calculated (line 4-5). Then, we look up the pre-computed records of distribution of the infrequent items, and update the predicted conditional probability. It is noticeable that the records of infrequent items can be shared for $Pr\{Error > X \mid n\}$ with different $n$.

**Algorithm 3:** Optimized Error Predictor

**Input:** the number of colliding distinct items $n$; error threshold $X$; frequency distribution of frequent items $\mathcal{D}_l$ and infrequent items $\mathcal{D}_s$; the portion of frequent items $p$; simulation rounds for infrequent items $T$;
**Output:** $Pr\{Error > X \mid Z = n\}$

```
1  cnt ← 0, simRounds ← 0, pred ← 0;
2  dist ← Get_Infrequent_Dist(n);
3  while pred has not converged do
4      n_l ← Binomial(n, p);
5      sum_l ← sum of n_l frequencies sampled from D_l;
6      cnt += Card({f | f ∈ dist[n − n_l] ∧ f > X − sum_l});
7      simRounds += T;
8      pred ← cnt/simRounds;
9  return pred;
10 Function Get_Infrequent_Dist(n):
11     for i = 1 to n do
12         for j = 1 to T do
13             dist[i][j] ← sum of i frequencies sampled from D_s;
14     return dist;
```

## IV. Generalize to Other Sketches

SketchConf can be generalized to other order-independent sketches. We show how to generalize to Count sketches in §IV-A, Tower sketches in §IV-B, and Nitro sketches in §IV-C. It is worth attention that, the idea of Monte Carlo simulation, reusing simulation results and the optimization of separating frequent and infrequent items are general for these order-independent sketches. When it comes to concrete operations such as how to conduct Monte Carlo simulation for a specific kind of sketch, as different sketches have different ways of insertion and querying results, SketchConf must adjust the concrete operations.

### A. Count Sketch

The Count sketch [37] consists of a $d \times w$ counter matrix, and we call the rows $R_1, \cdots, R_d$. There are $2d$ hash functions, where $h_1, \cdots, h_d$ hash items to one counter in each row, and $s_1, \cdots, s_d$ hash items to $\{+1, -1\}$. When inserting item $s$, for $R_i$, the hashed counter $R_i[h_i(s)]$ is added with $s_i(s)$. When querying the frequency of item $s$, the median of $R_i[h_i(s)] \cdot s_i(s)$ is reported.

The error of frequency estimation for Count sketches also comes from the hash colliding items. However, Count sketches are different from CM sketches in two aspects: 1) each item updates the counter with $\{+1, -1\}$ according to hash function $s_i$; 2) Count sketches use median estimator for query, while CM sketches use minimum estimator. For the first difference, when conducting Monte Carlo simulation, we should sample the colliding items that hash to $+1$ and those that hash to $-1$ separately, by sampling from binomial distribution. For the second difference, we only consider Count sketches with uneven $d$. Suppose an item $s$ has frequency of $f$. The median of estimated frequency has an error larger than $X$ indicates

that: more than half of the estimation are either larger than $f + X$, or less than $f - X$. As the tail error probability for a single row can be predicted with Monte Carlo simulation, we can compute the tail error probability for median estimator with the binomial expansion.

$$P_{median} = \sum_{i=\lceil d/2 \rceil}^{d} C_d^i P_{single}^i (1 - P_{single})^{d-i}$$

Here the $C_d^i$ is the combinatorial number, and $P_{single}$ is the tail error probability for a single row. The simulation results can also be reused for Count sketches, because for Count sketches the conditional probability is still only related to workload characteristics. The optimization is available. The simulation of the sum frequency for items hashed to $+1$ (or $-1$) is exactly the same as in CM sketches, and therefore the optimization can be applied to accelerate this process.

### B. Tower Sketch

The Tower sketch [38] consists of $d$ rows of counters and $d$ hash functions. The Tower sketch uses different counter size for each row, and sets the memory consumption for all counter rows to the same. This means the row with a larger counter size comprises of fewer counters. The insertion and query operations for the Tower sketch are similar to the CM sketch, except for one special case: the maximum value for a counter (*e.g.*, $2^\delta - 1$ for a $\delta$-bits counter). For insertion, the counter with maximum value will not be updated. For querying, the counter with maximum value stands for infinity.

The configurable parameters for Tower sketches include the number of counter arrays $d$, the number of bits for one array. We use fixed counter size of 32 bits, 16 bits, 8 bits and 4 bits. To predict error of Tower sketch under a specific configuration, we observe that event $Error > X$ occurs in two situations: 1) the sum frequency of colliding items is larger than $X$, and 2) the sum frequency will cause the counter to be maximum value (infinity). Suppose the sum frequency of colliding items is $s$, and the array has counters with $\delta$ bits, we have:

$$\begin{aligned} Pr\{Error > X\} &= max\{Pr\{s > X\}, Pr\{s \geqslant 2^\delta - 1\}\} \\ &= Pr\{s > min\{X, 2^\delta - 2\}\} \end{aligned}$$

Therefore, it can be predicted in the same way as CM sketches, and the only difference is that $X$ should be adjusted. Reusing simulation results and the optimization are available for Tower sketches.

### C. Nitro Sketch

The Nitro sketch [33] utilizes the sampling technique to accelerate insertion for CM sketches and Count sketches. The Nitro sketch sets a sampling rate $p$. For insertion, each counter array samples items with probability $p$, and sampled items update one hashed counter with $p^{-1}$ (CM sketches), or $\{+p^{-1}, -p^{-1}\}$ (Count sketches). When querying item's frequency, Nitro sketches report the median of $d$ hashed counters.

