JoinSketch: A Sketch Algorithm for Accurate and Unbiased Inner-Product Estimation

Feiyu Wang
Peking University

Qizhi Chen
Peking University

Yuanpeng Li
Peking University

Tong Yang
Peking University

Yaofeng Tu
ZTE Corporation

Lian Yu
Peking University

Bin Cui
Peking University

ABSTRACT

The inner-product estimation is the base of many important tasks in various big data scenarios, including measuring the similarity of streams in data stream processing, estimating join size in the database, and analyzing cosine similarity in various applications. Sketch, as a class of probabilistic algorithms, is promising in inner-product estimation. However, existing sketch solutions suffer from low accuracy due to neglecting the high skewness of real data. In this paper, we design a new sketch algorithm for accurate and unbiased inner-product estimation, namely JoinSketch. To improve accuracy, JoinSketch consists of multiple components and records items with different frequencies in different components. We theoretically prove that JoinSketch is unbiased and has lower variance than the well-known AGMS and Fast-AGMS sketch. The experimental results show that JoinSketch improves the accuracy of inner-product by 10 times on average while maintaining a comparable throughput. All code is open-sourced at Github [1].

ACKNOWLEDGMENTS

The research was partially supported by the National Natural Science Foundation of China (No. 61872011).

1 INTRODUCTION

In many big data scenarios, the data comes as a stream at high speed. There is a growing interest in processing and analyzing data streams in a single pass to offer statistic of the data stream, including frequencies [2, 3], heavy hitters [4, 5], heavy changes [6, 7], *e.t.c.* [8–11]. The inner-product of two data streams is an important statistic for data stream analysis, which is defined as the inner-product of their frequency vectors and the inner-product is equal to the size of the join of two data streams (see details in Section 2.1). We need to track the inner-product of two data streams in many scenarios. First, in data stream scenarios, the inner-product can be used to measure the similarity of two data streams, which is important in network measurement and data mining applications. For example, in data stream scenarios such as network flows in routers and web clicks in servers, there is a need to track the inner-product of different streams to help analyze the current running situation of the network. Second, it is significant in database systems to estimate the join size for the query optimizer[12–15]. In some cases of database systems, we need to treat all attribute values from a large table as a data stream[16] because the size of database tables is too large that we can only process them in one pass. Third, the cosine similarity of two data streams can be derived from the inner-product and is helpful for some data analysis tasks. However, it is impracticable and unnecessary to track the exact inner-product in data stream scenarios because of the high time cost and space cost to compute the exact statistic.

Researchers turn to probabilistic data structures for fast and accurate inner-product estimation. However, designing an appropriate algorithm is a great challenge because of the high speed and the huge size of data streams. Meanwhile, unbiased estimation is required in some distributed scenarios because biased estimation will lead to error accumulation and unbiased estimation is of theoretical elegance. Hence, the ideal inner-product estimation algorithms are supposed to meet three requirements. First, the algorithms have to process the data in one pass, and the algorithms are supposed to be very fast since the data stream comes at a rather high speed. Second, the accuracy of inner-product estimation should be high enough under small memory usage because the available memory in real scenarios such as routers is very limited. Third, the estimation provided by the algorithm is supposed to be unbiased.

Sketches are a class of hash-based probabilistic algorithms which is appropriate for data stream processing. There are several works focusing on sketch-based solutions for inner-product estimation, including the AGMS sketch [17, 18], the Fast-AGMS sketch [19], the Count-Min sketch [2], *e.t.c.* [20, 21].

The AGMS sketch [17, 18] uses a single counter to estimate the item [1] frequency of a data stream. It increments/decrements the counter with an equal probability when inserting an item. To estimate the inner-product of two data streams, one can simply multiply the AGMS sketch counters associated with the two data streams. However, the AGMS sketch suffers from a big variance and thus a high estimation error. To reduce the variance, researchers use multiple counters and take the median number as the estimation, at the cost of low throughput. Based on the AGMS sketch, the Fast-AGMS sketch [19] uses multiple hash functions to locate the counters to update, which significantly accelerates the insertion operation. The Count-Min sketch consists of an array of counters and is associated with multiple hash functions. It only increments the hashed counters when inserting an item. The inner-product is estimated by adding up the products of the corresponding counters of two Count-Min sketches. These algorithms are designed as universal algorithms which are capable of the inner-product estimation for data streams of various data distributions. However, in the scenarios of real data, their accuracy is usually poor because the real data often obeys unbalanced distribution. Real data usually consists of a few frequent items and many infrequent items. Hash

[1] We use "item" to represent an element in a data stream. A data stream is made of many items, and each item could appear more than once. For example, the item can be a 5-tuple in network measurement or a value from a database table.

collisions involving frequent items worsen the accuracy of inner-product estimation a lot. The Skimmed sketch [20] and the Red sketch [21] propose to estimate the inner-product by estimating the inner-product of frequent items and infrequent items separately. However, they require extensive computation to find frequent items before estimating the inner-product and need to get all item IDs in advance, which means these solutions are not one-pass and not practical consequently.

We propose JoinSketch to provide accurate, fast, and unbiased inner-product estimation for data streams. JoinSketch is based on a key observation that the real data often obeys unbalanced distribution such as Zipf [22, 23]. To take advantage of the natural characteristic of real data, we design JoinSketch to distinguish frequent items from the whole data and record them separately from infrequent items to improve accuracy.

Real data often obeys unbalanced distribution and is high-skewed in many scenarios. Most data items are infrequent, while only a few data items are very frequent. The mixture of frequent items and infrequent items is the key resource of the estimation error because the error of the inner-product estimation will be huge if hash collisions, especially the ones between frequent items and hash collisions between frequent items and infrequent items occur. The hash collisions can be classified into 3 types, (a) hash collisions between frequent items, (b) hash collisions between frequent items and infrequent items, and (c) hash collisions between infrequent items. Different types of hash collisions account for the inner-product estimation error to different extents. We illustrate how different types of hash collisions affect the inner-product estimation in Figure 1. Let's consider a data stream F and it consists of 6 distinct items. The frequency vector of F is $f = (f_1, f_2, ..., f_6) = (1000, 1000, 1, 1, 1, 1)$ where f_i represents the frequency of i-th item e_i. We take the Count-Min sketch with 5 counters and one hash function as an example. For brevity and convenience, we consider inner-product between F and itself. The true value of the inner-product is $J = f \odot f = 2,000,004$. As shown in Figure 1(a), type (a) hash collisions lead to a large error, frequent items e_1 and e_2 are over-estimated by 1000, and the inner-product is $\hat{J}_a = 4,000,004$, which is about two times as the true value. As shown in fig. 1(b), type (b) hash collisions bring big error to the frequency estimation of the infrequent item e_4 because e_4 is hashed to the same counter as the frequent item e_2. The estimated frequency of e_4 is 1001, which is 1000 times larger than its true frequency. The inner-product estimation is $\hat{J}_b = 2,002,004$. Figure 1(c) is an ideal situation where there is only a type (c) hash collision between two infrequent items e_3 and e_5. The estimated value of inner-product is $\hat{J}_c = 2,000,006$. Type (c) hash collisions lead to very small errors in inner-product estimation, which is acceptable.

JoinSketch consists of three components: the frequent part, the medium part and the infrequent part. The frequent part is a hash table used to record frequent items accurately because the frequent items are few yet important. The infrequent part is a Fast-AGMS sketch used to record infrequent items. The infrequent part only costs a small amount of memory since the frequency of items in infrequent part is so low that we can use small counters in infrequent part. For example, we can use only 8-bit counters in the infrequent part. The medium part is the key component of JoinSketch. It distinguishes frequent items on the basis of item frequency. JoinSketch firstly inserts an item to the medium part temporarily. If it grows up to exceed a predefined threshold T, it will be recorded in the frequent part as a frequent item. Otherwise, it is likely to be eliminated to the infrequent part as an infrequent item if there is no room for new-come items in the medium part.

JoinSketch stores items separately in three components. For every component of JoinSketch, we query the estimation of partial inner-product between it and every component of another JoinSketch which is constructed from another data stream. Thus, the inner-product can be derived from nine pieces (see details in Section 3.3). It is notable that the frequent part and the medium part record the frequency with no error. The infrequent part is a Fast-AGMS sketch and provides unbiased inner-product estimation. Combining the above two characteristics, it can be proved that JoinSketch provides unbiased inner-product estimation (see details in Section 4.1).

The advantages of JoinSketch over existing solutions are twofold. On the one hand, by separating frequent items and infrequent items, we improve the accuracy by reducing hash collisions. To be specific, we totally eliminate type (a) hash collisions and type (b) hash collisions because frequent items are recorded in the frequent part. On the other hand, since the frequencies of items in the infrequent part are small, we can use smaller counters than existing sketches, which means more counters under the same memory usage. More counters lead to fewer type (c) hash collisions between infrequent items. To sum up, JoinSketch improves accuracy by reducing all of the 3 types of hash collisions simultaneously.

The experimental results show that JoinSketch maintains unbiasedness, and the error is 10 times on average smaller than the state-of-the-art on high-skewed datasets. Even on datasets with little skewness, JoinSketch can still do better than existing algorithms. All code is open-sourced at Github [1].

Key Contributions:

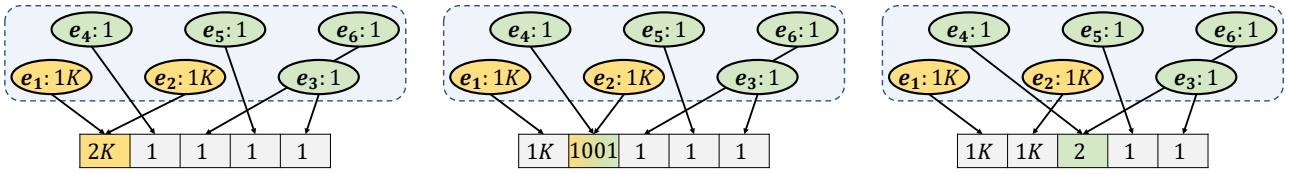
- We propose JoinSketch based on separating frequent items and infrequent items to the accuracy of the inner-product estimation.
- We theoretically prove that the estimation given by JoinSketch is unbiased. And we give a mathematical analysis of the variance of the estimation.
- We conduct extensive experiments to evaluate the performance of JoinSketch on various synthetic and real-world datasets. The results show that on high-skewed datasets, the error of JoinSketch is 10 times on average smaller than the state-of-the-art.

2 BACKGROUND

In this section, we first present the definition of the inner-product estimation, then introduce the well-known AGMS and Fast-AGMS sketch, which are the basis of our JoinSketch.

2.1 Problem Definition

Let F be a data stream with S items and G be another data stream. We use e_i to represent a data item in a data stream. Assume D is the domain of all items. $|D| = N$ and $D = \{e_{\beta_1}, ..., e_{\beta_i}, ..., e_{\beta_N}\}$. $F = [e_1, ..., e_i, ..., e_S]$, where each item e_i belongs to D. Note that items in D are distinct, and items in F or G may not be. For the data stream F, we define the frequency vector $f = (f_1, ..., f_i, ..., f_N)$



(a) Hash collisions between frequent items lead to great error.