The configurable parameters are the same as CM sketches or Count sketches. The estimation error of an item $s$ in Nitro sketch comes from two aspects: the other sampled items that collides with $s$, and the sampling error of $s$ itself. For an single array of Nitro sketch of CM version, suppose an item $e$ with frequency $f$ is hashed to one counter. The value of the counter can be expressed as $p^{-1}f_p + p^{-1}S$, where $f_p$ is the number of times $e$ is sampled, and $S$ is the sum frequency of colliding sampled items. The estimation error consists of the collision error $p^{-1}S$ and the sampling error $p^{-1}f_p - f$. To apply our error predictor, for colliding sampled items, we first estimate the sum frequency of colliding sampled items $S$, where we can apply Monte Carlo simulation. Then we draw from binomial distribution $B(S, p)$, and multiply it with $p^{-1}$ to get the sampled size. For the sampling error of $s$, we can also conduct Monte Carlo simulation: draw $s$ from frequency distribution, and simulate sampling error. Therefore Monte Carlo simulation can accurately predict error of Nitro sketches. Reusing simulation results is available, as the conditional probability is only associated with frequency distribution and sampling rate $p$. For the optimization, it can be applied to accelerate the estimation of collision errors. As the sampling error is associated with the queried item itself, the optimization cannot be applied to accelerate sampling error.

## V. Applying SketchConf to Real Scenarios

Above we have discussed how to configure sketches under given and fixed workload characteristics. However, in real scenarios of data stream processing, the workload characteristics may not be explicitly provided. Moreover, the workload characteristics may change over time, which introduces additional challenges for the application of SketchConf. In this section, we discuss how to address the challenges caused by unknown and changeable workloads.

When applying sketching algorithms for streaming data, as the streaming data may go unbounded, a practical and common solution is to divide the streaming data into time windows of fixed length, each processed by a sketch individually. Ideally, we wish to provide an appropriate sketch configuration for each time window that matches the corresponding workload characteristic. However, applying SketchConf will face three challenges as follows. First, as the workload information is the necessary input of SketchConf, we should estimate the information of the unknown workload in each time window with low overhead. Second, the optimal configuration for each time window may vary slightly due to small changes in the workload, while frequent re-configuration of sketches may incur additional overhead. Therefore, we should avoid unnecessary re-configuration of sketches. Third, the abrupt change of workload characteristics may lead to severe violation of user specified error constraints. In such situations, we should take emergency methods to guarantee the users' requirements.

For the first challenge, our idea is to estimate workload information based on the sketch of the previous time window. At the start time of current time window, we can apply MRAC [35] algorithms to estimate the number of distinct items and the frequency distribution from the built sketch of the previous time window. We can use the estimated workload information of the previous time window as the prediction

of future workload, and apply SketchConf to generate sketch configuration. The process will finish before the current time window ends, as both MRAC and SketchConf is efficient. For the next time window, we will apply the new sketch configuration.

For the second challenge, we propose a technique to monitor the errors of the sketch results in real time. The technique is based on the observation from [46], which shows that the estimation errors of sketches can be precisely approximated by the distribution of sketch counters. Specifically, take a $d \times w$ CM sketch as an example. Suppose the proportion of counters greater than $K$ in the first row is $p_K$. The probability that the error of the estimated frequency is greater than $K$ is approximately $p_K^d$. Therefore, we maintain an error counter for each error constraints $Pr\{Error > X_i\} < \delta_i$, which records the number of counters that exceed $X_i$ in the first row. Maintaining such error counters in real time incurs little overhead: when the item increments one counter in the first row of the sketch from $X_i$ to $X_i + 1$, we update the error counter. After the time window ends, we recover tail errors from the error counters. Only if the tail errors are larger/smaller than user specified constraints to some thresholds, *e.g.*, $Pr\{Error > X_i\} > \delta_i(1 + T_a)$ or $Pr\{Error > X_i\} < \delta_i(1 - T_a)$, do we activate SketchConf to re-configure the sketch of the next time window. Otherwise, we consider the tail errors go through normal fluctuations.

For the third challenge, our idea is to stop the current time window immediately when abrupt events lead to severe violation of users' demand. Since we can monitor the error of the sketch in the current time window in real time, we use the threshold $T_b$ to decide whether the current tail errors severely violate users' demands. Specifically, for one user specified error constraint $Pr\{Error > X_i\} < \delta_i$, if the current tail error $Pr\{Error > X_i\}$ has exceeded $\delta_i(1 + T_b)$, it is considered as severe violation. Note that $T_b$ is different from $T_a$. $T_b$ should be set much larger than $T_a$, which is used to avoid extreme worse cases. When the severe violation is detected, the current time window $w_c$ is ended, and a new time window starts with the same sketch configuration as the last window. At the same time, we urgently execute MRAC on the sketch of window $w_c$ for workload analysis, use SketchConf to re-configure sketches. Note that the execution of MRAC and SketchConf require time, and if the sketch error in the current window exceeds the $T_b$ threshold again, we end the window and start a new one to ensure a consistent guarantee of error constraints. Once the re-configuration is over, we start a new time window with the new sketch configuration. The time cost of re-configuration is small because of the efficiency of MRAC and SketchConf. Besides, the abrupt change of workload characteristics are usually rare, and therefore we think such a complex solution is acceptable.

## VI. Experimental Results

### A. Experimental Setup

**Datasets: 1) Synthetic Dataset:** the synthetic dataset is generated according to the Zipf distribution. We vary the skewness parameter from $0.0$ to $1.5$. Each synthetic dataset contains $32$M items with around $1$M distinct items. **2) IP Trace Dataset:** it contains one hour of anonymous network traces collected from the Equinix-Chicago monitor in 2018 [50]. We use the source IP and destination IP as the ID of items. We divide it into 1-minute intervals, and each contains around $27$M items and $85$K distinct items. **3) WebDocs Dataset:** it is built from a spidered collection of web documents [51]. It filters out html tags and the most common words in the documents, and use remaining terms as items. We use part of the dataset, which contains around $64$M items and $32.9$M distinct items.

**The competitors:** We compare our SketchConf with the following competitors: theory-based solutions, hash table solutions, learning-based sketches, and the baseline algorithm.

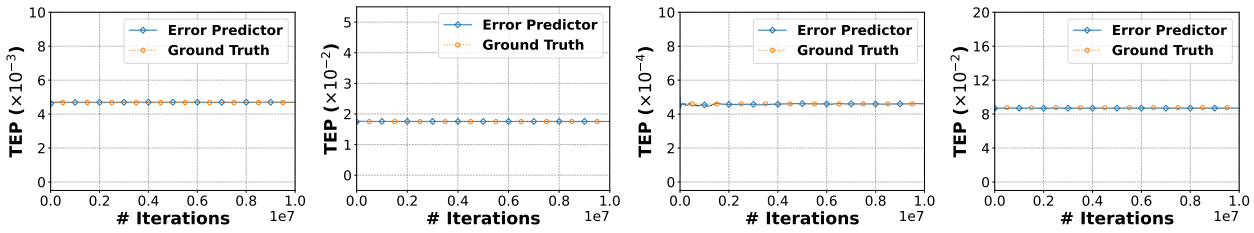
For theory-based solution, we use the theoretical configurations in [8], [48] for CM sketches and Count sketches, respectively. For hash table solutions, we implement a hash table with chaining to record items' frequency with full accuracy. For the learning-based sketches, we implement the heavy-hitter predictor in [44], and for the sketch part we use theory-based configurations and SketchConf configurations. The baseline algorithm is built on the state-of-the-art work for sketch error estimation [46], which provides a posterior error estimation by examining the distribution of the result counters. The baseline conducts binary search on memory, and enumerate configurations as SketchConf does. For each searched configuration, the baseline builds a sketch, inserts items into the sketch, and uses the distribution of result counters to estimate tail error probability. It repeats multiple times and uses convergent results as predicted errors. Note that the baseline should build the sketch first, because the error estimation algorithm is posterior and can only be used on the inserted sketch record.

**Implementation:** We implement SketchConf for sketches of CM, Count, Nitro, Tower in C++. We conduct evaluations on a server with dual 18-core CPUs (36 threads, Intel Core i9-10980XE CPU @3.00GHz) and 125GB DRAM memory. In all experiments, we use Farm Hash [52] to implement hash functions. Our source code is available at Github [53].

### B. Experiments on the Error Predictor

In this section, we evaluate the error predictor in III-A and show that our Monte Carlo based error predictor can precisely predict tail error probability.

**Settings:** We use the synthetic dataset with Zipf skewness parameter $1.0$ for this experiment and evaluate the error predictor for $Pr\{Error > 100\}$. For CM, Count and Tower sketches, we set the number of rows $d = 3$ and the number of columns $w = 200000$. For Nitro sketches, we use the CM version with $d = 3$ and $w = 200000$, and set the sampling rate $p = 0.1$. We conduct the error predictor with $10$ million iterations of Monte Carlo simulations, and plot the predicted tail error probability as the number of iterations grows. The ground truth is obtained by building a sketch, directly inserting




(a) CM sketches (b) Count sketches (c) Tower sketches (d) Nitro sketches

Fig. 4: Experiments on the error predictor (TEP represents tail error probability).

all the item in the dataset into it and querying each distinct items to calculate the real tail error rate.

**Experimental Results (Figure 4):** For CM sketches, after 10 million iterations, the predicted tail error probability is $0.004698$ and the ground truth is $0.004681$. For Count sketches, the predicted tail error probability is $0.017554$ and the ground truth is $0.017508$. For Tower sketches, the predicted tail error probability is $0.000461$ and the ground truth is $0.000460$. For Nitro sketches, the predicted tail error probability is $0.086967$ and the ground truth is $0.087547$.

**Analysis:** The experiment results show that our algorithm can predict tail error probability accurately. For all four sketches, the gaps between our prediction and the ground truth are smaller than $0.66\%$.

In the following experiments we conduct Monte Carlo simulation in batches, each including $1000$ iterations. We stop the simulation if the fluctuation of predicted error in the current batch and in previous batches is less than $0.0001$, and output the predicted error.

### C. Experiments on Reusing Simulation Results

In this section, we evaluate how reusing simulation results improves time efficiency for SketchConf in the configuration search, regardless of workload information.

**Settings:** We use the synthetic datasets that follow the Zipf distribution. We vary the skewness parameters from $0.0$ to $1.5$, and each dataset contains 32M items and around 1M distinct items. We compare the time cost of SketchConf with and without reusing simulation results. We use three error constraints that the output configurations should satisfy simultaneously: i) $\Pr\{Error > 100\} < 0.01$, ii) $\Pr\{Error > 200\} < 0.005$, iii) $\Pr\{Error > 300\} < 0.001$.

**Experimental results (Figure 5):** When the Zipf skewness parameter ranges from $0.0$ to $1.5$, our reuse technique reduces reduce $81.63\%$ to $99.96\%$ of time cost for CM sketches, $73.85\%$ to $98.62\%$ for Count sketches, $94.43\%$ to $99.88\%$ for Tower sketches, $86.85\%$ to $99.74\%$ for Nitro sketches.

**Analysis:** The experimental results show that the technique of reusing simulation results can improve time efficiency for all four kinds of sketches. The reduced time cost can be up to $99.96\%$. With the reusing simulation results technique, the search for sketch configuration can be finished in the order of 10s.

### D. Experiments on the Optimization

In this section, we apply the optimization of separating frequent and infrequent items to four kinds of sketches, and show how the optimization improves time efficiency. Besides, we discuss how the choice of $T$ (the number of repeated simulations on infrequent items, as shown in algorithm 3) impact the performance of the optimization.