(b) Hash collisions between frequent items and infrequent items lead to relatively big error.

(c) Hash collisions between infrequent items lead to small error, which is acceptable.

Figure 1: Types of Hash Collisions.

where f_i represents the frequency of the item e_{β_i}. Similarly, we have the frequency vector of G, and $g = (g_1, ..., g_i, ..., g_N)$. The inner-product of two data streams F and G is defined as

$$J = f \odot g = \sum_{i=1}^{N} f_i \cdot g_i. \tag{1}$$

Join predicate between F and G outputs all tuples (e_i, e_j) where $e_i = e_j$ and $e_i \in F,\ e_j \in G$. The inner-product is equivalent to the join size.

2.2 Sketch-Based Inner-product Estimation

Sketches are a variety of probabilistic data structures to approximate some statistical characteristics of big data. Sketch algorithms are widely used in big data scenarios, especially in high-speed data stream processing and analyzing. The AGMS sketch[17, 18] and Fast-AGMS sketch [19] are typical sketch algorithms for the task of inner-product estimation.

2.2.1 AGMS Sketches. The AGMS sketch [17, 18] is the first sketch-based algorithm for inner-product estimation. An AGMS sketch consists of only a single counter $sk(F)$ that summarizes all of the frequency information of a data stream. The AGMS sketch is associated with ξ, a family of $\{+1, -1\}$ random variables and 4-wise independent. For every item e_i in data stream $F = [e_1, e_2, ..., e_S]$, the AGMS sketch first calculates $\xi(e_i)$ and then add it to its single counter. The sketch counter $sk(F)$ can be calculated as follows:

$$sk(F) = \sum_{e_i \in F} \xi(e_i). \tag{2}$$

The standard technique to estimate the inner-product is to construct AGMS sketches for data streams F and G, respectively, with the same random function ξ. The inner-product of data stream F and G can be estimated as:

$$\hat{J} = Est(J) = sk(F) \times sk(G). \tag{3}$$

The estimator suffers from a big variance. Thus, it is required to use multiple independent single AGMS sketches to improve accuracy by taking the median or average of these sketches. However, such technique leads to poor throughput and thus is impractical.

2.2.2 Fast-AGMS Sketches. A Fast-AGMS sketch [19] consists of an array of m counters. Besides the random function ξ, the Fast-AGMS sketch has a hash function h, which is used to hash an item to a random counter. For item e_i in data stream F, the Fast-AGMS sketch first calculates $h(e_i)$ and updates the $h(e_i)\%m$-th counter (denoted as $sk(F)[h(e_i)\%m]$) by adding $\xi(e_i)$. The Fast-AGMS sketch is capable to estimate the frequency of e_i by the product $sk(F)[h(e_i)\%m] \times \xi(e_i)$. The hash function h helps reduce the number of counters to be updated when inserting a new item. Compared with the AGMS sketch, under the same space usage, the Fast-AGMS sketch has the same variance as the AGMS sketch but lower update and query time complexity.

As for inner-product estimation, the Fast-AGMS sketches for data streams F and G are constructed in advance with the same hash function h and random function ξ. The inner-product estimation is the summation of the product of corresponding counters of the two Fast-AGMS sketches. In other words, if we view the Fast-AGMS sketch as a column vector, the estimation can be written as:

$$\hat{J} = Est(J) = \sum_{i=1}^{m} sk(F)[i] \times sk(G)[i] = \overrightarrow{sk(F)}^{\mathrm{T}} \cdot \overrightarrow{sk(G)}. \tag{4}$$

The Fast-AGMS sketch also suffers from hash collisions. If two or more items with high frequency are hashed into the same counter, the accuracy of the inner-product estimation will be poor. In practice, researchers usually use the median estimation of multiple Fast-AGMS sketches to improve accuracy.

3 JOINSKETCH

In this section, we first present the rationale of JoinSketch. Then we describe the data structure and operations of JoinSketch. After that, we show how JoinSketch estimates the inner-product of two data streams. Finally, we present some optimization techniques for JoinSketch. We list the symbols used frequently in this paper in Table 1.

Table 1: Symbols Frequently Used in This Paper.

Symbols	Meaning
F, G	a data stream
f, g	a frequency vector
D	the domain of items and $D = \{e_{\beta_1}, e_{\beta_2}, ..., e_{\beta_N}\}$
N	the cardinality of D and $N = \|D\|$
f_i	the frequency of i-th item e_{β_i}
T	a predefined threshold for frequent items
J	the inner-product of F and G
$\hat{J}$	the estimated inner-product of F and G
$MP[i]$	the i^{th} bucket in the medium part
$B[i][e].counter$	the counter of item e if e exists in $B[i]$
$H(.)$	the hash function used in the frequent part
$H_m(.)$	the hash function used in the medium part
$h_i(.)$	the i^{th} hash function in the infrequent part
$\xi_i(.)$	the i^{th} random function in the infrequent part

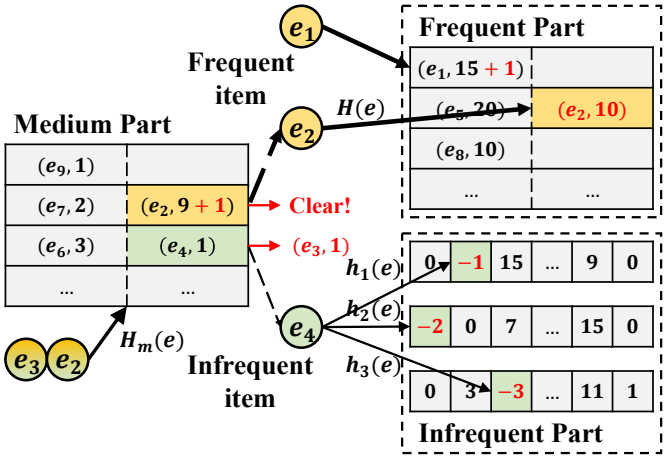


Figure 2: Data Structure of JoinSketch.

3.1 Rationale of JoinSketch

The key idea of JoinSketch is to distinguish frequent items and infrequent items from mixed data to improve the accuracy of inner-product estimation. JoinSketch consists of three components: the infrequent part, the frequent part, and the medium part. The infrequent part is a Fast-AGMS sketch used to record infrequent items. The frequent part is a hash table used to record frequent items. The medium part is the key component of JoinSketch, which separates items based on their frequency. It is used to distinguish frequent items from all data items. We organize these three components as shown in Figure 2. When inserting an item, JoinSketch first accumulates it in the medium part. If the frequency of an item grows big enough and exceeds a predefined threshold T, it is supposed to be a frequent item and be stored in the frequent part. Otherwise, it is supposed to be stored in the medium part or the infrequent part if there is no room for new-come items in the medium part.

3.2 Data Structure and Operations

3.2.1 Data Structure. As shown in Figure 2, the data structure of JoinSketch consists of three components, including the frequent part, the medium part and the infrequent part from top to bottom.

Frequent part: The frequent part FP is a hash table of k buckets and is associated with a hash function $H(.)$. Each bucket of the frequent part consists of c entries. Each entry stores an item and its current frequency.

Infrequent part: The infrequent part IFP is a Fast-AGMS sketch. Specifically, the infrequent part consists of d arrays (IFP_1,IFP_2,$\cdots$, IFP_d). Each array consists of w counters and is associated with a hash function $h_i(.)$ and a random function $\xi_i(.)$.

Medium part: The medium part MP is the key component of JoinSketch. As shown in Figure 2, the data structure of the medium part is an array of l buckets, and each bucket includes m entries. Each entry is composed of an item and a counter. The medium part is associated with a hash function $H_m(.)$.

3.2.2 Operations. JoinSketch supports two operations: inserting an item and looking up the frequency of an item.

Algorithm 1: Insertion of medium part.

Input: Item e

```
1  h ← H_m(e)
2  if e ∈ MP[h] then
3      MP[h][e].counter ← MP[h][e].counter + 1
4      if MP[h][e].counter < T then
5          return
6      else
7          insert ⟨e, MP[h][e].counter⟩ to FP
8          clear MP[h][e]
9          return
10 else if MP[h] is not full then
11     insert ⟨e, 1⟩ to an empty entry of MP[h]
12     return
13 else
14     y ← argmin_y(MP[h][y].counter)
15     insert ⟨y, MP[h][y].counter⟩ to IFP
16     claer MP[h][y] and insert ⟨e, 1⟩ to MP[h]
17     return
```

Insertion: When an item e is inserted, JoinSketch first checks whether it is stored in the frequent part. If so, we simply increment its counter in the frequent part. Otherwise, the item will be inserted into the medium part. The medium part hashes e to bucket $MP[h]$ using an associated hash function $H_m(.)$, where $h = H_m(e)$. There are three different cases according to whether bucket $MP[h]$ contains item e. The pseudo-code is shown in algorithm 1.

***Case 1** (line 2-9):* If bucket $MP[h]$ contains item e, the counter of item e will be increased by 1. Then we check the updated counter. If the updated frequency of item e is less than threshold T, the insertion ends. Otherwise, we insert e into the frequent part with current frequency and remove it from the medium part.

***Case 2** (line 10-12):* If bucket $MP[h]$ does not contain item e but there exists at least one empty entry, we insert item e into an empty entry of bucket $MP[h]$.

***Case 3** (line 13-17):* If bucket $MP[h]$ does not contain item e and it is full, we need to evict an entry to make room for item e. We select the smallest item y in bucket $MP[h]$. Item y is believed to be an infrequent item, and is then inserted into the infrequent part.

Example I: As shown in Figure 2, the frequent part and the medium part of JoinSketch consist of multiple buckets. Each bucket consist of 2 entries. The infrequent part is a Fast-AGMS sketch with three arrays. The threshold of frequent item $T = 10$. When inserting item e_1 to JoinSketch, we first check whether e_1 is in the frequent part: we compute hash function $H(e_1)$ to locate the 2^{nd} bucket in the frequent part. Since e_1 is in the bucket, we simply increment the counter by 1 to 16.

Example II: When inserting item e_2 to JoinSketch, e_2 is not in the frequent part. Therefore, we insert it to the medium part: we compute hash function $h_m(e_2)$ to locate the 2^{nd} bucket in the medium part. e_2 is in the bucket, therefore, we increment the corresponding counter by 1 to 10. After that, we compare the counter with the

threshold T. $10 \geqslant T$, therefore, we insert e_2 with frequency 10 to the frequent part, and remove it from the medium part.

Example III: When inserting item e_3 to JoinSketch, e_3 is not in the frequent part. Therefore, we insert it to the medium part: we locate it to the 3^{rd} bucket. The bucket is full, therefore, we remove the least frequent item e_4 from the bucket, and insert it to the infrequent part. Finally, we insert e_3 with frequency 1 to the bucket.