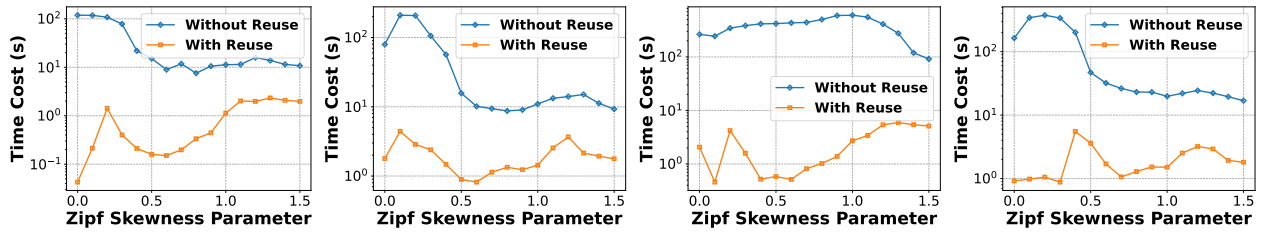
**Settings:** Our optimization is designed for skewed frequency distribution, so we conduct experiments on the synthetic dataset with skewness parameter ranging from $1.0$ to $1.5$ to show the effect of the optimization. To show the impact of $T$, we use the synthetic dataset with skewness parameter $1.0$, and conduct evaluations on CM sketches and Count sketches. The output configurations should follow three error constraints simultaneously: i) $\Pr\{Error > 100\} < 0.01$, ii) $\Pr\{Error > 200\} < 0.005$, iii) $\Pr\{Error > 300\} < 0.001$, the same as in §VI-C.

**Effects of the optimization (Figure 6):** The experiment results show that, with optimization, SketchConf achieves lower time cost for CM sketches, Count sketches and Tower sketches, and comparable performance for Nitro sketches. When Zipf skewness parameter ranges from $1.0$ to $1.5$, our optimization can reduce $49.39\%$ to $63.78\%$ of time cost for CM Sketches, $42.84\%$ to $68.66\%$ for Count Sketches, and $12.48\%$ to $45.23\%$ for Tower sketches. For Nitro sketches, the time cost is comparable. The reason is that, the simulation of Nitro sketches requires an additional computation of Binomial distribution due to the sampling, and thus is more time-consuming. For other sketches, the bottleneck lies in the addition of frequencies, and when simulating infrequent items the cost is small. For Nitro sketches, the bottleneck lies in the Binomial distribution computations, and separating frequent and infrequent items means two times of Binomial distribution computations, which offsets the benefits brought by the optimization.

**Impact of $T$ (Figure 7):** The experimental results show that, $T$, the number of repeated simulation of infrequent items, has little impact on the performance of the optimization. For CM sketches, when changing $T$ from 250 to 2000, the time cost of SketchConf ranges between $0.476124$s and $0.573137$s, and the memory of output configurations ranges between 1996KB to 2033KB. For Count sketches, when changing $T$ from 250 to 2000, the time cost of SketchConf ranges between $0.61$s and $0.71$s, and the memory of output configurations ranges between 2613KB to 2668KB.

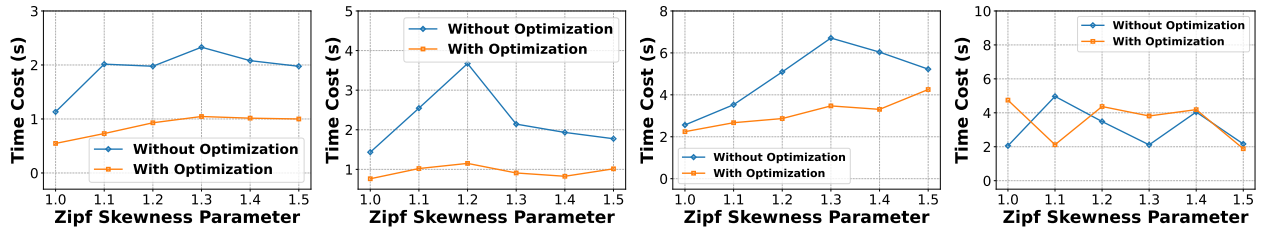
### E. Experiments on Configurations

In this section, we evaluate the effectiveness of SketchConf in the aspects of output configurations and search time cost on



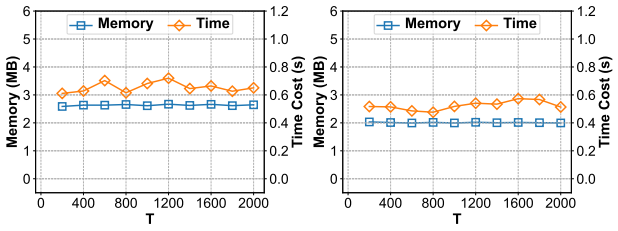

(a) CM sketches (b) Count sketches (c) Tower sketches (d) Nitro sketches

Fig. 5: Experiments on reusing simulation results.



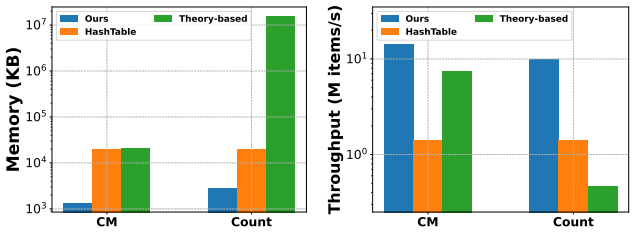

(a) CM sketches (b) Count sketches (c) Tower sketches (d) Nitro sketches

Fig. 6: Experiments on the optimization.




(a) CM sketches (b) Count sketches

Fig. 7: Experiments on the impact of parameter $T$.

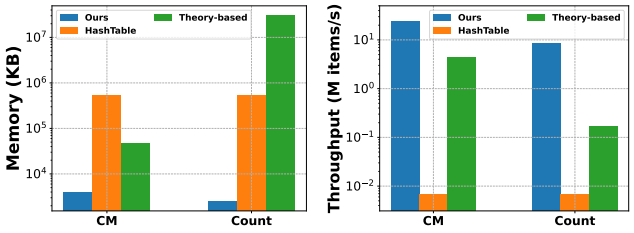



(a) Memory Consumption (b) Throughput

Fig. 8: Comparison with theory-based configurations and hash table solutions on the IP Trace Dataset.

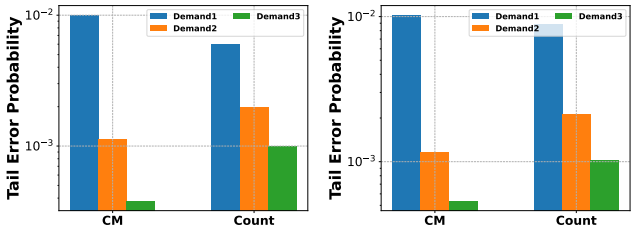
the real datasets. We further evaluate the impact of the number of error constraints on SketchConf.

**Settings:** We conduct experiments of frequency estimation on the IP Trace Dataset and WebDocs Dataset. We compare



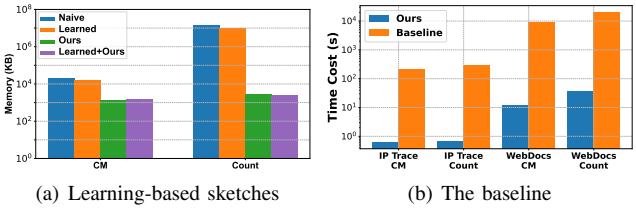

(a) Memory Consumption (b) Throughput

Fig. 9: Comparison with theory-based configurations and hash table solutions on the WebDocs Dataset.




(a) The IP Trace Dataset (b) The WebDocs Dataset

Fig. 10: Real tail errors of SketchConf in two real-world datasets.




(a) Learning-based sketches (b) The baseline

Fig. 11: Comparison with learning-based sketches and the baseline.

the sketch solutions using SketchConf configurations with the sketch solutions using theory-based configurations, hash table solutions and learning-based sketches. As prior work [44] trains learning-based sketches with the IP Trace Dataset, we only compare SketchConf with learning-based sketches on the IP Trace Dataset. For search time cost, we compare with the baseline algorithm. The output configurations should satisfy three error constraints, which are the same as those used in §VI-C and §VI-D. For the impact of the number of error constraints, we vary the number of error constraints and evaluate SketchConf on the synthetic dataset with skewness parameter 1.0. We enable the optimization for SketchConf in this experiments.

**Comparison with theory-based configurations and hash table solutions on the IP Trace Dataset (Figure 8, 10(a)):** On the IP Trace Dataset, the configurations output by SketchConf

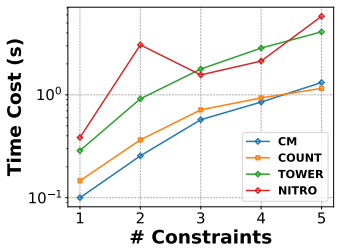


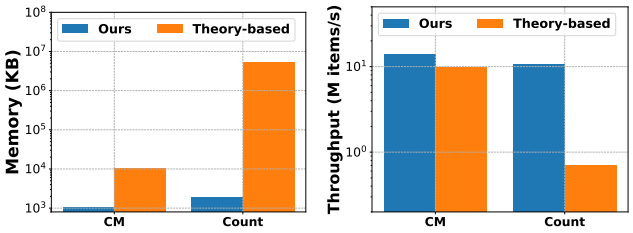

Fig. 12: Evaluation on the number of error constraints.

take up 1.32MB and 2.74MB for CM sketches and Count sketches, respectively, and the insertion throughput is 14.24M items/s and 9.96M items/s, respectively. Sketch solutions with SketchConf configurations save 93.41%, 99.98% memory and achieve 1.94×, 21.35× throughput compared with theory-based configurations, and save 93.37%, 86.30% memory and achieve 10.22×, 7.14× throughput compared with hash table solutions. It is worth noting that the theory-based configurations may consumes more memory than the fully accurate hash table solutions. This is because the theoretical bounds are loose, especially for real-world datasets with high skewness. Beside, for SketchConf configurations, the real tail error probabilities match the three demands.

**Comparison with theory-based configurations and hash table solutions on the WebDocs Dataset (Figure 9, 10(b)):** On the WebDocs Dataset, the configurations generated by SketchConf take up 3.86MB and 2.41MB for CM sketches and Count sketches, respectively, and the insertion throughput is 23.83M items/s and 8.67M items/s, respectively. Sketch solutions with SketchConf configurations save 91.89%, 99.99% memory, and achieve 5.37×, 50.57× throughput compared with theory-based configurations, and save 99.25%, 99.53% memory and achieve 3457×, 1257× throughput compared with hash table solutions. Beside, for SketchConf configurations, the real tail error probabilities match the three demands.

**Comparison with learning-based sketches (Figure 11(a)):** For theory-based configurations, sketches take up 19.77MB, 14.11GB, respectively. With learning-based techniques the memory consumption is reduced to 16.13MB, 9.40GB, respectively. Compared with learning-based sketches, SketchConf improves memory efficiency by 12.36×, 3491×, because the sketch part still has much space for refinement. It is also noticeable that our SketchConf is orthogonal to the learning techniques, as SketchConf can be utilized to generate the configurations for the sketch part. If the learning techniques show good performance, combining learning techniques with SketchConf may further improve the memory efficiency. For Count sketches the combination reduces memory by 1.13× compared with simply running SketchConf.