Discussion on the data structure: Our solution is a 3-part design. According to reviewer comments, we can combine the frequent part and medium part into one, and get a 2-part design. Indeed, the 2-part design is simple and easy to deploy. 3-part design is a little more complicated but fine-grained. The frequent part is supposed to store items with larger frequency than the medium part. Meanwhile, we use bigger counters in the frequent part and smaller counters in the medium part, which can further save memory usage. In section 3.4.3, we present an optimization technique using fingerprints, which can also benefit from the 3-part design.

Algorithm 2: Lookup of JoinSketch

Input: Item e
Output: The frequency estimation of item e

```
1 ret ← 0
2 if e ∈ FP[H(e)] then
3     ret+ = FP[e].counter
4 else if e ∈ MP[H_m(e)] then
5     ret+ = MP[H_m(e)][e].counter
6 for i = 1 → d do
7     S[i] ← IFP_i[h_i(e)] × ξ_i(e)
8 ret+ = median_{1≤i≤d}(S[i])
9 return ret
```

Lookup (frequency estimation): Besides the inner-product estimation (see details in Section 3.3), one can use JoinSketch to estimate an item's frequency. The pseudo-code for lookup operation is shown in Algorithm 2. JoinSketch initialize ret with 0 in the beginning (*line 1*). To look up the frequency of an item e, JoinSketch first checks whether item e exists in the frequent part or the medium part (*line 2-5*). If so, add the corresponding counter's value to ret. Afterwards, JoinSketch will look up e in the infrequent part. The Infrequent part is associated with d hash functions. JoinSketch locates these d hashed counters and then adds the median value of the counters to ret (*line 6-8*). JoinSketch returns ret as the frequency estimation of item e.

Discussion on Lookup: Note that we always query the infrequent part for frequency estimation. The reason is as follows. For an arbitrary item e, 1) if it does not exist in the frequent part or the medium part, we can tell that all of its instances are recorded in the infrequent part. In this case, JoinSketch only needs to look up its frequency in the infrequent part. 2) If e exists in the frequent part or the medium part, however, we are faced with a more complicated situation. The counter value will be the partial frequency of e if some instances of e have been evicted to the infrequent part before it grows into a frequent item. 2.1) If JoinSketch never evicts e into infrequent part, the counter value should be the true value of e's frequency. 2.2) Otherwise, it is a under estimation. Because we do not use additional flag to indicate whether the above eviction occurs to keep the data structure concise, we choose to always look up the infrequent part to obtain unbiased frequency estimation. In addition, it is also feasible to only return the counter value as the estimated frequency and we compare the accuracy of these two methods in Section 6.

Discussion on frequency estimation and inner-product estimation: Indeed, JoinSketch is a frequency sketch that does inner-product estimation. Using current evaluation metrics, nevertheless, we observe that accurate frequency estimation does not always lead to accurate inner-product estimation. Prior work usually evaluates the performance of sketch algorithms using metrics of AAE (absolute average error) and ARE (average relative error). Unfortunately, these metrics can not reflect the accuracy for inner-product estimation of sketches. An example is as follows. Consider we have two data streams F and G in which there are two items e_1 and e_2. The frequency vectors of F and G are $f = (10000, 10)$ and $g = (10001, 11)$, respectively. Assume the sketch gives an error-free estimation $\hat{g} = g$ while the estimation of f is not error-free.

- **Example I**: Consider two cases and the frequency estimation is $\hat{f}_{case1} = (11000, 11)$, $\hat{f}_{case2} = (10001, 1010)$. The **AAE** of both the cases is $\frac{1000+1}{10000+10}$. The inner-product estimation is $\hat{J} = \hat{f}_{case1} \odot \hat{g} = 110,011,121$ for case 1 and $\hat{J} = \hat{f}_{case2} \odot \hat{g} = 100,031,111$ for case 2. **The same AAE** for frequency estimation leads to inner-product estimation in sharp contrast.
- **Example II**: Consider two cases and the frequency estimation is $\hat{f}_{case1} = (11000, 11)$, $\hat{f}_{case2} = (10000, 12)$. The **ARE** of both the cases is 10%. The inner-product estimation is $\hat{J} = \hat{f}_{case1} \odot \hat{g} = 110,011,121$ for case 1 and $\hat{J} = \hat{f}_{case2} \odot \hat{g} = 100,010,132$ for case 2. **The same ARE** for frequency estimation leads to inner-product estimation in sharp contrast.

The above example shows that accurate frequency estimation does not indicate the accurate inner-product estimation. It is still an open question to design new appropriate metrics. We get an insight that we need to obtain a higher accuracy for frequent items than infrequent items, which motivates us to separate frequent items from infrequent items. JoinSketch is proposed based on the separation of items.

3.3 Inner-product Estimation

Given two data streams F and G, we first construct JoinSketch for them, *i.e.*, we insert all items in F and G into JoinSketch respectively. We name them $JoinSketch_F$ and $JoinSketch_G$. As shown in Figure 2, items are stored in the frequent part, the medium part and the infrequent part. Thus, the inner-product estimation can be obtained by adding up the inner-product of nine pieces, including (1) frequent-frequent, (2) frequent-medium, (3) frequent-infrequent, (4) medium-frequent, (5) medium-medium, (6) medium-infrequent, (7) infrequent-frequent, (8) infrequent-medium, and (9) infrequent-infrequent.

Formally, we define frequent vector $f_F = (f_{F1}, f_{F2}, ..., f_{FN})$ where $f_{Fi} \geq T$ for $i = 1, ..., N$, medium vector $f_M = (f_{M1}, f_{M2}, ..., f_{MN})$ where $f_{Mi} < T$ and infrequent vector $f_I = (f_{I1}, f_{I2}, ..., f_{IN})$. Let the frequent vector f_F be the partial frequency vector of what has been recorded in the frequent part. Let the medium vector f_M be the

partial frequency vector of what has been recorded in the medium part. Note that the f_F is not the frequency vector of the frequent items, but represents the frequency vector of the instances recorded in the frequent part. For example, if an item e_{β_i} is never evicted to the infrequent part before it grows to be a frequent item, f_{Fi} will be the true value of the frequency of e_{β_i}. We have $f_{Fi} = f_i$, $f_{Mi} = 0$ and $f_{Ii} = 0$. Once JoinSketch evicts e_{β_j} to the infrequent part, some instances of this items are recorded in the infrequent part. In this case, $f_{Fj} < f_j$ and $f_{Ij} > 0$. Similarly, the f_M is the frequency vector of the instances which are recorded in the medium part. Since an instance of any item is recorded in one and only one part of JoinSketch, we have $f = f_F + f_M + f_I$ and $g = g_F + g_M + g_I$. The inner-product of F and G can be calculated by

$$\begin{aligned} J = f \odot g &= (f_F + f_M + f_I) \odot (g_F + g_M + g_I) \\ &= f_F \odot g_F + f_F \odot g_M + f_F \odot g_I \\ &+ f_M \odot g_F + f_M \odot g_M + f_M \odot g_I \\ &+ f_I \odot g_F \ + f_I \odot g_M \ + f_I \odot g_I \\ &= J_{FF} + J_{FM} + J_{FI} + J_{MF} + J_{MM} + J_{MI} + J_{IF} + J_{IM} + J_{II}. \end{aligned} \tag{5}$$

These nine addends in Equation (5) correspond to the nine pieces above. JoinSketch estimates the inner-product by estimating the nine pieces respectively. For piece (1) J_{FF}, JoinSketch compares every item recorded in the frequent part of $JoinSketch_F$ with the one recorded in the frequent part of $JoinSketch_G$. For the same item, JoinSketch multiplies the corresponding counters and sum up all products. We estimate (2) J_{FM} (4) J_{MF} (5) J_{MM} using the same method. For piece (9) J_{II}, JoinSketch sums up all of the multiplications of corresponding counters of the infrequent part, which is the same as the Fast-AGMS sketch. JoinSketch traverses the frequent parts to estimate (3) J_{FI} and (7) J_{IF}. For each item in the frequent part of $JoinSketch_F$, we look up the estimated frequency of the item in the infrequent part of $JoinSketch_G$ and then estimate (3) J_{FI}. Similarly, we traverse the frequent part of $JoinSketch_G$ and then get (7) J_{IF}. The same method is applied to estimate (6) J_{MI} and (8) J_{IM}. By adding the results of the above nine pieces, we get the inner-product estimation of data stream F and data stream G.

3.4 Optimizations

3.4.1 Extension of Frequent Part. Ideally, the size of the frequent part should match the number of frequent items. If the size of frequent part is too small, a number of frequent items may be lost; if the size of frequent part is too large, memory will be wasted. Estimating the number of frequent items in advance is difficult, so we decide to dynamically extend the size of the frequent part. The method is borrowed from ElasticSketch [24]. If a bucket in the frequent part is full, we copy the frequent part and merge the frequent part and the copied one together as the new frequent part. Suppose the old frequent part contains k buckets. The new one contains $2k$ buckets, and thus we change the hash function from $H(.)\%k$ to $H(.)\%(2k)$. After the extension, half of the items should be removed. The removal operation can be done incrementally.

3.4.2 Using SIMD Instructions. The medium part and the frequent part consist of l buckets and each bucket consists of m entries. When inserting an item, it is hashed into a bucket. Afterward, we need to scan all entries in this bucket to determine whether or not the item exists in this bucket, which is costly. To improve insertion performance, we use SIMD (Single Instruction Multiple Data) instructions to scan a specific hashed bucket [25, 26]. With SIMD, we can scan and compare multiple entries with a single instruction. In order to make JoinSketch compatible with SIMD, we set the number of entries $m = 4$ or $m = 8$.

3.4.3 Fingerprint. We use fingerprints instead of the full item key in order to save memory usage. The fingerprint of an item e is a fixed-length hash value of the item. For example, we can use a hash function H_{fp} to calculate the fingerprint and the fingerprint of e is $H_{fp}(e)$. We use the fingerprint to save memory footprint if the item key is long. The usage of fingerprint, however, is likely to bring about fingerprint collisions which would downgrade the accuracy of inner-product estimation. Hence, we use longer fingerprints in the frequent part to avoid fingerprint collisions as much as possible. We set the length of fingerprint in the frequent part $L_F = 32$. And we set the length in the medium part $L_M = 22$. The reason for using 22-bit fingerprint is that in the experiment we find that 10-bit counters is big enough for the medium part, and the remaining 22 bits of the 32-bit variable can be used as the fingerprint.

3.4.4 Picking the threshold T. In terms of picking the threshold T, one feasible method is to initialize T with a moderate value and adjust it according to the status of the frequent part. The initial value of T can be set according to the total number of items and the estimated number of distinct items (if available). For example, one can use the average frequency or a small portion of all items as the initial value. JoinSketch checks the number of items in the frequent part periodically. If the number of items in the frequent part is small (e.g. $\leq rz \cdot kc$, r is a constant, z is the number of items inserted to JoinSketch and kc is the total number of entries in the frequent part), it suggests that the threshold T is too high to find the frequent items and JoinSketch will lower the threshold T. If the frequent part is extended frequently, which indicates that many items which are not so frequent are inserted to the frequent part, JoinSketch will take a higher T. In this way, JoinSketch will take a proper threshold value to separate frequent and infrequent items.

3.5 Extension to Multi-Way Joins

To show how to extend JoinSketch to multi-way join size estimation, we first introduce the concept of attributes. An item in data streams may consist of several attributes. Let $e_i.A_j$ be the attribute A_j of the item e_i, and let $F.A_j$ be the attribute A_j of the data stream F. An example of multi-way join is like

$$F \bowtie G \bowtie H \text{ where } F.A_1 = G.A_1 \wedge G.A_2 = H.A_2 \tag{6}$$

The typical work for multi-way joins is Compass [16], which uses multi-dimensional Fast-AGMS sketches. Essentially, JoinSketch is KV tables (the frequent part and the medium part) and a Fast-AGMS sketch (the infrequent part). The infrequent part perfectly fits into Compass. As for the KV tables, we modify them for multi-way join as follows. We replace the item key in the frequent part and the medium part with multiple item keys which are involved in the join. For example, we record A_1 for data stream F, A_2 for data

stream H, and both A_1 and A_2 for data stream G. We can obtain the inner-product estimation in the same way as 2-way join.

4 THEORETICAL ANALYSIS

4.1 Unbiasedness of JoinSketch

Theorem 1. *The inner-product estimation of two data streams given by the standard version of JoinSketch is unbiased.*

Proof. Suppose f and g are frequency vectors of two data streams F and G. $\hat{J}$ is the inner-product estimation given by JoinSketch and the true value of the inner-product is J. According to Section 3.3, the inner-product estimation is obtained by

$$\hat{J} = \hat{J_{FF}} + \hat{J_{FM}} + \hat{J_{FI}} + \hat{J_{MF}} + \hat{J_{MM}} + \hat{J_{MI}} + \hat{J_{IF}} + \hat{J_{IM}} + \hat{J_{II}}. \tag{7}$$

The frequent part and the medium part record a part of all instances of an frequent item with no error according to their definition, hence we have $\hat{f_F} = f_F$, $\hat{f_M} = f_M$, $\hat{g_F} = g_F$ and $\hat{g_M} = g_M$. Therefore, we have

$$\begin{aligned} \hat{J_{FF}} &= \hat{f_F} \odot \hat{g_F} = f_F \odot f_F = J_{FF} \\ \hat{J_{FM}} &= \hat{f_F} \odot \hat{g_M} = f_F \odot f_M = J_{FM} \\ \hat{J_{MF}} &= \hat{f_M} \odot \hat{g_F} = f_M \odot f_F = J_{MF} \\ \hat{J_{MM}} &= \hat{f_M} \odot \hat{g_M} = f_M \odot f_M = J_{MM}. \end{aligned} \tag{8}$$

Further, note that

$$\begin{aligned} \hat{J_{FI}} &= \hat{f_F} \odot \hat{g_I} = f_F \odot \hat{g_I} \\ \hat{J_{MI}} &= \hat{f_M} \odot \hat{g_I} = f_M \odot \hat{g_I} \\ \hat{J_{IF}} &= \hat{f_I} \odot \hat{g_F} = \hat{f_I} \odot g_F \\ \hat{J_{IM}} &= \hat{f_I} \odot \hat{g_M} = \hat{f_I} \odot g_M. \end{aligned} \tag{9}$$

Fast-AGMS gives unbiased estimation for both item frequency and inner-product. Therefore, $\mathbb{E}(\hat{f_I}) = f_I$, $\mathbb{E}(\hat{g_I}) = g_I$ and $\mathbb{E}(J_{II}) = J_{II}$. Hence, the estimation for the remaining five pieces is unbiased as well. We have that

$$\begin{aligned} \mathbb{E}(\hat{J}) &= \mathbb{E}(\hat{J_{FF}}) + \mathbb{E}(\hat{J_{FM}}) + \mathbb{E}(\hat{J_{FI}}) + \mathbb{E}(\hat{J_{MF}}) + \mathbb{E}(\hat{J_{MM}}) \\ &\quad + \mathbb{E}(\hat{J_{MI}}) + \mathbb{E}(\hat{J_{IF}}) + \mathbb{E}(\hat{J_{IM}}) + \mathbb{E}(\hat{J_{II}}) \\ &= J_{FF} + J_{FM} + J_{FI} + J_{MF} + J_{MM} + J_{MI} + J_{IF} + J_{IM} + J_{II} \\ &= J. \end{aligned} \tag{10}$$

Therefore, the estimation given by JoinSketch is unbiased. □

Analysis on optimizations: We present several optimization techniques in Section 3.4. The extension of frequent part and using SIMD instructions don't affect the unbiasedness of the inner-product estimation. The fingerprint, however, will affect the unbiasedness. We analyze the issue on the fingerprint in Section 4.4.

4.2 Variance of JoinSketch

As mentioned in Section 4.1, JoinSketch provides unbiased estimation of the inner-product. In this section, we prove the estimation offered by JoinSketch is of less variance, and thus JoinSketch improves estimation accuracy compared with prior arts. We start from the variance of the estimation given by the Fast-AGMS sketch.

Lemma 2. *Consider a Fast-AGMS sketch with n_{Fast} counters. The variance of the inner-product estimation (denoted as $\hat{J_{Fast}}$) is*

$$Var[\hat{J_{Fast}}] \le 2||f||_2^2||g||_2^2/n_{Fast} = B_{Fast} \tag{11}$$

according to [27] where f and g is the frequency vector of data streams F and G.

Theorem 3. *The bound of the variance of inner-product estimation $\hat{J}$ given by JoinSketch satisfies that*

$$Var[\hat{J}] \le \left(||f_U||_2^2||g_I||_2^2 + ||g_U||_2^2||f_I||_2^2 + 2||f_I||_2^2||g_I||_2^2\right)/n$$

where $f_U = f_F + f_M$, $g_U = g_F + g_M$ and n is the number of counters in the infrequent part.

Proof. JoinSketch stores the data stream in three components. Note that the frequent part and the medium part only store the part of all instances of a frequent item with no error. We can consider the two parts as a whole and thus obtain the inner-product estimation provided by JoinSketch $\hat{J}$ from

$$\hat{J} = \hat{J_{UU}} + \hat{J_{UI}} + \hat{J_{IU}} + \hat{J_{II}} \tag{12}$$

where $\hat{J_{UU}} = \hat{f_U} \odot \hat{g_U}$ and $f_U = f_F + f_M$. Since the f_F and f_M are independent from f_I, the variance of $\hat{J}$ consists of four parts.

$$Var[\hat{J}] = Var[\hat{J_{UU}}] + Var[\hat{J_{UI}}] + Var[\hat{J_{IU}}] + Var[\hat{J_{II}}]. \tag{13}$$

Since there is no error in frequency vector f_F and f_M, $\hat{J_{UU}} = J_{UU}$ and the variance of first part is

$$Var[\hat{J_{UU}}] = 0. \tag{14}$$

The variance of the second part $\hat{J_{UI}}$ can be derived based on the variance of frequency estimation of the Fast-AGMS sketch. The variance of frequency estimation of the Fast-AGMS sketch is

$$Var[\hat{f_i}] \le ||f||_2^2/n \tag{15}$$

where f is the frequency vector and n is the number of counters in the Fast-AGMS sketch. Combined with Equation (15)

$$\begin{aligned} Var[\hat{J_{UI}}] &= Var[\hat{f_U} \odot \hat{g_I}] \\ &= f_U \odot Var[\hat{g_I}] \le ||f_U||_2^2||g_I||_2^2/n. \end{aligned} \tag{16}$$

The third part $\hat{J_{IU}}$ is symmetric to the second part $\hat{J_{UI}}$. The variance of $\hat{J_{IU}}$ is

$$Var[\hat{J_{IU}}] \le ||g_U||_2^2||f_I||_2^2/n. \tag{17}$$

The fourth part $\hat{J_{II}}$ is the estimation from the infrequent part which is a Fast-AGMS sketch. The formula of the fourth part's variance is similar to Equation (11).

$$Var[\hat{J_{II}}] \le 2||f_I||_2^2||g_I||_2^2/n. \tag{18}$$

Substituting the above results into Equation (13), the bound of variance of the inner-product estimation given by JoinSketch is

$$\begin{aligned} Var[\hat{J}] =& Var[\hat{J_{UU}}] + Var[\hat{J_{UI}}] + Var[\hat{J_{IU}}] + Var[\hat{J_{II}}] \\ \le& \left(||f_U||_2^2||g_I||_2^2 + ||g_U||_2^2||f_I||_2^2 + 2||f_I||_2^2||g_I||_2^2\right)/n \\ =& B. \end{aligned} \tag{19}$$

□

Theorem 4. *The bound of JoinSketch is less than the bound of Fast-AGMS, i.e., $B \le B_{Fast}$ if $n \ge n_{Fast}$*

Proof. Since $f_i \geq 0$ and $g_i \geq 0$, we have

$$\begin{aligned}
B_{Fast} =& 2||f||_2^2||g||_2^2/n_{Fast} = 2||f_U + f_I||_2^2||g_U + g_I||_2^2/n_{Fast} \\
=& 2(||f_U||_2^2 + ||f_I||_2^2 + 2f_U \odot f_I)(||g_U||_2^2 + ||g_I||_2^2 + 2g_U \odot g_I)/n_{Fast} \\
=& 2(||f_U||_2^2||g_U||_2^2 + ||f_U||_2^2||g_I||_2^2 + ||f_I||_2^2||g_U||_2^2 + ||f_I||_2^2||g_I||_2^2 \\
&+ 4f_U \odot f_I\ g_U \odot g_I + 2f_U \odot f_I||g||_2^2 + 2g_U \odot g_I||f||_2^2)/n_{Fast} \\
=& \left(||f_U||_2^2||g_I||_2^2 + ||g_U||_2^2||f_I||_2^2 + 2||f_I||_2^2||g_I||_2^2\right)/n_{Fast} \\
&+ \text{Rem}/n_{Fast}
\end{aligned} \tag{20}$$

where Rem stands for the word *remain* for brevity and $\text{Rem} > \left(||f_U||_2^2||g_I||_2^2 + ||g_U||_2^2||f_I||_2^2 + 2||f_U||_2^2||g_U||_2^2\right)/n_{Fast}$.

$$\begin{aligned}
\frac{B_{Fast}}{B} =& \frac{2||f||_2^2||g||_2^2/n_{Fast}}{\left(||f_U||_2^2||g_I||_2^2 + ||g_U||_2^2||f_I||_2^2 + 2||f_I||_2^2||g_I||_2^2\right)/n} \\
=& \frac{n}{n_{Fast}}\left(1 + \frac{\text{Rem}}{||f_U||_1^2||g_I||_2^2 + ||g_U||_1^2||f_I||_2^2 + 2||f_I||_2^2||g_I||_2^2}\right).
\end{aligned} \tag{21}$$

Assume that $n \geq n_{Fast}$, *i.e.*, the number of counters in the infrequent part of JoinSketch is equal or greater than the number of counters in the Fast-AGMS. No matter how much memory the frequent part and the medium part consumes, $\text{Rem} > 0$ holds. Therefore, $B_{Fast}/B > 1$. The bound of JoinSketch is less than the bound of Fast-AGMS, *i.e.*, $B \leq B_{Fast}$. □

As stated above, because the frequency of items stored in the infrequent part is relatively low, we use small counters in the infrequent part. Under the same memory constraint, the assumption $n \geq n_{Fast}$ usually holds. We show that JoinSketch has a smaller bound of the variance than Fast-AGMS. Note that B_{Fast}/B may be very big in some cases and it is highly related to how big $||f_U||_2^2$ and $||g_U||_2^2$ are. Intuitively, the more skewed the data is, the bigger the ratio B_{Fast}/B is. It implies that JoinSketch is supposed to perform much better when the data is high-skewed.