**Comparison with Baseline (Figure 11(b)):** For CM Sketches, SketchConf takes 0.62s and 12.32s to search for configurations on IP Trace and WebDocs Dataset respectively, while the baseline takes 214s and 8815.09s. For Count Sketches, SketchConf takes 0.66s and 37.34s, while the baseline takes 291s and $2.04 \times 10^4$s respectively. SketchConf achieves 345.16× and 715.51× lower time cost for CM sketches, and 440.91× and 546.33× lower time cost for Count sketches.




(a) Memory Consumption (b) Throughput

Fig. 13: Experiments on finding top-100 items.

**Analysis:** The reason why SketchConf can generate better configurations compared with theory-based ones is that, SketchConf can output the configurations that exactly match the error constraints. In contrast, the theory-based configurations often achieve accuracy that is far beyond the user's requirement. Therefore, the configurations of SketchConf are more memory-efficient, and also achieve higher throughput due to higher cache utility. One reason why SketchConf consumes less time than the baseline algorithm is that, SketchConf conducts Monte Carlo simulation instead of building sketches by inserting all items. Reusing Monte Carlo simulation and the optimization also provide much speedup.

**Impact of the number of error constraints (Figure 12):** The experimental results show that SketchConf can support multiple error constraints efficiently. When the number of error constraints grows from 1 to 5, the search time cost of SketchConf increase from 0.09s to 1.32s for CM sketches, from 0.15s to 1.16s for Count sketches, from 0.29s to 4.10s for Tower sketches, and from 0.38s to 5.80s for Nitro sketches.

### *F. Experiments on the Top-k Applications*

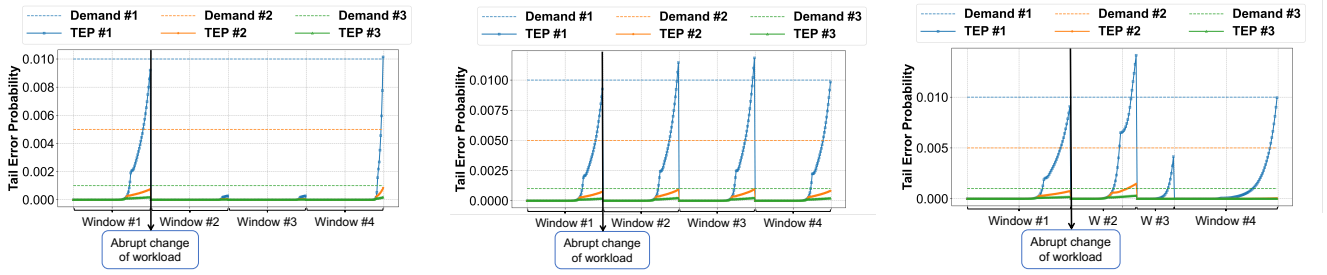
In this section, we show how SketchConf improves the effectiveness of sketches in the top-k applications.

**Settings:** We run sketching algorithms and maintain a heap of size 100 to find top-100 items on the IP Trace Dataset. Suppose in the IP Trace Dataset, the frequency difference of 100-th and 101-th largest items is $X$. To achieve 99% precision, the sketching algorithms should ensure that the probability of estimation error larger than $X$ is no more than 1%. We compare the performance of SketchConf and theory-based configurations under such error constraints.

**Experiments Results (Figure 13):** As shown in figure, compared with theory-based configurations, SketchConf reduces memory consumption by 9.83×, 2723× in sketches of CM, Count, respectively, and provide 1.45×, 15.27× speed up respectively. The precision of finding top-100 items is no less than 99%. For applications such as top-k, SketchConf can ensure the precision requirements while improving performance in both memory consumption and throughput.

### *G. Experiments on the unknown and changeable workload*

In this section, we simulate the unknown and changeable workload when applying SketchConf to streaming data processing. We consider applying CM sketches in time-based window scenarios, *e.g.*, building a new sketch for each new time window, as stated in §V. To simulate the changeable workload, we use the synthetic dataset for each time window,




(a) Workload change that leads to errors lower than the demands (b) Workload change that leads to errors higher than the demands (c) Workload change that leads to severe violation of demands

Fig. 14: Simulations for unknown and changeable workloads, where "TEP" stands for "Tail Error Probability".

and adjust the parameters like the skewness parameter and the number of items.

**Settings:** The sketch configurations should satisfy the following three error constraints at the same time: i) $\Pr\{Error > 100\} < 0.01$, ii) $\Pr\{Error > 200\} < 0.005$, iii) $\Pr\{Error > 300\} < 0.001$. For the following simulations, we use the CM sketch with 3 rows and 152226 columns at the beginning. In window 1, we use the synthetic dataset with skewness parameter 1.0, 30M items and 1M distinct items as the streaming data, and change workload for the following windows. We set $T_a = 0.2$ and $T_b = 0.5$ (as defined in §V). We run the EM algorithm for 1 iteration for MRAC, since each iteration takes a relative long time to complete, and our results show that 1 iteration suffices to produce precise estimation for re-configuration.

**Simulation of errors lower than user demands (Figure 14(a)):** The solid lines indicate the real-time monitored TEPs (tail error probabilities) and the dotted lines represent the user's demands. At the end of window 1, we find that all of the three TEPs don't exceed $\delta_i(1+T_a)$ and TEP #1 is close to demand #1 (i.e. bigger than $\delta_1(1-T_a)$), so it is unnecessary to re-configure the sketch. For the following windows, we change the skewness parameter to 1.2. At the end of window 2, we find that all of the three TEPs are much less than user's demands (i.e. less than $\delta_i(1-T_a)$), which indicates we can meet the demands with less memory consumption, so in window 3 we execute MRAC and SketchConf to calculate the new configuration for the sketch. Before window 3 ends, SketchConf outputs a new configuration with 4 rows and 37474 columns. In window 4 we use the new configuration, which still meets the user's demand with 67.18% less memory consumption.

**Simulation of errors higher than user demands (Figure 14(b)):** For the following windows, we still use the skewness parameter of 1.0, but increase the number of items per window to 32M. At the end of window 2, we find that the TEP #1 is significantly higher than user's demand (i.e. bigger than $\delta_0(1+T_a)$), which indicates we need to re-configure the sketch to meet the user's demand, so in window 3 we execute MRAC and SketchConf to calculate the new configuration for the sketch. Before window 3 ends, SketchConf outputs a new configuration with 3 rows and 167078 columns. In window 4 we use the new configuration, which can meet the user's demands.

**Simulation of severe violation of user demands (Figure 14(c)):** For the following windows, we change the skewness parameter to 0.8. During window 2, we find that TEP #1 exceeds demand #1 to an extent (i.e. bigger than $\delta_0(1+T_b)$), which indicates we need to urgently stop the current window and conduct re-configuration. Along with window 3, we execute MRAC and SketchConf to calculate the new configuration for the sketch. When the re-configuration ends, we start window 4 with the new sketch configuration with 5 rows and 197383 columns. The real-time monitored errors indicate that the new configuration can meet the user's demands.

**Memory overhead:** The memory overhead of the MRAC algorithm depends on the flow distribution. In our experiments above, MRAC typically needs tens of megabytes of memory. However, in real world scenario, the significant change of flow distribution happens not that often, and only then need we use MRAC to estimate the new distribution and reconfigure the sketch. So the memory consumption of MRAC won't have much impact to the whole system.

## VII. Conclusion

Providing appropriate parameter configurations for sketching algorithm is critical for the applications of sketches in various scenarios. This paper proposes an automatic sketch configuration framework, SketchConf, which supports order-independent sketches. SketchConf utilizes the technique of Monte Carlo simulation to build precise and efficient error predictor. To speed up the configuration search, SketchConf uses the technique of reusing simulation results. For highly skewed data, SketchConf can enable the optimization of separating frequent and infrequent items to further reduce the time cost. When applied to real scenarios, we provide practical solutions to deal with the unknown and changeable workloads. Experimental results show that SketchConf can output configurations efficiently, and the output configurations consume much less memory and achieve better insertion throughput compared with theory-based configurations.

## Acknowledgement

We would like to thank the anonymous reviewers for their valuable suggestions. This work is supported by Key-Area Research and Development Program of Guangdong Province 2020B0101390001, and National Natural Science Foundation of China (NSFC) (No. U20A20179, 61832001).

## REFERENCES


[1] Kaiyu Li and Guoliang Li. Approximate query processing: What is new and where to go? a survey on approximate query processing. *Data Science and Engineering*, 3:379–397, 2018.

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

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

[4] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proc. ACM SIGMOD*, pages 286–296, 2004.

[5] Zhewei Wei, Ge Luo, Ke Yi, Xiaoyong Du, and Ji-Rong Wen. Persistent data sketching. In *SIGMOD*, 2015.

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

[7] Graham Cormode and Marios Hadjieleftheriou. Methods for finding frequent items in data streams. *The VLDB Journal*, 19(1):3–20, 2010.

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

[9] Minos Garofalakis, Daniel Keren, and Vasilis Samoladas. Sketch-based geometric monitoring of distributed stream queries. *Proceedings of the VLDB Endowment*, 6(10):937–948, 2013.

[10] Benedikt Sigurleifsson, Aravindan Anbarasu, and Karl Kangur. The count-min sketch data structure and its uses within computer science, 2019.

[11] Amit Goyal and Hal Daumé III. Approximate scalable bounded space sketch for large data nlp. In *Proceedings of the 2011 Conference on Empirical Methods in Natural Language Processing*, pages 250–261, 2011.

[12] Amit Goyal, Hal Daumé III, and Graham Cormode. Sketch algorithms for estimating point queries in nlp. In *Proceedings of the 2012 joint conference on empirical methods in natural language processing and computational natural language learning*, pages 1093–1103, 2012.

[13] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *NSDI 2013*, 2013.

[14] Zaoxing Liu, Antonis Manousis, and et al. One sketch to rule them all: Rethinking network flow monitoring with univmon. In *Proc. ACM SIGCOMM*, 2016.

[15] Partha Talukdar and William Cohen. Scaling graph-based semi supervised learning to large number of labels using count-min sketch. In *Artificial Intelligence and Statistics*, pages 940–947. PMLR, 2014.

[16] Fabon Dzogang, Thomas Lansdall-Welfare, Saatviga Sudhahar, and Nello Cristianini. Scalable preference learning from data streams. In *Proceedings of the 24th International Conference on World Wide Web*, pages 885–890, 2015.

[17] Stuart Schechter, Cormac Herley, and Michael Mitzenmacher. Popularity is everything: A new approach to protecting passwords from {Statistical-Guessing} attacks. In *5th USENIX Workshop on Hot Topics in Security (HotSec 10)*, 2010.

[18] Heejung Yang and Chin-Wan Chung. Efficient iceberg query processing in sensor networks. *The Computer Journal*, 57(12):1834–1851, 2014.

[19] Qi George Zhao, Mitsunori Ogihara, Haixun Wang, and Jun Jim Xu. Finding global icebergs over distributed data sets. In *Proc. ACM SIGMOD-SIGACT-SIGART*, 2006.

[20] M. Gurmeet Singh and M. Rajeev. Approximate frequency counts over data streams. In *Proc. VLDB*, pages 346–357, 2002.

[21] Yang Yang, Wenjie Zhang, Ying Zhang, Xuemin Lin, and Liping Wang. Selectivity estimation on set containment search. *Data Science and Engineering*, 4(3):254–268, 2019.