4.3 Effectiveness of Finding Frequent Items

In this section, we provide theoretical analysis on the effectiveness of finding frequent items for JoinSketch. Every bucket in the medium part is the same as each other, and every bucket is independent of each other. We only analyze one bucket and the items hashed to it. Assume that the number of entries in this bucket is M. Before the formal analysis, we make assumptions about the data stream to simplify the problem. Since the number of frequent items is rather few compared to the number of all distinct items, we assume the data stream hashed to the bucket contains only one frequent item x with frequency t and all of the other items in this data stream appear for one time. The data stream contains N items in total. We use x_i to represent item x that appears for the i-th time. We use k_i to represent the number of other items between 2 consecutive items x_i and x_{i+1}. W.L.O.G., we assume x_0 is the first item, and x_t is the last item in this data sequence. Obviously, we have $k_i \geq 0$ and $\sum_{i=1}^{t-1} k_i + t = N$.

Theorem 5. *The probability of finding x a frequent item is*

$$\mathbb{P}(found) = 1 - \prod_{i=1}^{t-1}\left(1 - \left(1 - \frac{1}{M}\right)^{k_i}\right). \tag{22}$$

Proof. Since frequencies of items other than x is 1, once the frequency of x in the medium part is equal or greater than 2, x will not be replaced anymore. It will grow bigger and bigger and the medium part will find out it is a frequent item. When x_1 comes, it will be inserted into a random entry. Afterward, every item (k_1 items in total) before x_2 will bring out a replacement of a random entry. The probability of replacing x is $\frac{1}{M}$ for every replacement. So the probability of that x is not replaced by other items until x_2 is $(1 - \frac{1}{M})^{k_1}$. The probability of replacing x_1 is $1 - (1 - \frac{1}{M})^{k_1}$.

If x_1 is replaced by other item before x_2 comes, the medium part can not distinguish item x as a frequent item when x_2 comes. Instead, x_2 will bring out a replacement and insert item x with frequency 1 into a random entry. The situation between x_2 and x_3 is the same as the situation between x_1 and x_2. The probability of that x is replaced by other items before x_3 is $1 - (1 - \frac{1}{M})^{k_2}$. When x_3 comes, the probability for the medium part not to find out x a frequent item is $(1 - (1 - \frac{1}{M})^{k_1}) \times (1 - (1 - \frac{1}{M})^{k_2})$. Similarly, the probability for the medium part not to find out x a frequent item after the last item x_t comes is

$$\mathbb{P}(replace\ all) = \prod_{i=1}^{t-1}\left(1 - \left(1 - \frac{1}{M}\right)^{k_i}\right). \tag{23}$$

Therefore, the probability of finding x as a frequent item is

$$\mathbb{P}(found) = 1 - \mathbb{P}(replace\ all) = 1 - \prod_{i=1}^{t-1}\left(1 - \left(1 - \frac{1}{M}\right)^{k_i}\right). \tag{24}$$

□

Now we consider the error of the frequent part produced by the medium part. Define $\hat{t}$ is the frequency of x recorded in the medium part or the frequent part. The error $er = t - \hat{t}$. Note that the frequent part and the medium part never provide over estimation so that $er \geq 0$.

Theorem 6. $\mathbb{E}(er) = \sum_{j=1}^{t} j \times \prod_{i=1}^{j+1}\left(1 - (1 - \frac{1}{M})^{k_i}\right)$.

Proof. After the frequency of x grows to 2, the following item x will not produce any error. If x_i makes the frequency grow to 2, the error $er = i - 2$. Therefore, we have $\mathbb{P}(er = j), j = 0, ..., t-2$

$$\mathbb{P}(er = j) = \prod_{i=1}^{j+1}\left(1 - \left(1 - \frac{1}{M}\right)^{k_i}\right). \tag{25}$$

Hence, the expectation of er

$$\mathbb{E}(er) = \sum_{j=1}^{t} j \times \mathbb{P}\left(er = j\right) = \sum_{j=1}^{t} j \times \prod_{i=1}^{j+1}\left(1 - \left(1 - \frac{1}{M}\right)^{k_i}\right). \tag{26}$$

□

Theorem 7. *If the frequency of item x follows Poisson distribution $\mathbb{P}(\frac{t}{N})$, $\mathbb{E}(er) = \left(ln\frac{M}{M-1} \cdot \frac{N}{t}\right)^2$.*

Proof. If the frequency of item x follows Poisson distribution $\mathbb{P}(\frac{t}{N})$, the interval k_i between x_i and x_{i+1} follows exponential distribution $E(\frac{t}{N})$. Therefore, we have $k_i \sim E(\lambda) = E(\frac{t}{N})$ and k_i is independent from k_j for $\forall i \neq j$.

Hence, we have

$$\begin{aligned}\zeta =&\mathbb{E}\left(1-\left(1-\frac{1}{M}\right)^{k_i}\right) = 1-\mathbb{E}\left(1-\frac{1}{M}\right)^{k_i}\\ =&1-\int_0^{\infty}\left(1-\frac{1}{M}\right)^{k_i}\lambda e^{-\lambda k_i}\mathrm{d}k_i = \frac{\ln M-\ln(M-1)}{\ln M-\ln(M-1)+\frac{t}{N}}.\end{aligned} \tag{27}$$

Since $\zeta < 1$, we have

$$\begin{aligned}\mathbb{E}(er) &= \sum_{j=1}^{t} j\times\mathbb{P}(er=j) = \sum_{j=1}^{t} j\times\zeta^{j+1}\\ &\approx \frac{\zeta^2}{(1-\zeta)^2} = \left(\ln\frac{M}{M-1}\cdot\frac{N}{t}\right)^2.\end{aligned} \tag{28}$$

□

According to Equation (28), if the number of entries in a bucket (M) is bigger or the proportion of frequent items to all items is bigger (t/N), $\mathbb{E}(er)$ will be smaller.

4.4 Analysis on Fingerprint

The fingerprints in the medium part and the frequent part are used to identify different items and reduce memory costs. However, fingerprints bring about the problem of fingerprint collisions. If two or more items, especially frequent items, have the same fingerprint, the accuracy of the estimation will be degraded a lot. In this section, we analyze how fingerprints affect the accuracy of the inner-product estimation. Consider a set of N items. A_i denotes the random event that there is no fingerprint collision among i distinct independent items.

Lemma 8. *If the length of the fingerprint is l, the probability of fingerprint collision between two items is $\mathbb{P}(A_2) = 1-\frac{2^l}{(2^l)^2}$.*

Theorem 9. *The probability of no fingerprint collision between N items is $\mathbb{P}(A_N) = \frac{\prod_{i=1}^{N}(2^l-i)}{2^{lN}}$.*

Proof. We derive $\mathbb{P}(A_N)$ from the formula of conditional probability. We have

$$\begin{aligned}\mathbb{P}(A_N) &= \mathbb{P}(A_{N-1})\times\mathbb{P}\left(A_N|A_{N-1}\right) = \mathbb{P}(A_{N-1})\times\frac{2^l-N}{2^l}\\ &= \frac{\prod_{i=1}^{N}\left(2^l-i\right)}{2^{lN}}.\end{aligned} \tag{29}$$

□

Table 2: The probability of no hash collision.

Probability	$N=4$	$N=16$	$N=64$	$N=128$
$l=16$	$1.5E-04$	$2.0E-03$	$3.1E-02$	$1.2E-01$
$l=32$	$2.3E-09$	$3.2E-08$	$4.8E-07$	$1.9E-06$

As shown in Table 2, the probability of fingerprint collisions is rather small when the length of fingerprint $l = 32$. If fingerprint collisions occur, JoinSketch will regard two or more items as the same item. In this case, the reported inner-product estimation is expected to be slightly larger. Such an error is small and can be much smaller when using more bits for the fingerprints.

5 APPLICATIONS

JoinSketch is proposed for accurate and fast inner-product estimation in data stream scenarios. In Section 5.1, we describe the applications of JoinSketch in data stream scenarios. JoinSketch can be applied in more one-pass scenarios. In Section 5.2 and Section 5.3, we discuss how to apply JoinSketch in one-pass scenarios of database and cosine similarity.

5.1 Applications in Data Stream Processing

The inner-product of data streams is an important statistic for data stream processing. For example, we need to analyze the correlation between two data streams in many large-scale network measurement systems. To be specific, tracking the join size of abnormal traffic on several routers can help network administrators analyze the current running status of the network system. If a link failure happens, a practical network measurement system should be able to allow us to locate the link failure as soon as possible. In the scenario, the data stream processing is proposed to be real-time and fast enough. JoinSketch is suitable for data streams' inner-product estimation. We can deploy JoinSketch in measurement nodes (e.g., IP routers). The function of JoinSketch is to provide key statistics of data flows through the router and send the measurement result to the controller node. The estimation of the inner-product will then be used to analyze the real-time running status of the network.

5.2 Applications in Database

Inner-product estimation is an essential step in multi-way join. Most systems perform multi-way join by binary join algorithms, *i.e.*, they iteratively select two tables and join them into intermediate relations. However, a poor join plan may lead to a large volume of intermediate relations and result in high computation overhead. Therefore, many existing solutions [16, 20, 28, 29] present to use sketches to estimate join size in advance and avoid poor plans. JoinSketch supports join size estimation. Given two tables and join predicates, we build a JoinSketch for each table. Then, we estimate the inner-product of the two tables as the join size.

5.3 Applications in Cosine Similarity

Cosine similarity is a key metric in many fields of data science, including data mining, natural language processing, recommendation systems and so on. Cosine similarity computation, however, is often the bottleneck in some applications with massive volumes of data. Fortunately, it is acceptable to use the estimated cosine similarity instead of the true value in some cases. For example, researchers propose to estimate cosine similarity of data streams using the AGMS sketch in [30].