[22] Yuhan Wu, Siyuan Dong, Yi Zhou, Yikai Zhao, Fangcheng Fu, Tong Yang, Chaoyue Niu, Fan Wu, and Bin Cui. Kvsagg: Secure aggregation of distributed key-value sets. In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*. IEEE, 2023.

[23] Tong Yang, Haowei Zhang, Jinyang Li, Junzhi Gong, Steve Uhlig, Shigang Chen, and Xiaoming Li. Heavykeeper: An accurate algorithm for finding top-*k* elephant flows. *IEEE/ACM Transactions on Networking*, 27(5):1845–1858, 2019.

[24] Daniel Kifer, Shai Ben-David, and Johannes Gehrke. Detecting change in data streams. In *VLDB*, volume 4, pages 180–191. Toronto, Canada, 2004.

[25] Daniel Gmach, Jerry Rolia, Ludmila Cherkasova, and Alfons Kemper. Workload analysis and demand prediction of enterprise data center applications. In *2007 IEEE 10th International Symposium on Workload Characterization*, pages 171–180. IEEE, 2007.

[26] Berk Atikoglu, Yuehai Xu, Eitan Frachtenberg, Song Jiang, and Mike Paleczny. Workload analysis of a large-scale key-value store. In *Proceedings of the 12th ACM SIGMETRICS/PERFORMANCE joint international conference on Measurement and Modeling of Computer Systems*, pages 53–64, 2012.

[27] Zhichao Cao, Siying Dong, Sagar Vemuri, and David HC Du. Characterizing, modeling, and benchmarking {RocksDB} {Key-Value} workloads at facebook. In *18th USENIX Conference on File and Storage Technologies (FAST 20)*, pages 209–223, 2020.

[28] Arik Rinberg, Alexander Spiegelman, Edward Bortnikov, Eshcar Hillel, Idit Keidar, Lee Rhodes, and Hadar Serviansky. Fast concurrent data sketches. *ACM Transactions on Parallel Computing*, 9(2):1–35, 2022.

[29] Lior Zeno, Dan RK Ports, Jacob Nelson, Daehyeok Kim, Shir Landau-Feibish, Idit Keidar, Arik Rinberg, Alon Rashelbach, Igor De-Paula, and Mark Silberstein. {SwiSh}: Distributed shared state abstractions for programmable switches. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*, pages 171–191, 2022.

[30] Jaeyeon Jung, Balachander Krishnamurthy, and Michael Rabinovich. Flash crowds and denial of service attacks: Characterization and implications for cdns and web sites. In *Proceedings of the 11th international conference on World Wide Web*, pages 293–304, 2002.

[31] Qi Huang, Helga Gudmundsdottir, Ymir Vigfusson, Daniel A Freedman, Ken Birman, and Robbert van Renesse. Characterizing load imbalance in real-world networked caches. In *Proceedings of the 13th ACM Workshop on Hot Topics in Networks*, pages 1–7, 2014.

[32] Brian F Cooper, Adam Silberstein, Erwin Tam, Raghu Ramakrishnan, and Russell Sears. Benchmarking cloud serving systems with ycsb. In *Proceedings of the 1st ACM symposium on Cloud computing*, pages 143–154, 2010.

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

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

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

[36] Tong Yang, Jie Jiang, Peng Liu, Qun Huang, Junzhi Gong, Yang Zhou, Rui Miao, Xiaoming Li, and Steve Uhlig. Elastic sketch: adaptive and fast network-wide measurements. In *SIGCOMM*, 2018.

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

[38] Kaicheng Yang, Yuanpeng Li, Zirui Liu, Tong Yang, Yu Zhou, Jintao He, Tong Zhao, Zhengyi Jia, Yongqiang Yang, et al. Sketchint: Empowering int with towersketch for per-flow per-switch measurement. In *2021 IEEE 29th International Conference on Network Protocols (ICNP)*, pages 1–12. IEEE, 2021.

[39] Tao Li, Shigang Chen, and Yibei Ling. Per-flow traffic measurement through randomized counter sharing. *IEEE/ACM Transactions on Networking*, 20(5):1622–1634, 2012.

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

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

[42] Florin Rusu and Alin Dobra. Statistical analysis of sketch estimators. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, pages 187–198, 2007.

[43] Amirali Aghazadeh, Ryan Spring, Daniel LeJeune, Gautam Dasarathy, Anshumali Shrivastava, et al. Mission: Ultra large-scale feature selection using count-sketches. In *International conference on machine learning*, pages 80–88. PMLR, 2018.

[44] Chen-Yu Hsu, Piotr Indyk, Dina Katabi, and Ali Vakilian. Learning-based frequency estimation algorithms. In *International Conference on Learning Representations*, 2019.

[45] Gregory T Minton and Eric Price. Improved concentration bounds for count-sketch. In *Proceedings of the twenty-fifth annual ACM-SIAM symposium on Discrete algorithms*, pages 669–686. SIAM, 2014.

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

[47] Masoud Moshref, Minlan Yu, Ramesh Govindan, and Amin Vahdat. Scream: Sketch resource allocation for software-defined measurement. In *Proceedings of the 11th ACM Conference on Emerging Networking Experiments and Technologies*, pages 1–13, 2015.

[48] Sergei Vassilvitskii. Lecture 3: Counting on streams, 2017.

[49] Russel E Caflisch. Monte carlo and quasi-monte carlo methods. *Acta numerica*, 7:1–49, 1998.

[50] Anonymized internet traces 2018. https://catalog.caida.org/details/dataset/passive_2018_pcap. Accessed: 2022-6-29.

[51] Real-life transactional dataset. http://fimi.ua.ac.be/data/.

[52] Farmhash. https://github.com/google/farmhash.

[53] Source code of SketchConf. https://github.com/SketchConf/SketchConf.