JoinSketch can be also applied to estimate the cosine similarity in data stream scenarios. To be specific, cosine similarity can be derived from inner-product as shown below:

$$cos(F,G) = \frac{f\odot g}{\sqrt{(f\odot f)(g\odot g)}}. \tag{30}$$

After constructing JoinSketch for F and G, we can derive cosine similarity using three inner-product estimations.

6 EXPERIMENTAL RESULTS

In this section, we provide experimental results of JoinSketch. We present experimental setup in Section 6.1. First, data stream scenarios are the main and the most critical scenarios of JoinSketch. We show the performance of JoinSketch in data stream scenarios compared with prior arts. Second, we demonstrate a few properties of JoinSketch itself through experiments, including stability and throughput. Third, we show the performance of JoinSketch in finding frequent items and frequency estimation. Finally, we analyze the influence of parameters and give recommended settings. We show that JoinSketch has an advantage over existing algorithms when the data is skewed and the memory is limited.

6.1 Experimental Setup

6.1.1 Datasets.

1) CAIDA dataset: CAIDA Anonymized Internet Trace [31] is a data stream of anonymized IP trace collected in 2018. Each item is identified by its source IP (4 bytes) and destination IP (4 bytes).

2) TPC-DS dataset: The TPC Benchmark™ DS (TPC-DS) [32] is a decision support benchmark that models several generally applicable aspects of a decision support system, including queries and data maintenance. The benchmark provides a representative evaluation of the System Under Test's (SUT) performance as a general-purpose decision support system.

3) MovieLens dataset: The MovieLens datasets [33] are widely used in education, research, and industry. These datasets are a product of member activity in the MovieLens movie recommendation system, an active research platform that has hosted many experiments since its launch in 1997.

4) Zipf datasets: We generate synthetic datasets of Zipf distribution with different parameters and every dataset contains 32,000,000 items in completely random order.

5) Zipf with shifting: Two Zipf datasets with the same distribution will have frequent items with the same id. A shifting of k means that the i^{th} most frequent item in the original dataset is the $(i+k)\%N^{th}$ most frequent item in the shifted dataset, where N is the number of distinct items. We use a pair of Zipf dataset and shifted Zipf dataset to evaluate the impact of different correlations. The larger the shifting is, the less correlated the datasets are.

6) Zipf with different data arrival orders: In the scenarios of traditional database systems, the physical layout of data is often ordered, semi-ordered, or clustered, which leads to different data arrival orders. We reshuffle the Zipf datasets to generate datasets of different data arrival orders. We sort the items by their ID to generate ordered datasets. We swap every item with one of the 100 items closest to it based on the ordered datasets to generate clustered datasets. We swap every item with one of the 1000 items closest to it based on the ordered datasets to generate semi-ordered datasets.

6.1.2 Platform and implementation. We evaluate all algorithms on a server with 18-core CPUs (36 threads, Intel CPU i9-10980XE @3.00 GHz) with 128GB 3200MHz DDR4 memory and 24.75MB L3 cache. We implement all algorithms with C++ and build them with g++ 7.5.0 (Ubuntu 7.5.0-6ubuntu2) and the -O3 option. The hash functions we use are 32-bit Murmur Hash [34].

6.1.3 Metrics.

1) Absolute Error (AE): $\frac{1}{\Psi}\sum\left|J-\hat{J}\right|$, where J is the true value of inner-product, $\hat{J}$ is the estimated value, and Ψ is the number of testing rounds.

2) Relative Error (RE): $\frac{1}{\Psi}\sum\left|J-\hat{J}\right|/J$.

3) Variance (Var): $\frac{1}{\Psi}\sum\left(J-\hat{J}\right)^2$. We use variance to measure the stability of the algorithm.

4) Throughput (Mops): Million operations per second.

5) Maximum Absolute Error and Minimum Absolute Error: We use them to measure the algorithm's best- and worst-case errors.

6) Precision Rate (PR): $\frac{\text{Reported top-}k}{\text{Reported items}}$. We use the precision rate to evaluate the ability to find frequent items.

7) Average Relative Error (ARE): $\frac{1}{|\mathbf{K}|}\sum_{e_{\beta_i}\in\mathbf{K}}|f_{e_{\beta_i}}-\hat{f_{e_{\beta_i}}}|/f_{e_{\beta_i}}$, where $f_{e_{\beta_i}}$ is the real frequency of item e_{β_i}, $\hat{f_{e_{\beta_i}}}$ is the estimated frequency of e_{β_i}, and $\mathbf{K}$ is the query set.

8) Average Absolute Error (AAE): $\frac{1}{|\mathbf{K}|}\sum_{e_{\beta_i}\in\mathbf{K}}|f_{e_{\beta_i}}-\hat{f_{e_{\beta_i}}}|$, where $\mathbf{K}$, $f_{e_{\beta_i}}$ and $\hat{f_{e_{\beta_i}}}$ are the same as those defined in ARE.

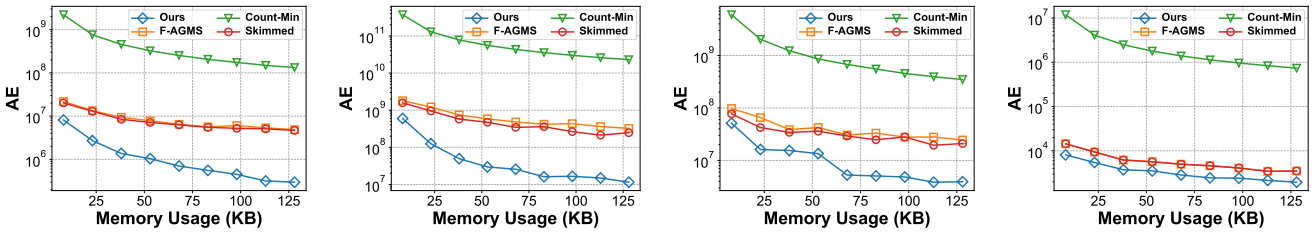
6.1.4 Parameter setting.

We compare JoinSketch with the Fast-AGMS sketch [19] (F-AGMS for short), the Count-Min sketch [35, 36], and the Skimmed sketch [20]. We set the number of hashes $d = 3$ by default, and all results are averaged over 50 runs with different hash seeds. For the CAIDA dataset, we set the threshold T to 400 for both JoinSketch and the Skimmed sketch. For the Zipf datasets, the threshold we set is adjusted according to the distribution's parameter α.

6.2 Experiments on Accuracy

Impact of memory size (Figure 3): We evaluate the accuracy of JoinSketch and its competitors. The memory allocated to these algorithms ranges from 8KB to 128KB on each dataset. We find that JoinSketch always achieves the best accuracy. The Count-Min sketch needs more memory to work well, while in practice, we don't always have that much memory. The accuracy of the Fast-AGMS sketch and the Skimmed sketch is comparable. Because the separated frequent items are not accurate, the performance of the Skimmed sketch is always inferior to JoinSketch, and even sometimes inferior to the Fast-AGMS sketch. Compared to the Skimmed sketch, the error of the frequent items found by JoinSketch is so small that JoinSketch achieves the best performance. When the memory is large enough, the average absolute error of JoinSketch is up to 15× smaller than that of the Fast-AGMS sketch. On average, JoinSketch is 10 times better than the Fast-AGMS sketch.

In addition to the CAIDA and the generated Zipf datasets, we also conduct experiments on the TPC-DS dataset and MovieLens dataset, which corresponds to the applications in database systems and cosine similarity mentioned in Section 5.2 and Section 5.3. These two datasets are in the form of key-value pairs, where key is the data item and the value is the frequency of the item. The experimental results show that JoinSketch can still achieve good performance. The TPC-DS dataset is a benchmark for database systems that has a bit of skewness. Therefore, JoinSketch still performs well since the frequent items are separated from the infrequent items. We use the Movielens dataset to evaluate the performance of JoinSketch



(a) AE on CAIDA. (b) AE on Zipf ($\alpha = 0.8$). (c) AE on TPC-DS. (d) AE on MovieLens.

Figure 3: AE on various datasets.

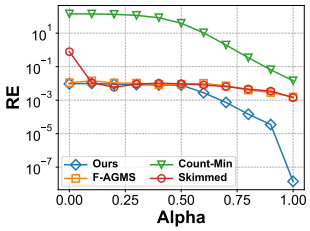


Figure 4: Impact of dataset skewness.

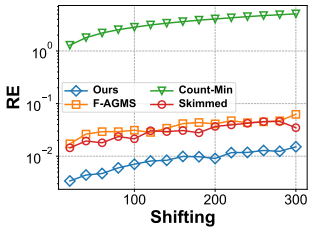


Figure 5: Impact of dataset correlation.

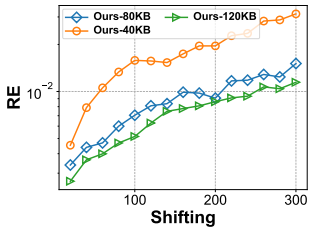


Figure 6: Dataset correlation v.s. memory usage.

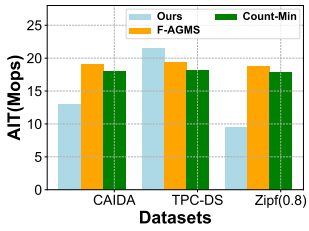
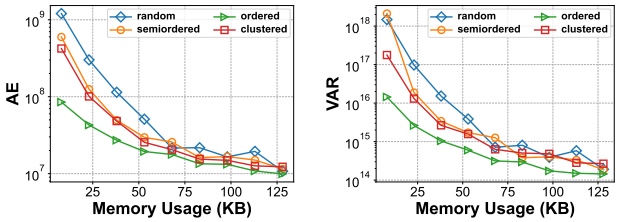


Figure 7: Throughput.

on cosine similarity estimation as in [30]. The dataset consists of movie ratings on a 5-star scale, with half-star increments. As shown in Figure 3(d), the absolute error of JoinSketch is small. This dataset only has ratings from 0 to 5 in the vector, but even for such a dataset with little skewness, JoinSketch still outperforms the Fast-AGMS sketch and the Count-Min sketch. On these datasets, the Skimmed sketch has almost no optimization effect.

Impact of dataset skewness (Figure 4): To simulate data with different skewness, we conduct experiments on the datasets of Zipf distribution with alpha ranging from 0.0 to 0.9. We fixed the memory size of all algorithms to 80KB. We use RE instead of AE because the true value of the inner-product changes as the alpha changes. We find that as the skewness of the dataset increases, the RE of each algorithm decreases. This is because of the nature of the Zipf distribution. As alpha increases, the number of distinct items in the data stream decreases, and the true value of the inner-product increases considerably. Both of the above factors will lead to smaller RE. From Figure 4, we can clearly see that JoinSketch achieves the best accuracy. The higher the skewness of the dataset is, the greater the advantage of JoinSketch is.

Impact of dataset correlations (Figure 5 and Figure 6): We generate two Zipf datasets with $\alpha = 0.8$ and shift one of them by different parameters between 0 and 300. The memory used by the algorithms is fixed to 80KB and the accuracy is measured with RE. The experimental results show that when the datasets are less correlated, the accuracy of all algorithms becomes lower, which indicates that the inner-product estimation becomes more difficult. More shifting degrades the accuracy of JoinSketch because when the intersection between the frequent items of two datasets is small, the benefit of separating frequent items and infrequent items is small. JoinSketch, however, always achieves the best accuracy among competitors. We also study the impact of memory usage and data correlations together. Figure 6 shows that more memory usage improves the accuracy of JoinSketch while the impact of dataset correlations remains the same.



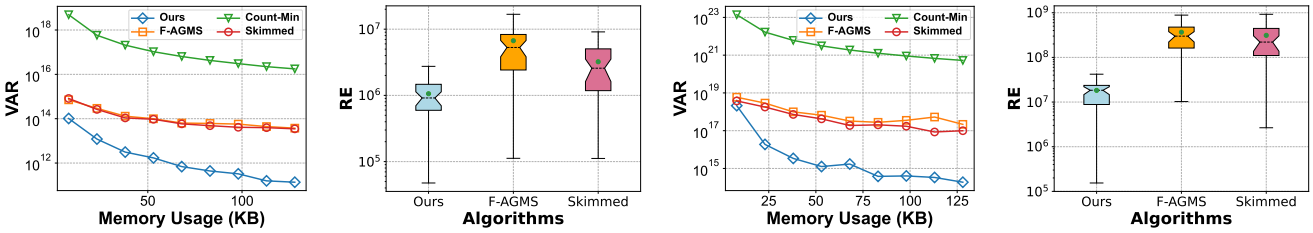
(a) AE. (b) Variance.

Figure 8: Impact of data arrival orders.

Impact of data arrival orders: We study the impact of data arrival orders on JoinSketch because the data arrival skew will affect the effectiveness of separating the frequent items for JoinSketch. As shown in Figure 8, data arrival orders affect the performance of JoinSketch. The random dataset is the worst case because many instances of frequent items may be evicted to the infrequent part before they grow frequent. JoinSketch achieves the best accuracy on the ordered dataset because when the dataset is ordered, JoinSketch can find frequent items efficiently. The accuracy of JoinSketch on clustered and semi-ordered datasets is between the ordered and random ones, which is consistent with how much ordered these datasets are. More ordered datasets lead to better performance of JoinSketch.

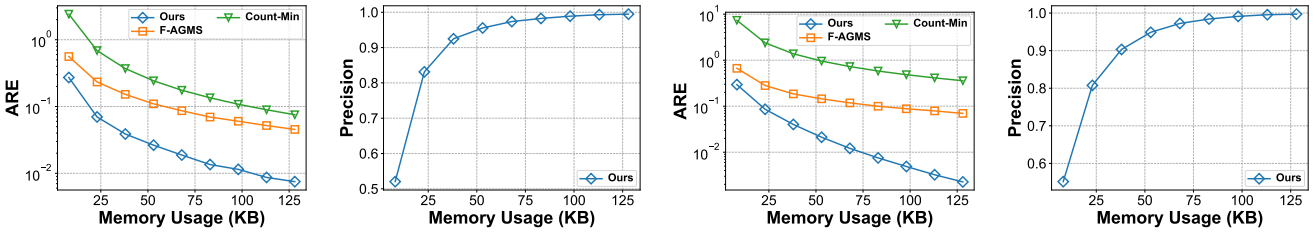
6.3 Experiments on Stability

An important indicator to measure the quality of an algorithm is the variance because the algorithm with less variance leads to more stability. We evaluate the variance of JoinSketch and its competitors for 100 rounds on the CAIDA and Zipf datasets with fixed memory of 80KB. The results are shown in Figure 9. From Figure 9(a) and Figure 9(c), we can see that JoinSketch has a clear advantage on



(a) Variance on CAIDA. (b) Stability on CAIDA. (c) Variance on Zipf (α = 0.8) (d) Stability on Zipf (α = 0.8)

Figure 9: Experiments on stability.



(a) ARE on CAIDA. (b) Precision on CAIDA. (c) ARE on Zipf (α=0.8). (d) Precision on Zipf (α = 0.8).

Figure 10: Accuracy of finding frequent items.

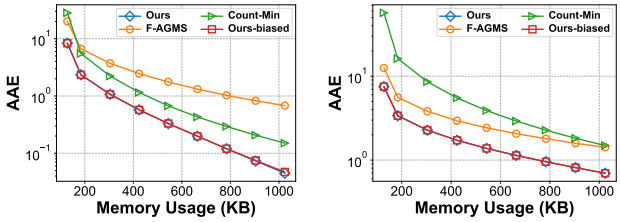
the variance. Besides the variance, Figure 9(b) and Figure 9(d) show that JoinSketch also performs better than other algorithms in terms of max, min, and median. Benefiting from separating the frequent items, the stability of JoinSketch for inner-product estimation are significantly better than other algorithms.

6.4 Experiments on Throughput

We evaluate the throughput of each algorithm on different datasets. The operation of JoinSketch is more complex, so it generally takes more time to insert an item than other algorithms. But with SIMD optimization, JoinSketch can sometimes achieve better throughput. As shown in Figure 7, we can see that JoinSketch, the Fast-AGMS sketch, and the Count-Min sketch have comparable throughput. The Skimmed sketch is not a one-pass algorithm, so we don't compare its throughput with other algorithms.

6.5 Experiments on Finding Frequent Items

Finding frequent items is a fundamental problem in data stream processing. JoinSketch is not specific to this problem, but can report frequent items approximately using the frequent part. We conducted two experiments related to finding frequent items, and the two experiments can also reflect the efficiency of our algorithm on separating the frequent items and the infrequent items. First, we query the frequency of the 500 most frequent items in the CAIDA and Zipf (α = 0.8) datasets. As shown in Figure 10(a) and Figure 10(c), the ARE of JoinSketch is the smallest, which means that JoinSketch performs very well in the frequency estimation of the 500 most frequent items. We then evaluate how many items of the top 500 frequent items appear in the frequent part, and Figure 10(b) and Figure 10(d) show that JoinSketch is very accurate in finding the frequent items. In fact, when the memory is more than 30KB, the precision can reach more than 90%, which means that most of the frequent items are successfully classified into the frequent part.



(a) AAE on CAIDA. (b) AAE on Zipf (α = 0.8).

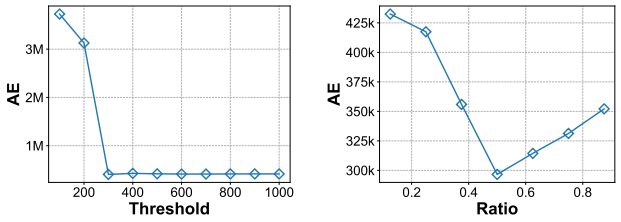
Figure 11: Frequency estimation.

6.6 Experiments on Frequency Estimation

Figure 11 shows the experimental results of frequency estimation on CAIDA and Zipf (α = 0.8). The line of Ours-biased represents the lookup operation which does not query an item in the infrequent part if it exists in the frequent part or the medium part. JoinSketch outperforms the Count-Min sketch and the Fast-AGMS sketch in terms of frequency estimation. The performance of JoinSketch is nearly the same as JoinSketch-biased, which means that few instances of frequent items are evicted to the infrequent part compared with the instances recorded in the frequent part and the medium part.

6.7 Experiments on Parameters

Impact of Threshold T (Figure 12(a)): JoinSketch needs to set a threshold T. It affects which items go into the frequent part. In fact, threshold T is strongly related to the skewness of the dataset. Our goal is to filter out the frequent items in the dataset. On the one hand, if the threshold T is small, some infrequent items will be considered as frequent items, which will downgrade the accuracy of JoinSketch. On the other hand, if the threshold T is set too large, frequent items will be considered infrequent and the counters in the medium part may overflow, which will also downgrade the accuracy



(a) Effect of Threshold T. (b) Effect of memory allocation.

Figure 12: Impact of different parameters of JoinSketch.

of JoinSketch. Figure 12(a) shows the results of JoinSketch on the CAIDA dataset with threshold T ranging from 100 to 1000. The results are in line with the above discussion. When the threshold is greater than 300, the difference is not big. This is because the error of items with a frequency greater than 300 in the CAIDA dataset entering the frequent part is relatively small. The reason that there is no overflow caused by a large threshold is that the size of the counter in the medium part is set to 10 bits in our algorithm optimization. The threshold is less than 1024, and there will be no overflow on the CAIDA dataset.

Impact of memory allocation (Figure 12(b)): The memory used by JoinSketch is divided into three parts. The memory of the frequent part should be positively related to the number of frequent items, so it actually depends on the threshold T. Assuming the memory of frequent part is set according to the threshold T, we consider only the size ratio of the medium part and the infrequent part. We evaluate the performance of JoinSketch with $\frac{\text{Medium Part size}}{\text{Total size}}$ ranging from $\frac{1}{8}$ to $\frac{7}{8}$ on the CAIDA dataset. As shown in fig. 12(b), we can find that when the memory of the medium part is small, the error is large because the size of the medium part affects the effectiveness of finding the frequent items. When the infrequent part is too small, the error of the infrequent items due to hash collisions can also lead to inaccurate estimation. Therefore, we should set a moderate memory ratio of the medium part to the infrequent part.

7 RELATED WORK

This section first discusses related work for the inner-product estimation and the join size estimation and then presents algorithms for finding frequent items and sketch algorithms designed for skewed data streams.

7.1 Inner-product Estimation

There are three mainstream inner-product size estimation techniques in the literature: histograms, sampling techniques, and sketches.

7.1.1 Histograms. Histograms[37–40] are common column statistics that provide information about the data distribution of column data in database. It divides the domain of an attribute into several buckets and assumes a uniform distribution within each bucket.

7.1.2 Sampling Techniques. Sampling techniques[41] are widely used in inner-product/join size estimation. The cross-product sampling scheme[41] is believed to give the best estimation out of the simple sampling schemes. However, sampling techniques are sensitive to skewed and sparse data, while skewed data are common in real scenarios. To address this drawback, researchers propose Bifocal[42] sampling algorithm and End-biased[43] sampling algorithm. Correlated sampling is proposed in [44], which is a part of CS2 algorithm and [45] improved correlated sampling.

7.1.3 Sketches. Sketches [35, 36, 46] are especially appropriate for the scenarios of data streams. There are several pieces of research focusing on the inner-product estimation using sketches. The basic AGMS sketch is first presented in [17, 18]. Dobra *et al.* [47, 48] extends AGMS to multi-way join size estimation. The Fast-AGMS sketch [19] preserves a matrix of basic AGMS counters to improve accuracy and efficiency simultaneously and achieves the best performance according to [27, 36]. The Skimmed sketch[20] and the Red sketch [21] propose to estimate the inner-product by estimating the inner-product of frequent items and infrequent items separately. The Skimmed sketch first builds a Fast-AGMS sketch for a data stream. Then it goes through the domain of the data stream to find frequent items. The Skimmed sketch and the Red sketch need to go through the domain of the data stream before estimating the inner-product, which means these multi-pass techniques are not practical. [28] extends the sketch-based method to join sketch estimation subject to filters. [16] proposes an online query optimizer exclusively based on sketches in a real database system and proposes to reorganize arrays of counters into a matrix to support multi-way join using the Fast-AGMS sketch. [29] introduces bound sketches that provide theoretical upper bounds for cardinality estimation.

7.2 Finding Frequent Items

There are many solutions in finding frequent items, including Misra-Gries algorithm [49], Lossy counting [50], SpaceSaving [4], Unbiased SpaceSaving [51], *e.t.c.* [5]. They report high accuracy for the frequent items but report 0 for the infrequent items and are thus not suitable for inner-product estimation.

7.3 Separating Frequent and Infrequent Items

Real data often obeys unbalanced data distribution such as Zipf [22, 23]. There are a number of sketch algorithms that record frequent and infrequent items separately in the literature, such as ASketch [25], ColdFilter [26], ElasticSketch [24], and so on [10, 52–57]. After adjustment to the scenarios of inner-product, they can be applied to estimate inner-product of data streams. However, the adjustment may require a lot of hard work because these algorithms are not particularly designed for inner-product estimation. Most of them cannot provide an unbiased estimation.

8 CONCLUSION

This paper proposes an algorithm called JoinSketch for inner-product estimation. It can provide accurate, fast, and unbiased inner-product estimation for data streams. By separating frequent and infrequent items, JoinSketch improves the accuracy of inner-product estimation, especially when the data is highly-skewed. We prove mathematically the unbiasedness of inner-product estimation given by JoinSketch and that it has lower variance than the prior art, Fast-AGMS sketch. We conduct extensive experiments on various real-world and synthetic datasets. Our experimental results show that JoinSketch maintains unbiasedness, and the error is 10 times on average smaller than the state-of-the-art on high-skewed datasets. JoinSketch outperforms the state-of-the-art with respect to both accuracy and stability.

REFERENCES

[1] Related source code. https://github.com/JoinSketch/JoinSketch.

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

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

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

[5] Ben Basat Ran, Gil Einziger, Roy Friedman, and Yaron Kassner. Heavy hitters in streams and sliding windows. In *Proc. IEEE INFOCOM 2016*, 2016.

[6] K. Balachander, S. Subhabrata, Z. Yin, and C. Yan. Sketch-based change detection: methods, evaluation, and applications. In *Proceedings of the 3rd ACM SIGCOMM conference on Internet measurement*, pages 234–247. ACM, 2003.

[7] Robert Schweller, Zhichun Li, Yan Chen, et al. Reversible sketches: enabling monitoring and analysis over high-speed data streams. *IEEE/ACM Transactions on Networking (ToN)*, 15(5):1059–1072, 2007.

[8] Philippe Flajolet, Éric Fusy, Olivier Gandouet, and Frédéric Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. In *Discrete Mathematics and Theoretical Computer Science*, pages 137–156. Discrete Mathematics and Theoretical Computer Science, 2007.

[9] S. Venkataraman, D. Xiaodong Song, P. B. Gibbons, and A. Blum. New streaming algorithms for fast detection of superspreaders. In *NDSS*, 2005.

[10] Zheng Zhong, Shen Yan, Zikun Li, Decheng Tan, Tong Yang, and Bin Cui. Burstsketch: Finding bursts in data streams. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2375–2383, 2021.

[11] Yinda Zhang, Jinyang Li, Yutian Lei, Tong Yang, Zhetao Li, Gong Zhang, and Bin Cui. On-off sketch: A fast and accurate sketch on persistence. *Proceedings of the VLDB Endowment*, 14(2):128–140, 2020.

[12] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proceedings of the VLDB Endowment*, 9(3):204–215, 2015.

[13] Viktor Leis, Bernhard Radke, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. Query optimization through the looking glass, and what we found running the join order benchmark. *The VLDB Journal*, 27(5):643–668, 2018.

[14] Shumo Chu, Magdalena Balazinska, and Dan Suciu. From theory to practice: Efficient join query evaluation in a parallel database system. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*, pages 63–78, 2015.

[15] Kyle Deeds, Dan Suciu, Magda Balazinska, and Walter Cai. Degree sequence bound for join cardinality estimation. *arXiv preprint arXiv:2201.04166*, 2022.

[16] Yesdaulet Izenov, Asoke Datta, Florin Rusu, and Jun Hyung Shin. Compass: Online sketch-based query optimization for in-memory databases. In *Proceedings of the 2021 International Conference on Management of Data*, pages 804–816, 2021.

[17] Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. *Journal of Computer and system sciences*, 58(1):137–147, 1999.

[18] Noga Alon, Phillip B Gibbons, Yossi Matias, and Mario Szegedy. Tracking join and self-join sizes in limited storage. *Journal of Computer and System Sciences*, 64(3):719–747, 2002.

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

[20] Sumit Ganguly, Minos Garofalakis, and Rajeev Rastogi. Processing data-stream join aggregates using skimmed sketches. In *International Conference on Extending Database Technology*, pages 569–586. Springer, 2004.

[21] Sumit Ganguly, Deepanjan Kesh, and Chandan Saha. Practical algorithms for tracking database join sizes. In *International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 297–309. Springer, 2005.

[22] David MW Powers. Applications and explanations of Zipf's law. In *Proc. EMNLP-CoNLL*. Association for Computational Linguistics, 1998.

[23] Lada A Adamic and Bernardo A Huberman. Power-law distribution of the world wide web. *science*, 287(5461):2115–2115, 2000.

[24] 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 *Proceedings of the 2018 Conference of the ACM Special Interest Group on Data Communication*, pages 561–575, 2018.

[25] Pratanu Roy, Arijit Khan, and Gustavo Alonso. Augmented sketch: Faster and more accurate stream processing. In *Proceedings of the 2016 International Conference on Management of Data*, pages 1449–1463, 2016.

[26] Yang Zhou, Tong Yang, Jie Jiang, Bin Cui, Minlan Yu, Xiaoming Li, and Steve Uhlig. Cold filter: A meta-framework for faster and more accurate stream processing. In *SIGMOD Conference*, 2018.

[27] 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.

[28] David Vengerov, Andre Cavalheiro Menck, Mohamed Zait, and Sunil P Chakkappen. Join size estimation subject to filter conditions. *Proceedings of the VLDB Endowment*, 8(12):1530–1541, 2015.

[29] Walter Cai, Magdalena Balazinska, and Dan Suciu. Pessimistic cardinality estimation: Tighter upper bounds for intermediate join cardinalities. In *Proceedings of the 2019 International Conference on Management of Data*, pages 18–35, 2019.

[30] Konstantin Kutzkov, Mohamed Ahmed, and Sofia Nikitaki. Weighted similarity estimation in data streams. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 1051–1060, 2015.

[31] The CAIDA UCSD Anonymized Internet Traces 2018. https://www.caida.org/catalog/datasets/passive_dataset/.

[32] Meikel Poess. *TPC-DS*, pages 1–8. Springer International Publishing, Cham, 2018.

[33] F. M. Harper and J. A. Konstan. The movielens datasets. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2015.

[34] Murmur hashing source codes. https://github.com/aappleby/smhasher/blob/master/src/MurmurHash3.cpp.

[35] Graham Cormode, Minos Garofalakis, Peter J Haas, Chris Jermaine, et al. Synopses for massive data: Samples, histograms, wavelets, sketches. *Foundations and Trends® in Databases*, 4(1–3):1–294, 2011.

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

[37] Yannis E Ioannidis and Stavros Christodoulakis. On the propagation of errors in the size of join results. In *Proceedings of the 1991 ACM SIGMOD International Conference on Management of data*, pages 268–277, 1991.

[38] Viswanath Poosala and Yannis E Ioannidis. Selectivity estimation without the attribute value independence assumption. In *VLDB*, volume 97, pages 486–495. Citeseer, 1997.

[39] Yannis E Ioannidis and Stavros Christodoulakis. Optimal histograms for limiting worst-case error propagation in the size of join results. *ACM Transactions on Database Systems (TODS)*, 18(4):709–748, 1993.

[40] Yannis E Ioannidis and Viswanath Poosala. Balancing histogram optimality and practicality for query result size estimation. *Acm Sigmod Record*, 24(2):233–244, 1995.

[41] Peter J Haas, Jeffrey F Naughton, S Seshadri, and Arun N Swami. Fixed-precision estimation of join selectivity. In *Proceedings of the twelfth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, pages 190–201, 1993.

[42] Sumit Ganguly, Phillip B Gibbons, Yossi Matias, and Avi Silberschatz. Bifocal sampling for skew-resistant join size estimation. In *Proceedings of the 1996 ACM SIGMOD international conference on management of data*, pages 271–281, 1996.

[43] Cristian Estan and Jeffrey F Naughton. End-biased samples for join cardinality estimation. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 20–20. IEEE, 2006.

[44] Feng Yu, Wen-Chi Hou, Cheng Luo, Dunren Che, and Mengxia Zhu. Cs2: a new database synopsis for query estimation. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, pages 469–480, 2013.

[45] TaiNing Wang and Chee-Yong Chan. Improved correlated sampling for join size estimation. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, pages 325–336. IEEE, 2020.

[46] Peiqing Chen, Dong Chen, Lingxiao Zheng, Jizhou Li, and Tong Yang. Out of many we are one: Measuring item batch with clock-sketch. In *Proceedings of the 2021 International Conference on Management of Data*, pages 261–273, 2021.

[47] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Processing complex aggregate queries over data streams. In *Proc. ACM SIGMOD*, pages 61–72. ACM, 2002.

[48] Alin Dobra, Minos Garofalakis, Johannes Gehrke, and Rajeev Rastogi. Sketch-based multi-query processing over data streams. In *International Conference on Extending Database Technology*, pages 551–568. Springer, 2004.

[49] Jayadev Misra and David Gries. Finding repeated elements. *Science of computer programming*, 2(2):143–152, 1982.

[50] Gurmeet Singh Manku and Rajeev Motwani. Approximate frequency counts over data streams. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pages 346–357. Elsevier, 2002.

[51] Daniel Ting. Data sketches for disaggregated subset sum and frequent item estimation. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1129–1140, 2018.

[52] 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.

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

[54] Bohan Zhao, Xiang Li, Boyu Tian, Zhiyu Mei, and Wenfei Wu. 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*, pages 2285–2293, 2021.

[55] 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.

[56] 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.

[57] Yuanpeng Li, Xiang Yu, Yilong Yang, Yang Zhou, Tong Yang, Zhuo Ma, and Shigang Chen. Pyramid family: Generic frameworks for accurate and fast flow size measurement. *IEEE/ACM Transactions on Networking*, 30(2):586–600, 2021.